UNIVERSITY OF CALIFORNIA,
IRVINE

Exploiting Graph Cutsets for Sampling-Based Approximations in Bayesian Networks

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Science

by

Bozhena Petrovna Bidyuk

Dissertation Committee:
Professor Rina Dechter, Chair
Professor Padhraic Smyth
Professor Max Welling

2006

The dissertation of Bozhena Petrovna Bidyuk
is approved and is acceptable in quality
and form for publication on microfilm:

_____

_____

_____

Committee Chair

University of California, Irvine
2006

To My Parents
To My Husband and My Children
To All Cats

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENT

It has been a great pleasure working with the faculty, staff, and students at the University of California, Irvine, during my tenure as a doctoral student. This work would never have been possible if it were not for the freedom I was given to pursue my own research interests, thanks in large part to the guidance, mentoring, and constructive criticism provided by my advisor and committee chair Rina Dechter. Also many thanks to David Eppstein for his input on the matters of algorithmic theory. I also wish to express my gratitude to Kris Bolcer and Milena Wypchlak, our wonderful graduate student advisors, and to all ICS faculty for creating and inspiring environment in our department, especially Mike Dillencourt, Lubomir Bic, Sandra Irani, and George Lueker. Thanks also to my fellow AI researchers at UCI including Robert Mateescu, Radu Marinescu, and Vibhav Gogate for many insightful discussions.

CURRICULUM VITAE

Bozhena Petrovna Bidyuk

1996 B.S. Summa Cum Laude, Information and Computer Science
University of California-Irvine

1998 M.S., Information and Computer Science
School of Information and Computer Science
University of California-Irvine

2006 Ph.D., Information and Computer Science
University of California-Irvine
Dissertation: “Exploiting Graph Cutsets for Sampling-Based Approximations in Bayesian Networks”
Advisor: Rina Dechter

ABSTRACT OF THE DISSERTATION

Exploiting Graph Cutsets for Sampling-Based Approximations in Bayesian Networks

By

Bozhena Petrovna Bidyuk

Doctor of Philosophy in Information and Computer Science

University of California, Irvine, 2006

Professor Rina Dechter, Chair

Automated reasoning with graphical models has found many practical applications in domains such as planning, vision, speech recognition, genetic linkage analysis, diagnostics, and many others. Graphical models, combining graph theory and probability theory, facilitate a compact and structured representation for problems with uncertainty and provide a mechanism for answering queries such as computing the probability of an event given observations.

Several exact algorithms for reasoning with graphical models exist. However, exact computation is not always possible due to prohibitive time and memory demands. In general, computing exact posterior marginals and even approximating posterior marginals within a desired degree of precision is NP-hard. In practice, we often choose methods that can quickly compute approximate answers to Bayesian queries, trading accuracy for speed. Approximation methods include algorithms for approximate inference, stochastic sampling, network simplifications (simplifying the structure of the underlying graph), and variational approximations. We often obtain a more flex-

ible computation scheme, balancing complexity and accuracy, by combining exact and approximate computation. This dissertation focuses on combining search with existing sampling and bounding methods yielding two new schemes for approximating and bounding posterior marginals in Bayesian networks. Those two new schemes, cutset sampling and any-time bounds, exploit the network structure to bound the complexity of exact computation.

Cutset sampling for computing approximate posterior marginals samples only a subset of variables, a cutset of the underlying graph. Since reducing the size of the sampling set results in lower sampling variance, cutset-based sampling converges faster than sampling on a full set of variables. Two variants of cutset sampling algorithm were developed. One, based on Gibbs sampling, is a general approach to *collapsed* Gibbs sampling in Bayesian networks. The second algorithm implements the likelihood weighting on a cutset. The proposed any-time bounds framework is an any-time scheme for computing bounds on posterior marginals. It enumerates a subset of cutset tuples and performs exact inference over these tuples and then bounds the remaining probability mass.

Both methods exploit the problem's underlying network structure to control the time and space complexity of the computations. They focus on finding a cutset of the graph such that the complexity of exact reasoning is bounded when the cutset variables are assigned. The dissertation proposes a new algorithm for finding a minimum cost cutset that yields the specified complexity bound on exact inference.

Chapter 1

Introduction and Overview

1.1 Introduction

The ability to establish cause-effect relationships is the cornerstone of human cognitive development. One of the first cause-effect relationships a child learns, for example, is that a toy falls down when released. Furthermore, the child learns that when a toy falls down, the mother rushes to pick it up. Arranging the two learned relationship into a hierarchy, the child can predict and plan a sequence of events such as invent a delightful game of throw-and-pick-up-the-toy. As time goes on, the child learns more complex relationships between the elements in the surrounding world, yet, the tendency to connect the events into a network of cause-effect relationships remains.

The attraction of Bayesian networks [96] is that they allow us to express the cause-effect relationships or the direction of influence between components of the systems using graph representation. Namely, the nodes in the graph are the variables representing the elements of the system and the edges show the direction of influence. The hierarchy of relationships in a network allows us to model interactions in large complex systems in a compact way that is easy for a human brain to relate to.

The cause-effect relationships are not always deterministic though. While the law of

gravity reinforces the child's learning that all solid objects fall down without support, he or she also discovers, for example, that the dark clouds sometimes bring the rain and sometimes pass by without a drop of water. Thus, the notion of the uncertainty of the outcome emerges. As the learned relationships become more complex, and perhaps more variables are taken into consideration, the model of the world becomes more refined and predictions more accurate. With the help of the satellite imaging and computer simulations, we can predict weather better, but the element of uncertainty usually remains as it is often either impractical or impossible to take into account "everything". Bayesian networks accommodate this inherent uncertainty in our life as well. It is expressed in the form of conditional probability tables or probability density functions describing the relationships between the variables.

Thus, Bayesian networks draw from the same fundamental principles as the human learning and knowledge representation. Unlike human brain, which takes millions of years of evolution to develop the neural connections necessary to support any new models, Bayesian networks can be learned and analyzed quickly with the aid of the computers. Providing Bayesian network as input, we can delegate to the computers the task of "reasoning" about the probability of the outcome, thus, automating the "thinking" process and enabling the processing of large volumes of data.

The only trouble remaining is that, as we refine our models of the world, the complexity of the computation spirals out of control. While the prediction of rain based on the color of the cloud might not be always accurate, it is easy to obtain. The forecast based

on the analysis of the weather system over the area is more rerfined, but may require a distributed computer simulation.

Since even powerful supercomputers have limited resources, computing exact answers in complex models becomes simply infeasible. Then, our options are to either simplify the model or to apply an algorithm that computes the answer faster, but with less precision. Both approaches have been investigated by many researchers in the last decade. This dissertation contributes to the area of research devoted to the development of approximate algorithms. It defines two new algorithms, one for approximating and one for bounding the answers to queries over Bayesian networks with discrete variables. Exploiting the structural properties of the networks and combining the search and exact computation, the proposed algorithms improves over existing methods.

In the remainder of this chapter, we define essential terminology and provide background information on Bayesian networks followed by overview of the thesis and summary of the results.

1.2 Background and Overview

In this section, we define essential terminology and provide background information on Bayesian networks. We use upper case letters without subscripts, such as X, to denote sets of variables and lower case letters without subscripts to denote an instantiation of a group of variables (e.g. x indicates that each variable in set X is assigned a value). We use an upper case letter with a subscript, such as X_i, to denote a single variable and a lower case letter

with a subscript, such as x_i, to denote an instantiated variable (e.g. x_i denotes an arbitrary value in the domain of X_i and means $X_i = x_i$). Given a set of variables $x = \{x_1, ..., x_i, ...\}$, we use $x_{-i} = x \backslash x_i$ to denote x with element x_i removed. $\mathcal{D}(X_i)$ denotes the domain of variable X_i. A superscript in a subscripted lower case letter would be used to distinguish different specific values for a variable, i.e., $\mathcal{D}(X_i) = \{x_i^1, x_i^2, ...\}$.

1.2.1 Bayesian networks

DEFINITION **1.2.1 (graph concepts)** *A* **directed graph** *is a pair $D =<V, E>$, where $V = \{X_1, ..., X_n\}$ is a set of nodes, or variables, and $E = \{(X_i, X_j) | X_i, X_j \in V\}$ is the set of edges. A directed graph is* **acyclic** *if it has no directed cycles; that is, for any node X_i, there is no nonempty directed path starting and ending on X_i.*

Given $(X_i, X_j) \in E$, X_i is called a **parent** of X_j, and X_j is called a **child** of X_i. The set of X_i's parents is denoted $pa(X_i)$, or pa_i, while the set of X_i's children is denoted $ch(X_i)$, or ch_i. The **family** of X_i includes X_i and its parents. The **moral graph** of a directed graph D is the undirected graph obtained by connecting the parents of all the nodes in D and removing the arrows. A node Y_j is a **descendant** of node X_i if there is a directed path from node X_i to node Y_j. A node Y_j is an **ancestor** of node X_i if there is a directed path from node Y_j to node X_i. A node X_i in a directed graph D is called a **root** if no edges are directed into X_i. A node X_i in a directed graph D is called a **leaf** if all of its adjacent edges are directed into X_i.

DEFINITION **1.2.2 (cycle-cutset)** *The underlying graph G of a directed graph D is the undirected graph formed by ignoring the directions of the edges in D. A* **cycle** *in an undirected graph G is a path whose two end-points coincide. A graph is* **singly connected** *(also*

called a **poly-tree***), if its underlying undirected graph has no cycles. Otherwise, it is called* **multiply connected**. *A* **cycle-cutset** *of an undirected graph is a subset of nodes in the graph that, when removed, results in a graph without cycles.*

DEFINITION **1.2.3 (loop-cutset)** *A* **loop** *in a directed graph D is a subgraph of D whose underlying graph is a cycle. A vertex v is a* **sink** *with respect to loop $\mathcal{L}$ if the two edges adjacent to v in $\mathcal{L}$ are directed into v. A vertex that is not a sink with respect to a loop $\mathcal{L}$ is called an* allowed *vertex with respect to $\mathcal{L}$. A* **loop-cutset** *of a directed graph D is a set of vertices that contains at least one allowed vertex with respect to each loop in D.*

DEFINITION **1.2.4 (belief networks)** *Let $X = \{X_1, ..., X_n\}$ be a set of random variables over multi-valued domains $\mathcal{D}(X_1), ..., \mathcal{D}(X_n)$. 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 a set of conditional probability tables (CPTs) associated with each X_i. $P(X_i|pa_i)$ is a conditional probability distribution of X_i conditional on parent instantiation pa_i. The BN represents a joint probability distribution having the product form:*

$$P(x_1, ..., x_n) = \prod_{i=1}^{n} P(x_i|x_{pa(X_i)})$$

An evidence e is an instantiated subset of variables E.

Bayesian networks facilitate a compact representation of the joint probability distribution over a set of variables X. Instead of enumerating all possible instantiations of X, we only enumerate the instantiations of parents of each node X_i in corresponding conditional probability table (CPT). Thus, the storage requirement is reduced from $d^{|X|}$ to $|X| * d^{max_i|pa(X_i)|}$ where d is the maximum domain size. A sample Bayesian network and corresponding moral graph are shown in Figure 1.1.

The structure of the directed acyclic graph reflects the dependencies between the variables. The parents of a variable X_i together with its children and parents of its children form a *Markov blanket* ma_i of node X_i. Given its Markov blanket, the probability

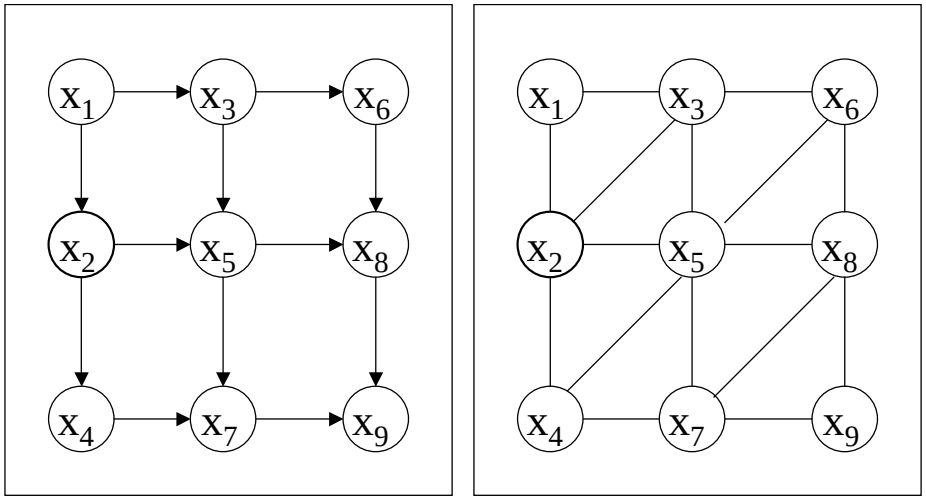


Figure 1.1: A sample Bayesian network (left) and its moral graph (right).

distribution of X_i is independent from the rest of the variables in the network. Namely, $P(x_i|x_{-i}) = P(x_i|ma_i)$. For more information see [96].

Also important in practice is the notion of a relevant subnetwork of X_i. In many algorithms, we can reduce the complexity of computation by limiting the computation to the subgraph that is the relevant subnetwork for X_i.

DEFINITION **1.2.5 (Relevant Subnetwork)** *A variable X_i in DAG G over X is* **irrelevant (barren)** *w.r.t. a subset $Z \subset X$ if $X_i \notin Z$ and X_i only has irrelevant descendants (if any). The relevant subnetwork of G w.r.t. a subset Z is the subgraph of G obtained by removing all variables that are irrelevant w.r.t Z.*

We will sometimes refer to the relevant subnetwork w.r.t. variable X_i and evidence E as the relevant subnetwork of X_i.

Other graph representations of reasoning problems (including Bayesian networks) can be useful in the analysis of the complexity of the algorithms for Bayesian networks.

DEFINITION **1.2.6 (Primal-, dual-,hypergraph of a problem)** *The* **primal** *graph G=<X,E>of a reasoning problem <X,F> has the variables X as its nodes and an arc connects two nodes if they appear in the scope of the same function $f \in F$. A* **dual** *graph of a reasoning problem has the scopes of the functions as the nodes and an arc connects two nodes if the corresponding scopes share a variable. The arcs are labelled by the shared variables. The* **hypergraph** *of a reasoning problem has the variables X as nodes and the scopes as edges. There is a one-to-one correspondence between the hypergraph and the dual graphs of a problem.*

The primary queries over Bayesian networks are:

1. *Belief Updating*: given evidence e and a variable $X_i \in X$, find the posterior probability distribution $P(X_i|e)$;

2. *Belief Revision (MPE)*: find **m**ost **p**robable **e**xplanation for evidence e of E, namely, find a maximum probability assignment to the unobserved variables $Y = X \backslash E$:

$$y \leftarrow \arg\max_y P(y|e)$$

3. *MAP*: find **m**aximum **a p***osteriori* hypothesis, namely find a maximum probability assignment to a subset of unobserved variables $Y \subset X \backslash E$ given evidence $E = e$:

$$y \leftarrow \arg\max_y P(y|e)$$

This collection of queries are often referred to as the "inference" task in Bayesian networks.

Exact inference in Bayesian networks is NP-hard [22, 111]. Furthermore, finding an approximate solution for any of the tasks above with a fixed error bound is also NP-hard [24, 4]. More recent results have demonstrated that between the three tasks, belief updating is harder than belief revision, and MAP is the hardest:

$$MPE < BeliefUpdating < MAP$$

To be exact, MPE remains NP-complete, Belief Updating was shown to be $\#P$-complete [107] as the task is similar to that of counting the number of solutions, and MAP was shown to be NP^{PP}-complete [92]. MPE and belief updating can be solved in linear time when the network is singly-connected (has no loops) using a belief propagation algorithm proposed by Pearl [96]. However, as Park [92, 94] showed, MAP task remains hard even when MPE and belief updating become easy (for example, in poly-tree networks).

This work is focused on the Belief Updating task, in particular, on finding posterior marginals of singleton variables $P(x_i|e)$. Next, we review the exact methods for belief updating in Bayesian networks.

1.3 Algorithms for Exact Inference

In this section we briefly review exact computation methods for answering Bayesian network queries. There are two primary types of algorithms for Bayesian queries: inference-based and search-based. We review inference-based methods in subsection 1.3.1. The search-based cutset conditioning is presented in subsection 1.3.2.

1.3.1 Inference Algorithms

Belief propagation algorithm performs belief updating in singly-connected Bayesian networks in time and space linear in the size of the input [96]. In loopy networks, the two main approaches for belief updating are *cutset conditioning* [96] and *tree clustering* [75]. We will introduce the principles of belief propagation in Section 1.3.1 followed by a brief

description of clustering algorithms in Section 1.3.1 and the conditioning method in Section 1.3.2.

Iterative Belief Propagation (IBP)

Belief propagation (BP) is an iterative message-passing algorithm that performs exact inference for singly-connected (poly-trees) Bayesian networks [96]. In each iteration, every node X_i sends a message $\pi_j(X_i)$ to each child X_j and receives a message $\lambda_j(X_i)$ from each child X_j:

$$\pi_j(x_i) = \alpha \prod_{k \neq j} \lambda_k(x_i) \sum_{pa_i} P(x_i \mid pa_i) \prod_{u_l \in pa_i} \pi_i(u_l) \tag{1.1}$$

$$\lambda_j(x_i) = \alpha \sum_{x_j \in \mathcal{D}(X_j)} \prod_{k} \lambda_k(x_j) \sum_{pa_j \setminus x_i} P(x_j \mid pa_j) \prod_{u_l \in pa_j \setminus x_i} \pi_i(u_l) \tag{1.2}$$

where α is a normalization constant. Figure 1.2 shows the exchange of messages between a variable X_i, its parents $U_1, ..., U_n$ and its children $Y_1, ..., Y_m$.

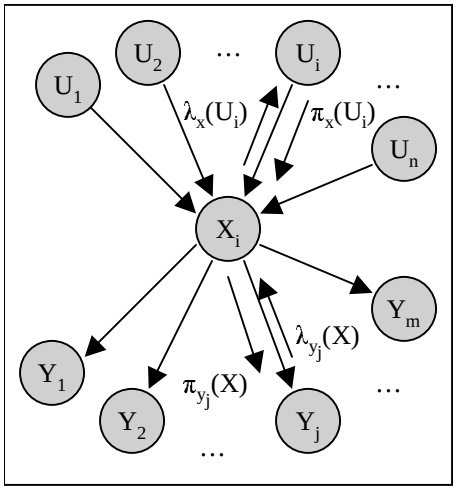


Figure 1.2: Propagation of messages in belief propagation.

Upon convergence, we obtain posterior marginal $P(x_i|e)$ as follows:

$$P(x_i|e) = \alpha \prod_k \lambda_k(x_i) \sum_{pa_i} P(x_i \mid pa_i) \prod_{u_l \in pa_i} \pi_i(u_l)$$

The message-passing order can be organized so that BP converges in two iterations. Applied to Bayesian networks with loops, the algorithm usually iterates longer (until it may converge) and hence, is known as Iterative Belief Propagation (IBP) or loopy belief propagation. IBP provides no guarantees on convergence or quality of approximate posterior marginals but was shown to perform well in practice [104, 91]. It is considered the best algorithm for inference in coding networks [39, 71] where finding the most probable variable values equals the decoding process [88]. Algorithm IBP requires linear space and usually converges fast if it converges. In our benchmarks, IBP converged within 25 iterations or less (see Section 2.5).

Clustering Methods

The join-tree clustering approach (JTC) refers to a family of algorithms including join-tree propagation [75, 59] and bucket-tree elimination [29, 28]. The idea is to first obtain a tree decomposition of the network into clusters of functions connected as a tree and then propagate messages between the clusters in the tree. We can obtain a tree decomposition of the graph by first, moralizing the graph (connect all parents of each node and drop the edge direction), and then eliminating nodes in some order from last to first, connecting all of its preceding neighbors. The complexity of the resulting graph is characterized by the graph's induced width.

DEFINITION **1.3.1 (induced-width)** *The* **width of a node** *in an ordered undirected graph is the number of the node's neighbors that precede it in the ordering. The* **width of an ordering** d*, denoted* $w(d)$*, is the width over all nodes. The* **induced width** *of an ordered graph,* $w^*(d)$*, is the width of the ordered graph obtained by processing the nodes from last to first. When node X is processed, all its preceding neighbors are connected. The resulting graph is called* **induced graph** *or* **triangulated graph**.

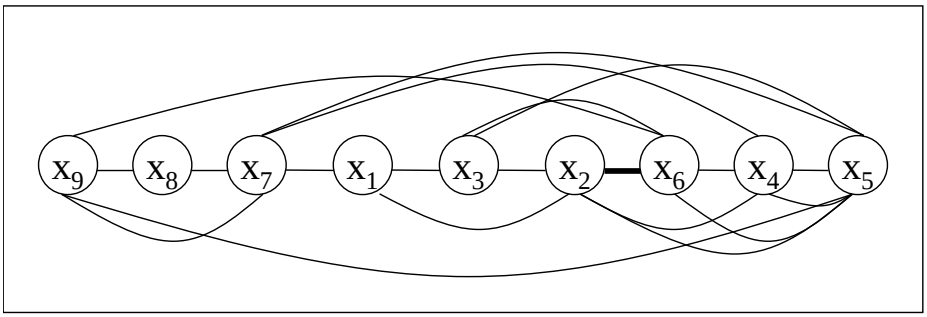


Figure 1.3: Induced graph for a sample Bayesian network shown in Figure 1.1 along ordering d=$\{X_5, X_4, X_6, X_2, X_3, X_1, X_7, X_8, X_9\}$, $w^* = 4$, with added edge $\{X_2, X_6\}$.

For the sample network in Figure 1.1, along ordering $\{X_5, X_4, X_6, X_2, X_3, X_1, X_7, X_8, X_9\}$, the induced width $w^* = 4$ after we add edge $\{X_2, X_6\}$. The induced ordered graph is shown in Figure 1.3.

A tree-decomposition is a singly-connected undirected graph whose nodes, also called clusters, contain subsets of variables and input functions defined over those variables. A tree-decomposition must contain each function once and satisfy running intersection property [86]. Formally:

DEFINITION **1.3.2 (tree-decomp., cluster-tree, tree-width)** *Let* $R = <X, D, F>$ *be a reasoning problem with its hypergraph* $\mathcal{H} = <X, F>$*. (We abuse notation when we identify a function with its scope). A* **tree-decomposition** *for* R *(resp., its hypergraph* $\mathcal{H}$*) is a triple* $< T, \chi, \psi >$*, where* $T = <V, E>$ *is a tree, and* χ *and* ψ *are labelling functions which associate with each vertex* $v \in V$ *two sets,* $\chi(v) \subseteq X$ *and* $\psi(v) \subseteq F$ *such that:*

1. *For each function $f_i \in F$, there is* exactly *one vertex $v \in V$ such that $f_i \in \psi(v)$, and $scope(f_i) \subseteq \chi(v)$.*
2. *For each variable $X_i \in X$, the set $\{v \in V | X_i \in \chi(v)\}$ induces a connected subtree of T. This is also called the running intersection property.*

Given two adjacent vertices V_i and V_j of a tree-decomposition, the separator *of V_i and V_j is defined as $sep(i,j) = \chi(V_i) \cap \chi(V_j)$.*

We will often refer to a node and its functions as a *cluster* and use the term *tree decomposition* and *cluster tree* interchangeably.

For a unifying perspective of tree-decomposition schemes see [63]. We will outline the approach next.

Given a tree-decomposition T of the network, the message propagation over this tree can be synchronized. We select any one cluster as the root of the tree and propagate messages up and down the tree. A message $m_{i,j}$ from cluster V_i to neighbor V_j is a function over the separator set $sep(i,j)$ that is a marginalization of the product of all functions in V_i and all messages that V_i received from its neighbors besides V_j:

$$m_{i,j} = \Downarrow_{sep(i,j)} \Big(\bigotimes_{f \in cluster(u), f \neq m_{j,i}} f \Big) \tag{1.3}$$

where $cluster(u) = \psi(u) \cup \{m_{j,k} | (V_j, V_k) \in T\}$.

We compute the posteriors for each $X_k \in V_i$ by taking a product of all functions in the cluster V_i and marginalising out all other variables:

$$P(X_k, e) = \Downarrow_{X_k} \Big(\bigotimes_{f \in cluster(u)} f \Big) \tag{1.4}$$

Assuming that the maximum number of variables in a cluster is $w + 1$ and maximum domain size is d, the time and space required to process one cluster is $O(d^{(w+1)})$.

Since the maximum number of clusters is bounded by $|X| = n$, the complexity of variable-elimination algorithms and cluster-tree propagation schemes is time and space $O(N \cdot d^{(w+1)})$. The parameter w, the maximum cluster size minus 1, is called the tree-width of the tree decomposition.

DEFINITION **1.3.3 (tree-width)** *The* **tree-width** *of a tree-decomposition* $< T, \chi, \psi >$ *is the maximum size of node* χ_i *minus* 1*, i.e.,* $\max_{v \in V} |\chi(v)|$*-1 [6]. The* **tree width** *of a graph* G *denoted* $tw(G)$ *is the minimum width over all possible tree decompositions of G* [6].

We will sometimes denote the optimal tree-width of a graph by tw^*. It is well known that the tree-width of a graph is identical to its induced-width. We use the notation of tree-width and induced-width interchangeably.

The task of finding the tree-width of a graph is NP-complete [6]. In the past 2 decades, substantial research focused on designing exact and approximate algorithms for finding the tree-width [10, 112, 49].

Bucket elimination is a special case of join-tree clustering where messages are passed from leaves to root along a bucket-tree [29, 28]. Given a variable ordering, the algorithm partitions functions into buckets, each associated with a single variable, corresponding to clusters in a join-tree. A function is placed in the bucket of its latest argument in the ordering. The algorithm processes each bucket, top-down, from the last variable to the first, by a variable elimination procedure that computes a new function using combination and marginalization operators. The new function is placed in the closest lower bucket whose variable appears in the new function's scope. In a generalized elimination scheme, known as bucket-tree elimination, a second pass along the bucket tree can update every bucket in

the tree [23, 63].

1.3.2 Cutset Conditioning

When the tree-width w of the Bayesian network is too large, (e.g., when the requirements of inference schemes such as bucket elimination and join-tree clustering (JTC) exceed available memory), we can switch to the alternative *cutset conditioning* scheme presented in [96, 109, 97]. The idea of this scheme is to select a subset of variables $C \subset X \backslash E$, called a *cutset*, and obtain posterior marginals for any node $X_i \in X \backslash C, E$ by:

$$P(x_i|e) = \sum_{c \in \mathcal{D}(C)} P(x_i|c, e) P(c|e) \tag{1.5}$$

Eq. (1.5) implies that we can enumerate all instantiations over C, perform exact inference via a join-tree algorithm for each cutset instantiation c to obtain $P(x_i|c, e)$ and $P(c|e)$ and then sum up the results.

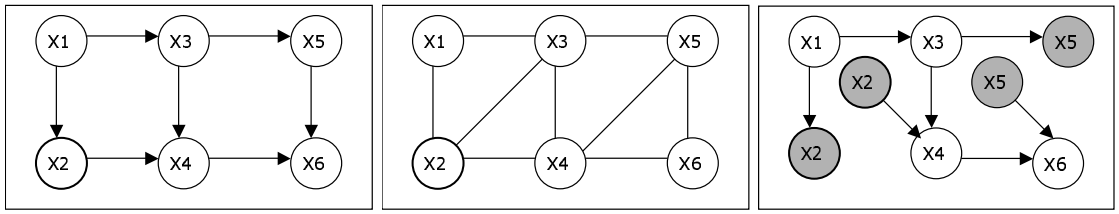


Figure 1.4: Bayesian network (left), its moral graph(center), and conditioned poly-tree (right) (conditioned on $C = \{X_2, X_5\}$).

Observations break down the dependencies in a Bayesian network. When a node X_i is observed (conditioned on), we can transform the network, while preserving all dependencies, as follows. We first remove all directed edges between the node X_i and its children.

Then, for each child $ch_j(X_i)$, create a copy X_{ij} of node X_i and add a directed edge from X_{ij} to $ch_j(X_i)$. If instantiated nodes form a loop-cutset, then the Bayesian network can be transformed into an equivalent singly-connected network. The graph on the right in Figure 1.4 shows a poly-tree network that is equivalent to the original network, shown in Figure 1.4 on the left, conditioned on loop-cutset $C = \{X_2, X_5\}$.

Hence, when loop-cutset nodes are observed, probabilities $P(x_i|c, e)$ and $P(c|e)$ can be computed via BP in time and space linear in the size of the network. The total computation time is exponential in the size of the loop-cutset because we have to enumerate all the cutset instantiations.

To minimize the cutset-conditioning time, we usually try to find a minimal loop-cutset, i.e., the one that contains the smallest number of variables. The problem is NP-hard since the minimal vertex cover, which is known to be NP-hard [42], can be reduced to the minimal loop-cutset problem [114]. A factor 4 approximation algorithm for finding a minimum loop-cutset was proposed in [7], i.e., the algorithm guarantees that the resulting loop-cutset is no more than a factor of 4 larger than optimal. A more refined version of the problem is to find a loop-cutset with the smallest number of tuples. Becker and Geiger [9] showed that it can be reduced to finding the minimum weighted vertex feedback set (MWVFS) where the weight of a variable equals the size of its domain. Of course, it is also NP-hard [61]. Using the reduction to MWVFS, Becker and Geiger [9] proposed a factor 2 approximate algorithm[1] for finding the minimum weight loop-cutset, thus, improving

[1] In UAI proceedings, it is stated that the algorithm is a factor 4. However, it was re-evaluated later.

on the result of Bar-Yehuda et. al. [7]. Subsequently, they showed that their deterministic algorithm can be improved by randomizing the greedy selection step and running algorithm multiple times [8, 3].

It is well-known that the minimum induced width w^* of the network is always less than the size of the smallest loop-cutset [13, 29]. Namely, $w^* + 1 \leq |C|$ for any C. Thus, inference algorithms (e.g., bucket elimination) are never worse and often are better than cutset conditioning time-wise. However, when w^* is large we must resort to cutset conditioning search, trading space for time. The optimal solution is a hybrid search and inference approach that conditions on the smallest cutset C such that the induced width w_C of the graph conditioned on C is small enough to perform exact inference whose complexity is bounded exponentially by w_C.

DEFINITION **1.3.4 (w-cutset)** *Given a Bayesian network $\mathcal{B}$ over X and evidence $E \subset X$, a subset of nodes $C \subset X \setminus E$ is a w-***cutset** *in $\mathcal{B}$ if the induced width of $\mathcal{B}$ conditioned on C, E is w.*

If C is a w-cutset, the quantities $P(x_i|c, e)$ and $P(c|e)$ can be computed in time and space exponential in w. The resulting scheme requires memory exponential in w and time $O(d^{|C|} \cdot N \cdot d^{(w+1)})$ where N is the size of the network and d is the maximum domain size. The w-cutset conditioning scheme generalizes loop-cutset conditioning and allows tuning the performance to the available system memory resource via w.

In [51] the idea is applied to the Pathfinder system [53], where the conditioning set is restricted to the set of diseases. Hybrid scheme combining conditioning with clustering algorithms has been explored further in [109]. Rish and Dechter [102, 103] proposed

a scheme for combining conditioning and variable elimination for propositional theories. They combined Directional Resolution, a variable elimination scheme for SAT, with the Davis-Putnam search procedure. In [74], Larrosa and Dechter extended that approach to general constraint optimization tasks. An alternative hybrid scheme combining conditioning and elimination in the computations in a single cluster of a clique-tree was proposed by Dechter and Fattah [30].

Given w, finding a minimal w-cutset for the hybrid scheme is hard. Several greedy heuristic approaches can be found in [43, 15, 16]. We elaborate more in Section 2.2.5.

1.4 Sampling methods for Bayesian networks

When the complexity of exact algorithms for answering Bayesian queries renders those methods infeasible in practice, we resort to approximate methods. Sampling algorithms are commonly used for approximate reasoning in Bayesian networks as they require linear amount of memory and guarantee convergence to exact values with time. In this section, we review briefly sampling methods for Bayesian networks focusing on likelihood weighting and Gibbs sampling.

Consider a Bayesian network $\mathcal{B}$ over X. Let E denote a subset of evidence variables and e denote the observed values. A sample $x^{(t)}$ is an assignment to all the variables in X. Superscript t denotes a sample number and $x_i^{(t)}$ is the value of X_i in sample t. Let $P(X)$ denote the probability distribution defined by $\mathcal{B}$. Since we want to sample from distribution $P(X)$, it is also called *target* distribution. Let $f(X)$ denote a function of in-

terest over variables X. Function $f(X)$ can represent any of the typical Bayesian queries, including the posterior marginal distribution $P(X_i|e)$. Given a set of T independent and identically distributed (i.i.d.) samples $x^{(1)}$, $x^{(2)}$, ..., $x^{(T)}$ from distribution $P(X)$, the expectation $E[f(X)] = \hat{f}_T(X)$ of $f(X)$ is defined as follows:

$$\hat{f}_T(X) = \frac{1}{T}\sum_{t=1}^{T} f(x^{(t)}) \tag{1.6}$$

For conciseness, we use $\hat{f}_T$ to denote $\hat{f}_T(x)$ and f to denote $f(x)$. Following the *law of large numbers*, it can be shown that:

$$\lim_{T\to\infty} \hat{f}_T = f$$

The convergence rate is derived from central limit theorem (CLT):

$$\sqrt{T}(\hat{f}_T - f) \to N(0, \sigma^2)$$

where $\sigma^2 = Var\{f(x)\}$. As a result, the "error term" in $\hat{f}_T$ is proportional to $O(T^{-1/2})$ and does not depend on the dimensionality of X. This makes sampling methods attractive for solving problems that are otherwise intractable, including inference in complex Bayesian networks. The limiting factors are the increase in sampling variance with the dimensionality of X and the difficulty in generating samples directly from the distribution $P(X)$. We will address the variance reduction techniques in Section 1.5. Next, we will summarize the basic sampling schemes for Bayesian networks.

In the absence of evidence, we can sample from the target distribution $P(X)$ using Logic sampling [54] described as follows. We sample variables in topological order of the

network. Namely, each node is processed after its parents. First, we sample the values of the root nodes from their priors. Any subsequent variable X_i is sampled from $P(X_i|pa_i)$ where pa_i denotes the set of parents of variable X_i. Since the parents of node X_i are sampled first, their values are fixed by the time we need to sample node X_i. Thus, we only need to look up $P(X_i|pa_i)$ in the CPT of node X_i. The samples produced that way are independent and drawn from $P(X)$ and the value of function $f(X)$ can be estimated using expression (1.6). In particular, if $f(X)$ is the posterior marginal $P(x_i)$, the estimator computation is reduced to counting the portion of samples where $X_i = x_i$.

When evidence is present, the target distribution $P(X)$ in a Bayesian network is the posterior joint distribution $P(X|e)$ and it is typically unavailable. We can, in principle, apply Logic sampling ignoring evidence, but we have to discard (reject) samples where sampled values conflict with actual evidence values. The resulting scheme is referred to as Rejection sampling [48]. In this case, many samples may be wasted, especially if the probability of evidence is small. Two families of algorithms addressing this issue have evolved. One is importance sampling scheme [110] and the other is Markov Chain Monte Carlo sampling.

The main idea behind importance sampling is that we sample from a *sampling* distribution $Q(X)$ that is different from the target distribution $P(X|e)$ and then weigh the samples. The convergence speed and the accuracy of the estimates obtained by importance sampling depend on how close the sampling distribution $Q(X)$ is to the target distribution $P(X|e)$. We review importance sampling in more detail and describe one of its simplest

variants, likelihood weighting, in Section 1.4.1.

An alternative approach is to generate dependent samples using Markov Chain Monte Carlo (MCMC) sampling. In this case, each sample represents the state of the system and generating a sample $x^{(t+1)}$ after sample $x^{(t)}$ is equivalent to a system state transition. The key is to define the state transition function such that the stationary distribution of the Markov Chain converges to the target distribution $P(X|e)$. We will describe MCMC methods further in Section 1.4.2 with focus on Gibbs sampling.

1.4.1 Importance Sampling Algorithms

As already mentioned, importance sampling schemes draw independent samples from a trial distribution $Q(X)$, which is different from target distribution $P(X)$. Generally, $Q(X)$ is selected so that it will be easy to compute.

Samples drawn from $Q(X)$ are weighed against the target distribution $P(X)$ in order to obtain an estimate of a function of interest $f(X)$. The weight of a sample can be obtained by first computing the expectation of $f(X)$:

$$E_P[f(X)] = \sum_{x \in X \setminus E} f(x)P(x) = \sum_{x \in X \setminus E} f(x)\frac{P(x)}{Q(x)}Q(x) = E_Q[f(X)\frac{P(X)}{Q(X)}]$$

Consequently, given T samples from $Q(X)$, we can estimate $f(X)$ as follows:

$$\hat{f}(X) = \frac{1}{T}\sum_{t=1}^{T}\frac{P(x^{(t)})}{Q(x^{(t)})}f(x^{(t)}) \tag{1.7}$$

The Eq. (1.7) above defines an *unbiased* estimator of $f(X)$. The ratio $w^{(t)} = \frac{P(x^{(t)})}{Q(x^{(t)})}$ is the

weight of sample $x^{(t)}$. So, we can re-write Eq. (1.7) as follows:

$$\hat{f}(X) = \frac{1}{T}\sum_{t=1}^{T} f(x^t)w^{(t)} \tag{1.8}$$

In many cases, a *biased* estimator is used instead:

$$\hat{f}(X) = \frac{\sum_{t=1}^{T} f(x^t)w^{(t)}}{\sum_{t=1}^{T} w^{(t)}} \tag{1.9}$$

The biased estimator may be preferred because it allows to compute the weights $w^{(t)}$ only up to a normalization constant. For example, when estimating posterior marginals in Bayesian networks, our target distribution usually is $P(X|e)$, but computing posterior probability $P(x^{(t)}|e)$ maybe hard. Instead, we can compute a joint probability $P(x^{(t)}, e)$ yielding a sample weight within constant $P(e)$:

$$w^{(t)} = \frac{P(x^{(t)}, e)}{Q(x^{(t)})} = P(e)\frac{P(x^{(t)}|e)}{Q(x^{(t)})}$$

Additionally, biased estimator in Eq. (1.9) often has a smaller mean squared error [81].

A common query of interest in Bayesian networks is to find the probability of evidence $P(e)$. Let $Y = X \backslash E$. Then, the expectation of $P(e)$ can be expressed as follows:

$$E_P[P(e)] = \sum_y P(y, e) = \sum_y \frac{P(y, e)}{Q(y, e)} Q(y, e) = E_Q[\frac{P(y, e)}{Q(y, e)}]$$

Using importance sampling approach, we can draw T samples from some distribution $Q(X)$, weigh them against $P(X, e)$ and obtain the following *unbiased* estimate $\hat{P}(e)$ of $P(e)$:

$$\hat{P}(e) = \frac{1}{T}\sum_{t=1}^{T} \frac{P(x^{(t)}, e)}{Q(x^{(t)})} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)} \tag{1.10}$$

In a similar manner, but counting only those samples where $X_i = x_i$, we can obtain an expression for the sampling estimate $\hat{P}(x_i, e)$ of $P(x_i, e)$ for $X_i \in X \backslash E$ by:

$$\hat{P}(x_i, e) = \frac{1}{T} \sum_{t=1}^{T} w^{(t)} \delta(x_i, x^{(t)})$$

where $\delta(x_i, x^{(t)})$ is the Dirac delta function. Namely, $\delta(x_i, x^{(t)}) = 1$ if $x_i = x_i^{(t)}$ and $\delta(x_i, x^{(t)}) = 0$ otherwise.

Another query is to compute posterior marginal distribution $P(X_i|e)$ which can be estimated using the ratio:

$$\begin{aligned} P(X_i|e) &= \frac{P(X_i, e)}{P(e)} \\ E[P(x_i|e)] &= \frac{E[P(x_i, e)]}{E[P(e)]} \end{aligned}$$

Substituting the unbiased estimators for $P(x_i, e)$ and $P(e)$ defined above, we get:

$$\hat{P}(x_i|e) = \frac{\frac{1}{T} \sum_{t=1}^{T} w^{(t)} \delta(x_i, x^{(t)})}{\frac{1}{T} \sum_{t=1}^{T} w^{(t)}} = \frac{\sum_{t=1}^{T} w^{(t)} \delta(x_i, x^{(t)})}{\sum_{t=1}^{T} w^{(t)}} \tag{1.11}$$

We can rewrite the above as:

$$\hat{P}(x_i|e) = \alpha \sum_{t=1}^{T} w_i^{(t)} \delta(x_i, x^{(t)}) \tag{1.12}$$

where α is a normalization constant. Note that in Eq. (1.11) and (1.12), we reuse the samples generated to estimate $P(e)$ in estimating $P(x_i, e)$. It is also possible to estimate $P(e)$ and $P(x_i, e)$ using two separate sets of samples [19, 25, 26]. The resulting estimate of $P(x_i|e)$ was shown to have a lower variance, but at the cost of doubling the amount of computation required. Consequently, in this work, we rely on the estimation method defined by Eq. (1.11) and (1.12).

The above importance sampling estimators are guaranteed to converge to their target values as long as the condition $P(x|e) \neq 0 \Rightarrow Q(x) \neq 0$ holds. While maintaining the condition above, it is desirable that $Q(X)$ reflects as many zeros of $P(X|e)$ as possible. If the distribution $P(X)$ has many zeros while $Q(X)$ remains positive, the algorithm often generates many samples having $P(X) = 0$ which are then discarded.

Generic Importance Sampling Scheme
Input: A belief network $\mathcal{B}$, variables $X = (X_1, ..., X_n)$, evidence $E = e$, initial importance function $Pr^0(X\backslash E)$, the desired number of samples T, the updating interval l, and the score arrays for every node.
Output: A set of samples $\{x^{(t)}\}$, $t = 1...T$.
1. Generate sampler order of the nodes.
2. Initialize: $k \leftarrow 0$, $S = \{\}$, $\forall X_i \in X\backslash E$, $\forall x_i \in \mathcal{D}(X_i)$, $w(x_i^{(t)}) = 0$.
3. Compute Samples:
For $t \leftarrow 1$ to T do:
3.1 $x^{(t)} \leftarrow$ generate a sample according to $Pr^k(X\backslash E)$
3.2 $S \leftarrow S \bigcup x^{(t)}$
3.3 $w^{(t)} \leftarrow$ compute weight of sample $x^{(t)}$
3.4 Update score arrays:
$\forall X_i \in X\backslash E$, $w(x_i^{(t)}) \leftarrow w(x_i^{(t)}) + w^{(t)}$
3.5 if (t mod $l == 0$) then
3.6 $k \leftarrow k + 1$
3.7 update importance function $Pr^k X\backslash E$ based on S
3.8 end if
End For
4. Normalize the score arrays for every node:
$\forall X_i \in X\backslash E$, $P(X_i|e) = \alpha w(x_i^{(t)})$

Figure 1.5: A generic *Importance Sampling* Scheme

A generic importance sampling scheme is defined in Figure 1.5. Step 1 of the algorithm is to create a sampling order. The schemes sampling from the prior distribution, such as Logic sampling and Rejection sampling process nodes in topological order. However,

a different sampling order may be selected to obtain a better sampling distribution that is closer to the target distribution. For example, backward sampling [41] attempts to change the sampling order so that evidence variables are sampled earlier. In [49] and [90] the variables are sampled in reverse elimination order; the sampling distribution of a variable is obtained by computing a product of all the functions in the variable's bucket and summing out all other variables. Gogate and Dechter [49] initialize buckets with functions obtained by Iterative Join-Graph Propagation (IJGP) [31]. Moral and Salmeron [90] utilize standard bucket elimination procedure but they approximate large functions by a probability tree.

In step 2, we initialize sample counters and set of samples S. In steps 3.1 and 3.2, we generate a new sample $x^{(t)}$ and add it to the set of samples S. In step 3.3, we compute the sample weight $w^{(t)}$ and, subsequently, update individual score arrays of each variable $X_i \in X \backslash E$ in step 3.4. Finally, steps 3.5-3.8 describe the optional operation of updating the sampling distribution based on generated samples. The updating step is performed every l samples where l is usually selected heuristically or empirically. In Logic sampling and Rejection sampling, this step is omitted. However, since initial sampling distribution is often very different from the target distribution, dynamic updating can substantially improve the convergence speed of importance sampling. The algorithms that incorporate the updating operation in steps 3.5-3.8 are often referred to as *adaptive* or *dynamic* importance sampling and include such methods as self-importance sampling, heuristic importance sampling [110], and, more recently, AIS-BN [21] and EPIS-BN [120]. A dynamic updating step is also incorporated in the algorithm of Moral and Salmeron [90]. The procedures for

updating the sampling probabilities vary. We describe the principles of adaptive importance sampling on the example of AIS-BN algorithm.

AIS-BN algorithm is based on the observation that if we could sample each node in topological order from distribution $P(X_i|pa_i, e)$, then the resulting sample would be drawn from target distribution $P(X|e)$. Since this distribution is unknown for any variable that has observed descendants, AIS-BN initializes importance function to some $Pr^0(X)$ defined by a collection of sampling distributions $Pr^0(X_i|pa_i, e)$. In the paper [21], authors experimented with setting $Pr^0(X_i|pa_i, e)$ equal to $P(X_i|pa_i)$ and a uniform distribution. They reported better convergence rates when the initial distribution is uniform.

The objective of AIS-BN is to update each distribution $Pr^k(X_i|pa_i, e)$ so that the next sampling distribution $Pr^{k+1}(X_i|pa_i, e)$ will be closer to $P(X_i|pa_i, e)$ than $Pr^k(X_i|pa_i, e)$. The updating formula, applied after generating every l samples, is as follows:

$$Pr^{k+1}(x_i|pa_i, e) = Pr^k(x_i|pa_i, e) + \eta(k) \cdot (Pr'(x_i|pa_i, e) - Pr^k(x_i|pa_i, e))$$

where $\eta(k)$ is a positive function that determines the learning rate and $Pr'(x_i|pa_i, e)$ is an estimate of $P(x_i|pa_i, e)$ based on the last l samples. When $\eta(k) = 0$ (lower bound), the importance function is not updated. When $\eta(k) = 1$ (upper bound), the old function is discarded so that $Pr^{k+1}(x_i|pa_i, e) = Pr'(x_i|pa_i, e)$. The convergence speed is directly related to $\eta(k)$. If it is small, the convergence will be slow due to the large number of updating steps needed to reach a local minimum.

Next, we describe likelihood weighting [40, 110]. It is a simple importance sampling algorithm that does not attempt to modify sampling distribution, but illustrates the core

principles of importance sampling.

Likelihood Weighting

Likelihood weighting [40, 110] samples from a distribution that is close to prior. It begins with a network without evidence and assigns values to nodes in topological order, similar to Rejection sampling. The unobserved nodes are assigned the same way as in logic sampling. First, root nodes are sampled from their prior distributions. Then, all other nodes $X_i \in X \backslash E$ are sampled from distribution $P(X_i|pa_i)$.However, if $X_i \in E$, then X_i is assigned its observed value. The sampling distribution of likelihood weighting can be described as follows:

$$Q(X) = \prod_i P(X_i|pa_i) \mid_{E=e}$$

Consequently, the weight $w^{(t)}$ of each sample t can be computed as follows:

$$w^{(t)} = \frac{P(x,e)}{Q(x)} = \frac{\prod_{x_i \in x} P(x_i|pa_i)}{\prod_{x_i \in x} Q(x_i|pa_i)}$$

where $\forall x_i \in x \backslash e$, $Q(x_i|pa_i) = P(x_i|pa_i)$ and $\forall e_i \in e$, $Q(x_i|pa_i) = 1$. Since for $\forall x_i \in x \backslash e$, the factors in the numerator and denominator of the fraction will cancel out, leaving the following expression for $w^{(t)}$:

$$w^{(t)} = \prod_{e_i \in e} P(e_i|pa_i)$$

When sampling, we initialize weight $w^{(t)} = 1$. As we process nodes, whenever we encounter an evidence variable E_i with observed value e_i, we compute its probability $P(e_i|pa_i^{(t)})$ conditional on the current assignment of values to its parents $pa_i^{(t)}$, fix its value $E_i = e_i$, and update the current sample weight $w^{(t)} = w^{(t)} \cdot P(e_i|pa_i^{(t)})$.

Likelihood weighting has lower rejection rate than Rejection sampling because it simply fixes the evidence node values. Hence, it converges faster. However, it usually converges slower then any of the adaptive importance sampling schemes, especially when probability of evidence $P(e)$ is small.

1.4.2 Gibbs sampling for Bayesian networks

The basic idea behind all Markov Chain Monte Carlo (MCMC) methods, including Gibbs sampling, is to simulate a Markov chain in the state space of $X = \{X_1, ..., X_n\}$ so that the stationary distribution of the chain is the target distribution $P(X)$. Hence, the number of states corresponds to the number of possible instantiations of sampled variables and is exponential in the number of variables. The transition from state $x^{(t)} = \{x_1^{(t)}, ..., x_n^{(t)}\}$ to state $x' = \{x'_1, ..., x'_n\}$ is defined by a transition probability function $T(x^{(t)}, x') = P(x = x'|x^{(t)})$. Markov chain-based sampling was first proposed by Metropolis et al. [89] and is known as Metropolis sampling. The only requirement of Metropolis algorithm is that the state transition function is symmetric, which can be expressed mathematically as follows:

$$T(x, x') = T(x', x)$$

That is, the transition rule is restricted to a subset of transition functions where the chance of obtaining x' from x is equal to the chance of sampling x from x'. Hastings generalized the algorithm [52] so that the transition function is not necessarily symmetric. In Hastings' generalization, the main requirement on the transition function is that $T(x, y) > 0$ if and only if $T(y, x) > 0$. The generalized scheme is known as Metropolis-Hastings algorithm.

Gibbs sampling is a special MCMC scheme introduced by Geman and Geman [45]. It uses the conditional distribution $P(x_i|x_{-i}, e)$ as state transition rule which guarantees that the stationary distribution of the Markov chain is $P(X|e)$.

Ordered Gibbs Sampler
Input: A belief network $\mathcal{B}$, variables $X = \{X_1, ..., X_n\}$, and evidence $E \subset X$, $E = e$.
Output: A set of samples $\{x^{(t)}\}$, $t = 1...T$.
1. **Initialize:** Assign random value $x_i^{(0)}$ to each variable $X_i \in X \backslash E$ from $\mathcal{D}(X_i)$. Assign evidence variables their observed values.
2. **Generate samples:**
For t = 1 to T, generate a new sample $x^{(t)}$:
    For each $X_i = X \backslash E$, compute a new value $x_i^{(t)}$:
        Compute distribution $P(X_i|x_{ma_i}^{(t)})$ and sample $x_i^{(t)} \leftarrow P(X_i|x_{ma_i}^{(t)})$.
        Set $X_i = x_i^{(t)}$.
    End (for i)
End (for t)

Figure 1.6: A *Gibbs sampling* Algorithm

Given a Bayesian network over the variables $X = \{X_1, ..., X_n\}$, and evidence e, Gibbs sampling [45, 47, 85] generates a set of samples $\{x^{(t)}\}$ from $P(X|e)$ as follows. The values of the evidence variables remain fixed. Given sample $x_i^{(t)}$ (the current state of Markov chain), a new value for variable X_i can be sampled from probability distribution $P(X_i|x_{-i}^{(t)})$, denoted $x_i \leftarrow P(X_i|x_{-i}^{(t)})$. The first sample can be initialized at random. The next sample $x_i^{(t+1)}$ is generated from the previous sample $x_i^{(t)}$ following one of two schemes.

Random Scan Gibbs Sampling. Given a sample $x^{(t)}$ at iteration t, pick a variable X_i at random and sample a new value x_i from the conditional distribution $x_i \leftarrow P(X_i|x_{-i}^{(t)})$ leaving other variables unchanged.

Systematic Scan (Ordered) Gibbs Sampling. Given a sample $x^{(t)}$, sample a new value for each variable in some order:

$$\begin{aligned}
x_1 &\leftarrow P(X_1|x_2^{(t)}, x_3^{(t)}, ..., x_n^{(t)}) \\
x_2 &\leftarrow P(X_2|x_1^{(t+1)}, x_3^{(t)}, ..., x_n^{(t)}) \\
&\cdots \\
x_i &\leftarrow P(X_i|x_1^{(t+1)}, ..., x_{i-1}^{(t+1)}, x_{i+1}^{(t)}, ..., x_n^{(t)}) \\
&\cdots \\
x_n &\leftarrow P(X_n|x_1^{(t+1)}, x_2^{(t+1)}, ..., x_{n-1}^{(t+1)})
\end{aligned}$$

For conciseness, we can write $P(x_i|x_{-i}^{(t)}) = P(x_i|x_1^{(t+1)}, ..., x_{i-1}^{(t+1)}, x_{i+1}^{(t)}, ..., x_n^{(t)})$. In Bayesian networks, the conditional distribution $P(X_i|x_{-i}^{(t)})$ is dependent only on the assignment to the Markov blanket ma_i of variable X_i. Thus, $P(x_i|x_{-i}^{(t)})$=$P(x_i|x_{ma_i}^{(t)})$ where $x_{ma_i}^{(t)}$ is the restriction of $x^{(t)}$ to ma_i. Given a Markov blanket of X_i, we obtain the sampling probability distribution $P(X_i|x_{ma_i}^{(t)})$ by computing the conditional probability $P(x_i|x_{ma_i}^{(t)})$ for each $x_i \in \mathcal{D}(X)$, as defined in [96]:

$$P(x_i|x_{ma_i}^{(t)}) = \alpha P(x_i|x_{pa_i}^{(t)}) \prod_{\{j|X_j \in ch_j\}} P(x_j^{(t)}|x_{pa_j}^{(t)}) \tag{1.13}$$

where α is a normalization constant.

Thus, generating a complete new sample requires $O(n \cdot r)$ multiplication steps where r is the maximum family size and n is the number of variables.

A two-component Gibbs sampler, with components X_1 and X_2 sampled in turns, X_1 from $P(X_1|x_2)$ and X_2 from $P(X_2|x_1)$, is known as *data augmentation scheme*.

Gibbs sampling distribution converges to $P(X|e)$ as the number of samples increases

as long as the corresponding Markov chain is *ergodic*, namely it is *aperiodic* and *irreducible* [96, 44, 85].

DEFINITION **1.4.1 (Aperiodic)** *A Markov chain is said to be* aperiodic *if the maximum common divider of the number of steps it takes for the chain to come back to the starting point (any) is equal to one. [81]*

DEFINITION **1.4.2 (Irreducible)** *A Markov chain is* irreducible *if the chain has nonzero probability (density) to move from one position in the state space to any other position in a finite number of steps. [81]*

DEFINITION **1.4.3 (Ergodic)** *An aperiodic, irreducible Markov chain is called* ergodic.

The aperiodicity means that the chain does not have regular loops where every k_i steps we return to state S_i. The irreducibility guarantees that we can get to any state S_j from any state S_i with non-zero probability and thus, will be able to visit (as number of samples increases $T \rightarrow \infty$) all statistically important regions of state space. The conditions are almost always satisfied as long as all probabilities are positive [117]. If a finite-state Markov chain is irreducible and aperiodic, then it converges to its stationary distribution regardless of the initial state. Thus, the Markov chain induced by Gibbs sampler is guaranteed to converge to the target distribution $P(X|e)$ regardless of its initial state as long as all probabilities are positive.

Assuming it takes $\approx K$ samples for a Markov chain to converge to its stationary distribution, the first K samples may be discarded to ensure that the collected samples properly represent distribution $P(X|e)$. The time spent computing K discarded samples is referred to as "burn-in" time. However, determining K is hard [60]. In general, the "burn-

in" is optional in the sense that the convergence of the estimates to the correct posterior marginals does not depend on it. Gibbs sampling algorithm is given in Figure 1.6.

The estimate $\hat{f}_T(X)$ of any function $f(X)$ over T samples can be computed using an *ergodic average*:

$$\hat{f}_T(X) = \frac{1}{T}\sum_{t=1}^{T} f(x^t) \tag{1.14}$$

When the convergence conditions are satisfied, $\hat{f}_T(X)$ is guaranteed to converge to the exact value $f(X)$. In other words, $|\hat{f}_T(x) - f(x)| \to 0$ as $T \to \infty$.

Our focus is on computing the posterior marginal distribution $P(X_i|e)$ for each $X_i \in X\backslash E$. The posterior marginals can be estimated using either a *histogram estimator*:

$$\hat{P}(X_i = x_i|e) = \frac{1}{T}\sum_{t=1}^{T} \delta(x_i, x^{(t)}) \tag{1.15}$$

or a *mixture estimator*:

$$\tilde{P}(X_i = x_i|e) = \frac{1}{T}\sum_{t=1}^{T} P(x_i|x_{-i}^{(t)}) \tag{1.16}$$

The histogram estimator corresponds to counting samples where $X_i = x_i$, namely, $\delta(x_i, x^{(t)}) = 1$ if $x_i^{(t)} = x_i$ and equals 0 otherwise. The name "mixture" stems from the fact that expression (1.16) is a mixture of posterior distributions. Since $P(x_i|x_{-i}^{(t)})$ =$P(x_i|x_{ma_i}^{(t)})$, the mixture estimator is simply an average of the conditional probabilities:

$$\hat{P}(x_i|e) = \frac{1}{T}\sum_{t=1}^{T} P(x_i|x_{ma_i}^{(t)}) \tag{1.17}$$

Gelfand and Smith [44] have pointed out that since the mixture estimator is based on estimating conditional expectation, its sampling variance is smaller due to the Rao-Blackwell theorem. However, their proof was based on the assumption that the samples

are independent which is clearly not true in the case of Gibbs sampling. Liu, Wong, and Kong [79] proved the variance reduction in the case of Gibbs sampler using the notion of auto-covariance, which can be defined as follows:

THEOREM **1.4.1 ([81], Theorem 6.6.2, p.145)** *Let $x^{(0)}, ..., x_n^{(T)}$ be consecutive samples generated by the random scan under stationarity, and let i be the random variable representing the random index in the updating scheme. For $f(x) \in L^2(P(X))$, the lag-n autocovariance between $f(x^{(0)})$ and $f(x^{(n)})$ is a non-negative monotone decreasing function of n. It can be written as:*

$$cov[f(x^{(0)}), f(x^{(T)})] = Var[E[E...[E[f(x)|i, x_{-i}]|x]...]] \tag{1.18}$$

where there are T conditional expectations taken alternately on i, x_{-i} and x.

Subsequently, we can obtain the following expressions for the variances of the estimates expressed in Eq. (1.15) and (1.16):

$$m^2 Var(\hat{I}) = m\sigma_0^2 + 2(m-1)\sigma_1^2 + ... + 2\sigma_{m-1}^2 \tag{1.19}$$

$$m^2 Var(\tilde{I}) = m\sigma_1^2 + 2(m-1)\sigma_2^2 + ... + 2\sigma_m^2 \tag{1.20}$$

where $\sigma_k^2 = cov[f(x^{(0)}, x^{(k)}]$. Comparing the two variances, it is clear that each term in (1.20) is exactly one lag behind the corresponding term in (1.19). Because covariances are non-negative and monotonous, it follows that $Var[\tilde{I}] \leq Var[\hat{I}]$.

As mentioned above, when the Markov chain is ergodic, $\hat{P}(X_i|e)$ will converge to the exact posterior marginal $P(X_i|e)$ as the number of samples increases. It was shown in [105] that the random scan Gibbs sampler can be expected to converge faster than the systematic scan Gibbs sampler. Ultimately, the convergence rate of Gibbs sampler depends on the correlation between two consecutive samples [77, 108, 79].

1.5 Variance Reduction Schemes

All sampling schemes are known to benefit from reducing the dimensionality of sampling space which leads to the reduction in the sampling variance and requires fewer samples to achieve the same accuracy of the estimates. Let X denote a set of all sampling variables. Assume we can decompose X into two subsets Y and Z. Then, we can decompose the sampling probability $P(X|e)$ as follows:

$$P(x|e) = P(y, z|e) = P(z|y, e)P(y|e)$$

If the probability $P(z|y, e)$ is easy to compute, then we can easily estimate $P(x|e)$ using the estimate of $P(y|e)$. In the case of importance sampling, our target distribution becomes $P(Y|e)$ and our sampling distribution is some distribution $Q(Y)$. The sample weight $w^{(t)}$ can be computed as:

$$w^{(t)} = \frac{P(y^{(t)}, e)}{Q(y^{(t)})}$$

yielding the following estimators for the posterior marginals:

$$\hat{P}(y_i|e) = \alpha \sum_{t=1}^{T} w^{(t)} \delta(y_i, y^{(t)})$$

$$\hat{P}(z_i|e) = \alpha \sum_{t=1}^{T} w^{(t)} P(z_i|y^{(t)}, e)$$

We can show that the estimates obtained by marginalising a subset of variables Z have lower variance and require fewer samples to achieve the same accuracy:

THEOREM **1.5.1 ([83], Theorem 6.3, p. 13)** *Let $P(y,z)$ and $Q(y,z)$ be two probability distributions such that the support of P is a subset of that of Q. Then:*

$$Var_Q[\frac{P(Y,Z)}{Q(Y,Z)}] \geq Var_Q[\frac{P(Y)}{Q(Y)}]$$

where $P(Y) = \sum_z P(Y,Z)$ and $Q(Y) = \sum_z Q(Y,Z)$ are marginal distributions. The variances are taken with respect to Q.

A proof, due to Rao-Blackwell theorem, can be found in [36] and [83]. Subsequently, importance sampling on a subset of variables is often referred to as Rao-Blackwellised importance sampling [34, 83].

Integrating out a subset of variables also improves the convergence of MCMC schemes. Reducing the number of sampled variables in Gibbs sampler is known as *collapsing* the Gibbs sampler. Intuitively, since the convergence rate of Gibbs sampler is determined by the maximal correlation between the states of two consecutive Gibbs iterations, removing strongly correlated variables from the sampling set reduces the correlation between samples. Alternatively, the convergence rate of Gibbs sampler can be improved by *blocking*, i.e., grouping variables together and sampling them simultaneously. If two variables are strongly correlated, we can reduce the correlation by sampling them together. Efficient blocking scheme for Bayesian networks have been investigated in [57, 67].

Given a joint probability distribution over three random variables X, Y, and Z, the two variance-reduction schemes for Gibbs sampler can be illustrated as follows:

1. Standard Gibbs:

$$x^{(t+1)} \leftarrow P(X|y^{(t)}, z^{(t)}) \quad (1.21)$$
$$y^{(t+1)} \leftarrow P(Y|x^{(t+1)}, z^{(t)}) \quad (1.22)$$
$$z^{(t+1)} \leftarrow P(Z|x^{(t+1)}, y^{(t+1)}) \quad (1.23)$$

2. Collapsed (variable Z is marginalized):

$$x^{(t+1)} \leftarrow P(X|y^{(t)}) \tag{1.24}$$
$$y^{(t+1)} \leftarrow P(Y|x^{(t+1)}) \tag{1.25}$$

3. Blocking by grouping X and Y together:

$$(x^{(t+1)}, y^{(t+1)}) \leftarrow P(X, Y|z^{(t)}) \tag{1.26}$$
$$z^{(t+1)} \leftarrow P(Z|x^{(t+1)}, y^{(t+1)}) \tag{1.27}$$

Although both blocking and collapsing improve convergence of Gibbs sampler, collapsing usually yields an estimator with smaller variance than blocking. Liu, Wong, and Kong [79] showed that for a Gibbs sampler with three variables the variance Var_c of the estimator in a collapsed Gibbs sampler is smaller than the variance Var_b of the estimator in a blocking Gibbs sampler. Both collapsed and blocking Gibbs sampler estimates have smaller variances than the variance Var_g of the full Gibbs sampler. We get:

$$Var_c \leq Var_b \leq Var_g$$

We can extend the argument for the case of Gibbs sampler with n variables by examining the properties of the forward operator **F**:

$$\mathbf{F}h(X) = \int T(x, y)h(y)\, dy$$

over a Hilbert space $L^2(\pi)$. The $T(x, y)$ is a Markov Chain transition function. The proof, which will not provide here as it requires an elaborate analysis of the operator properties, is based on the fact that the norm of operator F is known to be related to the convergence rate of the Markov chain. It is possible to show that the norm of the operator F_c, corresponding to the collapsed Gibbs sampler, is smaller than the norm of the operator F_b, corresponding

to the blocking Gibbs sampler, both of which are smaller than the norm of the operator F_g corresponding to the standard Gibbs sampling scheme:

$$||F_c|| \leq ||F_b|| \leq ||F_g||$$

Subsequently, we can expect the collapsed Gibbs sampler to converge faster than blocking Gibbs sampler or the standard sampling scheme. Further analysis of the collapsed Gibbs sampler can be found in [38, 84, 80, 81].

Collapsed Gibbs sampler is sometimes referred to as Rao-Blackwellised Gibbs sampler [106] although the Rao-Blackwellised estimates are normally obtained from samples in the space of $X = Y \cup Z$ by restricting the conditioning set to Y. That is the variables in Y are assigned their sampled values while the variables in Z are marginalised out. For example, a Rao-Blackwellised estimate of function $f(X)$ over T samples from X can be computed as follows:

$$\hat{f}_T(X) = \frac{1}{T}\sum_{t=1}^{T} E[f(x)|y^{(t)}]$$

where $y^{(t)}$ is a restriction of assignment $x^{(t)}$ to Y, $y^{(t)} \subset x^{(t)}$. A special case of Rao-Blackwellised estimator is the Gibbs mixture estimator expressed in Eq. (1.16).

There are instances of problems where introducing a clever auxiliary variable can actually improve the performance of Gibbs sampler. This is the case of Swendsen-Wang algorithm [115] for Monte Carlo simulation from Ising model. While the model is defined over a set of variables X, the algorithm introduces an auxiliary variable U such that all variables in X are conditionally independent when U is observed. As a result, the variables in X are sampled together as a block. The algorithm of Swendsen-Wang then can be viewed

as a data augmentation scheme iterating between sampling from $P(b|x)$ and $P(x|b)$ where b is an expectation function of u [37, 55, 116]. The Swendsen-Wang algorithm using a "decoupling" auxiliary variable U outperforms plain Monte Carlo sampling over Ising model [115] and in statistical image processing [55].

It is worth noting that introducing a decoupling variable does not always improve convergence of the Gibbs sampler. It was shown that integrating out the decoupling variable from the bivariate Gaussian inference problem [81] or Gibbs motif finding algorithm [78] actually improves their convergence rates.

The efficiency of sampling from a lower-dimensional space in both importance sampling and Gibbs sampling is hindered by the computation overhead incurred from computing the necessary sampling probabilities. In the case of Bayesian networks, the task of marginalising out some variables is equivalent to belief updating where evidence variables and sampling variables are observed. Its time complexity is, therefore, exponential in the induced width of the network conditioned on instantiated variables (evidence and sampled). Consequently, so far, the collapsed sampling ideas have been applied in the context of Gibbs and importance sampling for only a few classes of Bayesian networks [36, 34, 5, 106] by exploiting the special properties of the embedded distributions.

1.6 Thesis overview and Results

Research is what I'm doing when I don't know what I'm doing.
-Wernher Von Braun

The central theme of this thesis is that efficiency can be gained by combining search and exact inference when answering queries in Bayesian networks. It is well-known that an assignment of values to a subset of variables, a cutset, can reduce the complexity of exact inference over the conditioned network. We call the action of assigning values to variables "conditioning." By enumerating all cutset instances and performing exact inference for each instance, we can obtain exact answers to Bayesian queries such as exact posterior marginals. However, enumerating all cutset tuples is infeasible when the cutset is large since the number of tuples grows exponentially with the number of variables in the cutset. This dissertation seeks to improve the efficiency of existing algorithms for approximating and bounding posterior marginals by enumerating only a subset of cutset tuples. We investigated three aspects of the problem:

1. Combining conditioning and exact inference for efficient sampling over a subset of variables (Chapter 2). We propose two cutset sampling schemes, Gibbs-based w-cutset sampling and likelihood weighting on a loop-cutset, that exploit network structure to bound the complexity of computing a sample in the cutset space using exact inference. The schemes produce estimates with lower sampling variance and faster convergence rates compared to sampling over a full set of variables. From the point of view of cutset conditioning, we enumerate only those cutset tuples that are generated by sampling. The probability of a cutset tuple is estimated by its frequency in the set of samples. In case of

likelihood weighting, the frequency is also weighted to reflect the fact that the sampling distribution is different from the target distribution.

2. Finding a minimal cutset that reduces the induced width of the network (Chapter 3) that can be used as a sampling set in Chapter 2 and as a conditioning set in Chapter 4.

3. Using conditioning to compute bounds on posterior marginals (Chapter 4). We derive the expressions for lower and upper bounds that yield a framework that computes exactly probabilities for a subset of cutset tuples and bounds the rest using a combination of exact inference and off-the-shelf bounding scheme. The resulting bounds converge to exact posterior marginals as the number of computed cutset tuples increases. The scheme outperforms previously proposed bounded conditioning algorithm [56] using the bound propagation plug-in [76]. It outperforms the bound propagation algorithm after processing a few thousand cutset tuples out of millions of tuples in the loop-cutset space.

The following three subsections summarize our contributions.

1.6.1 w-cutset Sampling (Chapter 2)

All sampling algorithms converge slowly in high-dimensional spaces due to an increase in sampling variance. By reducing the number of variables in a sample, we reduce the sampling variance and, consequently, require fewer samples to converge. The challenging task is to be able to sample a subset of variables in a time-efficient manner.

Contributions

Our contribution is in presenting a general, structure-based scheme which samples a subset of variables, a cutset, as opposed to sampling all unobserved variables. Sampling over a cutset in a Bayesian network requires computing the sampling probabilities by exact inference. Inference methods such as bucket elimination and tree-clustering are time and space exponential in the induced width w of the network. We exploit the property that conditioning on a subset of variables simplifies the network's structure reducing its induced width and allowing efficient query processing by inference. If the cutset C is a w-cutset, namely a subset of nodes such that when assigned, the induced-width of the conditioned network is w, the time and space complexity of computing the next sample is exponential in w.

Sampling over a cutset improves sampling efficiency (reduces variance) by reducing the size of the sampling space at the cost of more demanding exact inference. Thus, we can control the time-space trade-offs between sampling and exact inference by selecting a cutset having desired properties.

We demonstrate the effectiveness of cutset sampling using Gibbs sampling and likelihood weighting (LW). We compared the accuracy of the estimates produced by sampling over a cutset against sampling over all unassigned variables as a function of time. We showed that the cutset-based estimates usually converge faster despite the incurred computation overhead. Specifically, Gibbs sampling on a loop-cutset outperformed full Gibbs sampling in all benchmarks but one. Gibbs sampling on a w-cutset outperformed full Gibbs

sampling for a range of w values. Likelihood weighting on a cutset was outperformed by full likelihood weighting only in a network without evidence where the LW sampling distribution equals the target distribution.

Sampling over a cutset offers additional benefits when the network contains deterministic probabilities. In this case, plain Gibbs sampling does not converge; however, Gibbs sampling over a cutset converges as long as the Markov chain over the cutset is ergodic. In case of likelihood weighting, we observed that sampling over a cutset often results in lower rejection rates.

1.6.2 Finding Minimum w-cutset (Chapter 3)

The size of the w-cutset affects the performance of both w-cutset sampling and w-cutset conditioning. Given induced width bound w, it is preferable to select a minimal w-cutset for both w-cutset sampling and for w-cutset conditioning. The focus of Chapter 3 is on developing new techniques for finding the minimum w-cutset.

Contributions

Since a network conditioned on w-cutset yields a tree-decomposition having tree width w, we propose to start with a good tree-decomposition Tr of the network and seek the minimum w-cutset of Tr. We prove that the problem of finding a minimal w-cutset of tree-decomposition is NP-hard because any minimum set multi-cover problem (SMC) can be reduced to solving a minimum w-cutset problem. We also show that any w-cutset problem

can be reduced to set multi-cover problem which allows us to adapt a well-known greedy algorithm for solving minimum SMC (or minimum-cost SMC) to finding a minimum w-cutset (or minimum-cost w-cutset).

We investigate empirically several variants of the greedy algorithm for SMC and show that it consistently finds a w-cutset that is the same size or smaller than the cutset obtained by the well-performing loop-cutset algorithm [10] (adapted to the w-cutset problem) and the w-cutset algorithm proposed in [43].

1.6.3 Any-Time Bounds (Chapter 4)

Chapter 4 investigates how enumerating a subset of cutset tuples can be used to improve existing bounding algorithms, building upon the approaches of [56] and [76]. Our approach is to exploit properties of the distribution $P(C, e)$, where e is evidence, over cutset C when a small subset of cutset tuples contains most of the probability mass of $P(e) = \sum_{c \in \mathcal{D}(C)} P(c, e)$. A distribution that exhibits such properties is sometimes referred to as a peaked distribution. The idea is to find all the cutset tuples with high probability mass $P(c, e)$ and bound the probability mass distributed over the remaining tuples. The resulting scheme is any-time in the sense that the lower and upper bounds continue to improve with time as more cutset tuples are explored. The bounds are guaranteed to converge to exact posterior marginals. The successful realization of this approach depends on solving the two subproblems of finding the high probability tuples and bounding the unexplored tuples.

Contributions

Our contribution is a bounding framework, Any Time Bounds (ATB), that defines new expressions for the lower and upper bounds, derived from first principles, and allows one to plug-in any off-the-shelf bounding algorithm to bound the probability mass over the unexplored cutset tuples. It extends the ideas that are the foundation of the bounded conditioning algorithm where the probability mass of the unexplored tuples was bounded by the sum of their priors. The prior distribution was also used to guide the selection of high probability tuples. Our approach uses a more sophisticated scheme for selecting high probability tuples, namely, cutset sampling, and a more accurate bounding algorithm, namely, bound propagation, to bound the missing probability mass.

To improve the performance of the resulting combination scheme, we improve bound propagation in several ways. We exploit the relevant subnetwork properties of each node and also propose an algorithm for approximately solving the linear optimization problems, which are part of bound propagation algorithm, in order to reduce computation time. Thus, we plug into our bounding framework a version of bound propagation that exploits network properties to its advantage but relaxes the linear optimization problems and solves them without using the simplex solver.

We evaluate the performance of the resulting hybrid bounding scheme and bound propagation over a number of benchmarks. The results show that when the distribution $P(C, e)$ (or, equivalently, $P(C|e)$) indeed has a peaked shape, Any Time Bounds (ATB) outperforms bound propagation after exploring a few hundred cutset tuples. Even when

the distribution is relatively flat, ATB outperforms bound propagation after exploring a few thousand tuples. The results also show that cutset sampling is indeed effective at discovering high probability tuples. Using cutset conditioning to generate cutset tuples, we were able to accumulate over 90% of the weight of $P(e)$ in a few hundred to a few thousand tuples in several networks from UAI repository. In other networks, we accumulated up to 20-30% of probability mass of $P(e)$ after generating just a few hundred out of several millions of cutset tuples. The details are provided in the experimental section 4.6 of Chapter 4.

Chapter 2

Cutset sampling for Bayesian networks

The chapter presents a new sampling methodology for Bayesian networks that samples only a subset of variables and applies exact inference to the rest. Cutset sampling is a network structure-exploiting application of the Rao-Blackwellisation principle to sampling in Bayesian networks. It improves convergence by exploiting memory-based inference algorithms. It can also be viewed as an anytime approximation of exact cutset-conditioning algorithm [96]. Cutset sampling can be implemented efficiently when the sampled variables constitute a loop-cutset of the Bayesian network and, more generally, when the induced width of the network's graph conditioned on the observed sampled variables is bounded by a constant w. We demonstrate empirically the benefit of this scheme on a range of benchmarks.

2.1 Introduction

We have already defined basic principles of sampling in Bayesian network in Chapter 1 and introduced two families of sampling algorithms, Markov Chain Monte Carlo and importance sampling. When exact inference is impractical due to prohibitive time and memory demands, sampling is often the only feasible approach that offers performance guarantees. Given a Bayesian network over the variables $X = \{X_1, ..., X_n\}$, evidence $E = e$, and a

set of samples $\{x^{(t)}\}$ from $P(X|e)$, a function $f(X)$ can be estimated using the generated samples via *ergodic average*:

$$E[f(X)|e] \cong \frac{1}{T}\sum_{t} f(x^{(t)}) \tag{2.1}$$

where T is the number of samples. The estimate can be shown to converge to the exact value as T increases. The central query of interest over Bayesian networks is computing the posterior marginals $P(x_i|e)$ for each value x_i of variable X_i, also called *belief-updating*. For this query, the above equation reduces to counting the fraction of occurrences of $X_i = x_i$ in the samples.

As we mentioned previously, a significant limitation of all sampling schemes is that the statistical variance increases when the number of variables in the network grows and therefore the number of samples necessary for accurate estimation increases. In this chapter, we present a sampling scheme for Bayesian networks that reduces the sampling variance by sampling from a subset of the variables, i.e., *collapsing* the sampling set. This technique is also sometimes referred to as *Rao-Blackwellised* sampling. The fundamentals of sampling on a subset were developed in and [79] for Gibbs sampling and in [83, 36] for importance sampling. A related work on the reduction of variance in Rao-Blackwellised estimates was performed in [20] and [44].

The basic collapsed sampling scheme can be described as follows. Suppose we partition the space of variables X into two subsets C and Z. Subsequently, we can re-write any function $f(X)$ as $f(C, Z)$. If we can efficiently compute $P(C|e)$ and $E[f(C, Z)|c, e]$ (by summing out Z in both cases), then we can perform sampling on subset C only, generating

samples $c^{(1)}, c^{(2)}, ..., c^{(T)}$ and approximating the quantity of interest by:

$$E[f(X) \mid_e] \cong \frac{1}{T} \sum_t E[f(C, Z)|c^{(t)}, e] \tag{2.2}$$

If the function $f(X)$ is a posterior marginal of node $X_i \in X \backslash C, E$, then $f(X)|e = P(X_i|e)$, $E[f(C, Z)|c^{(t)}, e] = E[P(X_i)|c^{(t)}, e)] = P(X_i|c^{(t)}, e)$ and Eq. (2.2) becomes:

$$P(X_i|e) = \frac{1}{T} \sum_t P(X_i|c^{(t)}, e) \tag{2.3}$$

Rao-Blackwellised estimates have lower sampling variance and therefore require a smaller number of samples to converge to the target distribution. Yet, the cost of generating each sample may increase. Indeed, the principles of Rao-Blackwellised sampling have been applied only in a few classes of probabilistic models with specialized structure and probability distributions [69, 38, 84, 80, 32, 5, 106].

The contribution in this chapter is in presenting a general, structure-based scheme which applies the Rao-Blackwellisation principle to Bayesian networks. The idea is to exploit the property that conditioning on a subset of variables simplifies the network's structure allowing efficient query processing by inference algorithms. In general, exact inference by variable elimination [28, 29] or join-tree algorithms [75, 59] is time and space exponential in the induced-width w of the network. However, when a subset of the variables is assigned (i.e., conditioned upon) the induced-width of the conditioned network is reduced.

The idea of cutset sampling is to choose a subset of variables C such that conditioning on C yields a sparse enough Bayesian network having a small induced width to allow

exact inference. Since a sample is an assignment to a subset of variables, we can generate a new sample over the cutset variables efficiently over the conditioned network where the computation of $P(c|e)$ and $P(X_i|c,e)$ can be bounded. In particular, if the sampling set C cuts all the cycles in the network (i.e., it is a loop-cutset), inference over the conditioned network becomes linear. In general, if C is a w-cutset, namely a subset of nodes such that when assigned, the induced-width of the conditioned network is w, the time and space complexity of computing the next sample is proportional to $O(d^{w+1})$ where d is the maximum domain size.

The idea of exploiting properties of conditioning on a subset of variables has been first proposed for exact belief updating in the context of cutset-conditioning [96]. This scheme requires enumerating all instantiations of cutset variables. Therefore, if the size of the cutset is too big, sampling over the cutset space may be the right compromise. Thus, sampling on a cutset can also be viewed as an anytime approximation of the cutset-conditioning approach.

Collapsing of a sampling set can be applied in the context of any sampling algorithm. We will introduce the principles of cutset sampling in the context of Gibbs sampling [45, 47, 85], a Markov Chain Monte Carlo sampling method for Bayesian networks, and likelihood weighting [40, 110], an instance of importance sampling. The resulting cutset sampling algorithms are our main contribution in this chapter. Extension to any other sampling approach or any other graphical model, such as Markov networks, should be straight forward.

The chapter defines and analyzes the cutset sampling scheme and investigates empirically the trade-offs between sampling and exact inference over a variety of randomly generated networks, grid structure networks as well as known real-life benchmarks such as CPCS networks and coding networks. We show that cutset sampling converges faster than pure sampling in terms of the number of samples as dictated by theory and is also almost always time-wise cost effective on all the benchmarks tried. We also demonstrate the applicability of this scheme to some networks with deterministic probabilities, such as Hailfinder network and coding networks, where Markov chain generated by full Gibbs sampling is non-ergodic.

The contribution of the chapter presenting the cutset sampling starts in Section 2.2. Section 2.5 presents the empirical evaluation of cutset sampling. We also present an empirical evaluation of the sampling variance and the resulting standard error based on the method of *batch means* (for more details, see [46]). In section 2.6, we review previous application of Rao-Blackwellisation and section 2.7 provides summary and conclusions.

2.2 Cutset Sampling

As we discussed above, the convergence rate of Gibbs sampler can be improved via collapsing. Cutset sampling scheme is an efficient way of sampling from a subset of variables $C \subset X$, tying the complexity of sample generation to the structure of the Bayesian network.

2.2.1 Cutset sampling algorithm

The cutset sampling scheme partitions the variable set X into two subsets C and $X \backslash C$. The objective is to generate samples from space C=$\{C_1, C_2, ..., C_m\}$ where each sample $c^{(t)}$ is an instantiation of all the variables in C. Following the Gibbs sampling principles, we generate a new sample $c^{(t)}$ by sampling a value $c_i^{(t)}$ from the probability distribution $P(C_i|c_{-i}^{(t)}) = P(C_i|c_1^{(t+1)}, c_2^{(t+1)}, ..., c_{i-1}^{(t+1)}, c_{i+1}^{(t)}, ..., c_m^{(t)})$. We will use left arrow to denote that value c_i is drawn from distribution $P(C_i|c_{-i}^{(t)})$:

$$c_i \leftarrow P(C_i|c_{-i}^{(t)}, e) \tag{2.4}$$

If we can compute $P(C_i|c_{-i}^{(t)}, e)$ efficiently for each sampling variable $C_i \in C$, then we can generate samples efficiently. The relevant conditional distributions can be computed by exact inference whose complexity is tied to the network structure. We denote by JTC($\mathcal{B}$,X_i,e) a generic algorithm in the class of variable-elimination or join-tree clustering algorithms which, given a belief network $\mathcal{B}$ and evidence e, outputs the posterior probabilities $P(X_i|e)$ for variable $X_i \in X$ [75, 59, 28]. When the network's identity is clear, we will use $JTC(X_i, e)$.

Therefore, for each sampling variable C_i and for each value $c_i \in \mathcal{D}(C_i)$, we can compute $P(C_i, c_{-i}^{(t)}, e)$ via $JTC(C_i, c_{-i}^{(t)}, e)$ and obtain $P(C_i|c_{-i}^{(t)}, e)$ via normalization: $P(C_i|c_{-i}^{(t)}, e) = \alpha P(C_i, c_{-i}^{(t)}, e)$.

Cutset sampling algorithm that uses systematic scan Gibbs sampler is given in Figure 2.1. Clearly, it can be adapted to be used with the random scan Gibbs sampler as well.

Cutset Sampling
Input: A belief network $\mathcal{B}$, a cutset $C = \{C_1, ..., C_m\}$, evidence e.
Output: A set of samples c^t, $t = 1...T$.
1. **Initialize:** Assign random value c_i^0 to each $C_i \in C$ and assign e.
2. **Generate samples:**
For $t = 1$ to T, generate a new sample $c^{(t+1)}$ as follows:
For $i = 1$ to m, compute new value $c_i^{(t)}$ for variable C_i as follows:
a. Compute $JTC(C_i, c_{-i}^{(t)}, e)$.
b. Compute $P(C_i|c_{-i}^{(t)}, e) = \alpha P(C_i, c_{-i}^{(t)}, e)$.
c. Sample:

$$c_i^{(t+1)} \leftarrow P(C_i|c_{-i}^{(t)}, e) \tag{2.5}$$

End For i
End For t

Figure 2.1: w-*Cutset sampling* Algorithm

Steps (a)-(c) demonstrate how the algorithm generates sample $(t+1)$ from sample (t). For every variable $C_i \in C$ in sequence, the main computation is in step (a), where the distribution $P(C_i, c_{-i}^{(t)}, e)$ over C_i is generated. This requires executing JTC for every value $c_i \in \mathcal{D}(C_i)$ separately. In step (b), the conditional distribution is derived by normalization. Finally, step (c) samples a new value from the obtained distribution. Note that we only use $P(C_i|c_{-i}^{(t)}, e)$ as a short-hand notation for $P(C_i|c_1^{(t+1)}, ..., c_{i-1}^{(t+1)}, c_{i+1}^{(t)}, ..., c_k^{(t)}, e)$. Namely, when we sample a new value for variable C_i, the values of variables C_1 through C_{i-1} have already been updated.

We will next demonstrate the process using the special case of loop-cutset (the notion of loop-cutset is given in Definition 1.2.1).

Example 2.2.1 *Consider the belief network shown in Figure 2.2. The subset $\{A, D\}$ is a loop-cutset of the network. Assume that node $E = e$ is observed. Then, when sampling from the cutset $\{A, D\}$, we need to compute for the $(t+1)$ sample the probabilities $P(A|d^{(t)}, e)$ and $P(D|a^{(t+1)}, e)$. Since the conditioned network is a poly-tree (Figure 2.2, right), JTC reduces to Pearl's belief propagation algorithm and the distributions can be computed in linear time.*

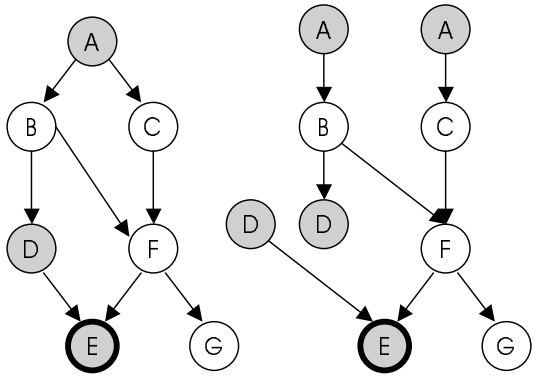


Figure 2.2: When nodes A and D in the loopy Bayesian network (left) are instantiated, the network can be transformed into an equivalent singly-connected network (right). In the transformation process, a replica of an observed node is created for each child node.

Specifically, assume A and D are bi-valued variables with domains $\{0, 1\}$. We begin the sampling process by initializing sampling variables to $a^{(0)}$ and $d^{(0)}$. Next, we compute new sample values $a^{(1)}, d^{(1)}$ as follows:

1. *a. Compute $P(A, d^{(0)}, e)$ using $JTC(A, d^{(0)}, e)$.*
   b. $P(A|d^{(0)}, e) = \alpha P(A, c^{(0)}, e)$ where:

$$\alpha = \frac{1}{P(a = 0, d^{(0)}, e) + P(a = 1, d^{(0)}, e)}$$

2. *Sample the new value value $a^{(1)}$ from $P(A|d^{(0)}, e)$:*

$$a^{(1)} \leftarrow P(A|d^{(0)}, e)$$

3. *a. Compute $P(D|a^{(1)}, e)$ using $JTC(C, a^{(1)}, e)$.*
   b. $P(D|a^{(1)}, e) = \alpha P(D, a^{(1)}, e)$ where:

$$\alpha = \frac{1}{P(d = 0, a^{(1)}, e) + P(d = 1, a^{(1)}, e)}$$

4. *Sample the next sample value $d^{(1)}$ from $P(D|a^{(1)}, e)$:*

$$d^{(1)} \leftarrow P(D|a^{(1)}, e)$$

The process above corresponds to two iterations of the inner loop in Figure 2.1. Steps 1-2, where we sample a new value for variable A, correspond to steps (a)-(c) of the first iteration. In the second iteration, steps 3-4, we sample a new value for variable D.

2.2.2 Estimating Posterior Marginals

Once a set of samples over a subset of variables C is generated, we can estimate the posterior marginals of any variable in the network using mixture estimator. For sampling variables, the estimator takes the form:

$$\hat{P}(C_i|e) = \frac{1}{T}\sum_{t=1}^{T} P(C_i|c_{-i}^{(t)}, e) \tag{2.6}$$

For variables in $X\backslash C$, the posterior marginal estimator is:

$$\hat{P}(X_i|e) = \frac{1}{T}\sum_{t=1}^{T} P(X_i|c^{(t)}, e) \tag{2.7}$$

where we can use $JTC(X_i, c^{(t)}, e)$ to obtain the distribution $P(X_i|c^{(t)}, e)$ over the input Bayesian network conditioned on $c^{(t)}$ and e as shown before.

If we maintain a running sum of the computed distributions $P(C_i|c_{-i}^{(t)}, e)$ and $P(X_i|c^{(t)}, e)$ during sample generation (Eq. (2.4)), the sums in the right hand side of Eq. (2.6) and (2.7) will be readily available. As we noted before, the estimators $\hat{P}(C_i|e)$ and $\hat{P}(X_i|e)$ are guaranteed to converge to their corresponding exact posterior marginals as T increases as long as the Markov Chain over the cutset C is ergodic.

While for the cutset variables the estimator is a simple ergodic average, for $X_i \in X\backslash C, E$ the convergence can be easily derived from first principles:

THEOREM **2.2.2** *Given a Bayesian network $\mathcal{B}$ over X, evidence variables E, and cutset C, and given a set of T samples $c^{(1)}, c^{(2)}, ..., c^{(T)}$ obtained via Gibbs sampling from $P(C|e)$, then for any $X_i \in X\backslash C, E$ assuming $\hat{P}(X_i|E)$ is defined by Eq. (2.7), $\hat{P}(X_i|e) \rightarrow P(X_i|e)$ as $T \rightarrow \infty$.*

Proof. By definition:

$$\hat{P}(X_i|e) = \frac{1}{T}\sum_{t=1}^{T} P(X_i|c^{(t)}, e) \tag{2.8}$$

Instead of summing over samples, we can rewrite the expression above to sum over all possible tuples $c \in \mathcal{D}(C)$ and group together the samples corresponding to the same tuple instance c. Let $q(c)$ denote the number of times a tuple $C = c$ occurs in the set of samples. It is easy to see that:

$$\hat{P}(X_i|e) = \sum_{c\in\mathcal{D}(C)} P(X_i|c, e)\frac{q(c)}{T} \tag{2.9}$$

since $\sum_{c\in\mathcal{D}(C)} q(c) = T$. And since the fraction $\frac{q(c)}{T}$ is a sampling estimator for the posterior marginal $\hat{P}(c|e)$, we get:

$$\hat{P}(X_i|e) = \sum_{c\in\mathcal{D}(C)} P(X_i|c, e)\hat{P}(c|e) \tag{2.10}$$

Since the Markov chain corresponding to Gibbs sampling over C is ergodic, $\hat{P}(c|e) \rightarrow P(c|e)$ as $T \rightarrow \infty$ and therefore:

$$\hat{P}(X_i|e) \rightarrow \sum_{c\in\mathcal{D}(C)} P(X_i|c, e)P(c|e) = P(X_i|e)$$

■

2.2.3 Complexity

The time and space complexity of generating samples and estimating the posterior marginals via cutset sampling is dominated by the complexity of JTC in line (a) of the algorithm. Only linear amount of additional memory is required to maintain the running sum of $P(C_i|c^{(t)}_{-i}, e)$ or $P(X_i|c^{(t)}, e)$ used in the posterior marginal estimators.

Sample Generation Complexity

Clearly, when JTC is applied to the network $\mathcal{B}$ conditioned on all the cutset variables C and evidence variables E, its complexity is time and space exponential in the induced width w of the conditioned network. It is $O(N \cdot d^{(w+1)})$ when C is a w-cutset (see Definition 1.3.4).

Using the notion of w-cutset, we can balance sampling and exact inference. At one end of the spectrum we have plain Gibbs sampling where sample generation is fast, requiring linear space, but may have high variance. At the other end, we have exact inference requiring time and space exponential in the induced width of the moral graph. In between these two extremes, we can control the time and space complexity using w as follows.

THEOREM **2.2.3 (Complexity of sample generation)** *Given a network $\mathcal{B}$ over X, evidence E, and a w-cutset C, the complexity of generating a new sample is time and space $O(|C| \cdot N \cdot d^{(w+2)})$ where d bounds the variables domain size and $N = |X|$.*

Proof. If C is a w-cutset and d is the maximum domain size, then the complexity of computing distribution $P(C_i|c_{-i}^{(t)}, e)$ over the conditioned network is $O(N \cdot d^{(w+1)})$. Since this operation must be repeated for each $c_i \in \mathcal{D}(C_i)$, the complexity of processing one variable is $O(N \cdot d \cdot d^{(w+1)}) = O(N \cdot d^{(w+2)})$. Finally, since ordered Gibbs sampling requires sampling each variable in the cutset, generating one sample is $O(|C| \cdot N \cdot d^{(w+2)})$. ■

Complexity of estimator computation

Posterior marginals for any cutset variable $C_i \in C$ are easily obtained at the end of sampling process without incurring additional computation overhead. As mentioned earlier, we only need to maintain a running sum of $P(C_i|c_{-i}^{(t)}, e)$ computed when a new value of

variable C_i is sampled. Estimating $P(X_i|e)$, $X_i \in X\backslash C, E$, using Eq. (2.7) requires computing $P(X_i|c^{(t)}, e)$ once a sample $c^{(t)}$ is generated. We can obtain $P(X_i|c^{(t)}, e)$ for all $X_i \in X\backslash C, E$ by applying JTC. We summarize the complexity of generating T samples and estimating all posterior marginals in the following theorem.

THEOREM **2.2.4 (Computing Marginals)** *Given a w-cutset C, the complexity of computing posteriors for all variables $X_i \in X\backslash E$ using T samples over the cutset variables is $O(T \cdot [|C| + d] \cdot N \cdot d^{(w+1)})$.*

Proof. As we showed in Theorem 2.2.3, the complexity of generating one sample is $O(|C| \cdot N \cdot d^{(w+2)})$. Once a sample $c^{(t)}$ is generated, the computation of the posterior marginals for the remaining variables requires computing $P(X_i|c^{(t)}, e)$ via $JTC(X_i, c^{(t)}, e)$ which is $O(N \cdot d^{(w+1)})$. The combined computation time for one sample is $O(|C| \cdot N \cdot d^{(w+2)} + N \cdot d^{(w+1)}) = O([|C| + d] \cdot N \cdot d^{(w+1)})$. Repeating the computation for T samples, yields $O(T \cdot [|C| + d] \cdot N \cdot d^{(w+1)})$. ■

Note that the space complexity of w-cutset sampling is bounded by $O(N \cdot d^{(w+1)})$.

Complexity of loop-cutset.

When the cutset C is a loop-cutset, algorithm JTC reduces to belief propagation [96] that computes the joint probability $P(C_i, c^{(t)}_{-i}, e)$ in linear time. We will refer to the special case as *loop-cutset sampling* and to the general as *w-cutset sampling*.

A loop-cutset is also a w-cutset where w equals the maximum number of unobserved parents. However, since processing poly-trees is linear even for large w, the induced width does not capture its complexity properly. The notion of loop-cutset could be better captured via the hyperwidth of the network [50, 63]. The hyperwidth of a poly-tree is 1 and therefore, a loop-cutset can be defined as a 1-hypercutset. Alternatively, we can express the

complexity via the input size M referring to the total size of conditional probability tables to be processed as follows:

THEOREM **2.2.5 (Complexity of loop-cutset sample generation)** *If C is a loop-cutset, the complexity of generating each sample is $O(|C| \cdot d \cdot M)$ where M is the size of the input.*

Proof. When a loop-cutset of a network is observed, JTC or belief propagation (BP) can compute the joint probability $P(c_i, c_{-i}^{(t)}, e)$ in linear time $O(M)$ [96] yielding total time and space of $O(|C| \cdot d \cdot M)$ for each sample. ■

2.2.4 Optimizing cutset sampling performance

Our analysis of the complexity of generating samples (Theorem 2.2.3) is overly pessimistic in assuming that each computation of the sampling distribution for each variable in the cutset is independent. While all variables may change a value when moving from one sample to the next, the change occurs one variable at a time in some sequence so that much of the computation can be retained when moving from one variable to the next.

We will now show that sampling all the cutset variables can be done more efficiently reducing the factor of $N \cdot |C|$ in Theorem 2.2.3 to $(N + |C| \cdot \delta)$ where δ bounds the number of clusters in the tree decomposition used by JTC that contains any node $C_i \in C$. We assume that we can control the order by which cutset variables are sampled.

Consider the following simple network with variables $X = \{X_1, ...X_N\}$, $Y = \{Y_1, ..., Y_{N-1}\}$ and CPTs $P(X_{i+1}|X_i, Y_i)$ and $P(Y_{i+1}|X_i)$ for every i, as shown in Fig-

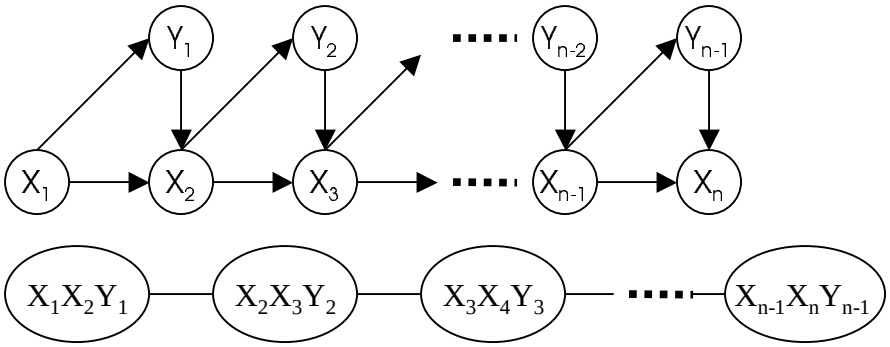


Figure 2.3: A Bayesian network (top) and a corresponding cluster-tree (bottom).

ure 2.3, top. The join-tree of this networks is a chain of cliques of size 3 given in Figure 2.3, bottom. Since Y is a loop-cutset, we will sample variables in Y. Let's assume that we use the ordering $Y_1, Y_2, ...Y_{N-1}$ to generate a sample. Given the current sample, all the cutset variables are assigned and we are ready to generate the next sample. We then apply JTC (or bucket-elimination) to the network whose cutset variables are assigned. This makes the network effectively singly-connected, namely, the actual number of variables in a cluster is 2. The algorithm sends message from the cluster containing X_N towards the cluster containing X_1. When cluster (X_1, X_2, Y_1) gets the relevant message from cluster (X_2, X_3, Y_2) we can sample Y_1. This can be accomplished by d linear computations in clique (X_1, X_2, Y_1) for each $y_i \in \mathcal{D}(Y_i)$ yielding the desired distribution $P(Y_1|.)$ (we can multiply all functions and incoming messages in this cluster, sum out X_1 and X_2 and normalize). If the cutset is a w-cutset, each computation in a single clique is $O(d^{(w+1)})$.

Once we have $P(Y_1|\cdot)$, Y_1 is sampled and assigned a value, y_1. Cluster $(X_1, X_2, Y_1 = y_1)$ then sends a message to cluster (X_2, X_3, Y_2) which now has all the information necessary to compute $P(Y_2|.)$ in $O(d^{(w+2)})$. Once $P(Y_2|.)$ is available, a new value $Y_2 = y_2$ is

sampled. The cluster than computes and sends a message to cluster (X_3, X_4, Y_3), and so on. At the end, we obtain a full sample via two message passes over the conditioned network having computation complexity of $O(N \cdot d^{(w+2)})$. This example can be generalized as follows.

THEOREM **2.2.6** *Given a Bayesian network having N variables, a w-cutset C a tree-decomposition used by JTC, and given a sample $c_1, ..., c_{|C|}$, a new sample can be generated in $O((N + |C| \cdot \delta) \cdot d^{(w+2)})$ where δ is the maximum number of clusters containing any node $C_i \in C$.*

Proof. Given a w-cutset C, by definition, there exists a tree-decomposition T_r of the network (that includes the cutset variables) such that when the cutset variables C are removed the number of variables remaining in each cluster of T_r is bounded by $w + 1$. Let's impose directionality on T_r starting at an arbitrary cluster that we call R as shown in Figure 2.4. Let T_{C_i} denote the connected subtree of T_r whose clusters include C_i. In Figure 2.4, for clarity, we collapse the subtree over C_i into a single node. We will assume that cutset nodes are sampled in depth-first traversal order dictated by the cluster tree rooted in R.

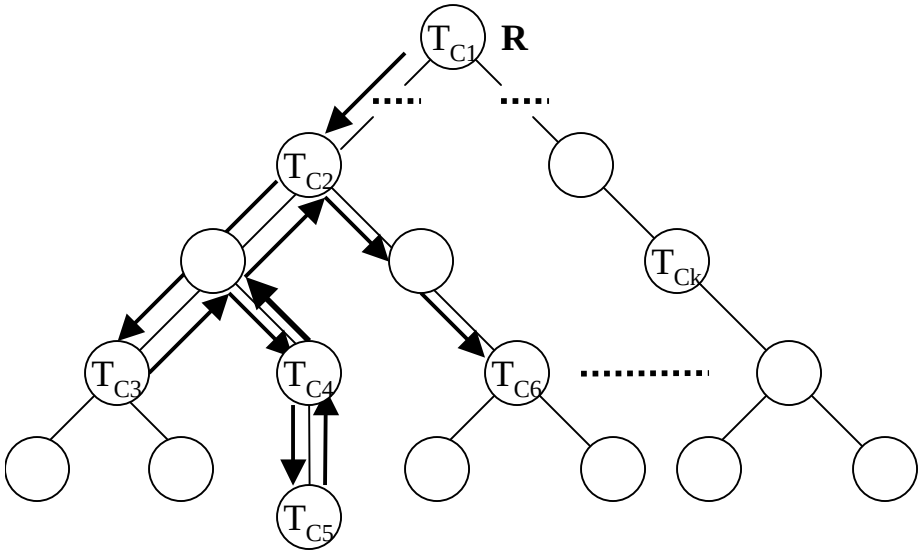


Figure 2.4: A cluster-tree rooted in cluster R where a subtree over each cutset node C_i is collapsed into a single node marked T_{C_i}.

Given a sample $c^{(t)}$, JTC will send messages from leaves of T_r towards the root cluster. We can assume without loss of generality that R contains cutset node C_1 which is the first to be sampled in $c^{(t+1)}$. JTC will now pass messages from root down only to

clusters restricted to T_{C_1} (note that $R \in T_{C_1}$). Based on these messages $P(C_1 = c_1, c_{-1}^{(t)})$ can be computed in $O(d^{(w+1)})$. We will repeat this computation for each other value of C_1 involving only clusters in T_{C_1} and obtain the distribution $P(C_1|\cdot)$ in $O(d^{(w+2)})$ and sample a new value for C_1. Thus, if C_1 appears in δ clusters, the number of message passing computations (after the initial $O(N)$ pass) is $O(\delta)$ and we can generate the first distribution $P(C_1|\cdot)$ in $O(\delta \cdot d^{(w+2)})$.

The next node in the depth-first traversal order is T_{C_2} and thus, the second variable to be sampled is C_2. The distance between variables C_1 and C_2, denoted $dist_{1,2}$, is the shortest path along T_r from a cluster that contains C_1 to a cluster that contains C_2. We apply JTC's mesage-passing along that path only which will take at most $O(dist_{1,2} \cdot d^{(w+1)})$. Then, to obtain the conditional distribution $P(C_2|\cdot)$, we will recompute messages in the subtree of T_{C_2} for each value $c_2 \in \mathcal{D}(C_2)$ in $O(\delta \cdot d^{(w+2)})$. We continue the computation in similar manner for other cutset nodes.

If JTC traverses the tree in the depth-first order, it only needs to pass messages along each edge twice (see Figure 2.4). Thus, the sum of all distances traveled is $\sum_{i=2}^{|C|} dist_{i,i-1} = O(N)$. What may be repeated is the computation for each value of the sampled variable. This, however, can be accomplished via message-passing restricted to individual variable's subtrees and is bounded by its δ. We can conclude that a new full sample can be generated in $O((N + |C| \cdot \delta) \cdot d^{(w+2)})$. ■

It is worthwhile noting that the complexity of generating a sample can be further reduced by a factor of $d/(d-1)$ (which amounts to a factor of 2 when d=2) by noticing that whenever we move from variable C_i to C_{i+1}, the joint probability $P(c_1^{(t+1)}, ..., c_i^{(t+1)}, c_{i+1}^{(t)}, ..., c_k^{(t)})$ is already available from the previous round and should not be recomputed. We only need to compute $P(c_1^{(t+1)}, ..., c_i^{(t+1)}, c_{i+1}, ..., c_k^{(t)})$ for $c_{i+1} \neq c_{i+1}^{(t)}$. Buffering the last computed joint probability, we only need to apply JTC algorithm $d-1$ times. Therefore, the total complexity of generating a new sample is $O((N + |C| \cdot \delta) \cdot (d-1) \cdot d^{(w+1)})$.

Figure 2.5 demonstrates the application of the enhancements discussed. It depicts the moral graph (a) and the corresponding join-tree (b) for the Bayesian network in Figure 2.2.

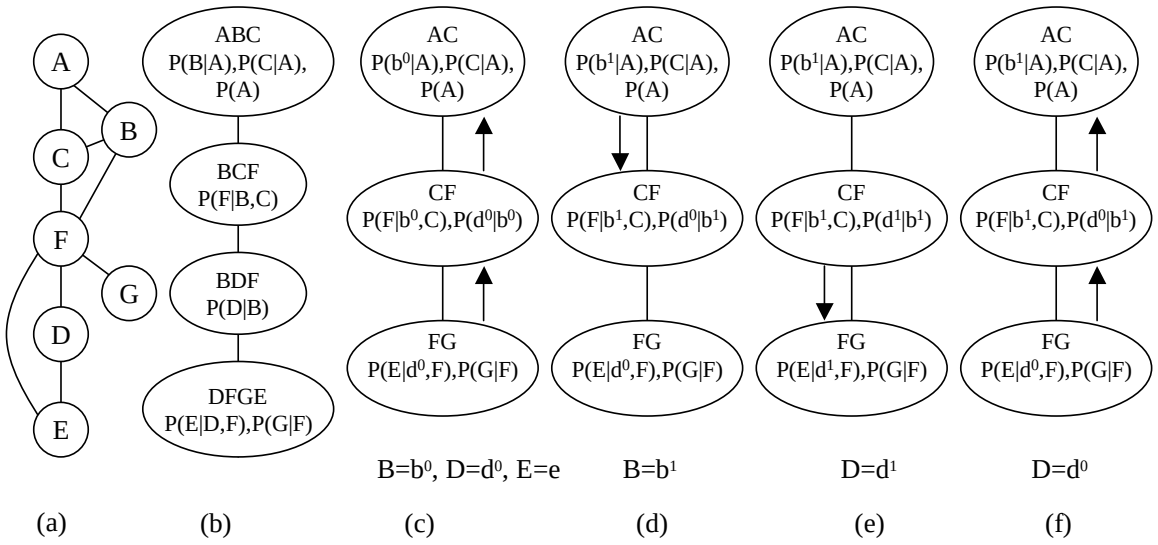


Figure 2.5: A join-tree of width 2 (b) for a moral graph (a) is transformed into a join-tree of width 1 (c) when evidence variable E and cutset variables B and D are instantiated (in the process, clusters BDF and BCF are merged into cluster CF). The clusters contain variables and functions from the original network. All nodes have domains of size 2, $\mathcal{D}(B) = \{b^0, b^1\}$, $\mathcal{D}(D) = \{d^0, d^1\}$. Starting with a sample $\{b^0, d^0\}$, messages are propagated in (c)-(e) to first, sample a new value of variable B (d) and then variable D (e). After that, messages are propagated up the tree to compute posterior marginals $P(\cdot|b^1, c^1, e)$ for the rest of the variables (f).

With evidence variable E removed, variables B and D form a 1-cutset. The join-tree of the network with cutset and evidence variables removed is shown in Figure 2.5 (c). Assuming that cutset variables have domains of size 2, we can initialize $B = b^0$ and $D = d^0$.

Selecting cluster AC as the root of the tree, JTC first propagates messages from leaves to the root as shown in Figure 2.5 (c) and then computes $P(b^0, d^0, e)$ in cluster AC. After setting $B = b^1$, updating all functions containing variable B, and propagating messages through the subtree of B (clusters AC and CF in Figure 2.5 (d)), we obtain $P(b^1, d^0, e)$. Normalizing the two joint probabilities, we obtain $P(b|d^0, e)$ and sample a new value of B. Assume we sampled value b^1.

Next we sample a new value for variable D. Thus, we need to compute $P(D|b^1, e)$. The joint probability $P(d^0, b^1, e)$ is readily available since it was computed for sampling a new value of B. Thus, we set $D = d^1$ and compute the second probability $P(d^1, b^1, e)$ updating functions in clusters CF and FG and sending an updated message from CF to FG (Figure 2.5 (e)). We obtain distribution $P(D|b^1, e)$ by normalizing the joint probabilities and sample a new value d^0 for D. Since the value has changed from latest computation, we update again functions in the clusters CF and FG and propagate updated messages in the subtree C_D (send message from CF to FG).

In order to obtain the distributions $P(\cdot|b^1, d^0, e)$ for the remaining variables A, C, F, and G, we only need to send updated messages up the join-tree, from FG to CF and then from CF to AC as shown in Figure 2.5 (f). The last step also serves as the initialization step for the next sample generation.

In this example the performance of cutset sampling is significantly better than its worst case. We have sent a total of 5 messages to generate a new sample while the worst case suggests at least $N \cdot |C| \cdot d = 3 \cdot 2 \cdot 2 = 12$ messages (here, N equals the number of clusters).

2.2.5 On finding w-cutset

Clearly, w-cutset sampling will be effective only when the w-cutset is small. This calls for the task of finding a minimum size w-cutset. The problem is NP-hard; yet, several heuristic algorithms have been proposed. We next briefly survey some of those proposals.

In [74], w-cutset is obtained when processing variables in the elimination order. The next node to be eliminated (selected using some triangulation heuristics) is added to the cutset if its current induced width (or degree) is greater than w. In [43], this idea has been augmented with various heuristics.

In [15], the variables are selected to be included in the cutset using greedy heuristics based on the node's basic graph properties (such as the degree of a node). One scheme starts from empty w-cutset and then heuristically adds nodes to the cutset until a tree-decomposition of width $\leq w$ can be obtained. The other scheme starts from a set $C = X \backslash E$ containing all nodes in the network as a cutset and then removes nodes from the set in some order. The algorithm stops when removing the next node would result in a tree decomposition of width $> w$.

In [16], it was proposed to first obtain a tree-decomposition of the network and then

find the minimal w-cutset of the tree-decomposition (also an NP-hard problem) via a well-known greedy algorithm used for set cover problem. This approach is shown to yield a smaller cutset than previously proposed heuristics and is used for finding w-cutset in our experiments (section 2.5.4) with a modification that a tree-decomposition is re-computed each time a node is removed from the tree and added to the w-cutset.

2.3 Rao-Blackwellised Likelihood Weighting

Given a Bayesian network over a set of variables X with evidence $E \subset X$, E=e, let $C \subset X \setminus E$ be a subset of variables in X, Z=$C \bigcup E$, and m=$|Z|$. Let o=$\{Z_1, ..., Z_m\}$ be a topological ordering of the variables. We can define likelihood weighting over Z as follows. Processing variables in order o, we sample value z_1 from distribution $P(Z_1)$, z_2 from $P(Z_2|z_1)$, and so on. For each $Z_i \in C$, we sample a value z_i from the distribution $P(Z_i|z_1, ..., z_{i-1})$. If $Z_i \in E$, we assign Z_i its observed value z_i. The latter sets $Q(z_i|z_1, ..., z_{i-1}) = 1$ for all $z_i \in e$. The sampling distribution $Q(Z)$ is:

$$Q(Z) = \prod_{Z_i \in C} P(Z_i|z_1, ..., z_{i-1}) \mid_{E=e} \tag{2.11}$$

The weight $w^{(t)}$ of sample t is given by:

$$w^{(t)} = \frac{P(z^{(t)})}{Q(z^{(t)})} = \frac{\prod_{Z_i \in Z} P(z_i^{(t)}|z_1^{(t)}, ..., z_{i-1}^{(t)})}{\prod_{Z_i \in C} P(z_i^{(t)}|z_1^{(t)}, ..., z_{i-1}^{(t)})} \tag{2.12}$$

After cancelling out the common factors in denominator and numerator, we get:

$$w^{(t)} = \prod_{Z_i \in E} P(e_i|z_1^{(t)}, ..., z_{i-1}^{(t)}) \tag{2.13}$$

During sampling, the weight (initialized to 1) is updated every time we encounter an evidence variable $Z_i \in E$ with observed value e_i using:

$$w^{(t)} \leftarrow w^{(t)} \cdot P(e_i|z_1, ..., z_{i-1}) \tag{2.14}$$

A outline of the likelihood weighting on a cutset is given in Figure 2.6.

Likelihood Weighting on a Cutset
Input: A Belief network $\mathcal{B}$, variables $X = (X_1, ..., X_n)$, evidence $E = e$, cutset $C \subset X \backslash E$, topological ordering $o = \{Z_1, ..., Z_m\}$ of variables in $Z = C \cup E$, the desired number of samples T, and the score arrays for every node.
Output: A set of samples $\{c^{(t)}\}$, $t = 1...T$.
1. **Initialize:** $\forall C_i \in C \backslash E$, $\forall c_i \in \mathcal{D}(X_i)$, $w(c_i^{(t)}) = 0$.
2. **Generate Samples:**
For $t \leftarrow 1$ to T do:
Initialize sample weight: $w^{(t)} \leftarrow 1$
For $i = 1$ to m sample a new value $z_i^{(t)}$ for a variable $Z_i \in Z$:
If $Z_i \in E$ then

$$w^{(t)} \leftarrow w^{(t)} \cdot P(z_i|z_1, ..., z_{i-1}) \tag{2.15}$$
$$Z_i \leftarrow z_i \in e \tag{2.16}$$

Else

$$sample\ z_i^{(t)} \leftarrow \begin{cases} P(z_i),\ if\ Z_i\ is\ a\ root \\ P(z_i|z_1, ..., z_{i-1}),\ otherwise \end{cases} \tag{2.17}$$
$$Z_i \leftarrow z_i^{(t)} \tag{2.18}$$

End If
End For(i)
Update score arrays: $\forall C_i \in C \backslash E$, $w(c_i^{(t)}) \leftarrow w(c_i^{(t)}) + w^{(t)}$
End For(t)

Figure 2.6: Algorithm likelihood weighting on a cutset (LWLC).

The main difference between likelihood weighting over cutset C and sampling over all variables X is in computing the sampling distributions. In the latter case, the distribution

$P(X_i|x_1, ..., x_{i-1}) = P(X_i|pa_i)$ is readily available in the conditional probability table of X_i. However, the sampling distribution $P(Z_i|z_1, ..., z_{i-1})$ for LWLC needs to be computed.

Consider the special case when $C \cup E$ is a loop-cutset. In this case, we can compute the probability $P(z)$=$P(c, e)$ in linear time and space using Pearl's belief propagation algorithm. We can show that we can also compute $P(Z_i|z_1, ..., z_{i-1})$ efficiently if we order the variables in Z topologically and restrict our attention to the relevant subnetwork of $Z_1, ..., Z_i$.

THEOREM **2.3.1** *Given Bayesian network over X, evidence $E \subset X$, and cutset $C \subset X \backslash E$, let $Z = C \cup E$ be a loop-cutset. If Z is topologically ordered, then $\forall Z_j \in Z$ the relevant subnetwork of $Z_1, ..., Z_j$ is singly-connected when $Z_1, ..., Z_j$ are observed.*

Proof. Proof by contradiction. Assume that the relevant subnetwork of $Z_1, ..., Z_j$ contains a loop L with sink S. Then, either $S = Z_q$ or S has a descendant Z_q, $1 \leq q \leq j$, (otherwise S is irrelevant). By definition of loop-cutset, $\exists C_m \in L$ s.t. $C_m \neq S$ and $C_m \in C \subset Z$. Therefore, C_m is an ancestor of Z_q. Since variables are topologically ordered and all loop-cutset nodes preceding Z_q are observed, C_m must be observed, thus, breaking the loop, yielding a contradiction. ■

Therefore, if C is a loop-cutset, we can compute the distributions $P(Z_i|z_1, ..., z_{i-1})$ for every $Z_i \in Z$ over the relevant subnetwork of Z_i in linear time and space.

Consequently, the complexity of computing a new sample is proportional to the number of variables in Z and the size of the input N. In summary:

THEOREM **2.3.2 (Complexity)** *Given a Bayesian network over X, evidence E, and a loop-cutset $C \subset X \backslash E$, the complexity of generating one sample using likelihood weighting over a cutset C is $O(|Z| \cdot N)$ where $Z = C \cup E$ and N is the size of the input network.*

When a sample $c^{(t)}$ is generated, we apply belief propagation algorithm once more to

obtain the posterior marginals, $P(X_i|c^{(t)}, e)$, for each remaining variable. Once T samples are generated, we obtain the posterior marginals estimates, similar to Eq. (1.11), by:

$$\hat{P}(c_i|e) = \alpha \sum_{t=1}^{T} w^{(t)} \delta(c_i, c^{(t)}),\ \forall C_i \in C$$

$$\hat{P}(x_i|e) = \alpha \sum_{t=1}^{T} w^{(t)} P(x_i|c^{(t)}, e),\ \forall X_i \in X \backslash C, E$$

2.3.1 Convergence

Likelihood weighting on a loop-cutset (LWLC) has a higher overhead in computing the distributions $P(Z_i|z_1, ..., z_{i-1})$ for $\forall Z_i \in Z$, compared with sampling on a full variable set. However, as mentioned earlier, it converges faster. because the estimates obtained by sampling from a lower-dimensional space have lower variance due to Rao-Blackwell theorem. That is:

$$Var\{\frac{P(Y,C)}{Q(Y,C)}\} \geq Var\{\frac{P(C)}{Q(C)}\}$$

where $P(C) = \sum_y P(Y,C)$ and $Q(C) = \sum_y Q(Y,C)$ [36, 83] A proof can be found in [36] and [83]. Consequently, fewer LWLC samples are needed to achieve the same accuracy as LW.

The information distance between target distribution $P(C|e)$ and sampling distribution $Q(C)$ in LWLC is smaller than the distance between $P(X|e)$ and sampling distribution $Q(X)$. We can show this for the KL-distance [72]:

$$KL(P(X), Q(X)) = \sum_x P(x) \log \frac{P(x)}{Q(x)} \tag{2.19}$$

THEOREM **2.3.3 (Reduced Information Distance)** *Given a Bayesian network expressing probability distribution $P(X)$, evidence E=e, and a cutset $C \subset X \backslash E$, let $Q(X)$ and $Q(C, E)$ denote the likelihood weighting sampling distribution over X and over C, E respectively. Then:*

$$KL(P(C|e), Q(C, E)) \leq KL(P(X|e), Q(X))$$

The proof is given in Appendix A.

2.4 Caching Sampling on a Cutset

Often, we can reduce the computation time of a sampling scheme by caching the generated samples and their probabilities. Caching LW values is of limited benefit since it uses probabilities stored in CPTs. However, in the case of LWLC, caching may compensate in part for the computation overhead. A suitable data structure for caching is a search-tree over the cutset C with a root node C_1. As new variable values are sampled and a partial assignment to the variables $C_1, ..., C_i$ is generated, LWLC traverses the search tree along the path $c_1, ..., c_i$. Whenever a new value of C_i is sampled, the corresponding tree branch is expanded and the current sample weight and the sampling distribution $P(C_i|z_1, ..., z_{i-1})$ are saved in the node C_i. In the future, when generating the same partial assignment $c_1, ..., c_i$, LWLC saves on computation by reading saved distributions from the tree. We will use LWLC-BUF to denote LWLC sampling scheme that uses a memory buffer to cache previously computed probabilities. LWLC-BUF can also update the sampling distributions $P(C_i|z_1, ..., z_{i-1})$ when dead-ends are discovered. Namely, if the algorithm finds that a partial instantiation $z_1, ..., z_i$, cannot be extended to a full tuple with non-zero probability,

then we set $P(C_i|z_1, ..., z_{i-1}) = 0$ and normalize the updated distribution.

Similar considerations apply to Gibbs sampling. The main difference is that we only need to cache joint probabilities $P(c, e)$ in the leaf nodes $C_m = c_m$, where $m = |C|$ and $c_m \in c$, corresponding to the instantiation c, and we don't need to cache any computed values in the inner nodes of the tree. We will denote resulting loop-cutset sampling scheme as LCS-BUF.

2.5 Experiments

In this section, we present empirical studies of cutset sampling algorithms for several classes of problems. We compare Gibbs-cutset sampling schemes, loop-cutset sampling (LCS) and w-cutset sampling, with traditional Gibbs sampling (Gibbs) on a full set of variables. We also compare the performance of full likelihood weighting (LW), sampling over all the variables, against likelihood weighting on a loop-cutset (LWLC) and buffered likelihood weighting on a loop-cutset (LWLC-BUF). In networks with positive distributions, we compare likelihood weighting side by side with Gibbs-based sampling schemes. For reference, we also report the performance of Iterative Belief Propagation (IBP) algorithm.

2.5.1 Methodology

Sampling Methodology

In all Gibbs-based sampling algorithms we restarted Markov Chain every T samples. The samples from each chain (batch) k are averaged separately:

$$\hat{P}_m(x_i|e) = \frac{1}{T}\sum_{t=1}^{T} P(x_i|c^{(t)}, e)$$

The final estimate is obtained as a sample average over M chains:

$$\hat{P}(x_i|e) = \frac{1}{M}\sum_{m=1}^{M} \hat{P}_m(x_i|e)$$

Restarting Markov chain is known to improve the sampling convergence rate. A single chain can become "stuck" generating samples from a single high-probability region without ever exploring large number of other high-probability tuples. By restarting a Markov chain at a different random point, sampling algorithm can achieve better coverage of the sampling space and find other high-probability regions. In our experiments, we did not observe any significant difference in the estimates obtained from a single chain of size $M \cdot T$ or M chains of size T and therefore, we only choose to report the results for multiple Markov chains. However, we rely on the independence of random values $\hat{P}_k(x_i|e)$ to estimate 90% confidence interval for $\hat{P}(x_i|e)$.

In our implementation of Gibbs sampling schemes, we use zero "burn-in" time (see section 1.4.2). One reason is that no reliable method for estimating burn-in time exists for discrete Bayesian networks. Second, our experimental results showed no positive indication that burn-in time would be beneficial. As we mentioned earlier, the first K samples

are thrown away on the assumption that the Markov Chain needs $\approx K$ samples to become stationary. Then, the remaining samples can be guaranteed to be drawn from target distribution. In practice, this gives the sampling algorithm a "pre-processing" time to find the high-probability regions in the distribution $P(C|e)$ in case the algorithm initially spends disproportionally large period of time in low probability regions. Discarding a large number of low-probability tuples obtained initially, the frequency of the remaining high-probability tuples is automatically adjusted to better reflect their weight.

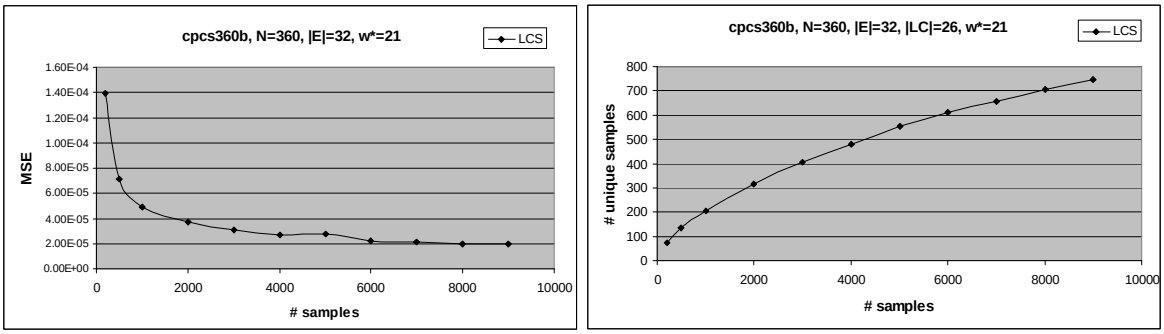


Figure 2.7: Comparing loop-cutset sampling MSE vs. number of samples (left) and and number of unique samples vs. number of samples (right) in cpcs360b, 20 instances.

In our benchmarks, we observed that both full Gibbs sampling and cutset sampling were able to find high probability tuples fast relative to the number of samples generated. For example, in one of the benchmarks, cpcs360b, the rate of generating unique samples, namely, the ratio of cutset instances that have not been seen to the number of samples, decreases over time. Specifically, loop-cutset sampling generates 200 unique tuples after first 1000 samples, additional 100 unique tuples while generating the next 1000 samples, and then the rate of generating unique tuples slows to 50 per 1000 samples in the range

from 2000 to 10000 samples as shown in Figure 2.7, right. That means that after the first few hundred samples, the algorithm spends most of the time revisiting high-probability tuples. In other benchmarks, the number of unique tuple instances generated increases linearly (as in cpcs54) and, thus, the tuples appear to be distributed nearly uniformly. In this case, there is no need for burn-in because there are no strongly-expressed heavy-weight tuples. Instead of using burn-in times, we sample initial variable values from the posterior marginal estimates generated by IBP in all of our experiments.

All experiments were performed on 1.8 GHz CPU.

Measures of Performance

For each problem instance defined by a Bayesian network having variables $X = \{X_1, ..., X_n\}$ and evidence $E \subset X$, $E = e$, we derived the exact posterior marginals $P(X_i|e)$ using bucket-tree elimination [29, 28] and computed the mean square error (MSE) of the approximate posterior marginals $\hat{P}(X_i|e)$ where MSE is defined by:

$$MSE = \frac{1}{\sum_{X_i \in X \setminus E} |\mathcal{D}(X_i)|} \sum_{X_i \in X \setminus E} \sum_{\mathcal{D}(X_i)} [P(x_i|e) - \hat{P}(x_i|e)]^2$$

We averaged MSE over all problem instances, each associated with different observation values.

While the mean square error is our primary accuracy measure, the results are consistent across other well-known measures such as average absolute error, KL-distance, and squared Hellinger's distance which we show only for loop-cutset sampling. The absolute

error Δ is averaged over all values of all unobserved variables:

$$\Delta = \frac{1}{\sum_{X_i \in X \setminus E} |\mathcal{D}(X_i)|} \sum_{X_i \in X \setminus E} \sum_{\mathcal{D}(X_i)} |P(x_i|e) - \hat{P}(x_i|e)|$$

KL-distance D_K between the distribution $P(X_i|e)$ and the estimator $\hat{P}(X_i|e)$ is defined as follows:

$$D_K(P(X_i|e), \hat{P}(X_i|e)) = \sum_{\mathcal{D}(X_i)} P(x_i|e) \log \frac{P(x_i|e)}{\hat{P}(x_i|e)}$$

For each benchmark instance, we compute the KL-distance for each variable $X_i \in X \setminus E$ and then average the results:

$$D_K(P, \hat{P}) = \frac{1}{|X \setminus E|} \sum_{X_i \in X \setminus E} D_K(P(X_i|e), \hat{P}(X_i|e))$$

The squared Hellinger's distance D_H between the distribution $P(X_i|e)$ and the estimator $\hat{P}(X_i|e)$ is obtained as:

$$D_H(P(X_i|e), \hat{P}(X_i|e)) = \sum_{\mathcal{D}(X_i)} [\sqrt{P(x_i|e)} - \sqrt{\hat{P}(x_i|e)}]^2$$

The average squared Hellinger's distance for a benchmark instance is the average of the distances between posterior distributions of one variable:

$$D_H(P, \hat{P}) = \frac{1}{|X \setminus E|} \sum_{X_i \in X \setminus E} D_H(P(X_i|e), \hat{P}(X_i|e))$$

The average errors for different network instances are then averaged over all instances of the given network (typically, 20 instances).

We also report the confidence interval for the estimate $\hat{P}(x_i|e)$ using approach similar to the well-known batch means method [18, 46, 113]. Since chains are restarted independently, the estimates $\hat{P}_k(x_i|e)$ are independent. Thus, the confidence interval can be

obtained by measuring the variance in the estimators $\hat{P}(X_i|e)$. We report results in Section 2.5.5.

2.5.2 Benchmarks

We experimented with several different benchmarks from Bayesian network repository and also random and coding networks. The characteristics of the benchmarks from Bayesian network repository are summarized in Table 2.1.

Table 2.1: Benchmarks' characteristics: N-number of nodes, w^*-induced width, $|LC|$-loop-cutset size, $P(e)$-average probability of evidence (over specified number of instances), T_{BE}-exact computation time by bucket elimination.

	$\mathbf{N}$	$\mathbf{w}^*$	$\mathbf{\|LC\|}$	# instances	$\mathbf{P(e)}$	$\mathbf{T_{BE}}$
cpcs54	54	15	6	20	0.08	1 sec
cpcs179	179	8	8	20	0.00004	2 sec
cpcs360b	360	21	26	20	5E-8	20 min
cpcs422b	422	22	47	20	1.5E-6	50 min
Hailfinder	56	5	5	20	0.05	0.05 sec
Pathfinder1	109	6	9	30	0.07	1 sec
Pathfinder2	135	4	4	30	0.06	0.01 sec
Link	724	15	142	30	0.07	325 sec

CPCS. We considered four CPCS networks derived from the Computer-based Patient Case Simulation system [95, 100]. CPCS network representation is based on INTERNIST 1 [43] and Quick Medical Reference (QMR) [42] expert systems. The nodes in CPCS networks correspond to diseases and findings and conditional probabilities describe their correlations. The exact inference time for cpcs422b is about 50 min. **cpcs54** network consists of $N = 54$ nodes and has a relatively large loop-cutset of size $|LC| = 15$ ($> 25\%$ of the nodes). Its induced width is 15. **cpcs179** network consists of $N = 179$ nodes. Its

induced width is $w^* = 8$. It has a small loop-cutset of size $|LC| = 8$ but with a relatively large corresponding adjusted induced width $w_{LC} = 7$. **cpcs360b** is a larger CPCS network with 360 nodes, adjusted induced width of 21, and loop-cutset $|LC| = 26$. Exact inference on cpcs360b averaged ~ 30 min. The largest network, **cpcs422b**, consisted of 422 nodes with induced width $w^* = 22$ and loop-cutset of size 47.

Hailfinder network is a small network with only 56 nodes. The exact inference in Hailfinder network is easy since its loop-cutset size is only 5. Yet, this network has some zero probabilities and, therefore, is a good benchmark for demonstrating the convergence of cutset sampling in contrast to Gibbs sampling.

Pathfinder is an expert system for providing assistance with the identification of disorders from lymph node tissue sections [53]. We experimented with two subsets of the network. Pathfinder1 contains 109 nodes, has induced width w^*=6, and a loop-cutset of size $|LC| = 9$. The larger network, Pathfinder2, contains 135 nodes, has induced width $w^* = 4$, and a loop-cutset of size 4. These two networks have many deterministic probabilities are only used in evaluation of likelihood weighting schemes.

Link is a model for the linkage between two genes [58]. It has 724 nodes and has induced width $w^* = 15$.

Random networks. We experimented with several classes of random networks: 2-layer networks, random networks, and grid networks. The random networks were generated with $N = 200$ binary nodes (domains of size 2) and R=50 root nodes with uniform priors. The first 100 nodes, $\{X_1, ..., X_{100}\}$, were designated as root nodes. Each non-root

node X_i , $i > 100$, was assigned 3 parents selected randomly from the list of predecessors $\{X_1, ..., X_{i-1}\}$. All nodes were assigned a domain of size 2 $\mathcal{D}(X_i) = \{x_i^0, x_i^1\}$. Evidence nodes E were selected at random from leaf nodes (nodes without children). We will refer to this class of random networks as multi-partite random networks to distinguish from bi-partite (2-layer) random networks.

The random 2-layer networks were generated with 50 root nodes (first layer) and 150 leaf nodes (second layer), yielding a total of 200 nodes. All nodes had domains of size 2. Each non-root node (second layer) was assigned 1-3 parents selected at random from the root nodes. Evidence nodes were selected at random among the non-root nodes. For both 2-layer and multi-partite random networks, the root nodes were assigned uniform priors while conditional probabilities were chosen randomly. Namely, each value $P(x_i^0|pa_i)$ was drawn from uniform distribution over interval $(0, 1)$ and used to compute the complimentary probability value $P(x_i^1|pa_i) = 1 - P(x_i^0|pa_i)$. The conditional probability table values were chosen randomly.

The grid networks of size 15x30 with 450 nodes were also constructed with uniform priors (on the single root node) and random conditional probability tables. Those networks had an average induced width of size 20 (exact inference using bucket elimination required about 30 minutes). Those networks had the most regular structure of all and the largest loop-cutset containing nearly a half of all the unobserved nodes. The evidence nodes were selected at random.

Coding networks. We experimented with coding networks having 200 nodes (50

coding bits, 50 parity check bits). Those networks have an average loop-cutset size of 26 and induced width of 21. The parity check matrix was randomized; each parity check bit had three parents. The Markov chains generated by Gibbs sampling over coding networks are not ergodic due to the deterministic parity check function. As a result, Gibbs sampling does not converge. However, the Markov chain resulting from Gibbs sampling over a subset of coding bits is ergodic and, thus, all of the cutset sampling schemes have converged as we will show in the next two sections.

In coding networks, all coding bits were observed (their values selected at random). In grid networks, evidence variables were selected at random among all network variables. In all other benchmarks, evidence nodes were selected at random among the leaf nodes. For each benchmark, we report on the chart title the number of nodes in the network N, average number of evidence nodes $|E|$, size of loop-cutset $|LC|$, and average induced width of the input instance w^*. We denote the induced width of the input as w^* to distinguish from induced width w of the network adjusted for its w-cutset.

2.5.3 Results for Loop-Cutset Sampling

In this section we compare loop-cutset sampling with pure Gibbs sampling and IBP. In all benchmarks, the cutset was selected so that the evidence and sampling nodes together constitute a loop-cutset of the network using the algorithm proposed in [8]. We show the accuracy of Gibbs and loop-cutset sampling as a function of time.

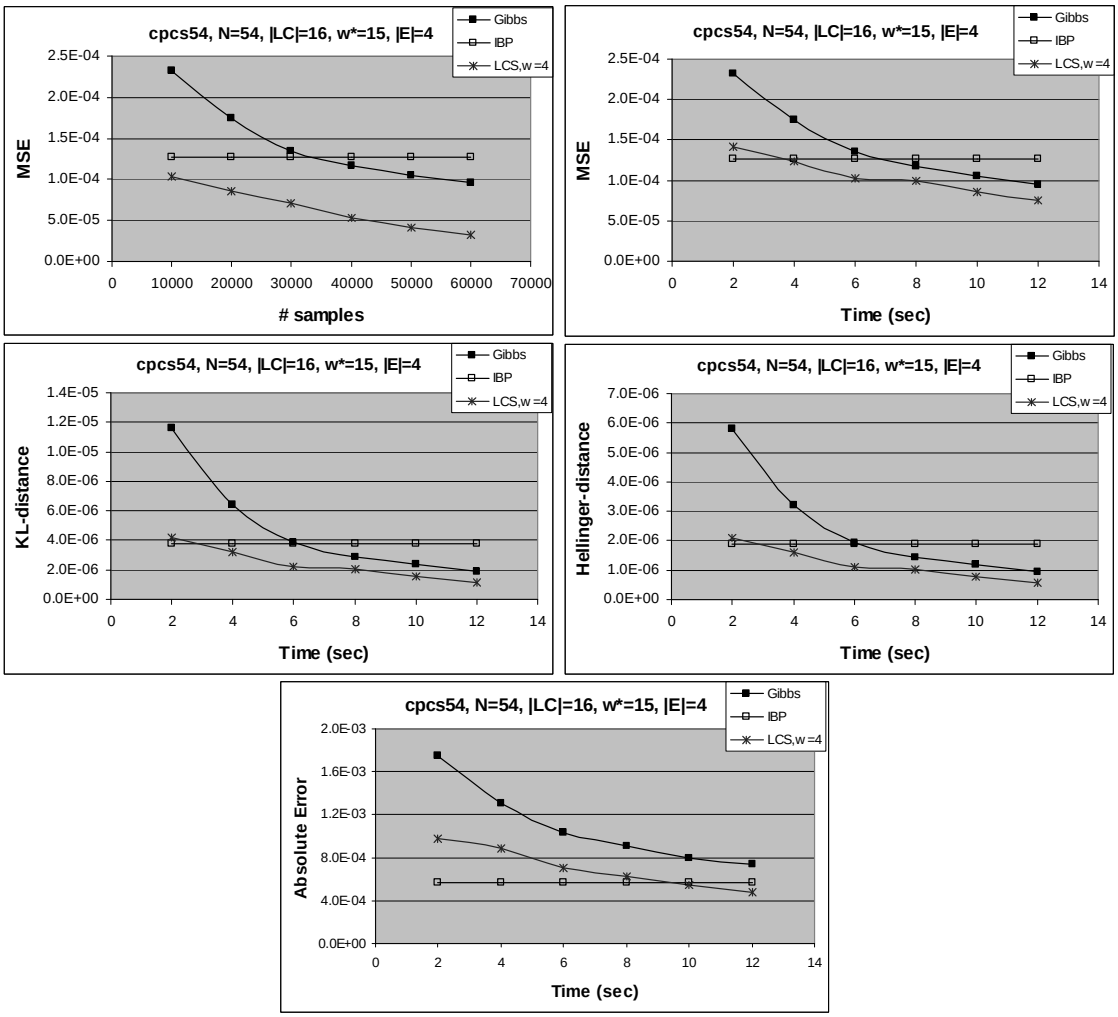


Figure 2.8: Comparing loop-cutset sampling (LCS), Gibbs sampling and IBP on cpcs54 network, averaged over 20 instances each, showing MSE as a function of the number of samples (top left) and time (top right) and KL-distance (middle left), squared Hellinger's distance (middle right), and an average error (bottom) as a function of time.

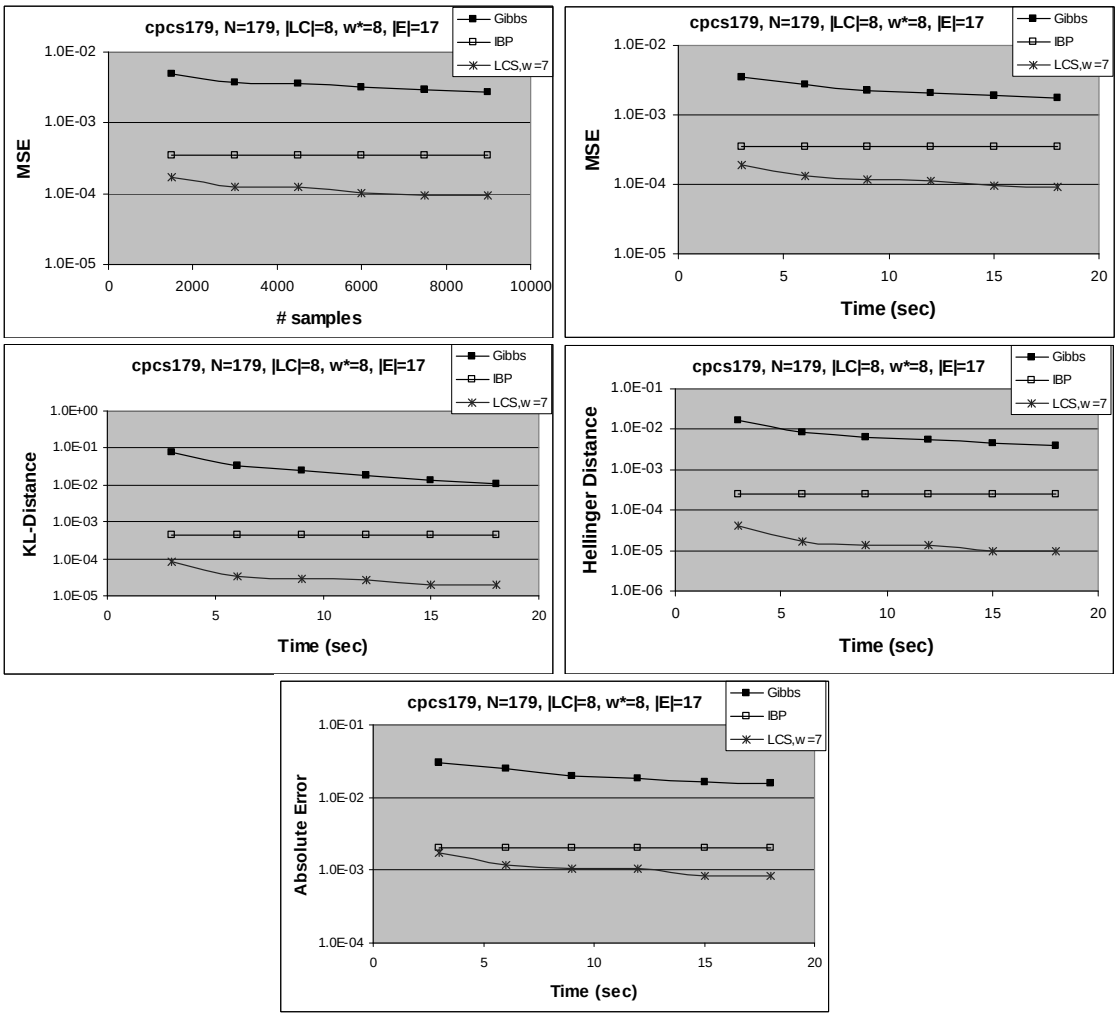


Figure 2.9: Comparing loop-cutset sampling (LCS), Gibbs sampling and IBP on cpcs179 network, averaged over 20 instances each, showing MSE as a function of the number of samples (top left) and time (top right) and KL-distance (middle left), squared Hellinger's distance (middle right), and an average error (bottom) as a function of time.

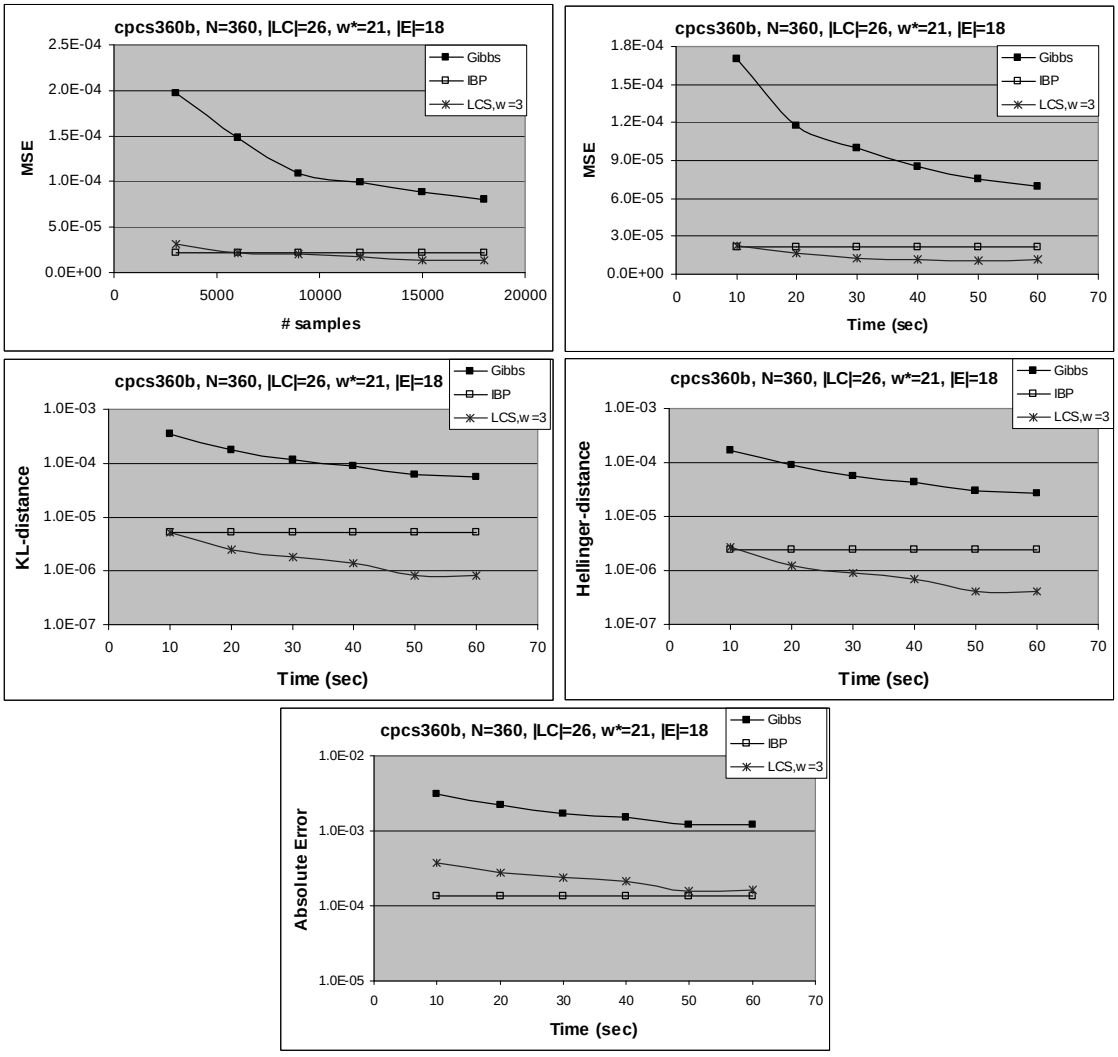


Figure 2.10: Comparing loop-cutset sampling (LCS), Gibbs sampling and IBP on cpcs360b network, averaged over 20 instances each, showing MSE as a function of the number of samples (top left) and time (top right) and KL-distance (middle left), squared Hellinger's distance (middle right), and an average error (bottom) as a function of time.

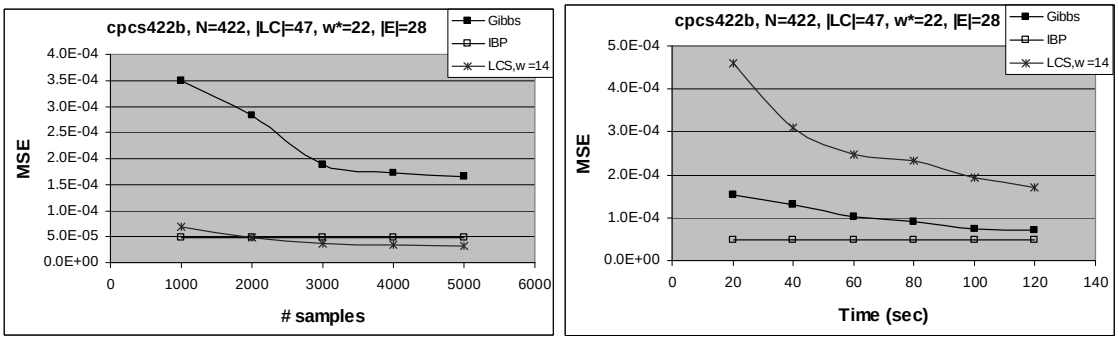


Figure 2.11: Comparing loop-cutset sampling (LCS), Gibbs sampling and IBP on cpcs422b network, averaged over 10 instances, showing MSE as a function of the number of samples (left) and as a function of time (right).

CPCS networks. The results are summarized in Figures 2.14-2.11. Each chart title specifies network name, number of nodes in the network N, the size of evidence set $|E|$, size of loop-cutset (sampling set) $|LC|$, and induced width w^* of the network instance (not taking into account either E or C). The loop-cutset curve in each chart is denoted LCS (for Loop Cutset Sampling). The induced width of the network w when loop-cutset nodes are observed appears next to the name. It is identical to the largest family size in the poly-tree generated when cutset variables are removed. We plot the time on the x-axis and the accuracy (MSE) on the y-axis. IBP curve is always a straight horizontal line because IBP reaches convergence fast (within seconds) and the results do not change after that. The curves corresponding to Gibbs sampling and loop-cutset sampling demonstrate the convergence of the sampling schemes with time. In three CPCS networks we observed that loop-cutset sampling converges much faster than Gibbs sampling. The only exception is cpcs422b (Figure 2.11, right) where loop-cutset sampling generates samples very slowly (2 samples/second) compared to Gibbs sampling (300 samples/second). The reason for

this discrepancy is that the induced width of the conditioned singly-connected network remains high ($w = 14$) due to large family sizes. Since computing sampling distribution is exponential in w, sampling a single variable is $O(2^{14})$ (all variables have domains of size 2). As a result, although loop-cutset sampling shows significant reduction in MSE when comparing the accuracy of two sampling schemes as a function of the number of samples (Figure 2.11, left), it is not enough to compensate for the two orders of magnitude difference in the loop-cutset rate of sample generation. For cpcs54 (Figure 2.8), cpcs179 (Figure 2.9), and cpcs360b (Figure 2.10) loop-cutset sampling achieves greater accuracy than IBP within 10 seconds or less. Note that our sampling time includes the pre-processing time of IBP.

Random networks. In random multi-part networks (Figure 2.12, top) and random 2-layer networks (Figure 2.12, middle), loop-cutset sampling always converged faster than Gibbs sampling. The results are averaged over 10 instances of each network type. In both cases, loop-cutset sampling achieved accuracy of IBP in 2 seconds or less. In 2-layer networks, Iterative Belief Propagation performed particularly poorly. Both Gibbs sampling and loop-cutset sampling obtained more accurate results in less than a second.

Coding Networks. The results for coding networks are shown in Figure 2.12, bottom. We computed MSE over all coding bits and averaged over 10 networks. As we noted earlier, coding networks contain deterministic parity check functions and as a result, Gibbs sampling does not converge while the Markov chain corresponding to sampling over a subspace of code bits only is ergodic and therefore, loop-cutset sampling over a subset of

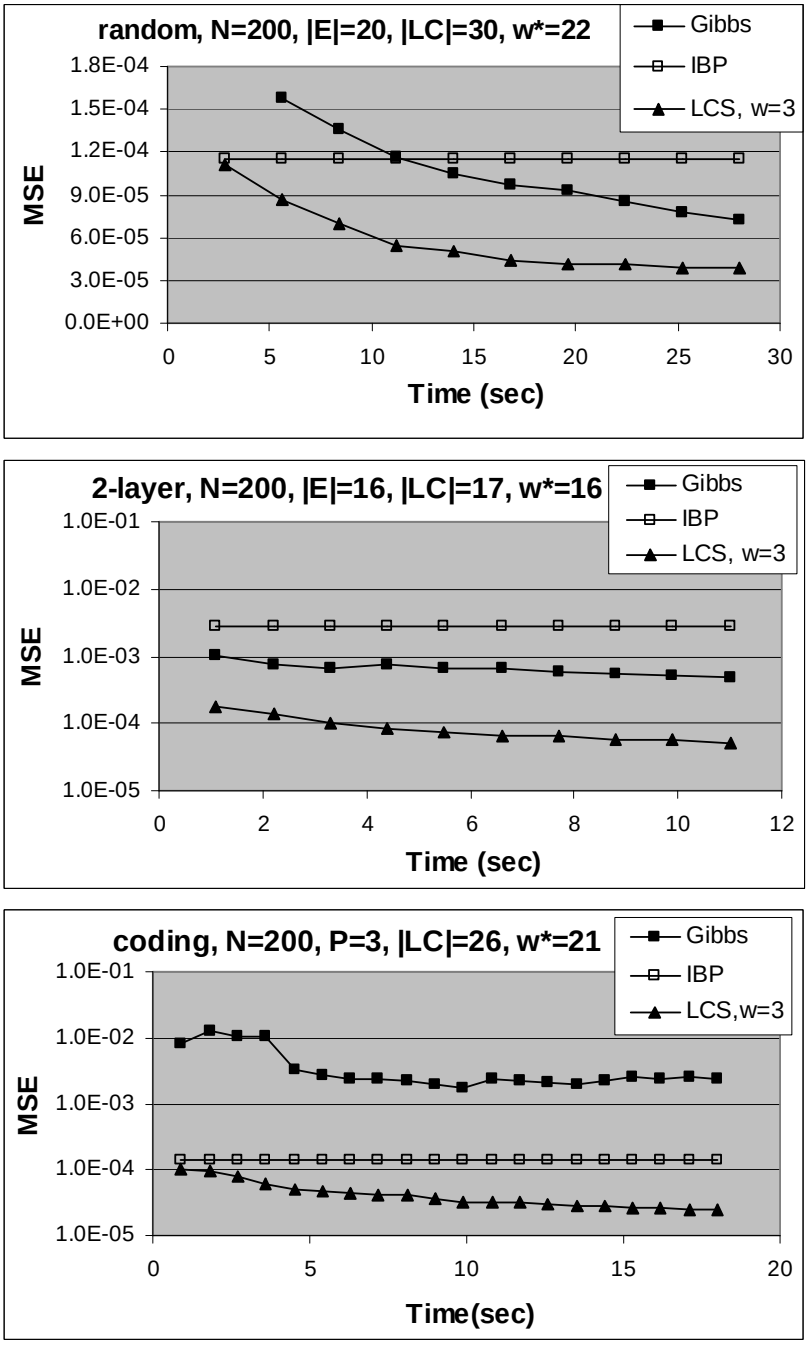


Figure 2.12: Comparing loop-cutset sampling, Gibbs sampling and IBP on random networks (top), 2-layer random networks (middle), and coding networks, σ=0.4 (bottom), averaged over 10 instances each. MSE as a function of time.

coding bits converges and even achieves higher accuracy than IBP with time. In reality, IBP is certainly preferable for coding networks since the size of the loop-cutset grows linearly with the number of code bits.

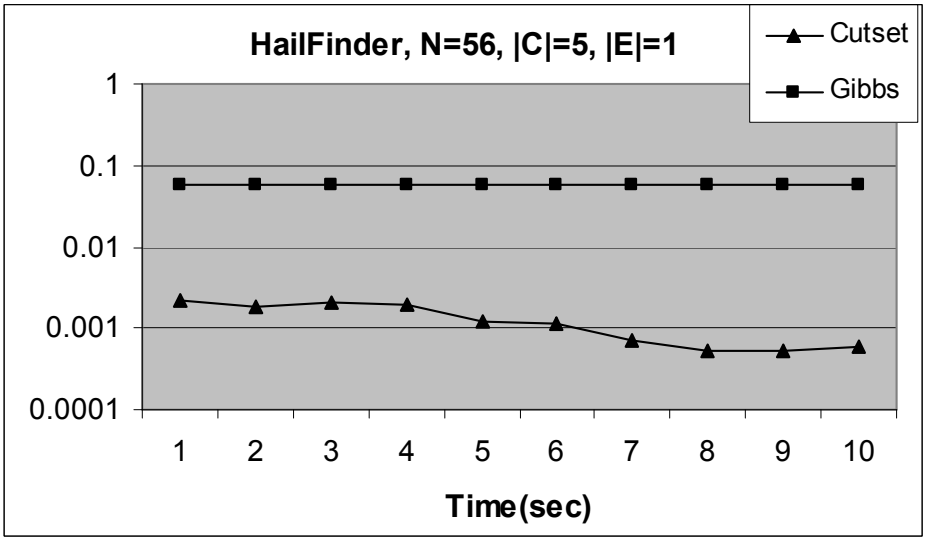


Figure 2.13: Comparing loop-cutset sampling and Gibbs sampling on Hailfinder network, 1 instance. MSE as a function of time.

Hailfinder network is easy to solve exactly. We used this benchmark as an example of a network where Gibbs sampling fails while cutset sampling converges to posterior marginals. Hailfinder network contains many deterministic relationships. Consequently, Markov chain generated by Gibbs sampling is non-ergodic and Gibbs sampling does not converge. However, it turns out the Markov chain corresponding to loop-cutset sampling is ergodic and, therefore, loop-cutset sampling converges to the exact posterior marginals, as we can see in Figure 2.13.

In summary, the empirical results demonstrate that loop-cutset sampling is cost-effective time-wise and superior to Gibbs sampling. We measured the ratio $R = \frac{M_g}{M_c}$ of

the number of samples M_g generated by Gibbs to the number of samples M_c generated by loop-cutset sampling in the same time period (it is relatively constant for any given network and only changes slightly between problem instances that differ with observations). For cpcs54, cpcs179, cpcs360b, and cpcs422b the ratios were correspondingly 2.5, 3.75, 0.7, and 150. We also obtained $R = 2.0$ for random networks and R=0.3 for random 2-layer networks. The ratio values > 1 indicate that Gibbs sampler generates samples faster than loop-cutset sampling which is usually the case. In those instances, variance reduction compensates for the increased computation time because fewer samples are needed to converge resulting in the overall better performance of loop-cutset sampling compared to Gibbs sampling. In some cases, however, the reduction in the sample size also compensates for the overhead computation in the sampling of one variable value so that loop-cutset sampling generates samples faster than Gibbs yielding ratio $R < 1$. Then, the improvement in the accuracy of cutset sampling is due to both larger number of samples and variance reduction.

2.5.4 w-cutset Sampling

In this section, we compare the general w-cutset scheme for different values of w against Gibbs sampling. The main goal is to study how the performance of w-cutset sampling varies with w. For completeness sake, we include results of loop-cutset sampling shown in section 2.5.3.

In this empirical study, we used the greedy algorithm for set cover problem, men-

tioned in section 2.2.5, for finding minimal w-cutset. We apply algorithm in such a manner that each $(w + 1)$-cutset is a proper subset of a w-cutset and, thus, can be expected to have a lower variance and converge faster than sampling on w-cutset in terms of number of samples required (following the Rao-Blackwellisation theory). We focus the empirical study on the trade-offs between cutset size reduction and the associated increase in sample generation time as we gradually increase the bound w.

We used the same benchmarks as before and included also grid networks. All sampling algorithms were given a fixed time bound. When sampling small networks, such as cpcs54 ($w^* = 15$) and cpcs179 ($w^* = 8$), where exact inference is easy, sampling algorithms were allocated 10 seconds and 20 seconds respectively. For larger networks we allocated 100-200 seconds depending on the complexity of the network which was only a fraction of exact computation time.

Table 2.2 reports the size of the sampling set used by each algorithm where each column reports the size of the corresponding w-cutset. For example, for cpcs360b, the average size of Gibbs sample (all nodes except evidence) is 342, the loop-cutset size is 26, the size of 2-cutset is 22, and so on. Table 2.3 shows the rate of sample generation by different algorithms per second. The table shows that in some special cases cutset sampling generated samples faster than Gibbs sampler. For example for cpcs360b, we see that loop-cutset sampling and 2-cutset sampling were able to generate 600 samples per second while Gibbs sampler was able to generate only 400 samples. We attribute this to the size of cutset sample (26 nodes or less as reported in Table 2.2) compared to the size of the Gibbs sample

Table 2.2: Markov chain sampling set size as a function of w.

	Sampling Set Size								
	Gibbs	LC	w=2	w=3	w=4	w=5	w=6	w=7	w=8
cpcs54	51	16	17	15	11	9	8	-	-
cpcs179	162	8	11	9	7	5	-	-	-
cpcs360b	342	26	22	19	16	15	14	13	-
cpcs422b	392	47	65	57	50	45	40	35	-
grid15x30	410	169	163	119	95	75	60	50	13
random	190	30	61	26	25	24	18	17	-
2layer	185	17	22	15	13	13	11	-	-
coding	100	26	38	23	18	18	-	-	-

Table 2.3: Average number of samples generated per second as a function of w.

	No. of Samples								
	Gibbs	LC	w=2	w=3	w=4	w=5	w=6	w=7	w=8
cpcs54	5000	2000, w=4	3000	2400	800	500	300	-	-
cpcs179	1500	400, w=7	400	150	40	10	-	-	-
cpcs360b	400	600, w=3	600	400	160	100	40	20	-
cpcs422b	300	2, w=14	200	150	90	50	30	15	-
grid15x30	2000	500, w=2	300	260	150	105	60	35	20
random	2000	1000, w=3	1400	700	450	300	140	75	-
2layer	200	700, w=3	900	320	150	75	40	-	-
coding	650	450, w=3	800	600	250	150	100	-	-

(over 300 nodes).

CPCS networks. We present two charts. One chart demonstrates the convergence over time for several values of w. The second chart depicts the change in the quality of approximation (MSE) as a function of w for two time points, at the half of the total sampling time and at the end of total sampling time. The performance of Gibbs sampling

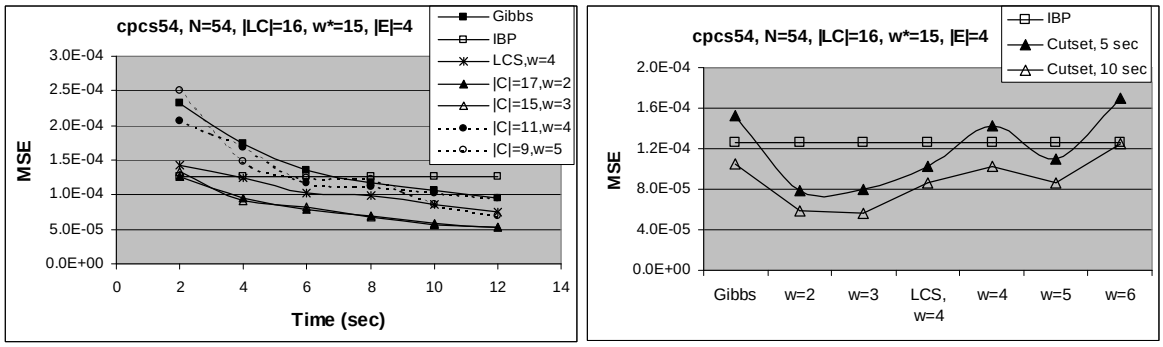


Figure 2.14: MSE as a function of time (left) and w (right) in cpcs54, 20 instances, time bound=12 seconds.

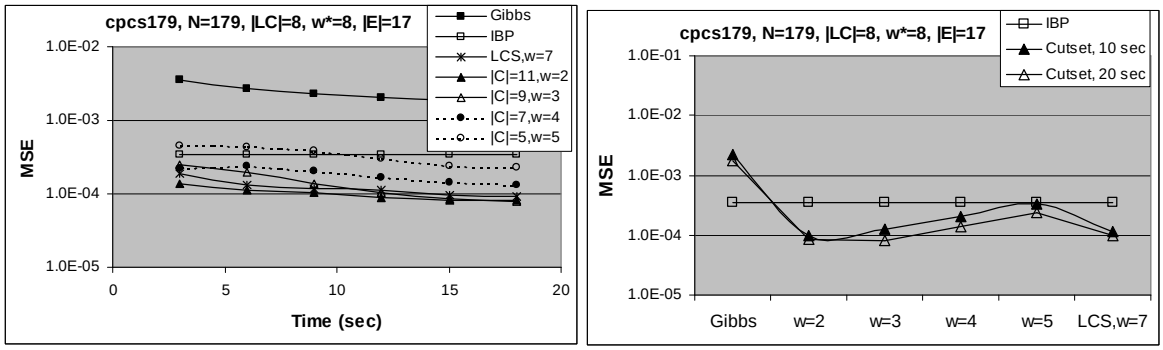


Figure 2.15: MSE as a function of time (left) and w (right) in cpcs179, 20 instances, time bound=12 seconds. Y-scale is exponential due to large variation in performance of Gibbs and cutset sampling.

and cutset sampling for cpcs54 is shown in Figure 2.14. The results are averaged over 20 instances with number of evidence variables in range $1-4$. The graph on the left in Figure 2.14 shows the mean square error of the posterior marginals as a function of time for Gibbs sampling, loop-cutset sampling, and w-cutset sampling for w=2, 3, 4, and 5. The second chart shows accuracy as a function of w. The first point corresponds to Gibbs sampling, other points correspond to cutset sampling with w ranging from 2 to 6. The loop-cutset result is embedded with the w-cutset values at $w = 4$. As explained in section 2.2.3, the loop-cutset corresponds to the w-cutset where w is the maximum number of parents in

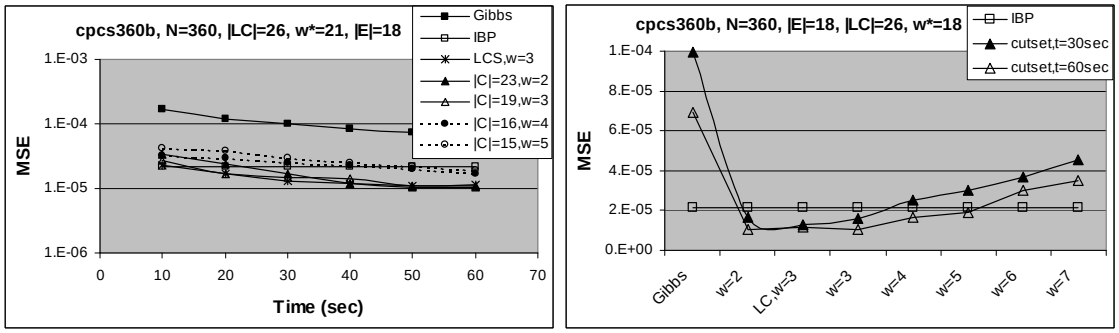


Figure 2.16: MSE as a function of time (left) and w (right) in cpcs360b, 20 instances, time bound=60 seconds. Y-scale is exponential due to large variation in performance of Gibbs and cutset sampling.

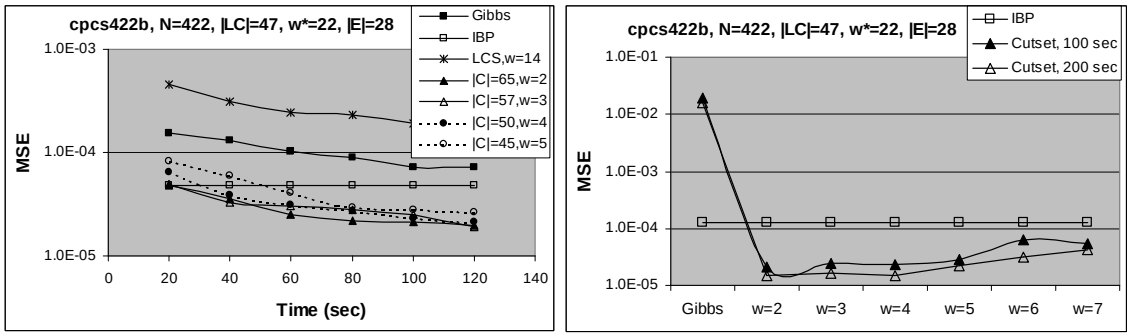


Figure 2.17: MSE as a function of time (left) and w (right) in cpcs422b, 20 instances, time bound=200 seconds. Y-scale is exponential due to large variation in performance of Gibbs and cutset sampling.

the network. Clearly, the best results were obtained by 2- and 3-cutset sampling followed by the loop-cutset sampling.

The results for cpcs179 are reported in Figure 2.15. The chart on the left shows that 2- and 3-cutset sampling are the most accurate having the lowest MSE curves. The loop-cutset curve falls in between 2- and 3-cutset at first and is outperformed by both 2- and 3-cutset after 12 seconds. The 4- and 5-cutset sampling results follow closely behind. Four curves corresponding to loop-cutset sampling and 2-, 3- and 4-cutset sampling fall below the IBP line which means that all three algorithm outperform IBP in the first seconds of

execution (IBP converges in less than a second). The 5-cutset outperforms IBP after about 12 seconds. All cutset sampling algorithms are superior to Gibbs sampling. In Figure 2.15 on the right, we see the accuracy results for all sampling algorithms after 10 seconds and 20 seconds. Clearly, the loop-cutset sampling and w-cutset sampling for w in range from 2 to 5 achieve greater accuracy than Gibbs at both checkpoints. In particular, while Gibbs MSE remains on the order of 1E-02, the MSE for 3-cutset sampling algorithms falls below 1E-04, in the range of 1E-05 after 12 seconds showing two orders of magnitude improvement over Gibbs. Those results are similar to AIS-BN sampling (based on importance sampling principles) results for cpcs179 reported in [21] (factoring into account the difference in the processor speed). Gibbs sampling usually cannot compete with importance sampling algorithms. Although it samples from the target distribution $P(X|e)$, it generates samples slower and does not achieve the same accuracy as importance sampling. The results for cpcs179 indicate that cutset sampling can compete with the state-of-the-art importance sampling algorithms.

In cpcs360b (Figure 2.16), loop-cutset sampling and 2- and 3-cutset sampling have similar performance. The accuracy of the estimates slowly degrades as w increases. Loop-cutset sampling w-cutset sampling substantially outperforms Gibbs sampling for all values w and exceed the accuracy of IBP within 1 minute.

cpcs422b is the largest of the CPCS networks with 422 nodes, the loop-cutset size $|LC| = 47$, and induced width $w^* = 22$. The results are shown in Figure 2.17. The network contains several large CPTs so that the minimum cluster size in any tree-decomposition is

15 unless the nodes in those large functions are observed. As we reported in section 2.5.3, its loop-cutset is relatively small $|LC|$=47 but the adjusted induced width of the network conditioned on the loop-cutset is 14 and thus, sampling just one new loop-cutset variable value is exponential in the big adjusted induced width. As a result, loop-cutset sampling computes only 2 samples per second while the 2-, 3- and 4-cutset, which are only slightly larger having 65, 57, and 50 nodes respectively (see Table 2.2), compute samples considerably faster at rates of 200, 150, and 90 samples per second (see Table 2.3). The 5-cutset that is closest to loop-cutset in size, $|C_5| = 45$, computes 50 samples per second which is more than an order of magnitude more than loop-cutset sampling. Since loop-cutset sampling generated very small number of samples in the time bound of 200 seconds, it performed worse than any other sampling algorithm, including Gibbs sampling (Figure 2.17, left). However, the w-cutset in cpcs422b was able to take advantage of the network structure to be time-wise cost-effective. The chart on the right in Figure 2.17 shows that w-cutset performed well in range of $w = 2 - 7$ and is far superior to Gibbs sampling. When allowed enough time, w-cutset sampling outperformed IBP as well. The IBP converged in 5 seconds. The 2-cutset and 3-cutset improved over IBP after 20 seconds, the 4-cutset after 30 seconds, and the 5-cutset after 50 seconds.

Random networks. Results from random multi-partite and 2-layer networks, 10 instances each, are shown in Figure 2.18. As we can see, w-cutset sampling substantially improves over Gibbs sampling and IBP reaching optimal performance for $w = 2 - 3$ for both classes of networks. In this range, its performance is similar to that of loop-cutset

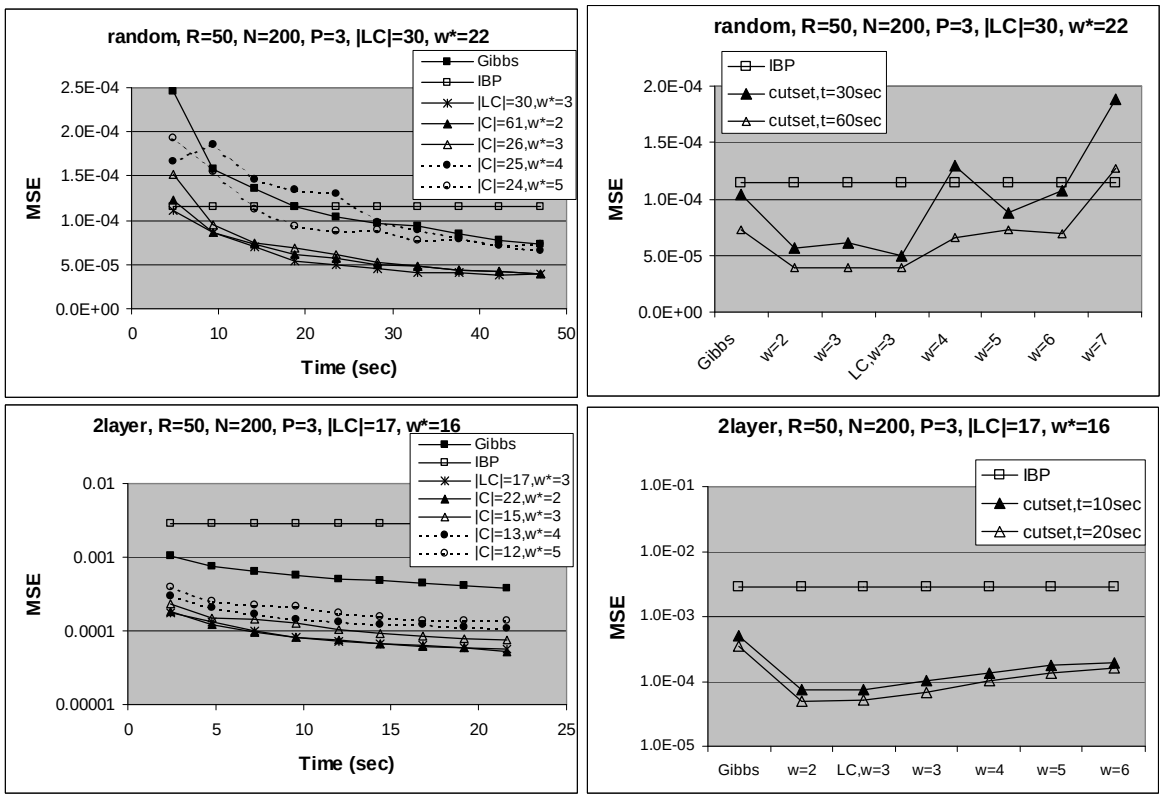


Figure 2.18: Random multi-partite networks (top) and 2-layer networks (bottom), 200 nodes, 10 instances. MSE as a function of the number of samples (left) and w (right).

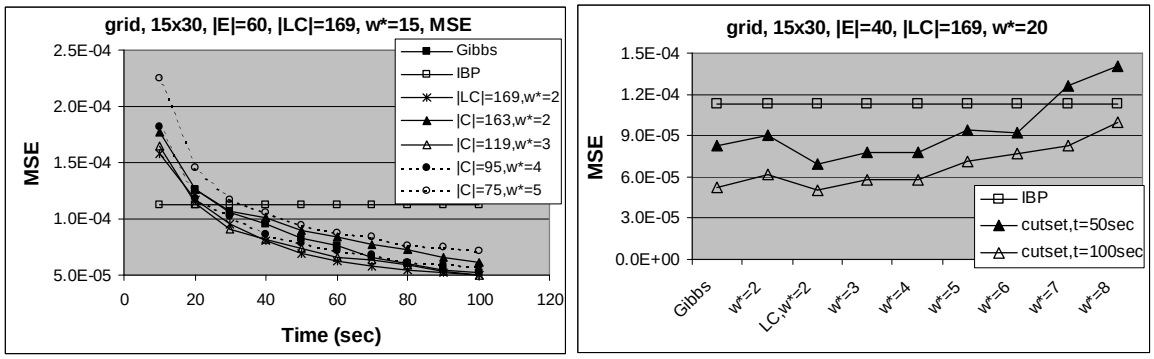


Figure 2.19: Random networks, 450 nodes, 10 instances. MSE as a function of the number of samples (left) and w (right).

sampling. In case of 2-layer networks, the accuracy of both Gibbs sampling and IBP is an order-of-magnitude less compared to cutset sampling (Figure 2.18, bottom right). The poor convergence and accuracy of IBP on 2-layer networks was observed previously [91].

Grid networks. Grid networks having 450 nodes (15x30) were the only class of benchmarks where full Gibbs sampling was able to produce estimates comparable to cutset-sampling (Figure 2.19). With respect to accuracy, Gibbs sampler, loop-cutset sampling, and 3-cutset sampling were best performers and achieved similar results. Loop-cutset sampling was the fastest and most accurate among cutset sampling schemes. Still, it generated samples about 4 times slower compared to Gibbs sampling (Table 2.3) since loop-cutset in this case is relatively large containing about half of all the nodes excluding the evidence. Thus, the reduction in variance compensates for sample computation overhead but it is not enough to outperform Gibbs sampling. The accuracy of loop-cutset sampling was closely followed by 2-, 3- and 4-cutset sampling slowly degrading as w increased further. Grid networks are an example of benchmarks with regular graph structure (that cutset sampling

cannot exploit to its advantage) and small CPTs (in a two-dimensional grid network each node has at most 3 parents) where Gibbs sampling is strong.

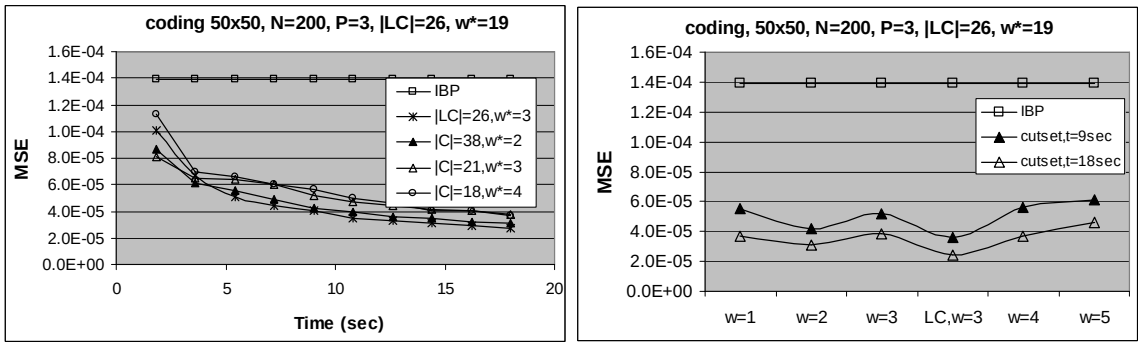


Figure 2.20: Coding networks, 50 code bits, 50 parity check bits, σ=0.4, 100 instances, time bound=6 minutes.

Coding Networks. The cutset sampling results for coding networks are shown in Figure 2.20. Here, the induced width varied from 18 to 22 allowing for exact inference. However, we additionally tested and observed that the complexity of the network grows exponentially with the number of coding bits (even after a small increase in the number of coding bits to 60 yielding a total of 240 nodes after corresponding adjustments to the number of parity-checking bits and transmitted code size, the induced width exceeds 24) while the time for each sample generation scales up linearly. We collected results for 10 networks (10 different parity check matrices) with 10 different evidence instantiations (total of 100 instances). In decoding, the Bit Error Rate (BER) is a standard error measure. However, we computed MSE over all unobserved nodes to evaluate the quality of approximate results more precisely. As mentioned before, since coding networks are not ergodic Gibbs sampling does not converge and is left off the charts. The charts in Figure 2.20 show

that loop-cutset is an optimal cutset for the coding networks whose performance is closely followed by 2-cutset sampling. As we saw earlier, cutset sampling outperforms IBP.

2.5.5 Computing an Error Bound

Second to the issue of convergence of sampling scheme is always the problem of predicting the quality of the estimates and deciding when to stop. In this section, we compare empirically the error intervals for Gibbs and cutset sampling estimates.

Gibbs sampling and cutset sampling are guaranteed to converge to the correct posterior distribution in networks with positive distributions. However, it is hard to estimate how many samples are needed to achieve a certain degree of convergence. It is possible to derive bounds on the absolute error based on sample variance for any sampling method if the samples are independent. In Gibbs and other MCMC methods, samples are dependent and we cannot apply the confidence interval estimate directly. In case of Gibbs sampling, we can apply the *batch means* method that is a special case of *standardized time series* method and is used by the BUGS software package (for more details, see [18, 46, 113]).

The main idea is to "split" a Markov chain of length $M \cdot T$ into M chains of length T. Let $\hat{P}_m(x_i|e)$ be an estimate derived from a single chain $m \in [1, ..., M]$ of length T (meaning, containing T samples) as defined in equations (2.6)-(2.7). The estimates $\hat{P}_m(x|e)$ are assumed *approximately* independent for large enough M. Assuming that convergence conditions are satisfied and the central limit theorem holds, the $\hat{P}_m(x|e)$ is distributed according to $N(E[P(x_i|e)], \sigma^2)$ so that the posterior marginal $\hat{P}(X_i|e)$ is obtained as an average of

Table 2.4: Individual Markov chain length as a function of w. The length of each chain M was adjusted for each sampling scheme for each benchmark so that the total processing time across all sampling algorithms was the same.

		Markov Chain Length T						
	Time	Gibbs	LC	w=2	w=3	w=4	w=5	w=6
cpcs54	20 sec	5000	2000	3000	2400	800	500	-
cpcs179	40 sec	1500	400	400	150	40	10	-
cpcs360b	100 sec	2000	3000	3000	2000	800	500	200
cpcs422b	200 sec	3000	20	2000	1500	900	500	250
grid15x30	100 sec	2000	500	300	260	150	105	60
random	50 sec	2000	1000	1400	700	450	300	140
2layer	20 sec	200	700	900	320	150	75	40
coding	20 sec	650	450	800	600	250	150	100

the M results obtained from each chain, namely:

$$\hat{P}(x|e) = \frac{1}{M}\sum_{m=1}^{M}\hat{P}_m(x|e) \tag{2.20}$$

and the sampling variance is computed as usually:

$$\sigma^2 = \frac{1}{M-1}\sum_{m=1}^{M}(\hat{P}_m(x|e) - \hat{P}(x|e))^2$$

An equivalent expression for the sampling variance is:

$$\sigma^2 = \frac{\sum_{m=1}^{M}\hat{P}_m^2(x|e) - M\hat{P}^2(x|e)}{M-1} \tag{2.21}$$

where σ^2 is easy to compute incrementally storing only the running sums of $\hat{P}_m(x|e)$ and $\hat{P}_m^2(x|e)$. Therefore, we can compute the confidence interval in the $100(1-\alpha)$ percentile used for random variables with normal distribution for small sampling set sizes. Namely:

$$P\left[P(x|e) \in [\hat{P}(x|e) \pm t_{\frac{\alpha}{2},(M-1)}\sqrt{\frac{\sigma^2}{M}}\right] = 1-\alpha \tag{2.22}$$

where $t_{\frac{\alpha}{2},(M-1)}$ is a table value from t distribution with $(M-1)$ degrees of freedom.

We used the batch means approach to estimate the confidence interval in the posterior marginals with one modification. Since we were working with relatively small sample sets (a few thousand samples) and the notion of *large enough* M is not well defined, we restarted the chain after every T samples to guarantee that the estimates $\hat{P}_m(x|e)$ were truly independent. The method of batch means only provides meaningful error estimates assuming that the samples are drawn from the stationary distribution. We assume that in our problems the chains mix fast enough so that the samples are drawn from the target distribution.

Table 2.5: Average absolute error Δ (measured) and estimated confidence interval $\Delta_{0.9}$ as a function of w over 20 Markov Chains.

		Average Error and Confidence Interval						
		Gibbs	LC	w=2	w=3	w=4	w=5	w=6
cpcs54	Δ	0.00056	0.00036	0.00030	0.00030	0.00040	0.00036	0.00067
	$\Delta_{0.9}$	0.00119	0.00076	0.00064	0.00063	0.00098	0.00112	0.00116
cpcs179	Δ	0.01577	0.00086	0.00074	0.00066	0.00113	0.00178	-
	$\Delta_{0.9}$	0.02138	0.00148	0.00111	0.00164	0.00235	0.00392	-
cpcs360b	Δ	0.00051	0.00011	0.00010	0.00008	0.00014	0.00012	0.00022
	$\Delta_{0.9}$	0.00113	0.00022	0.00023	0.00021	0.00030	0.00028	0.00046
cpcs422b	Δ	0.00055	-	0.00018	0.00020	0.00018	0.00027	0.00037
	$\Delta_{0.9}$	0.00119	-	0.00033	0.00035	0.00043	0.00060	0.00074
random	Δ	0.00091	0.00039	0.00119	0.00091	0.00099	0.00109	0.00113
	$\Delta_{0.9}$	0.00199	0.00080	0.00247	0.00205	0.00225	0.00222	0.00239
2layer	Δ	0.00436	0.00066	0.00063	0.00082	0.00117	0.00134	0.00197
	$\Delta_{0.9}$	0.00944	0.00145	0.00144	0.00185	0.00235	0.00302	0.00341
coding	Δ	-	0.00014	0.00019	0.00019	0.000174	-	-
	$\Delta_{0.9}$	-	0.00030	0.00035	0.00034	0.000356	-	-
grid15x30	Δ	0.00108	0.00099	0.00119	0.00091	0.00099	0.00109	0.00113
	$\Delta_{0.9}$	0.00248	0.00214	0.00247	0.00205	0.00225	0.00222	0.00239

We applied this approach to estimate error bound in Gibbs sampler and cutset sampler. We have computed a 90% confidence interval for the estimated posterior marginal

$P(x_i|e)$ based on the sampling variance of $P_m(x_i|e)$ over 20 Markov chains as described above. We computed sampling variance σ^2 from Eq. (2.21) and 90% confidence interval $\Delta_{0.9}(x_i)$ from Eq. (2.22) and averaged over all nodes:

$$\Delta_{0.9} = \frac{1}{N \sum_i |\mathcal{D}(X_i)|} \sum_i \sum_{x_i \in \mathcal{D}(X_i)} \Delta_{0.9}(x_i)$$

The estimated confidence interval can be too large to be practical. Thus, we compared $\Delta_{0.9}$ with the empirical average absolute error Δ:

$$\Delta = \frac{1}{N \sum_i |\mathcal{D}(X_i)|} \sum_i \sum_{x_i \in \mathcal{D}(X_i)} |\hat{P}(x_i|e) - P(x_i|e))$$

The objective of this study was to observe whether the computed confidence interval $\Delta_{0.9}$ (estimated absolute error) accurately reflects the true absolute error Δ, namely, to verify that $\Delta < \Delta_{0.9}$, and if so, then investigate empirically whether confidence interval for cutset-sampling estimates will be smaller compared to Gibbs sampling as we would expect due to variance reduction.

Table 2.5.5 presents the average confidence interval and average absolute error for our benchmarks. For each benchmark, the first row of results (row Δ) reports the average absolute error and the second row of results (row $\Delta_{0.9}$) reports the 90% confidence interval. Each column in Table 2.5.5 corresponds to a sampling scheme. The first column reports results for Gibbs sampling. The second column reports results for loop-cutset sampling. The remaining columns report results for w-cutset sampling for w in range $2 - 6$. The loop-cutset sampling results for cpcs422b are not included due to statistically insignificant number of samples generated by loop-cutset sampling. The Gibbs sampling results for

coding networks are left out because the network contains many deterministic distributions (as mentioned earlier) and Gibbs sampling does not converge.

We can see that for all the networks $\Delta < \Delta_{0.9}$ which validates our method for measuring confidence interval. In most cases the estimated confidence interval is no more than 2-3 times the size of average error and is relatively small. In case of cutset sampling, the largest confidence interval $\max \Delta_{0.9} = 0.00247$ is reported in grid networks for loop-cutset sampling. Thus, we can conclude that confidence interval estimate could be used as a criteria reflecting the quality of the posterior marginal estimate by the sampling algorithm in practice. Subsequently, comparing the results for Gibbs sampling and cutset sampling, we observe not only a significant reduction in the average absolute error, but also a similar reduction in the estimated confidence interval. Across all benchmarks, the estimated confidence interval of Gibbs sampler remains $\Delta_{0.9}$ >0.001. At the same time, for cutset sampling we obtain $\Delta_{0.9}$ <0.001 in 5 out of 8 classes of networks (excluded are cpcs179, grid, and 2-layer networks).

2.5.6 Likelihood Weighting on a Cutset

Next, we present empirical results comparing performance of likelihood weighting schemes. We use the same accuracy measures, namely, average absolute error and mean square error as we did in the case of Gibbs-based sampling schemes. Our benchmarks are two Pathfinder networks, Pathfinder1 and Pathfinder2, Link, and two CPCS networks, cpcs360b and cpcs422b.

Three of the benchmarks, Pathfinder networks and Link network, contain many deterministic probabilities and Gibbs-based sampling estimates do not converge over those networks. In the case of cpcs360b and cpcs422b, all probabilities are positive and, therefore, we compare likelihood weighting schemes against full Gibbs sampling (Gibbs) and Gibbs-based loop-cutset sampling (LCS).

In the Pathfinder and Link networks, the exact posterior marginals are easy to compute by bucket elimination since all three networks have small induced widths. However, they are hard for sampling because of the large number of deterministic relationships. Namely, the target sampling distribution has many zeros where sampling distribution remains positive. As a result, a large number of generated samples can have weight 0. Those samples do not contribute to the computation of the estimates and will be termed *rejected* (or discarded). The percentage of the rejected samples, called a *rejection rate*, is typically used to measure the number of discarded samples. Please note that the term *rejection rate* can have a different connotation when applied to different sampling algorithms. For example, in Metropolis sampling [89], we discard (reject) a sample if its probability is less than some threshold value.

A high rejection rate can significantly slow down the convergence of the sampling estimates. When the evidence is rare, we may need to generate a very large number of samples before we find a single sample of non-zero weight. When all the generated samples are rejected, we will say that the rejection rate is 100% and call the network instance *unresolved*.

Sampling Speed

We generated 30 instances of each Pathfinder network and Link network and 20 instances of cpcs360b and cpcs422b networks with different random observations among the leaf nodes. We generated more instances of deterministic networks since a number of those instances remained unresolved by full likelihood weighting. In Table 2.6, we report the speed of generating samples using LW, LWLC, and LWLC-BUF sampling schemes. As expected, LWLC generates far fewer samples than LW. Notably, the relative speed of LW and LWLC remains the same in the two Pathfinder networks and in Link network. By the time LW generates $100,000$ samples, LWLC generates 1200 samples. Table 2.6 also shows an order of magnitude improvement in the speed of generating samples by LWLC-BUF in cpcs360b, Pathfinder1, and Pathfinder2, a factor of 2 improvement in cpcs422b, and no change in the Link network. The improvement depends on the ratio of unique samples. The number of unique tuples in Pathfinder networks is only $\approx$1% of the total number of samples and, thus, 99% of the computation is redundant. However, in Link network, nearly all samples are unique. Hence, buffering was not beneficial.

Table 2.6: Average # of samples generated by LWLC and LWLC-BUF by the time LW generates $100,000$ samples.

	LW	**LWLC**	**LWLC-BUF**
cpcs360b	100000	2400	24000
cpcs422b	100000	25	50
Pathfinder1	100000	1200	12000
Pathfinder2	100000	1200	12000
Link	100000	1200	1200

Rejection Rates

Table 2.7: Average rejection rates for different benchmarks: **k** -# instances, out of 30, where rejection rate <100%, **R** - average rejection rate.

	LW		**LWLC**		**LW-BUF**	
	k	**R(%)**	k	**R(%)**	k	**R(%)**
PF1	30	47	30	6	30	0.01
PF2	28	77	30	26	30	0.05
Link	17	67	30	16	30	16

The rejection rates of the three likelihood weighting schemes in Pathfinder1, Pathfinder2, and Link are summarized in Table 2.7. For each benchmark, we report the number of instances k (out of 30), where the rejection rate <100%. As we can see, LW resolved all 30 instances of Pathfinder1 but only 28 instances of Pathfinder2 and only 17 instances of Link. LWLC and LWLC-BUF resolved all network instances.

Table 2.7 also reports the rejection rate R averaged over those instances where all three algorithms generated some samples with non-zero probabilities. As we can see, LW has high rejection rates in all benchmarks. The corresponding LWLC rejection rates are a factor of 3 or more smaller. Although lower rejection rate alone does not guarantee faster convergence, it helps compensate for generating fewer samples. The rejection rate of LWLC-BUF is two orders of magnitude lower than LWLC in Pathfinder networks but it is the same as LWLC in Link network (also because most of the samples are unique).

For a given network instance, the rejection rates of LW and LWLC remain unchanged. However, as LWLC-BUF learns zeros of the target distribution, its rejection rate decreases

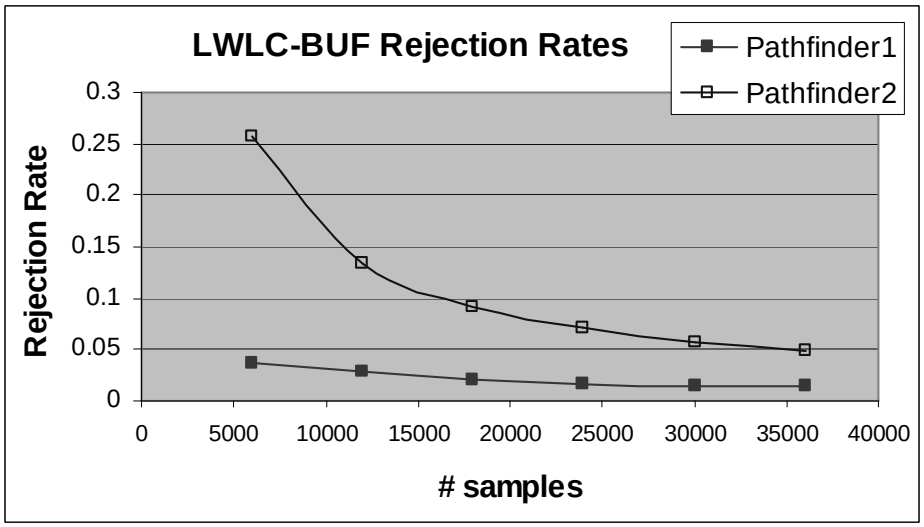


Figure 2.21: LWLC-BUF average rejection rate over 30 network instances in Pathfinder1 and Pahfinder2 as a function of the number of samples.

as the number of samples increases. Figure 2.21 demonstrates this on the example of Pathfinder networks.

Accuracy of the Estimates

Pathfinder1, Pathfinder2, and Link. The accuracy of the approximate posterior marginals for PathFinder1, Pathfinder2, and Link are shown in Figure 2.22. These three networks contain many deterministic probabilities and, subsequently, neither full Gibbs sampling nor loop-cutset sampling can be applied. Hence, we only compare the likelihood weighting schemes. The charts on the left in Figure 2.22 show the average absolute error, the charts on the right show the average MSE.

The comparative behavior of LW, LWLC, and LWLC-BUF sampling schemes is similar in all three networks. LWLC consistently converges faster than LW and outperforms

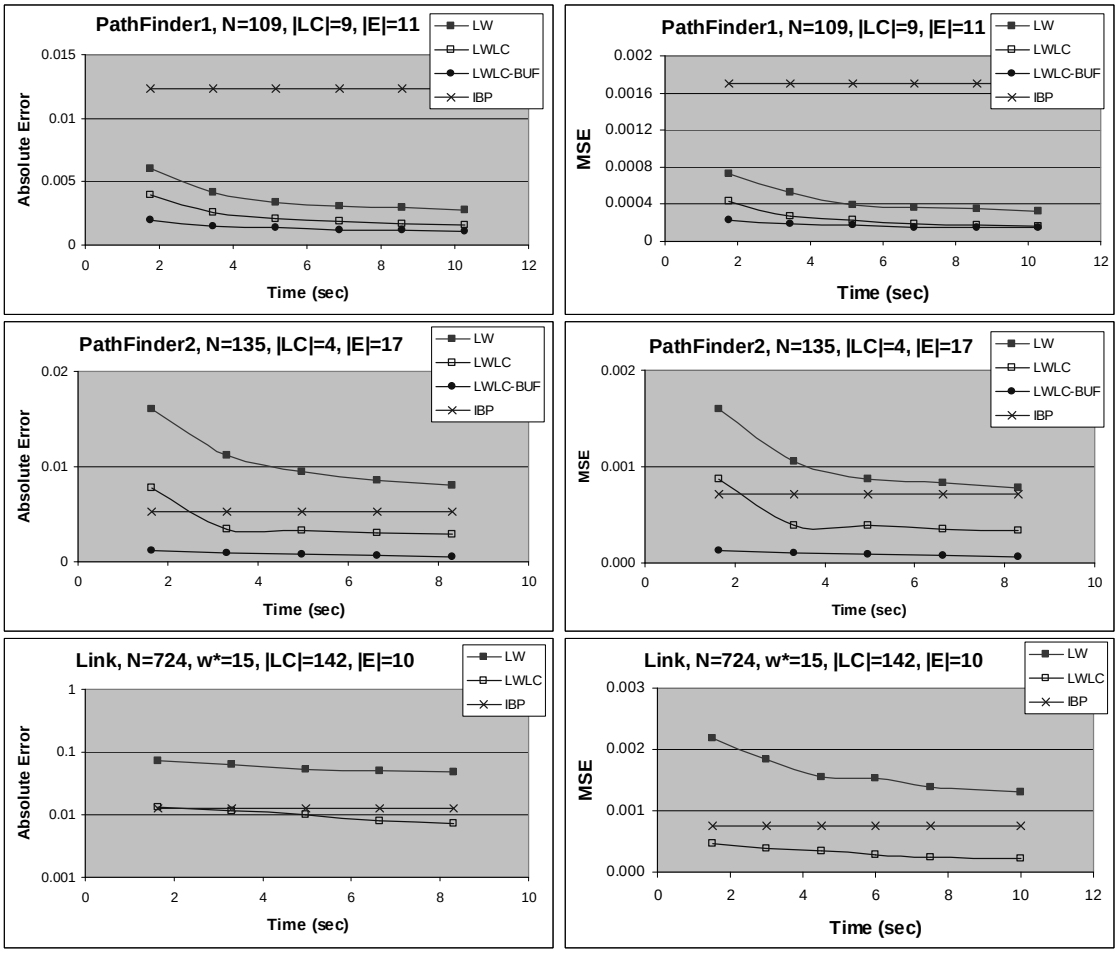


Figure 2.22: Average Error (left) and MSE (right) as a function of time for likelihood weighting (LW) and likelihood weighting on loop-cutset (LWLC), and IBP over 30 instances of Pathfinder1, 28 instances of Pathfinder2, and 17 instances of Link network.

IBP within 2 seconds. LW outperforms IBP within 2 seconds in Pathfinder1 and within 8 seconds in Pathfinder2. However, LW is considerably worse than IBP in Link network. LWLC-BUF converges faster than LWLC in Pathfinder1 and Pathfinder2 because it generates more samples and has a lower rejection rate. In Link network, their performance is the same and, thus, we only show the LWLC curve.

The PathFinder2 network was also used as a benchmark in the evaluation of AIS-BN algorithm [21], an adaptive importance sampling scheme. Although we experimented with different network instances, we can make a rough comparison. Within 60 seconds, AIS-BN computes MSE ≈ 0.0005. Adjusting for the difference in processor speed, the corresponding MSE of LWLC and LWLC-BUF are $\approx$0.004 and $\approx$0.00008, obtained in 6 seconds. Hence, AIS-BN and LWLC-BUF produce comparable results.

In the case of cpcs360b, we break down the analysis into two special cases. First, we compare performance of all the sampling schemes on the instance of the cpcs360b network without evidence. The results are shown in Figure 2.23. Without evidence, the sampling distribution of likelihood weighting equals the target distribution which is the prior distribution $P(X)$. Hence, we can expect that likelihood weighting schemes will perform very well. Indeed, we see in Figure 2.23 (top) that full likelihood weighting outperforms full Gibbs sampling by wide margin. The loop-cutset sampling schemes, LCS and LWLC, obtain very similar results. They improve over full likelihood weighting but only slightly as they compute an order of magnitude fewer samples. While full likelihood weighting computes $100,000$ samples, LWLC computes about $10,000$ samples and LCS computes only

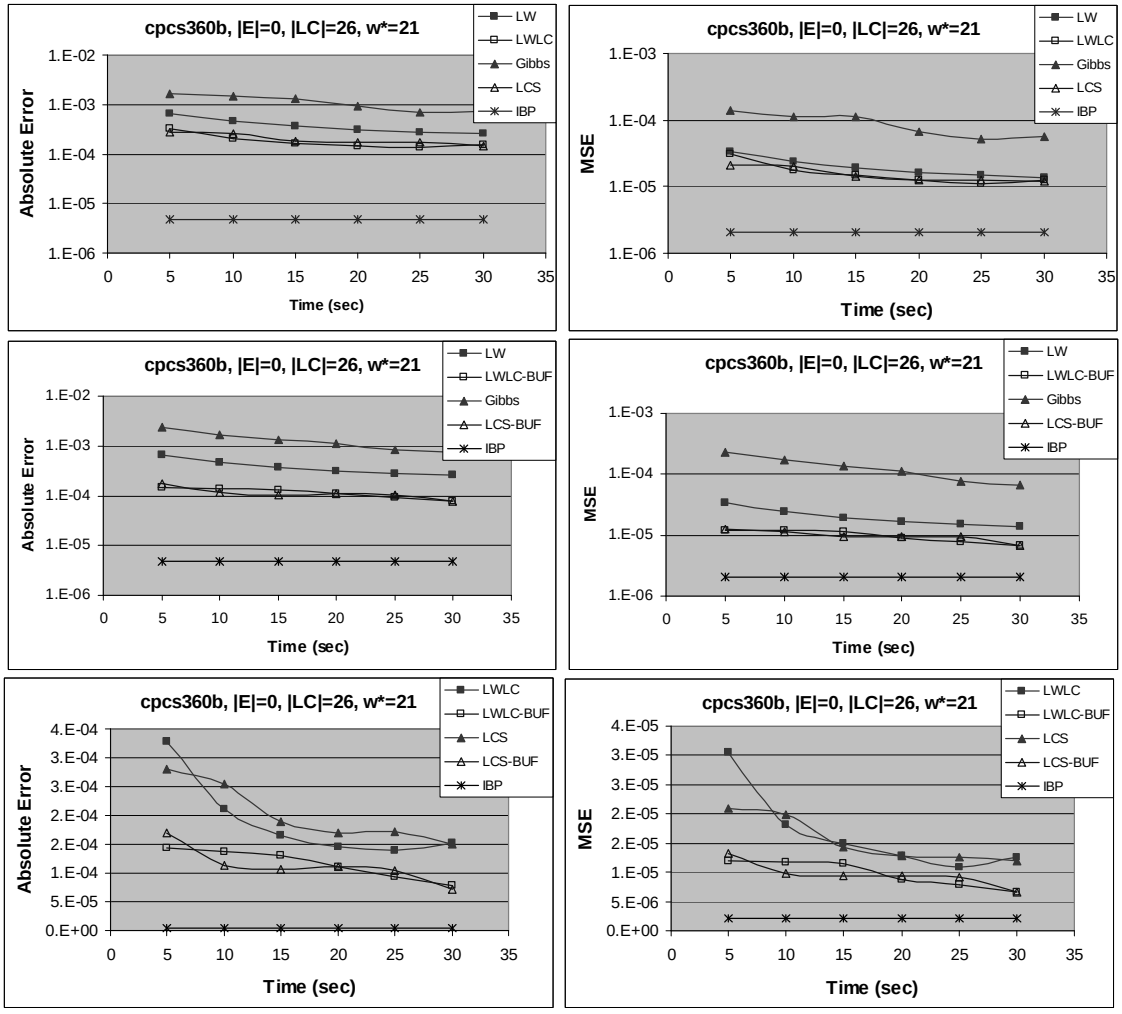


Figure 2.23: Comparing full Gibbs sampling (Gibbs), full likelihood weighting (LW), Gibbs-based loop-cutset sampling (LCS), likelihood weighting on loop-cutset (LWLC), buffered loop-cutset sampling (LCS-BUF), buffered likelihood weighting on loop-cutset (LWLC-BUF), and IBP over cpcs360b without evidence.

$6,000$ samples. Within the same time interval, the cutset-sampling schemes with caching, LCS-BUF and LWLC-BUF, generate on the order of $13,000$ samples and, hence, improve considerably over full likelihood weighting, as shown in Figure 2.23, middle, and over the cutset-sampling schemes without caching, as shown in Figure 2.23, bottom. Again, the performance of the two buffered cutset sampling schemes is very similar.

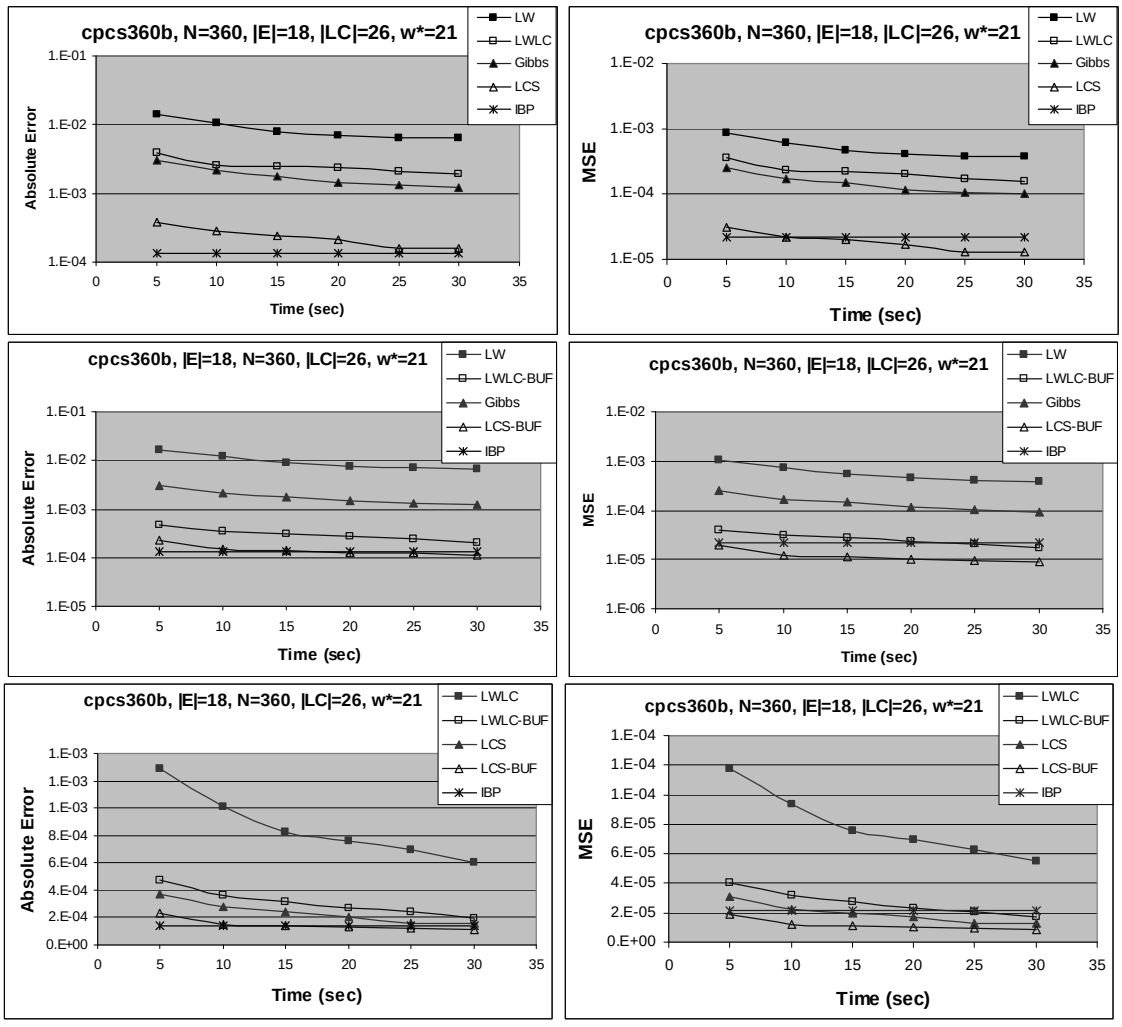


Figure 2.24: Average Error (left) and MSE (right) as a function of time for full Gibbs sampling (Gibbs), likelihood weighting (LW), loop-cutset sampling (LCS), likelihood weighting on loop-cutset (LWLC), buffered loop-cutset sampling (LCS-BUF), buffered likelihood weighting on a cutset (LWLC-BUF), and IBP over cpcs360b with evidence on leaf nodes.

Figure 2.24 shows results for cpcs360b network with randomly selected evidence among the leaf nodes only (nodes without children) averaging 18 evidence nodes per instance. In this case, the sampling distribution of the likelihood weighting remains equal to prior while the target distribution becomes $P(X|e)$. As Figure 2.24 shows, full Gibbs sampling and likelihood weighting now reverse the roles. That is, full likelihood weighting is the worst of all schemes. Full Gibbs sampling outperforms full likelihood weighting, although it generates an order of magnitude fewer samples, and the accuracy of the cutset sampling schemes is considerably better than Gibbs. LCS is the best of the four schemes. Both cutset sampling schemes improve with caching.

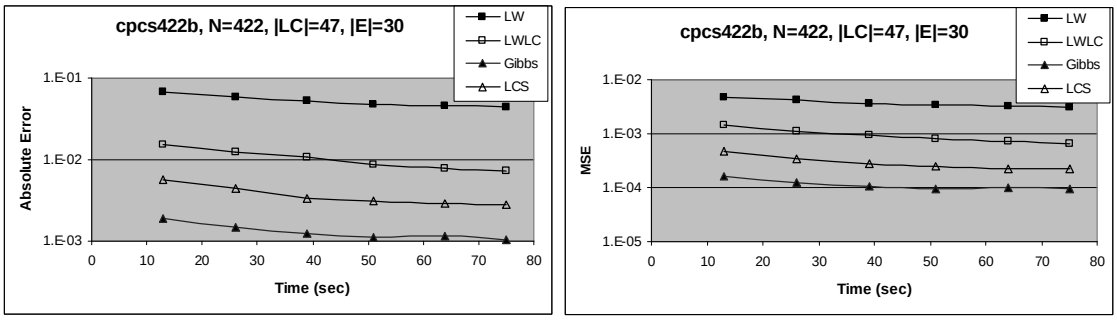


Figure 2.25: Average Error (left) and MSE (right) as a function of time for full Gibbs sampling (Gibbs), likelihood weighting (LW), loop-cutset sampling (LCS) and likelihood weighting on loop-cutset (LWLC) and IBP over cpcs422b with evidence on leaf nodes.

The experiments with cpcs422b consisted of 20 instances of the network with the average of 30 evidence nodes (selected among the leaf nodes) per instance. The results are shown in Figure 2.25. Likelihood weighting over loop-cutset outperforms full likelihood weighting although it still yields to loop-cutset sampling. Gibbs sampling outperforms loop-cutset sampling (for reasons discussed previously, in Section 2.5.3) and produces the

best results of the four schemes.

2.5.7 Summary

Our empirical evaluation of the performance of cutset sampling demonstrates that, except for grid networks, sampling on a cutset usually outperforms Gibbs sampling and offers a considerable anytime improvement over IBP on several networks. We show that convergence of cutset sampling in terms of number of samples dramatically improves as predicted theoretically. More importantly, the performance of cutset sampling remains superior to regular Gibbs sampling in most benchmarks when the sampling time is bounded, even though cutset sampling usually generates fewer samples within the given time interval than Gibbs sampler. The same observations apply in the case of likelihood weighting on a cutset. Although cutset-based likelihood weighting computes an order of magnitude fewer samples, it consistently outperforms full likelihood weighting.

Experiments clearly show that there exists a range of w-values where w-cutset sampling outperforms Gibbs sampler. The performance of w-cutset deteriorates when increasing w results in only a small reduction in the cutset size. An example is cpcs360b network where starting with w=4, increasing w by 1 results in the reduction of the sampling set by only 1 node (shown in Table 2.2).

We observe that the special case of loop-cutset is a good choice of cutset sampling as long as the induced width of network w_{LC} conditioned on loop-cutset is reasonably small. When w_{LC} is large (as in cpcs422b) the loop-cutset sampling is less efficient then w-cutset

sampling for $w < w_{LC}$.

The experiments demonstrate that the relative performance of Gibbs sampling and likelihood weighting depends on the properties of the input network. In particular, it depends on the choice of evidence. We considered two extreme cases, an instance of the network without evidence (most favorable for likelihood weighting) and multiple instances of the network with evidence concentrated in the leaf nodes (the least favorable for likelihood weighting). As expected, in the former case, likelihood weighting performed very well; in the latter case, it produced the worst results. Surprisingly, in cpcs360b Gibbs-based loop-cutset sampling is superior not only to full Gibbs sampling, but also full likelihood weighting and it compares favorably to likelihood weighting on a loop-cutset even when the sampling distribution equals the target distribution. It seems likely that the increased distances between the sampled variables in cutset sampling has an effect of reducing the dependence among samples and, thus, contributes to the faster convergence of the loop-cutset scheme.

2.6 Related Work

We mention here some related work. The idea of marginalising out some variables to improve efficiency of Gibbs sampling was first proposed in [79]. It was successfully applied in several special classes of Bayesian models. In [69], collapsing has been applied to the bivariate Gaussian problem with missing data. In [78], Gibbs sampling algorithm for finding repetitive motifs in biological sequences applies collapsing by integrating out two parame-

ters from the model. Similarly, Gibbs sampling set is collapsed in [38, 84, 80] for learning the nonparametric Bayes problem. In all of the instances above, special relationships between problem variables have been exploited to integrate several variables out resulting in a collapsed Gibbs sampling approach.

In the case of importance sampling, the effectiveness of collapsing of sampling set has been demonstrated in the context of Particle Filtering method for Dynamic Bayesian networks [33, 35, 34]. It was shown that sampling from a subspace combined with exact inference (Rao-Blackwellised Particle Filtering) yields a better approximation than Particle Filtering on the full set of variables. However, the study has been limited to observation of the effect in special cases where some of the variables can be integrated out easily, e.g., when the distributions of the marginalised variables could be computed analytically using a Kalman filter [36, 34, 5] or when the marginalised variables in a factored HMM became conditionally independent (when sampled variables are observed) due to the numerical structure of the CPTs [34].

Compared to this previous research work, our contribution is in defining a generic scheme for collapsing Gibbs sampling and likelihood weighting in Bayesian networks which takes advantage of the network's graph properties and does not depend on the specific form of the relationships between variables. As the cutset selection process can be automated, the proposed cutset sampling schemes can be applied to any Bayesian network.

In [57], sampling and exact inference were combined in a blocking Gibbs sampling scheme. Groups of variables were sampled simultaneously using exact inference to com-

pute the needed conditional distributions. Their empirical results demonstrate a significant improvement in the convergence of the Gibbs sampler over time. Yet, in proposed blocking Gibbs sampling, the sample contains all variables in the network. In contrast, cutset sampling reduces the set of variables that are sampled. As noted previously, collapsing produces lower variance estimates than blocking and, therefore, cutset sampling should require fewer samples to converge.

A different combination of sampling and exact inference for join-trees was described in [68] and [67]. Both papers proposed to sample the probability distribution in each cluster for computing the outgoing messages. In [67], Gibbs sampling was used only for large clusters to estimate the joint probability distribution $P(V_i), V_i \subset X$ in cluster i. The estimated $\hat{P}(V_i)$ is recorded instead of the true joint distribution to conserve memory. The motivation is that only high-probability tuples will be recorded while the remaining low-probability tuples are assumed to have probability 0. In small clusters, the exact joint distribution $P(V_i)$ is computed and recorded. However, the paper does not analyze the introduced errors or compare the performance of this scheme with standard Gibbs sampler or the exact algorithm. No analysis of error is given nor comparison with other approaches.

In [68], sampling is used to compute messages sent from cluster i to cluster j and the posterior joint distributions in a cluster-tree that contains both discrete and continuous variables. This approach subsumes [67] and includes rigorous analysis of the error in the estimated posterior distributions. The method has difficulties with propagation of evidence. The empirical evaluation is limited to two hybrid network instances and compares the qual-

ity of the estimates to those of likelihood weighting, an instance of importance sampling that does not perform well in presence of low-probability evidence.

2.7 Summary and Future Work

The paper presents the w-cutset sampling scheme, a general scheme for collapsing Gibbs sampler in Bayesian networks. We showed theoretically and empirically that cutset sampling improves the convergence rate due to sampling from lower-dimensional space and allows sampling from networks with deterministic probabilities as long as the Markov chain corresponding to sampling over cutset variables is ergodic. Using the induced width w as a controlling parameter, w-cutset sampling provides a mechanism for balancing sampling and exact inference.

We studied the power of cutset sampling when the sampling set is a loop-cutset and, more generally, when the sampling set is a w-cutset of the network (defined as a subset of variables such that, when instantiated, the induced width of the network is $\leq w$). The performance of w-cutset sampling was investigated as a function of the adjusted induced width w. The user can control the trade-offs between sampling and inference in w-cutset sampling by examining the performance of w-cutset sampling for different w values.

We also defined a cutset-based likelihood weighting. By reducing the dimensionality of the sampling space, we achieve reduction in the sampling variance. bbi-lwlc and also reduce the information distance (KL-distance) between the sampling and the target distributions.

Due to reduction in sampling variance, (and also KL-distance in the case of likelihood weighting), cutset sampling schemes require fewer samples to converge than regular sampling. Our experiments confirm faster convergence of cutset sampling as a function of the number of samples over a range of randomly generated and real benchmarks. We also demonstrate that both cutset sampling schemes, one based on Gibbs sampling and the other based on likelihood weighting, are superior or as good as corresponding full sampling schemes time-wise in all of our benchmarks. The cutset-based likelihood weighting scheme also has a lower rejection rate as compared to full likelihood weighting in the deterministic networks.

We improve the performance of cutset-sampling schemes by caching computed samples and their probabilities. In the case of cutset-based likelihood weighting, we also use caching to learn zeros of the target distribution and update the sampling distributions dynamically. Using the same approach, other adaptive importance sampling techniques could be incorporated in the cached version of likelihood weighting in the future.

We showed also that while loop-cutset sampling results are usually comparable to the best results of w-cutset sampling over a range of w values, in some instances 2-cutset and 3-cutset are smaller than loop-cutset and offer better performance although simple heuristics were used for finding the minimal w-cutset. Since the size of cutset and correlations between variables are two main factors contributing to the speed of convergence, w-cutset sampling maybe optimized further with the advancement of methods for finding minimal w-cutset. Another promising direction for future research is to incorporate the heuristics

for avoiding selecting strongly-correlated variables into a cutset since those correlations are driving factors in the speed of convergence of Gibbs sampling. Alternatively, we could combine sample collapsing with blocking.

In summary, w-cutset sampling scheme is a simple yet powerful extension of sampling in Bayesian networks that is likely to dominate regular sampling for any sampling method. While we focused on Gibbs sampling with better convergence characteristics, can be implemented with the cutset sampling principle.

Chapter 3

On finding minimal w-cutset

The complexity of a reasoning task over a graphical model is tied to the induced width of the underlying graph. In the previous chapters, we have already discussed that conditioning (assigning values) on a subset of variables yields a subproblem of the reduced complexity. If the assigned variables constitute a loop-cutset, the rest of the network is singly-connected and therefore can be solved by linear propagation algorithms. More generally, if the cutset and evidence variables form a w-cutset, then exact inference exponential in w. In this chapter, we address the problem of finding a minimal w-cutset in a graph. We relate the problem to that of finding the minimal w-cutset of a tree-decomposition. The latter can be mapped to the well-known *set multi-cover* problem. This relationship yields a proof of NP-completeness on one hand and a greedy algorithm for finding a w-cutset of a tree decomposition on the other. Empirical evaluation of the algorithms is presented.

3.1 Introduction

Reducing the complexity of exact inference by conditioning (assigning values) on a subset of variables is the principle at heart of the well-known loop-cutset conditioning algorithms for Bayesian networks [96] and for constraint networks [27]. Loop-cutset conditioning exploits the fact that if the assigned variables constitute a loop-cutset, the rest of the network

is singly-connected and can be solved by linear propagation algorithms. Recently, the idea of cutset-conditioning was extended to accommodate search on any subset of variables using the notion of w-cutset, yielding a hybrid algorithmic scheme of conditioning and inference parametrized by w [103]. The w-cutset is defined as a subset of nodes in the graph that, when observed, the graph has tree-width of w or less.

The hybrid *w-cutset-conditioning* algorithm applies search to the cutset variables and exact inference (e.g., bucket elimination [28]) to the remaining network. *Given a w-cutset C_w, the algorithm is space exponential in w and time exponential in $w + |C_w|$* [29]. The scheme was applied successfully in the context of satisfiability [103] and constraint optimization [74]. In Chapter 2, the notion of conditioning was explored for speeding up sampling algorithms in Bayesian networks in a scheme called *cutset-sampling*. The idea is to restrict sampling to w-cutset variables only (perform inference on the rest) and thus reduce the sampling variance.

Since the processing time of both search-based and sampling-based schemes grows with the size of the w-cutset, it calls for a secondary optimization task of finding a minimal-size w-cutset. Also of interest is the task of finding the full sequence of minimal w-cutsets, where w ranges from 1 to the problem's induced-width (or tree-width), so that the user can select the w that fits his/her resources. We call the former the *w-cutset problem* and the latter *the sequence w-cutset problem*. The w-cutset problem extends the task of finding minimum loop-cutset (e.g., a 1-cutset), a problem that received a fair amount of attention [9, 8, 119].

The chapter addresses the minimum size w-cutset and, more generally, the minimum weight w-cutset problem. First, we relate the size of a w-cutset of a graph to its tree-width and the properties of its tree-decompositions. Then, we prove that the problem of finding a minimal w-cutset of a given tree decomposition is NP-complete by reduction from the *set multi-cover* problem [119]. Consequently, we apply a well-known greedy algorithm (GWC) for set multi-cover problem to solve the minimum w-cutset problem. The algorithm finds w-cutset within $O(1 + \ln m)$ of optimal where m is the maximum number of clusters of size greater than $w + 1$ sharing the same variable in the input tree decomposition. We investigate its performance empirically and show that, with rare exceptions, GWC and its variants find a smaller w-cutset than the well-performing MGA loop-cutset algorithm [9] (adapted to the w-cutset problem) and a w-cutset algorithm (DGR) proposed in [43].

3.2 Minimal w-cutset of a Graph

We have defined the w-cutset of a graph in Definition 1.3.4 as a subset of variables such that, when observed, the induced width of the graph conditioned on w-cutset is $\leq w$. Clearly, a w-cutset is also a w'-cutset when $w' \geq w$. Further, if C is a w-cutset, then any superset S of C, is also a w-cutset. Next, we define the minimal w-cutset of a graph.

DEFINITION **3.2.1 (minimal w-cutset of a graph)** *The w-cutset C_w is* minimal *if no w-cutset of smaller size exists.*

For completeness, we also define the weighted w-cutset problem that generalizes minimum w-cutset problem (where all node weights are assumed the same) and gives a

refined definition of the complexity of cutset-based algorithms. For example, in w-cutset conditioning, the space requirements of exact inference is $O(d_{max}^w)$ where d_{max} is the maximum node domain size in graph G. The total time required to condition on w-cutset C is $O(d_{max}^w) \times |\mathcal{D}(C)|$ where $|\mathcal{D}(C)|$ is the size of the cutset domain space. The upper bound value $d_{max}^{|C|}$ on $|\mathcal{D}(C)|$ produces a bound on the computation time of the cutset-conditioning algorithm: $O(d_{max}^w) \times d_{max}^{|C|} = O(d_{max}^{w+|C|})$. In this case, clearly, we want to minimize the size of C. However, a more refined optimization task is to minimize the actual value of $|\mathcal{D}(C)|$:

$$|\mathcal{D}(C)| = \prod_{C_i \in C} |\mathcal{D}(C_i)|$$

Since the minimum of $|\mathcal{D}(C)|$ corresponds to the minimum of $\lg(|\mathcal{D}(C)|)$, we can solve this optimization task by assigning each node C_i cost $c_i = \lg |\mathcal{D}(X_i)|$ and minimizing the cost of cutset:

$$cost(C) = \lg |\mathcal{D}(C)| = \sum_{C_i \in C} \lg |\mathcal{D}(C_i)| = \sum_i c_i$$

Similar considerations apply in case of the w-cutset sampling algorithm. Here, the space requirements for the exact inference are the same. The time required to sample a node $C_i \in C$ is $O(d_{max}^w) \times |\mathcal{D}(C_i)|$. The total sampling time is $O(d_{max}^w) \times \sum_{C_i \in C} |\mathcal{D}(C_i)|$. To minimize the total processing time, we assign each node C_i cost $c_i = |\mathcal{D}(C_i)|$ and select the w-cutset of minimum cost:

$$cost(C) = \sum_{C_i \in C} |\mathcal{D}(C_i)|$$

DEFINITION **3.2.2 (weighted w-cutset of a graph)** *Given a reasoning problem $<X, F>$ where each node $X_i \in X$ has associated $cost(X_i) = q_i$, the cost of a w-cutset C_w is given by: $cost(C_w) = \sum_{C_i \in C_w} q_i$. The minimum weight w-cutset problem is to find a min-cost w-cutset.*

In practice, we can often assume that all nodes have the same cost and solve the easier minimal w-cutset problem which is our focus here. In section 3.3, we establish relations between the size of w-cutset of a graph and the width of its tree-decomposition. In section 3.4, we show that the problem is NP-hard even when finding a minimum w-cutset of a chordal graph (corresponding to a tree-decomposition of a graph).

3.3 w-cutset and Tree-Decompositions

In this section, we explore relationship between w-cutset of a graph and its tree-decomposition.

THEOREM **3.3.1** *Given a graph G=$<X, E>$, if G has a w-cutset C_w, then there is a tree-decomposition of G having a tree-width $tw \leq |C_w| + w$.*

Proof. If there exists a w-cutset C_w, then we can remove C_w from the graph yielding, by definition, a subgraph G' over $X \backslash C_w$ that has a tree decomposition T with clusters of size at most $w + 1$. We can add the set C_w to each cluster of T yielding a tree-decomposition with clusters of size at most $w + 1 + |C_w|$ and tree-width $w + |C_w|$. ■

We can conclude therefore that for any graph $tw^* \leq |C_i| + i$ for every i. Moreover,

THEOREM **3.3.2** *Given a graph G, if c_i^* is the size of a smallest i-cutset C_i^*, and tw^* is its tree-width, then:*

$$c_1^* + 1 \geq c_2^* + 2 \geq ... \geq c_i^* + i \geq ... \geq tw^* \tag{3.1}$$

Proof. Let us define $\Delta_{i,i+1} = c_i^* - c_{i+1}^*$, then we claim that $\Delta_{i,i+1} \geq 1$. Assume to the contrary that $c_i = c_{i+1}$, that is $D_{i,i+1}$=0. Since C_i^* is an i-cutset, we can build a tree

decomposition T with maximum cluster size $(i+1)$. Pick some $X_j \in C_i^*$ and add X_j to every cluster yielding tree decomposition T' with maximum cluster size $(i+2)$. Clearly, $C_i^* \backslash X_j$ is an $(i+1)$-cutset of size $c_i^* - 1 = c_{i+1}^* - 1$ which contradicts the minimality of C_{i+1}^*. ■

Given a graph $G = (V, E)$, the w-*cutset sequence* problem seeks a sequence of minimal j-cutsets where j ranges from 1 to the graph's tree-width: C_1^*,...,C_j^*,...,C_{tw^*} = ϕ. Let C'_w be a subset-minimal w-cutset, namely one that does not contain another w-cutset. If we have a w-cutset sequence, we can reason about which w to choose for applying the w-cutset conditioning algorithm or w-cutset sampling. Given a w-cutset sequence we define a function $f(i) = |C_i| + i$ where i ranges from 1 to tw. This function characterizes the complexity of the w-cutset conditioning algorithms where for each i, the space complexity is exponential in i and the time complexity is exponential in $f(i)$. The time-complexity suggests operating with i as large as possible while space consideration suggests selecting i as small as possible. Notice that for various intervals of i, $f(i)$ is constant, if $|C_i| = |C_{i+1}| + 1$. Thus, given a w-cutset sequence, we have that whenever $f(i) = f(i+1)$, then $w = i$ is preferred over $w = i + 1$. Alternatively, given a bound on the space-complexity expressed by r, we can select a most preferred w_p-cutset such that:

$$w_p(r) = \arg\min_j \{r = f(j)\}$$

In the empirical section 3.6, we demonstrate the analysis of function $f(i)$ and its implications.

THEOREM **3.3.3** *Given a tree-decomposition T=(V, E) where V=$\{V_1, ..., V_t\}$ is the set of clusters and given a constant w, a minimum w-cutset C_w^* of G satisfies:*

$$|C_w^*| \leq \sum_{i,|V_i|>w+1} (|V_i| - (w+1)) \tag{3.2}$$

Proof. From each cluster $V_i \in V$ of size larger than (w+1), select a subset of nodes $C_i \subset V_i$ of size $|C_i| = |V_i| - (w+1)$ so that $|V_i \backslash C_i| \leq w+1$. Let $C_w = \cup_{i,|V_i|>w+1} C_i$.
By construction, C_w is a w-cutset of G and: $c_w^* \leq |C_w| = |\cup_i C_i| \leq \sum_i |C_i| = \sum_{i,|V_i|>w+1} |V_i| - (w+1)$. ■

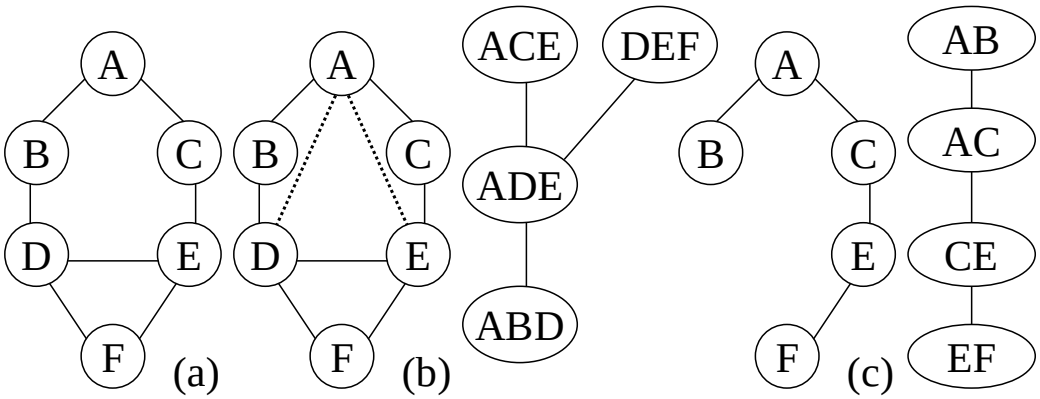


Figure 3.1: (a) Graph; (b) triangulated graph and corresponding tree decomposition of width 2; (c) graph with 1-cutset node $\{D\}$ removed and corresponding tree-decomposition.

Since a w-cutset yields a tree-decomposition having $tw = w$, it looks reasonable when seeking w-cutset to start from a good tree-decomposition and find its w-cutset (or a sequence). In particular, this avoids the need to test if a graph has $tw = w$. This task is equivalent to finding a w-cutset of a chordal (triangulated) graph.

DEFINITION **3.3.1 (A w-cutset of a tree-decomposition)** *Given a tree decomposition T=$<V, E>$ of a reasoning problem $<X, F>$ where V is a set of subsets of X then $C_w^T \subset X$ is a w-*cutset *relative to T if for every i, $|V_i \backslash C_w^T| \leq w+1$.*

We should note upfront, however, that a minimum-size w-cutset of T (even if T is optimal) is not necessarily a minimum w-cutset of G.

Example 3.3.4 *Consider a graph in Figure 3.1(a). An optimal tree decomposition of width* 2 *is shown in Figure 3.1(b). This tree-decomposition clearly does not have a* 1*-cutset of size* < 2*. However, the graph has a* 1*-cutset of size* 1*,* $\{D\}$*, as shown in Figure* 3.1*(c).*

On the other hand, given a minimum w-cutset of a graph G, removing the w-cutset from the graph yields a graph having $tw^* = w$. If not, then there exists a tree-decomposition of G over $X \backslash C_w$ having $tw < w$. Select such a tree and select a node in C_w that can be added to the tree-decomposition without increasing its tree-width beyond w. Such a node must exist, contradicting the minimality of C_w.

It is still an open question if every minimal w-cutset of graph G is a w-cutset of some minimum-width tree-decomposition of G.

3.4 Hardness of Minimal w-Cutset of a Tree Decomposition

While it is obvious that the general w-cutset problem is NP-complete (1-cutset is a loop-cutset and finding the minimal loop-cutset is known to be NP-complete), it is not clear that the same holds relative to a given tree-decomposition. We now show that, given a tree-decomposition T of a hypergraph $\mathcal{H}$, the w-cutset problem for T is NP-complete. We use a reduction from *set multi-cover* (SMC) problem.

DEFINITION **3.4.1 (Set Cover (SC))** *Given a pair $< U, S >$ where U is universal set and S is a set of subsets S=$\{S_1, ..., S_m\}$ of U, find a minimum set $C \subset S$ s.t. each element of U is covered at least once: $\cup_{S_i \in C} S_i = U$.*

DEFINITION **3.4.2 (Set Multi-Cover(SMC))** *Given a pair $< U, S >$ where U is a universal set and S is a set of subsets $S = \{S_1, ..., S_m\}$ of U, find a minimum cost set $C \subset S$ s.t. each $U_i \in U$ is covered at least $r_i > 0$ times by C.*

The SC is an instance of SMC problem when $\forall i, r_i = 1$.

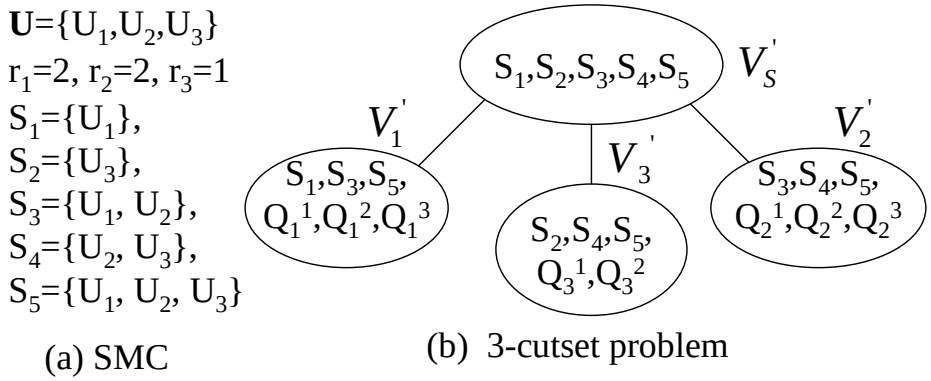


Figure 3.2: (a) A set multi-cover problem $<U, S>$ where U=$\{U_1, U_2, U_3\}$, S=$\{S_1, ..., S_5\}$, the covering requirements r_1=2, r_2=2, r_3=1. (b) Corresponding augmented tree decomposition T'=$<V', E>$ over S'=$\{S_1, ..., S_5, Q_1^1, Q_1^2, Q_1^3, Q_2^1, Q_2^2, Q_2^3, Q_3^1, Q_3^2\}$.

THEOREM **3.4.1 (NP-completeness)** *The problem* "Given a tree-decomposition T=$<V, E>$ and a constant k, does there exist a w-cutset of T of size at most k ?" *is NP-complete.*

Proof. Given a tree decomposition T=$<V, E>$ over X and a subset of nodes $C \in X$, we can verify in linear time whether C is a w-cutset of T by checking if $\forall V_i \in V$, $|V_i \backslash C| \leq w + 1$. Now we show that the problem is NP-hard by reduction from set multi-cover.

Assume we are given a set multi-cover problem $<U, S>$, where U=$\{X_1, ..., X_n\}$ and S=$\{S_1, ..., S_m\}$, a covering requirement $r_i > 0$ for each $U_i \in U$.

We define a cluster tree T=$<V, E>$ over S where there is a node $V_i \in V$ corresponding to each variable U_i in U that contains all subsets $S_j \in S$ that cover node X_i: $V_i = \{S_j \in S | X_i \in S_j\}$. Additionally, there is a node $V_S \in V$ that contains all subsets in S: $V_S = S$. Thus, $V = \{V_i | U_i \in U\} \cup V_S$. Denote $|V_i| = f_i$. The edges are added between each cluster $V_{i, i \neq s}$ and cluster V_S: $E = \{V_i V_S | U_i \in U\}$ to satisfy running intersection property in T.

Define w+1=$|S| - \min_i r_i = m - \min_i r_i$. For each $V_{i,i\neq s}$, since $|V_i|$=$f_i \leq m$ and $r_i > 0$, then $f_i - r_i \leq m - \min_i r_i \leq w + 1$. Consequently, $f_i \leq r_i + w + 1$.

For each V_i s.t. $f_i < r_i + w + 1$, define Δ_i=$r_i + w + 1 - f_i$ and augment cluster V_i with a set of nodes $\mathcal{Q}_i = \{Q_i^1 ... Q_i^{\Delta_i}\}$ yielding a cluster $V_i' = V_i \cup \mathcal{Q}_i$ of size $|V_i'|$=f_i'=$r_i + w + 1$.

We will show now that a set multi-cover $<U, S>$ has a solution of size k iff there exists a w-cutset of augmented tree decomposition T'=$<V', E>$ of the same size.The augmented tree for sample SMC problem in Figure 3.2(a) is shown in Figure 3.2(b).

Let C be a set multi-cover solution of size k. Then, $\forall U_i \in U$, $|C \cap V_i'| \geq r_i$ which yields $|V_i' \backslash C| \leq |V_i'| - r_i = f_i' - r_i = w + 1$. Since $|C| \geq \min_i r_i$, then $|V_S \backslash C| \leq |V_S| - \min_i r_i = m - \min_i r_i = w + 1$. Therefore, C is a w-cutset of size k.

Let C_w be a w-cutset problem of size k. If C_w contains a node $Q_j \in \mathcal{Q}_i$, we can replace it with some node $S_p \in V_i'$ without increasing the size of the cutset. Thus, without loss of generality, we can assume $C_w \subset S$. For each V_i' corresponding to some $U_i \in U$, let $C_i = C_w \cap V_i'$. By definition of w-cutset, $|V_i' \backslash C_w| \leq w + 1$. Therefore, $|C_i| \geq |V_i'| - (w + 1) = f_i' - (w + 1) = r_i$. By definition, C_w is a cover for the given SMC problem.

Minimum w-cutset problem is NP-hard by reduction from set multi-cover and is verifiable in linear time. Therefore, minimum w-cutset problem is NP-complete. ■

Example 3.4.2 *We demonstrate the steps for the SMC problem with U=$\{U_1, U_2, U_3\}$ and S=$\{S_1, ..., S_5\}$ shown in Figure 3.2(a). Define T=$<V, E>$, V=$\{V_1, V_2, V_3, V_s\}$, over S:*
V_1=$\{S_1, S_2, S_3\}$, f_1=3, V_2=$\{S_3, S_4, S_5\}$, f_2=3,
V_3=$\{S_2, S_4, S_5\}$, f_3=3, V_S=$\{S_1, ..., S_5\}$, f_S=5.
Then, $w = |S| - 1 - \min_i r_i = 5 - 1 - 1 = 3$. *Augment:*
V_1: Δ_1=w+1+$r_1 - f_1$=4+2-3=3, Q_1=$\{Q_1^1, Q_1^2, Q_1^3\}$.
V_2: Δ_2=w+1+$r_2 - f_2$=4+2-3=3, Q_2=$\{Q_2^1, Q_2^2, Q_2^3\}$.
V_3: Δ_3=w+1+$r_3 - f_3$=4+1-3=2, Q_3=$\{Q_3^1, Q_3^2\}$.
The augmented tree decomposition T' is shown in Figure 3.2(b). Any SMC solution such as C=$\{S_3, S_5\}$ is a 3-cutset of T and vice versa.

In summary, we showed that when w is not a constant the w-cutset problem is NP-complete. This implies that the w-cutset sequence problem over tree-decompositions is hard.

3.5 Algorithms for minimal w-cutset of a tree-decomposition

In this section, we propose two algorithms for finding a minimal w-cutset. One is the dynamic programming algorithm for finding the optimal solution (having exponential complexity). The other is a factor 2 approximation scheme based on a simple greedy algorithm.

3.5.1 An exact algorithm for minimal w-cutset of a tree-decomposition

Assume we are given a tree-decomposition T=<V, E> over X. Assume that the tree-width of T is w^* and we are given a target bound value w. For each cluster V_i s.t. $|V_i| > w + 1$, define $r_i = |V_i| - (w + 1)$. Let $C^*_w(T)$ denote minimum size w-cutset of T.

Before we define an algorithm for finding an exact minimum size w-cutset, we will outline a few pre-processing steps that can be taken to solve the "trivial" part of the problem and reduce its complexity.

THEOREM **3.5.1** *Given a tree-decomposition T=<V, E> over X of width w^* and a constant $w < w^*$, assume there is a cluster $V_i \in V$ s.t. $|V_i| > w+1$ and V_i is singly-connected in T. In other words, $r_i = |V_i| - (w + 1) > 0$ and V_i has only 1 neighbour V_j. Let S_{ij} be the separator between V_i and V_j: $S_{ij} = V_i \cap V_j$. If $|S_{ij}| \leq r_i$, then there is a minimum-size w-cutset $C^*_w(T)$ such that $S_{ij} \subset C^*_w(T)$.*

Proof. Assume some node $X_k \in S_{ij}$ is not in the minimum-size cutset. Then, $\exists X_q \in V_i$ s.t. $X_q \notin S_{ij}$ and $X_q \in C^*_w(T)$. Such a node must exist to satisfy r_i. If $X_q \notin S_{ij}$, then $\forall j \neq i$, $X_q \notin V_j$ and we can replace X_q with X_k in the cutset without increasing the size of the w-cutset. ■

As a consequence, whenever we have a singly-connected cluster V_i whose only sepa-

rator $|S_{ij}| \leq r_i$, we can safely add to cutset any subset S_i of nodes in V_i such that $S_{ij} \subset S_i$ and $|S_i| = r_i$. Then, we remove from T all nodes in S_{ij} as well as the cluster V_i and obtain a tree T'=<V', E'> where V' = $V \backslash S_{ij}$ where $C_w^*(T) = S_i \cup C_w^*(T')$. Thus, without loss of generality, we assume that for any singly-connected cluster V_i with some $r_i > 0$, its separator size $|S_{ij}| > r_i$.

A similar line of reasoning leads to the conclusion that given any singly-connected cluster V_i with neighbor V_j and their separator S_{ij} where $|S_{ij}| > r_i$, there is a minimum size cutset $C_w^*(T)$ that contains r_i nodes from S_{ij}: $C_w^*(T) \cap V_i \subset S_{ij}$. Thus, in our search for a minimum w-cutset, we can limit the search for $C_w^*(T) \cap V_i$ to the subsets of S_{ij}. That is the main idea behind the proposed recursive algorithm: remove all those singly-connected clusters V_i that have $r_i \leq |S_{ij}|$, select one of the remaining sinlgy-connected clusters V_i that have $r_i > |S_{ij}|$ enumerate all possible subsets f_i of S_{ij} of size r_i and solve the w-cutset problem for each T' over $X' = X \backslash f_i$. The exact $MinSizeCutset(T, w)$ algorithm is given in Figure 3.3.

The maximum depth of the recursion of procedure $MinSizeCutset(T, w)$ equals the number of clusters in T whose size $> w + 1$. At each recursion iteration our state-space is multiplied by $|F_i| = (|S_{ij}|, r_i)$. Since $|S_{ij}| \leq w^*$, then:

$$|F_i| \leq (w^*, r_i) \quad = \quad \frac{w^*!}{(w^* - r_i)! r_i!} = w^*(w^* - 1)...(w^* - r_i + 1) \leq (w^*)^{\sum_i r_i}$$

The total size of state-space explored is bounded by:

$$\prod_{i=1}^{|V|} (w^*)^{r_i} = (w^*)^{\sum_i r_i}$$

MinSizeCutset(T,w)
1. Remove all singly-connected clusters of size $\leq w+1$.

2. Solve all singly-connected clusters V_i s.t. $r_i \geq S_{ij}$ where S_{ij} is a separator between V_i and its neighbor V_j.

3. Select a singly-connected cluster V_i. Let S_{ij} be a separator between V_i and its neighbor V_j. Let F_i be a set of all possible subsets of size r_i from S_{ij}.
FOR EACH $f_i \in F_i$ DO
$T' = T \backslash f_i$
$\overline{f}_i = MinSizeCutset(T', w)$
END FOR
Let $m = \arg\min_{f_i \in F_i} |\overline{f}_i|$.
Return $C^*_w(T) = f_m \cup \overline{f}_m$.

Figure 3.3: Recursive minimum size w-cutset algorithm.

If we a looking to answer the decision problem *"Does the tree-decomposition T has a w-cutset of size k ?"*, we can stop the recursion when sum of the $r'_i s$ of the processed clusters reaches k. Then, the total number of states explored is bounded by $(w^*)^k$.

3.5.2 Algorithm GWC for minimum cost w-cutset

Next, we show that the problem of finding w-cutset can be mapped to that of finding set multi-cover. The mapping suggests an application of greedy approximation algorithm for set multi-cover problem to find w-cutset of a tree decomposition. When applied to a tree decomposition T=$<V, E>$ over X, it is guaranteed to find a solution within factor $O(1 + \ln m)$ of optimal where m is the maximum # of clusters of size $> (w+1)$ sharing the same node. To avoid loss of generality, we consider the weighted version of each problem.

The mapping is as follows. Given any w-cutset problem of a tree-decomposition

T=<V, E> over X, each cluster node $V_i \in V$ of the tree becomes a node of universal set U. A covering set S_{X_j}=$\{V_i \in V | X_j \in V_i\}$ is created for each node $X_j \in X$. The cost of S_{X_j} equals the cost of X_j. The cover requirement is $r_i = |V_i| - (w+1)$. Covering a node in SMC with a set S_{X_j} corresponds to removing node X_j from each cluster in T. Then, the solution to a set multi-cover is a w-cutset of T. Let C be a solution to the SMC problem. For each $U_i \in U$, the set C contains at least r_i subsets S_{X_j} that contain U_i. Consequently, since $U_i = V_i$, then $|V_i \cap C| \geq r_i$ and $|V_i \backslash C| \leq |V_i| - r_i = |V_i| - |V_i| + (w+1) = w+1$. By definition, C is a w-cutset. An example is shown in Figure 3.4. This duality is important because the properties of SC and SMC problems are well studied and any algorithms previously developed for SMC can be applied to solve w-cutset problem.

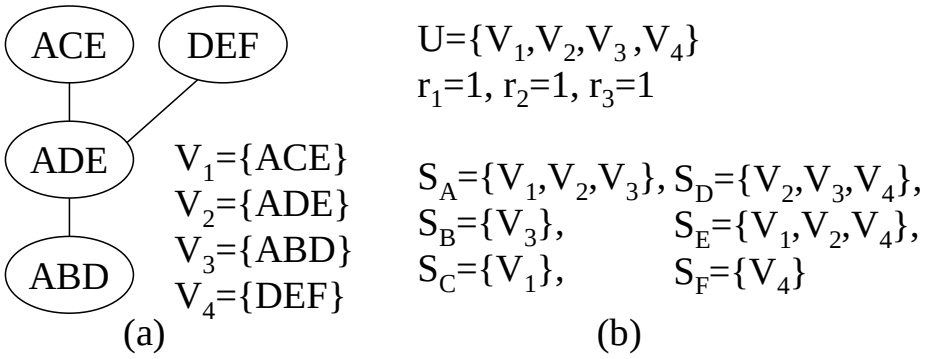


Figure 3.4: (a) A tree decomposition T=<V,E> where V=$\{V_1, ..., V_4\}$ over X=$\{A, B, C, D, E, F\}$; (b) the corresponding set multi-cover problem <U,S> where U=$\{V_1, V_2, V_3, V_4\}$ and S=$\{S_A, S_B, S_C, S_D, S_F\}$; here, set S_{X_i} contains a cluster V_j iff $X_i \in V_j$. The 1-cutset of T is a solution to the set multicover with covering requirements r_1=r_2=r_3=r_4=1: when node $V_i \in V$ is "covered" by set S_{X_i}, node X_i is removed from each cluster.

A well-known polynomial time greedy algorithm exists for weighted SMC [119] that chooses repeatedly set S_i that covers the most "live" (covered less than r_i times) nodes f_i at

the cost c_i: a set that minimizes the ratio c_i/f_i. In the context of w-cutset, f_i is the number of clusters whose size still exceeds $(w+1)$ and c_i is the cost of node X_i. As discussed earlier, c_i maybe defined as the size of the domain of node X_i or its log. When applied to solve the w-cutset problem, we will refer to the algorithm as GWC (Greedy W-Cutset). It is formally defined in Figure 3.5. We define here the approximation algorithm metrics:

DEFINITION **3.5.1 (factor δ approximation)** *An algorithm $\mathcal{A}$ is a* factor δ, $\delta > 0$, *approximation algorithm for minimization problem $\mathcal{P}$ if $\mathcal{A}$ is polynomial and for every instance $I \in D_{\mathcal{P}}$ it produces a solution s such that:* $cost(s) \leq \delta * cost_{OPT}(s), \delta > 1$.

GWC is a factor $O(1+\ln m)$ approximation algorithm [101] where m is the maximum number of clusters sharing the same node (same as the maximum set size in SMC).

Greedy w-Cutset Algorithm (GWC)
Input: A set of clusters $V = \{V_1, ..., V_m\}$ of a tree-decomposition over $X = \{X_1, ..., X_n\}$ where $\forall V_i \in V$, $V_i \subset X$; the cost of each node X_i is c_i.
Output : A set $C \subset X$ s.t. $|V_i \backslash C| \leq w$.
Set $C = \emptyset$, $t = 0$.
While $\exists V_i$ s.t. $|V_i| > w$ **do**
1. $\forall X_i \in X$, compute $f_i = |\{V_j\}|$ s.t. $|V_j| > w$ and $X_i \in V_j$.
2. Find node $X_i \in X$ that minimizes the ratio c_i/f_i.
3. Remove X_i from all clusters: $\forall V_i \in V, V_i = V_i \backslash X_i$.
4. Set $X = X \backslash X_i$, $C = C \cup \{X_i\}$.
End While
Return C

Figure 3.5: Greedy w-cuset Algorithm.

This bound is nearly the best possible for a polynomial algorithm due to strong inapproximability results for the set cover problem, the special case of set multi-cover problem. Approximation guarantee better than $O(\ln m)$ is not possible for any polynomial time algo-

rithm unless P=NP [12, 82]. Furthermore, $\exists C, \Delta_0$ s.t. for all $\Delta \geq \Delta_0$ no polynomial-time algorithm can approximate the optimum within a factor of $\ln \Delta - C \ln \ln \Delta$ unless P=NP [118].

3.6 Experiments

We compare empirically performance of the proposed GWC algorithm (Greedy W-Cutset) and its variants and two greedy heuristic algorithms, MGA (Modified Greedy Algorithm due to [10]) and DGR (Deterministic Greedy Algorithm due to [43]).

The GWC algorithm was implemented as described earlier picking at each iteration a node found in most clusters of size $> w + 1$ with a secondary heuristics (tie breaking) that selects the node contained in most of the clusters. Several variants of GWC with different tie breaking heuristics were tested that were allowed to rebuild a tree decomposition after removing a cutset node:

GWCA - breaks ties by selecting the node found in most of the clusters of the tree decomposition;

GWCM - breaks ties by selecting the node found in most of the clusters of maximum size;

GWCD - breaks ties by selecting the node of highest degree (the degree of the node is computed on the subgraph with all cutset nodes removed and all resulting singly-connected nodes removed). Note that GWC and GWCA only differ in that GWCA rebuilds a cluster-tree after removing a cutset node. Also note that MGA and GWCD have their primary and tie-breaking heuristics switched.

The MGA algorithm is adapted from [10]. It is a factor 2 approximation algorithm for finding minimum cost loop-cutset. MGA iteratively removes all singly-connected nodes from the graph and adds to cutset the node that minimizes cost to degree ratio. The algorithm stops when remaining subgraph is cycle-free. However, it can be easily adapted to finding minimal w-cutset for $w > 1$. For MGA, the only modification required to find w-cutset is to stop when original graph with cutset nodes removed can be decomposed into a cluster tree of width w or less (using min-fill heuristics). In our implementation, MGA algorithm uses the GWC heuristics to break ties: if two nodes have the same degree, the node found in most of the clusters of size $> w$ is added to the cutset.

The DGR algorithm is the Deterministic Greedy Algorithm for finding an elimination order of the variables that yields a tree-decomposition of bounded width defined in [43]. DGR obtains a w-cutset while computing the elimination order of the variables. When eliminating some node X yields a cluster that is too large (size $> w + 1$), the algorithm uses greedy heuristics to pick a cutset node among all the nodes that are not in the ordering yet. Specifically, the deterministic algorithm adds to the cutset a node X that maximizes expression $\sqrt{|N_X|}C_X$, where N_X is a set of neighbours of X that are not eliminated yet and $C_X = \prod_{U_i \in N_X} |\mathcal{D}(U_i)|$. As we ignore domain sizes in this empirical study, we defined $C_X = |N_X|$ in which case DGR adds to cutset a node of maximum degree in the subgraph over nodes that are not eliminated.

3.6.1 Benchmarks

We use Bayesian networks as input reasoning problems. In all experiments, we used a moral graph G of a Bayesian network $\mathcal{B}$ as the input to the minimal w-cutset problem. The tree-decomposition of G was obtained using min-fill algorithm [66].

Our benchmarks are two CPCS networks from UAI repository, cpcs360b with N=360 nodes and induced width w^*=22 and cpcs422b with N=422 nodes and induced width w^*=27, one instance each. Our other benchmarks are layered random networks, meaning that each node is assigned a set of parents selected randomly from previous layer. One set of random networks consisted of 4 layers of $L = 50$ nodes each, total of N=50x4=200 nodes, each node assigned $P = 3$ parents. The second set of random networks consisted of 8 layers of $L = 25$ nodes each, total of N=25x8=200 nodes, each node was assigned $P = 3$ parents. For random networks, the results are averaged over 100 instances.

3.6.2 Results

The results are presented in Table 3.1. For each benchmark, the table provides the five rows of results corresponding to the five algorithms (labelled in the second column). Columns 3-12 are the w-cutset sizes for the w-value. The upper half of the table entries provides results for w in range $[1, 10]$; the lower half of the table provides results for w in range $[11, 20]$. The results for cpcs360b and cpcs422b correspond to a single instance of each network. The result for random networks are averaged over 100 instances. The best entries

Table 3.1: w-cutset. Networks: I=cpcs360b, II=cpcs422b, III=4-layer random networks, L=50, N=200, P=3; IV =8-layer random networks, L=25, N=200, P=3.

	w	1	2	3	4	5	6	7	8	9	10
I	MGA	30	22	20	18	16	15	14	13	12	**10**
w*=20	DGR	36	22	19	18	16	**14**	**13**	**12**	**11**	**10**
	GWC	**27**	**20**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**
	GWCA	**27**	21	18	**16**	**15**	**14**	**13**	**12**	**11**	**10**
	GWCD	**27**	21	18	**16**	**15**	**14**	**13**	**12**	**11**	**10**
II	MGA	80	70	65	60	54	49	44	41	38	36
w*=22	DGR	84	70	63	54	49	43	38	32	27	23
	GWC	**78**	66	58	52	46	41	36	31	26	22
	GWCA	**78**	**65**	**57**	**51**	**45**	**40**	**35**	**30**	**25**	**21**
	GWCD	**78**	**65**	**57**	**51**	**45**	**40**	**35**	**30**	**25**	**21**
III	MGA	87	59	54	52	50	48	47	45	44	43
w*=49	DGR	80	57	52	50	48	46	44	43	42	40
	GWC	78	61	53	49	46	44	43	42	41	39
	GWCA	**74**	**56**	50	**47**	**44**	**42**	**41**	**39**	**38**	**37**
	GWCD	**74**	**56**	**49**	**47**	**44**	**42**	**41**	**39**	**38**	**37**
IV	MGA	99	74	69	66	63	61	59	56	54	51
w*=24	DGR	90	71	65	61	58	55	52	49	47	44
	GWC	93	77	68	63	59	55	52	49	46	43
	GWCA	87	**70**	**62**	**57**	**54**	**51**	**48**	**45**	**42**	**39**
	GWCD	**86**	**70**	**62**	**57**	**54**	**51**	**48**	**45**	**42**	**39**
	w	11	12	13	14	15	16	17	18	19	20
I	MGA	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
w*=20	DGR	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	GWC	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	GWCA	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	GWCD	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
II	MGA	33	30	28	**9**	**8**	7	6	5	4	**2**
w*=22	DGR	21	19	16	**9**	**8**	7	**5**	**4**	**3**	**2**
	GWC	19	16	13	10	**8**	**6**	**5**	**4**	**3**	**2**
	GWCA	**18**	**15**	**12**	**9**	**8**	**6**	**5**	**4**	**3**	**2**
	GWCD	**18**	**15**	**12**	**9**	**8**	**6**	**5**	**4**	**3**	**2**
III	MGA	41	40	39	37	36	35	34	33	31	30
w*=49	DGR	39	38	36	36	34	33	32	31	30	29
	GWC	38	37	36	35	34	33	32	31	30	29
	GWCA	**36**	35	**34**	**33**	**32**	**31**	**30**	**29**	**28**	**27**
	GWCD	**36**	**34**	**34**	**33**	**32**	**31**	**30**	**29**	**28**	**27**
IV	MGA	49	47	44	41	39	36	34	31	28	26
w*=24	DGR	41	38	36	33	31	28	25	23	21	19
	GWC	40	37	35	32	29	27	25	23	20	18
	GWCA	**37**	**34**	**32**	**30**	**27**	**25**	**23**	**21**	**19**	**17**
	GWCD	**37**	35	**32**	**30**	28	**25**	24	**21**	**19**	**17**

for each w are highlighted.

As Table 3.1 shows, it pays to rebuild a tree decomposition: with rare exceptions, GWCA finds a cutset as small as GWC or smaller. On average, GWCA, GWCM, and GWCD computed the same-size w-cutsets. The results for GWCM are omitted since they do not vary sufficiently from the others.

The performance of MGA algorithm appears to depend on the network structure. In case of cpcs360b, it computes the same size w-cutset as GWC variants for $w \geq 10$. However, in cpcs422b, MGA consistently finds larger cutsets except for w=20. On average, as reflected in the results for random networks, MGA finds larger cutset than DGR or any of the GWC-family algorithms. In turn, DGR occasionally finds a smaller cutset compared to GWC, but always a larger cutset compared to GWCA and GWCD, especially for small values of w.

Table 3.2: Function $f(i)$ for i=1...16, GWCA. Networks: I=cpcs360b, II=cpcs422b, III=4-layer random, L=50, N=200, P=3.

	f(i)									
i	1	2	3	4	5	6	7	8	9	10
I	28	23	21	20	20	20	20	20	20	20
II	79	67	60	55	50	46	42	38	34	31
III	75	57	53	51	49	48	48	47	47	47

	f(i)									
i	11	12	13	14	15	16	17	18	19	20
I	20	20	20	20	20	20	20	20	20	20
II	29	27	25	23	23	22	22	22	22	22
III	47	47	47	47	47	47	47	47	47	47

We measured the GWC algorithm approximation parameter M in all of our benchmarks. In cpcs360b and cpcs422b we have $M = 86$ and $M = 87$ yielding approximation factor of $1 + \ln M \approx 5.4$. In random networks, M varied from 29 to 47 yielding approximation factor $\in [4.3, 4.9]$. Thus, if C is the w-cutset obtained by GWC and C_{opt} is the

minimum size w-cutset, then on average:

$$\frac{|C|}{|C_{opt}|} \leq 5$$

3.6.3 Sequence w-cutset Results

Looking at the results as solutions to the sequence w-cutset problems, we can inspect the sequence and suggest good w's by analysing the function $f(i) = |C_i| + i$ as described in section 3.3. To illustrate this we focus on algorithm GWCA for CPC364, CPCS424 and 4-layer random networks (See Table 3.2).

For cpcs360b we observe a small range of values for $f(i)$, namely $f(i) \in \{20, 21, 23, 28\}$. In this case the point of choice is $w = 4$ because $f(1) = 28$, $f(2) = 23$, $f(3) = 21$ while at $i = 4$ we obtain reduction $f(4) = 20$ which stays constant for $i \geq 4$. Therefore, we can have the same time complexity for w-cuset as for exact inference ($w* = 20$) while saving a lot in space, reducing space complexity from exponential in 20 to exponential in 4 only. For w-cutset sampling this implies sampling 20 variables (out of 360) and for each variable doing inference exponential in 4.

The results are even more interesting for cpcs422b where we see a fast decline in time complexity with relatively slow decline in space complexity for the range $i = 1, ..., 11$. The decline is more moderate for $i \geq 11$ but is still cost-effective: for $i = 16$ we get the same time performance as $i = 20$ and therefore $i = 16$ represents a more cost-effective point.

Finally, for the case of 4-layer random networks, on average the function $f(i)$ decreases for $i = 1...8$ and then remains constant. This suggests that if space complexity

allows, the best point of operation is $w = 8$.

3.6.4 Monotonous w-cutset

We ran a second set of experiments where we used the same benchmarks and the same base algorithms. However, we modified the task of finding a single minimum size w-cutset for a fixed w to that of finding a family of w-cutsets for a range of w values. Given a graph G and its tree decomposition T of tree-width tw, each algorithm is first assigned an objective to find a w-cutset for $w = tw - 1$, then, for $w = tw - 2$, and so on. As a result, we obtain a family of cutsets C_{tw-1}, C_{tw-2}, ..., C_1 such that $\forall 1 \leq i < j \leq tw - 1$, $C_i \subset C_j$.

An important observation is that seeking a w-cutset of size $tw - 1$ is equivalent to solving the set cover problem (or its dual, the hitting set problem) since we just need to remove 1 node from each cluster of size $> tw + 1$. This may be important in practice and in fact allows us to find a family of w-cutsets faster because we do not need to start each time from the beginning. The results are shown in Table 3.3 where each algorithm name is prefixed with 'M' to indicate that we are searching for w-cutset *monotonously*. The algorithm DGR is not designed to update the cutset once a tree-decomposition is obtained and therefore, it is omitted in Table 3.3. Compared to Table 3.1, we observed that most of the time the monotonous algorithm finds a slightly larger cutset than its non-monotonous equivalent.

Table 3.3: Monotonous w-cutset. Networks: I=cpcs360b, II=cpcs422b, III=4-layer random, L=50, N=200, P=3; IV =6-layer random, L=25, N=150, P=3.

	w	1	2	3	4	5	6	7	8	9	10
I	MMGA	30	22	20	19	17	16	15	14	14	13
w*=22	MGWC	29	24	21	19	17	16	15	14	13	12
	MGWCA	28	21	20	19	18	17	16	16	15	12
	MGWCD	27	21	20	19	18	16	15	15	14	12
II	MMGA	80	70	65	60	54	49	44	41	38	36
w*=28	MGWC	79	68	61	54	48	43	38	34	29	25
	MGWCA	80	67	59	52	47	43	38	33	28	24
	GWCD	79	67	60	52	47	42	37	33	28	24
III	MMGA	88	59	53	51	49	48	46	45	43	42
w*=49	DGR	80	57	52	49	47	46	44	42	41	40
	MGWC	79	62	54	50	48	46	44	42	41	40
	MGWCA	74	56	50	47	45	43	42	41	40	39
	GWCD	73	56	50	48	46	45	44	43	42	41
IV	MMGA	73	51	47	45	43	41	39	38	36	34
w*=34	MGWC	67	54	48	43	41	38	36	34	31	29
	MGWCA	63	49	43	39	37	35	32	30	29	27
	MGWCD	61	49	43	40	38	35	33	32	30	28
	w	11	12	13	14	15	16	17	18	19	20
I	MMGA	12	12	10	8	7	6	5	4	3	2
w*=22	MGWC	11	10	9	8	7	6	5	4	3	2
	MGWCA	11	10	9	8	7	6	5	4	3	2
	MGWCD	11	10	9	8	7	6	5	4	3	2
II	MMGA	33	30	28	11	10	9	8	7	6	5
w*=28	MGWC	22	19	16	14	13	12	11	10	9	8
	MGWCA	21	19	15	12	10	9	9	8	8	8
	MGWCD	21	18	15	12	10	9	8	8	8	8
III	MMGA	42	40	39	38	37	36	34	33	32	31
w*=48	MGWC	38	37	36	35	34	33	32	31	30	29
	MGWCA	38	37	36	35	34	33	32	31	30	29
	MGWCD	40	39	38	36	35	34	33	32	31	30
IV	MMGA	32	31	29	27	26	24	22	21	19	17
	MGWC	27	25	23	21	19	17	16	14	12	11
	MGWCA	25	23	21	19	17	16	14	13	12	10
	MGWCD	26	24	23	21	20	18	17	15	14	12

3.7 Related Work and Conclusions

In this chapter, we formally defined the minimal w-cutset problem applicable to any reasoning problem with graphical model such as constraint networks and Bayesian networks. The minimum w-cutset problem extends the minimum loop-cutset problem corresponding to $w = 1$. The motivation for finding a minimal w-cutset is to bound the space complexity of the problem (exponential in the width of the graph) while minimizing the required additional processing time (exponential in the width of the graph plus the size of cutset). The loop-cutset problem corresponds to the well-known weighted vertex-feedback set problem and can be approximated within factor 2 of optimal by a polynomial algorithm. We show that the minimal w-cutset problem is harder by reduction from the set multi-cover problem [119]: the set multi-cover problem, and subsequently the w-cutset problem, cannot have a constant-factor polynomial approximation algorithm unless P=NP. Empirically, we show that the minimal loop-cutset heuristics based on the degree of a node is not competitive with the tree-decomposition of the graph.

To our knowledge, only heuristics related to the node elimination order were used before in finding a w-cutset. In [103, 74] and [43], the w-cutset is obtained while computing elimination order of the nodes. The next elimination node is added to the cutset in [103, 74] if its bucket size exceeds the limit. A similar approach was explored in [43] in DGR algorithm (presented in the empirical section) except that the cutset node was chosen heuristically among all the nodes that were not eliminated yet. As the empirical results demonstrate, DGR usually finds smaller cutset than MGA but bigger than GWC/GWCA/GWCD.

It is possible that the performance of DGR could be improved if it was allowed to re-start building a tree-decomposition after adding a variable to the w-cutset.

The research results presented in this chapter have been published in [16].

3.8 Future Work

The main objective of our future work is to find good heuristics for w-cutset problem that are independent from tree-decomposition of a graph since the minimal w-cutset of a tree-decomposition provides only an upper bound on the minimal w-cutset of a graph. So far, we only looked at the degree of the node as possible heuristics and found empirically that GWC heuristics are usually superior. There are also open questions remaining regarding the relationship between w-cutset of a graph and a w-cutset of its tree-decomposition. It is not clear, for example, whether the minimal w-cutset of a graph is a w-cutset of one of its minimum width tree-decompositions.

Chapter 4

Any-Time Bounding Scheme for Belief Updating

This section 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 [56] and bound propagation [76]. Following the principles of cutset conditioning [96], 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. Our empirical evaluation demonstrates the power of our new scheme over a collection of benchmarks. In particular, we show that our new scheme improves on the earlier schemes.

4.1 Introduction

Computing bounds on posterior marginals is a special case of approximating posterior marginals with a desired degree of precision. The latter problem is known to be NP-hard [24]. One way to deal with NP-hard problems is to develop anytime schemes. Such schemes can provide answers anytime and will give more accurate bounds with more time. Hence, we propose here an anytime bounding framework based on two previously proposed

bound computation schemes, bounded conditioning and bound propagation.

Bounded conditioning [56] is founded on the principles of the cutset-conditioning method [96]. Given a Bayesian network over X and a subset of variables $C = \{C_1, ..., C_k\}$ (e.g., a loop-cutset), we can obtain exact posterior marginals for $X_l \in X$ by enumerating over all cutset tuples $c^i \in \mathcal{D}(C)$ using the formula:

$$P(x_l|e) = \frac{\sum_{i=1}^{M} P(x_l, c^i, e)}{\sum_{i=1}^{M} P(c^i, e)} \tag{4.1}$$

The computation of quantities $P(x_l, c^i, e)$ 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, $M = \prod_{i=1}^{k} |\mathcal{D}(C_i)|$ where $\mathcal{D}(C_i)$ is the domain of node $C_i \in C$.

In [56], 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 $P(x_l, c^i, e)$ 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 [56] on the Alarm network with 37 nodes and a loop-cutset of size 5 (M=108). The empirical results demonstrate convergence of the algorithm as h increases and also indicate that the width of the bounds interval for a fixed h increases as more evidence is added.

Bound propagation scheme, proposed recently in [76], 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 on the Alarm network, Ising grid network, and regular bi-partite 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 bounds more accurately the mass of the unexplored tuples in polynomial time. Second, it uses cutset sampling (see Chapter 2) for finding high-probability cutset tuples. Finally, the any-time framework allows to plugin any scheme for bounding joint probabilities over unexplored tuples. In particular, utilizing an improved variant of bound propagation within our any-time framework yields greater accuracy as a function of time than either bounded conditioning or bound propagation alone.

Section 4.2 provides background on the previously proposed methods of bounded conditioning and bound propagation. Section 4.3 defines our ATB framework. In our derivation, we tie the computation of bounds on posterior marginals to bounding a polynomial number of probabilities $P(x_l, c_{1:q}, e)$ and $P(c_{1:q}, e)$ where $c_{1:q} = \{c_1, ..., c_q\}$ is an instantiation of a subset of the cutset variables. Section 4.4 discusses implementation issues concerning bounding of $P(x_l, c_{1:q}, e)$ and $P(c_{1:q}, e)$ with bound propagation. The search

for h high-probability cutset tuples using cutset sampling is given in Section 4.5. We report on our empirical results in Section 4.6 and draw final conclusions in Section 4.7.

4.2 Background

We continue to use the notation defined previously. Namely, we use upper case letters, such as X, to denote subsets of variables. In particular, we will use X to denote a set of all variables in a Bayesian network and C to denote a subset of X. Borrowing terms from graph theory, we will also refer to C as a cutset of the network's graph.

We will use upper case letter with subindex, such as X_l, to denote a single variable. We use lower case letters x and x_l to denote an instantiation of a set of variables or a single variable respectively. We will often denote a specific instantiation of subset of variables C as c^i, using the superindex i.

4.2.1 Bounded Conditioning

Bounded conditioning is an any-time scheme for computing posterior bounds in Bayesian networks [56]. It is derived from the loop-cutset conditioning method (see Eq. (4.1)). Given a Bayesian network over X, evidence $E \subset X$, $E = e$, a loop-cutset $C \subset X \backslash E$, and some node $X_l \in X$, the method computes exactly $P(c^i, e)$ and $P(x_l, c^i, e)$ for h cutset tuples with the highest prior weight $P(c^i)$ and bounds the rest using prior distribution.

Let M denote the total number of cutset tuples. In [56], the authors derive bounds

from the following formula:

$$P(x_l|e) = \sum_{i=1}^{M} P(x_l|c^i, e)P(c^i|e) = \sum_{i=1}^{h} P(x_l|c^i, e)P(c^i|e) + \sum_{i=h+1}^{M} P(x_l|c^i, e)P(c^i|e) \tag{4.2}$$

Deriving Lower Bound

Setting $\forall i > h$, $P(x_l|c^i, e) = 0$ in the Eq. (4.2) yields a lower bound on $P(x_l|e)$:

$$P(x_l|e) \geq \sum_{i=1}^{h} P(x_l|c^i, e)P(c^i|e) \tag{4.3}$$

Since $P(c^i|e)$ can be expressed as:

$$P(c^i|e) = \frac{P(c^i, e)}{\sum_{j=1}^{h} P(c^j, e) + \sum_{j=h+1}^{M} P(c^j, e)} \tag{4.4}$$

we can obtain a lower bound on $P(c^i|e)$ by replacing $P(c^j, e)$, $j > h$, in the denominator of the Eq. 4.4 with the upper bound value $P(c^j)$, yielding:

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

Substituting the right-hand size of Eq. 4.4 for $P(c^i|e)$ in Eq. (4.3) yields a lower bound on $P(x_l|e)$:

$$P^L(x_l|e) \triangleq \sum_{i=1}^{h} P(x_l|c^i, e)\frac{P(c^i, e)}{\sum_{j=1}^{h} P(c^j, e) + \sum_{j=h+1}^{M} P(c^j)} = \frac{\sum_{i=1}^{h} P(x_l|c^i, e)P(c^i, e)}{\sum_{j=1}^{h} P(c^j, e) + \sum_{j=h+1}^{M} P(c^j)} \tag{4.5}$$

Finally, since $P(x_l|c^i, e)P(c^i, e) = P(x_l, c^i, e)$, we get:

$$P^L(x_l|e) = \frac{\sum_{i=1}^{h} P(x_l, c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M} P(c^i)} \tag{4.6}$$

Deriving Upper Bound

Starting again from equation (4.2), an upper bound is obtained by setting $P(x_l|c^i, e) = 1$ for $i > h$ and replacing $P(c^i|e)$ with some upper bound, which we denote as $P^U(c^i|e)$. Namely:

$$P(x_l|e) \leq \sum_{i=1}^{h} P(x_l|c^i, e)P^U(c^i|e) + \sum_{i=h+1}^{M} P^U(c^i|e) \tag{4.7}$$

For $i \leq h$, we can obtain an upper bound $P^U(c^i|e)$ from Eq. (4.4) by dropping the $\sum_{i=h+1}^{M} P(c^i, e)$ from denominator. Substituting the resulting upper bound in Eq. (4.7) yields:

$$P(x_l|e) \leq \sum_{i=1}^{h} P(x_l|c^i, e)\frac{P(c^i, e)}{\sum_{j=1}^{h} P(c^j, e)} + \sum_{i=h+1}^{M} P^U(c^i|e) \tag{4.8}$$

Factoring $P(x_l|c^i, e)$ into the numerator and replacing $P(x_l|c^i, e)P(c^i, e)$ with $P(x_l, c^i, e)$, we transform Eq. (4.8) into:

$$P(x_l|e) \leq \frac{\sum_{i=1}^{h} P(x_l, c^i, e)}{\sum_{i=1}^{h} P(c^i, e)} + \sum_{i=h+1}^{M} P^U(c^i|e) \tag{4.9}$$

An upper bound $P^U(c^i|e)$ for $i > h$ can be obtained through a series of transformations which we detail in Appendix B, yielding the final upper bound on $P(x_l|e)$:

$$P^U(x_l|e) \;=\; \frac{\sum_{i=1}^{h} P(x_l, c^i, e)}{\sum_{i=1}^{h} P(c^i, e)} + \sum_{i=h+1}^{M} P(c^i) + \frac{[\sum_{i=h+1}^{M} P(c^i)]^2}{\sum_{i=1}^{h} P(c^i, e)} \tag{4.10}$$

Note that in the upper bound derivation in [56], the authors separate the h tuples into two groups. The first group contains the tuples for which bounded conditioning computes exactly both $P(c^i, e)$ and $P(x_l|c^i, e)$. The second group contains tuples for which bounded conditioning only computes exactly $P(c^i, e)$ and uses $P^L(x_l|c^i, e) = 0$ in the lower bound

formulation, and $P^U(x_l|c^i, e) = 1$ in the upper bound formulation. Let m, $m < h$, denote the number of tuples for which the algorithm computes $P(x_l|c^i, e)$ exactly. Then, the lower and upper bounds derived in Eq. (4.6) and (4.10) respectively become:

$$P^{L'}(x_l|e) = \frac{\sum_{i=1}^{m} P(x_l, c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M} P(c^i)} \tag{4.11}$$

$$P^{U'}(x_l|e) = \frac{\sum_{i=1}^{m} P(x_l, c^i, e)}{\sum_{i=1}^{h} P(c^i, e)} + \frac{\sum_{i=m+1}^{h} P(c^i, e)}{\sum_{i=1}^{h} P(c^i, e)} + \sum_{i=h+1}^{M} P(c^i) + \frac{[\sum_{i=h+1}^{M} P(c^i)]^2}{\sum_{i=1}^{h} P(c^i, e)} \tag{4.12}$$

It is clear that $P^L(x_l|e) \geq P^{L'}(x_l|e)$ and $P^U(x_l|e) \leq P^{U'}(x_l|e)$. In the future, we use $P^L(x_l|e)$ in Eq. (4.6) and $P^U(x_l|e)$ in Eq. (4.10) as the basis for comparison with our any-time bounds.

Clearly, the bounds expressed in Eq. (4.6) and (4.10) converge to the exact posterior marginals as $h \rightarrow M$. The convergence rate depends on the form of the distribution $P(C|e)$. The scheme was validated in [56] on the example of the Alarm network with 37 nodes. Its loop-cutset contains 5 nodes and the 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 evidence was added, processing the same 40 cutset tuples, the bounding intervals length increased. Specifically, with 3 and 4 evidence variables, the bounding interval length rose to ≈ 0.15. Hence, while the empirical results demonstrated the convergence of bounded conditioning, they also showed the deterioration in the scheme as more nodes are observed.

Note also that the upper bound in Eq. (4.10) can become greater than 1. Dropping

the first two addends from Eq. (4.10), we obtain:

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

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

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

This shows that $P^U(x_l|e)$, as defined in [56], can become arbitrarily large when $P(e)$ is small compared to $P(c^i)$.

4.2.2 Bound Propagation

Bound propagation (BdP) [76] 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 the posterior marginal $P(x_i|e)$. Let Y denote Markov blanket of node X_i $ma_i = Y = \{Y_1, ..., Y_k\}$. The idea is to compute posterior marginals via:

$$P(x_i|e) = \sum_{y_1,...,y_k} P(x_i|y_1, ..., y_k)P(y_1, ..., y_k|e) \tag{4.13}$$

where $P(x_i|y_1, ..., y_k)$ is an entry in the probability table of X_i 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:

$$\sum_{y_1,...,y_k} P(y_1, ..., y_k|e) = 1 \tag{4.14}$$

Further, $\forall y_j \in \mathcal{D}(Y_j)$:

$$\sum_{Y \setminus Y_j, Y_j = y_j} P(y_1, ..., y_k|e) = P(y_j|e)$$

Denoting arbitrary lower and upper bounds on $P(y_j|e)$ by $P^L(y_j|e)$ and $P^U(y_j|e)$ respectively, we can write:

$$P^L(y_j|e) \leq \sum_{Y \setminus Y_j, Y_j = y_j} P(y_1, ..., y_k|e) \leq P^U(y_j|e) \tag{4.15}$$

Hence, for each variable X_i, we have a linear optimization problem with the objective function $P(x_i|e)$, defined in Eq. (4.13), that is minimized or maximized with respect to all variables $P(y_1, ..., y_k|e)$. For each instantiation of the 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 a coefficient of the objective function. Therefore, the number of variables is exponential in the size of the Markov blanket. The constraints are defined in Eq. (4.14) (sum-to-1 constraint) and (4.15). For each variable Y_j in the Markov blanket of X_i, there will be $|\mathcal{D}(Y_j)|$ constraints of type Eq. (4.15). The total number of constraints equals $1 + \sum_j |\mathcal{D}(Y_j)|$.

Example 4.2.1 *Let X_i be bi-valued. Let $ma_i = \{A, B\}$. Let $\mathcal{D}(A) = \{0, 1\}$ and $\mathcal{D}(B) = \{0, 1, 2\}$. Let $P(X_i|A, B)$ be defined as follows:*

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

Denoting $P_{mn} = P(a = m, b = n|e)$, the objective function of the linear optimization problem can be defined as follows:

$$P(x_i|e) = 0.1P_{00} + 0.2P_{01} + 0.3P_{02} + 0.4P_{10} + 0.5P_{11} + 0.6P_{12}$$

s.t.

$$P_{00} + P_{01} + P_{02} + P_{10} + P_{11} + P_{12} = 1$$
$$P^L(a = 0|e) \leq P_{00} + P_{01} + P_{02} \leq P^U(a = 0|e)$$
$$P^L(a = 1|e) \leq P_{10} + P_{11} + P_{12} \leq P^U(a = 1|e)$$
$$P^L(b = 0|e) \leq P_{00} + P_{10} \leq P^U(b = 0|e)$$
$$P^L(b = 1|e) \leq P_{01} + P_{11} \leq P^U(b = 1|e)$$
$$P^L(b = 2|e) \leq P_{02} + P_{12} \leq P^U(b = 2|e)$$

First, for $\forall X_i \in X \backslash E$ the algorithm initializes $P^L(x_i|e)$ and $P^U(x_i|e)$ to 0 and 1. Then, processing variables one by one in some order, BdP solves linear minimization and maximization problems for each variable and updates the quantities P^L and P^U. This 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.

If we know *a priori* some lower and upper bounds $P^L(x_i|e)$ and $P^U(x_i|e)$ for some variable X_i, we can use those values in the initialization step of BdP. However, in this case, it is no longer guaranteed that the minimum and maximum of the linear optimization problem will be as good or better than the current bound values. Hence, we have to check for that before performing an updating step. We incorporate this generalization in the outline of the bound propagation algorithm in Figure 4.1.

The inputs to the algorithm are a Bayesian network 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_i. 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 $X_i \in X \backslash E$. Inside the repeat loop we define a single iteration of bound

propagation algorithm. In each iteration, $\forall X_i \in X \backslash E$ the algorithm computes a conditional probability table $P(X_i|ma_i)$ (step 1) and then solves the linear optimization problem for each value $x_i \in \mathcal{D}(X_i)$ (steps 2 and 3). 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 (step 4).

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:
For every $X_i \in X \backslash E$ do:
For every $x_i \in \mathcal{D}(X_i)$ do:

1. Compute conditional probability table over $ma_i = \mathbf{Y} = \{Y_1, ..., Y_k\}$:

$$P(x_i|y_1, ..., y_k) \leftarrow \alpha P(x_i|pa_i) \prod_j P(ch_j|pa_j),\ \forall y \in \mathcal{D}(\mathbf{Y})$$

2. Define constraints of a linear optimization problem:

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

$$P^L(y_j|e) \leq \sum_{Y \backslash Y_j, Y_j = y_j} P(y_1, ..., y_k|e) \leq P^U(y_j|e),\ \forall Y_j \in \mathbf{Y},\ \forall y_j \in \mathcal{D}(Y_j)$$

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

$$m_i \leftarrow \min \sum_{y_1,...,y_k} P(x_i|y_1, ..., y_k, e) P(y_1, ..., y_k|e) \quad (4.16)$$

$$M_i \leftarrow \max \sum_{y_1,...,y_k} P(x_i|y_1, ..., y_k, e) P(y_1, ..., y_k|e) \quad (4.17)$$

4. Update Bounds
if $m_i > P^L(x_i|e)$ then $P^L(x_i|e) \leftarrow m_i$
if $M_i < P^U(x_i|e)$ then $P^U(x_i|e) \leftarrow M_i$

Until Converged

Figure 4.1: Bound Propagation (BdP) Algorithm

Each iteration of the algorithm requires solving $|\mathcal{D}(X_i)|$ problems for each variable

$X_i \in X \backslash E$ (one for each value of X_i). The total number of linear problems per iteration is $O(N \cdot d)$ where $N = |X|$ and d is the maximum variable domain size. The maximum problem size is d^k variables, where k is the maximum number of variables in a Markov blanket of variable X_i, and $d \cdot k + 1$ constraints.

The paper [76] showed that BdP performed quite well on Ising grid and regular two-layer networks. The algorithm was also tested with the Alarm network without evidence. Notably, in a network without evidence, the marginal probabilities of root nodes equal their priors (which are given). In the case of Alarm network, BdP obtained small bounds interval for several variables but did not obtain good bounds for root nodes 11, 12, 13, and 14. This shows that BdP exploits local network structure but ignores the global network properties. In general, it is not guaranteed to compute good bounds even in a singly-connected network.

In practice, algorithm BdP as presented in [76] is feasible only for networks having bounded Markov blanket size, such as grid networks or regular random networks, since the number of variables in the optimization problem in Figure 4.1 grows exponentially with the size of the Markov blanket.

4.3 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 h most probable cutset tuples (with high probabilities $P(c|e)$), the probabilities of the h tuples are

evaluated exactly and the rest are upper and lower bounded.

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

The algorithm computes exactly the quantities $P(x_l, c^i, e)$, $X_l \in X \backslash E$, and $P(c^i, e)$ for $1 \leq i \leq h$ and bounds the sums $\sum_{i=h+1}^{M} P(x_l, c^i, e)$ 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 constant h, it replaces the sums over the tuples c^{h+1},...,c^M with a sum over a polynomial number (in h) of partially-instantiated cutset tuples. Details are provided in Section 4.3.1.

Second, in Section 4.3.2, we develop new expressions for lower and upper bounds on posterior marginals as a function of the lower and upper bounds on the joint probabilities $P(x_l, c_{1:q}, e)$ and $P(c_{1:q}, e)$. We assume in our derivation that there is an algorithm $\mathcal{A}$ that can compute those bounds.

We defer the problem of selecting high probability cutset tuples to Section 4.4 and the bounding scheme for $P(x_l, c_{1:q}, e)$ and $P(c_{1:q}, e)$ to Section 4.4.1.

4.3.1 Bounding the Number of Processed Tuples

We obtain the any-time bounding scheme starting with the cutset conditioning formula, similar to the way bounded conditioning was developed. Given a Bayesian network over a set of variables X, evidence $E \subset X$, $E = e$, a cutset $C \subset X \backslash E$, 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 the generated cutset tuples, $0 < h < M$. We can assume without loss of generality that we generated first h tuples in some enumeration of all tuples. Then, for a node X_l with $x'_l \in \mathcal{D}(X_l)$, we can re-write Eq. (4.1) separating the summation over the generated tuples 1 through h and the rest as:

$$P(x'_l|e) = \frac{\sum_{i=1}^{h} P(x'_l, c^i, e) + \sum_{i=h+1}^{M} P(x'_l, c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M} P(c^i, e)} \tag{4.18}$$

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

Consider a fully-expanded search tree of depth $|C|$ over the cutset search space expanded in the order C_1,...,C_k, $k = |C|$. A path from the root to the leaf at depth $|C|$ corresponds to a full cutset tuple. Assume that we mark all the tree edges on paths that correspond to the first h generated cutset tuples. Then the unexpanded tuples c^i, $i > h$, correspond to the unmarked leaves. We can obtain a truncated search tree by trimming unmarked leaves as follows:

DEFINITION **4.3.1 (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 $x^j = \{x_1^j, ..., x_n^j\}$, is 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.

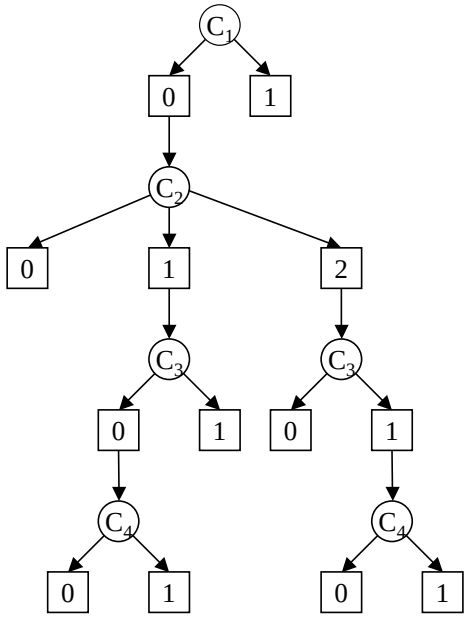


Figure 4.2: A search tree for cutset $C = \{C_1, ..., C_4\}$.

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, $q < |C|$, is a tuple $c_{1:q} = \{c_1, ..., c_q\}$ over a subset of the cutset variables. The full cutset is denoted c or $c_{1:k}$, where $k = |C|$.

An example of a truncated search tree is shown in Figure 4.2. Given a cutset C=$\{C_1, ..., C_4\}$ of size 4, $\mathcal{D}(C_1)$=$\{0, 1\}$, $\mathcal{D}(C_2)$=$\{0, 1, 2\}$, $\mathcal{D}(C_3)$=$\{0, 1\}$, $\mathcal{D}(C_4)$=$\{0, 1\}$, and four fully-instantiated tuples $\{0, 1, 0, 0\}$, $\{0, 1, 0, 1\}$, $\{0, 2, 1, 0\}$, $\{0, 2, 1, 1\}$, the remaining partially instantiated tuples are $\{0, 0\}$, $\{0, 1, 1\}$, $\{0, 2, 0\}$, and $\{1\}$.

It is easy to see that the number M' of truncated tuples 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_k can have no more than $(d-1)$ emanating leaves.

PROPOSITION **4.3.1** *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 can enumerate the partially instantiated tuples, denoting the j-th tuple $c^j_{1:q_j}$, $1 \leq j \leq M'$, where q_j is the tuple's length. Clearly, the probability mass over the cutset tuples c^{h+1}, ..., c^M can be captured via a sum over the truncated tuples. Namely:

PROPOSITION **4.3.2**

$$\sum_{i=h+1}^{M} P(c^i, e) = \sum_{j=1}^{M'} P(c^j_{1:q_j}, e) \tag{4.19}$$

$$\sum_{i=h+1}^{M} P(x'_l, c^i, e) = \sum_{j=1}^{M'} P(x'_l, c^j_{1:q_j}, e) \tag{4.20}$$

Therefore, we can bound the sums over tuples $h+1$ through M in Eq. (4.18) by bounding a polynomial number of partially-instantiated tuples.

4.3.2 Bounding the Probability over the Truncated Tuples

Now we will derive the expressions for bounding the posterior marginals. Replacing the summation over tuples $h+1$ through M with summation over the partially-instantiated tuples 1 through M' in Eq. (4.18), we get:

$$P(x'_l|e) = \frac{\sum_{i=1}^{h} P(x'_l, c^i, e) + \sum_{j=1}^{M'} P(x'_l, c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P(c^j_{1:q_j}, e)} \tag{4.21}$$

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 $P^L(x_l, c^i_{1:q_i}, e)$ and $P^U(x_l, c^i_{1:q_i}, e)$

for any value $x_l \in \mathcal{D}(X_l)$ of any unobserved variable X_l s.t. $P_A^L(c_{1:q}, e) \leq P(c_{1:q}, e) \leq P_A^U(c_{1:q}, e)$ and $P_A^L(x_l, c_{1:q}, e) \leq P(x_l, c_{1:q}, e) \leq P_A^U(x_l, c_{1:q}, e)$. In the future derivations, we sometimes drop the algorithm's name.

Deriving Lower Bounds

A brute force lower bound expression of Eq. (4.21) can be obtained by replacing each $P(x'_l, c^j_{1:q_j}, e)$ with its lower bound (reducing numerator) and each $P(c^j_{1:q_j}, e)$ with its upper bound (increasing denominator) yielding expression P^{L_1}:

$$P(x'_l|e) \geq \frac{\sum_{i=1}^{h} P(x'_l, c^i, e) + \sum_{j=1}^{M'} P_{\mathcal{A}}^L(x'_l, c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P_{\mathcal{A}}^U(c^j_{1:q_j}, e)} \triangleq P_{\mathcal{A}}^{L_1}(x'_l|e) \tag{4.22}$$

However, a tighter bound can be obtained if we apply additional transformations to Eq. (4.21) and prove a helpful lemma. First, we decompose $P(c^j_{1:q_j}, e)$, $0 \leq j \leq M'$, as follows:

$$P(c^j_{1:q_j}, e) = \sum_{x_l} P(x_l, c^j_{1:q_j}, e) = P(x'_l, c^j_{1:q_j}, e) + \sum_{x_l \neq x'_l} P(x_l, c^j_{1:q_j}, e) \tag{4.23}$$

Replacing $P(c^j_{1:q_j}, e)$ in Eq. (4.21) with the right-hand size expression in Eq. (4.23), we get:

$$P(x'_l|e) = \frac{\sum_{i=1}^{h} P(x'_l, c^i, e) + \sum_{j=1}^{M'} P(x'_l, c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P(x'_l, c^j_{1:q_j}, e) + \sum_{x_l \neq x'_l} \sum_{j=1}^{M'} P(x_l, c^j_{1:q_j}, e)} \tag{4.24}$$

We will use the following two lemmas:

LEMMA **4.3.1** *Given positive numbers $a > 0$, $b > 0$, $\delta \geq 0$, if $a < b$, then: $\frac{a}{b} \leq \frac{a+\delta}{b+\delta}$.* ■

LEMMA **4.3.2** *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 4.3.2 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. (4.24) the sums in both numerator and denominator contain $P(x'_l, c^j_{1:q_j}, e)$. Hence, we can apply Lemma 4.3.2. We will obtain a lower bound by replacing $P(x'_l, c^j_{1:q_j}, e)$, $1 \leq j \leq M'$, in Eq. (4.24) with corresponding lower bounds in both numerator and denominator, yielding:

$$P(x'_l|e) \geq \frac{\sum_{i=1}^{h} P(x'_l, c^i, e) + \sum_{j=1}^{M'} P^L_{\mathcal{A}}(x'_l, c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^L_{\mathcal{A}}(x'_l, c^j_{1:q_j}, e) + \sum_{x_l \neq x'_l} \sum_{j=1}^{M'} P(x_l, c^j_{1:q_j}, e)} \tag{4.25}$$

Subsequently, we replace $P(x_l, c^j_{1:q_j}, e)$, $x_l \neq x'_l$, with its upper bound, yielding:

$$P(x'_l|e) \geq \frac{\sum_{i=1}^{h} P(x'_l, c^i, e) + \sum_{j=1}^{M'} P^L_{\mathcal{A}}(x'_l, c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^L_{\mathcal{A}}(x'_l, c^j_{1:q_j}, e) + \sum_{x_l \neq x'_l} \sum_{j=1}^{M'} P^U_{\mathcal{A}}(x_l, c^j_{1:q_j}, e)} \triangleq P^{L_2}_{\mathcal{A}}(x'_l|e) \tag{4.26}$$

Hence, we have obtained two expressions for lower bound on $P(x'_l|e)$, P^{L_1} defined in Eq. (4.22) and P^{L_2} defined in Eq. (4.26). Both schemes are defined as a function of upper

and lower bounds derived by an algorithm $\mathcal{A}$. Neither bound dominates the other. In Section 4.3.3, we will define conditions under which P^{L_2} is tighter than P^{L_1}. In particular, we will show that P^{L_2} is always better than P^{L_1} if $|\mathcal{D}(X_l)| = 2$.

Deriving the Upper Bound Expression

The upper bound formulation can be obtained in a similar manner. Since both numerator and denominator in Eq. (4.24) contain addends $P(x'_l, c^j_{1:q_j}, e)$, using Lemma 4.3.2 we replace each $P(x'_l, c^j_{1:q_j}, e)$ with an upper bound $P^U_A(x'_l, c^j_{1:q_j}, e)$ yielding:

$$P(x'_l|e) \leq \frac{\sum_{i=1}^{h} P(x'_l, c^i, e) + \sum_{j=1}^{M'} P^U_{\mathcal{A}}(x'_l, c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^U_{\mathcal{A}}(x'_l, c^j_{1:q_j}, e) + \sum_{x_l \neq x'_l} \sum_{j=1}^{M'} P(x_l, c^j_{1:q_j}, e)} \tag{4.27}$$

Subsequently, replacing $P(x_l, c^j_{1:q_j}, e)$, $x \neq x'$, with a lower bound (reducing denominator), we obtain a new upper bound expression, P^{U_1}, on $P(x'_l|e)$:

$$P(x'_l|e) \leq \frac{\sum_{i=1}^{h} P(x'_l, c^i, e) + \sum_{j=1}^{M'} P^U_{\mathcal{A}}(x'_l, c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^U_{\mathcal{A}}(x'_l, c^j_{1:q_j}, e) + \sum_{x_l \neq x'_l} \sum_{j=1}^{M'} P^L_{\mathcal{A}}(x_l, c^j_{1:q_j}, e)} \triangleq P^{U_1}_{\mathcal{A}}(x'_l|e) \tag{4.28}$$

The derived bounds, P^{L_1}, P^{L_2}, and P^{U_1}, are never worse than those obtained by bounded conditioning, as we will show in Section 4.3.3. In particular, unlike the derivation in bounded conditioning, the upper bound above is guaranteed to be ≤ 1 for any lower and upper bounds on $P(x_l, c^j_{1:q_j}, e)$, even if we plugin the trivial 0 and 1 bounds.

Deriving Bounds for Cutset Nodes

The main difference in the formulation of the bounds for a cutset node C_l is that only a subset of the h cutset tuples will have $C_l = c'_l$. Therefore, the number of partially-instantiated tuples for different values of C_l may differ. Thus, we use Dirac δ-function in the denominator to indicate that summation is only over those cutset tuples where $C_l = c'_l$. i.e., $\delta(c^i, c'_l) = 1$ iff the value of variable C_l in a tuple c^i equals c'_l. We use subindex c_l in M'_{c_l} to denote the number of partially-instantiated tuples where $C_l = c_l$. We provide the detailed derivation in Appendix C. Here, we only summarize the results. For any $C_l \in C$, the following is an upper bound expression for $P(c'_l|e)$:

$$P_{\mathcal{A}}^{U_1}(c'_l|e) = \frac{\sum_{i=1}^{h} \delta(c^i, c'_l) P(c^i, e) + \sum_{j=1}^{M'_{c'_l}} P_{\mathcal{A}}^{U}(c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'_{c'_l}} P_{\mathcal{A}}^{U}(c^j_{1:q_j}, e) + \sum_{c_l \neq c'_l} \sum_{j=1}^{M'_{c_l}} P_{\mathcal{A}}^{L}(c^j_{1:q_j}, e)} \tag{4.29}$$

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

$$P_{\mathcal{A}}^{L_1}(c'_l|e) = \frac{\sum_{i=1}^{h} \delta(c^i, c'_l) P(c^i, e) + \sum_{j=1}^{M'_{c'_l}} P_{\mathcal{A}}^{L}(c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P_{\mathcal{A}}^{U}(c^j_{1:q_j}, e)} \tag{4.30}$$

$$P_{\mathcal{A}}^{L_2}(c'_l|e) = \frac{\sum_{i=1}^{h} \delta(c^i, c'_l) P(c^i, e) + \sum_{j=1}^{M'_{c'_l}} P_{\mathcal{A}}^{L}(c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'_{c'_l}} P_{\mathcal{A}}^{L}(c^j_{1:q_j}, e) + \sum_{c_l \neq c'_l} \sum_{j=1}^{M'_{c_l}} P_{\mathcal{A}}^{U}(c^j_{1:q_j}, e)} \tag{4.31}$$

Any-Time Bounds Architecture

Input: A belief network $(\mathcal{B})$, variables X, evidence $E \subset X$, cutset $C \subset X \backslash E$, constant h, algorithm $\mathcal{A}$ for computing lower and upper bounds.

Output: lower bounds P^L, upper bounds P^U.

1. Generate h cutset tuples.
2. Compute:

$$S \leftarrow \sum_{i=1}^{h} P(c^i, e)$$
$$S_l \leftarrow \sum_{i=1}^{h} P(x_l, c^i, e), \quad \forall x_l \in \mathcal{D}(X_l), \quad \forall X_l \in X \backslash (C, E)$$

3. Traverse partially-instantiated tuples:

3.1 Generate the truncated tree associated with the h tuples and let $c^1_{1:q_1}, ..., c^{M'}_{1:q_{M'}}$ be the M' partial assignments.

3.2 Compute:

$$S' \leftarrow \sum_{j=1}^{M'} P^U_{\mathcal{A}}(c^j_{1:q_j}, e)$$
$$LB_{\mathcal{A}}(x_l) \leftarrow \sum_{j=1}^{M'} P^L_{\mathcal{A}}(x_l, c^j_{1:q_j}, e), \quad \forall x_l \in \mathcal{D}(X_l), \quad \forall X_l \in X \backslash (C, E)$$
$$UB_{\mathcal{A}}(x_l) \leftarrow \sum_{j=1}^{M'} P^U_{\mathcal{A}}(x_l, c^j_{1:q_j}, e), \quad \forall x_l \in \mathcal{D}(X_l), \quad \forall X_l \in X \backslash (C, E)$$

4. Compute bounds:

$$P^L(x'_l|e) = \max \left\{ \begin{array}{c} \frac{S_l + LB_{\mathcal{A}}(x_l)}{S + S'} \\ \frac{S_l + LB_{\mathcal{A}}(x'_l)}{S + LB_{\mathcal{A}}(x'_l) + \sum_{x_l \neq x'_l} UB_{\mathcal{A}}(x_l)} \end{array} \right. \tag{4.32}$$

$$P^U(x'_l|e) = \frac{S_l + UB_{\mathcal{A}}(x'_l)}{S + UB_{\mathcal{A}}(x'_l) + \sum_{x_l \neq x'_l} LB_{\mathcal{A}}(x_l)} \tag{4.33}$$

Figure 4.3: Any-Time Bounds Architecture

Properties of ATB

The any-time bounding scheme is summarized in Figure 4.3. In steps 1 and 2, we generate h fully-instantiated cutset tuples and compute exactly probabilities $P(c^i, e)$ and $P(X_l, c^i, e)$ for $i \leq h$, $\forall X_l \in X \backslash (C, E)$. In step 3, we compute bounds on partially instantiated tuples using algorithm $\mathcal{A}$. In step 4, we compute the bounds on the posterior marginals. Given the lower and upper bounds computed using algorithm $\mathcal{A}$, the upper bound is computed using expression (4.33). The lower bound is defined by the maximum of the two lower bound expressions in Eq. (4.32).

Example 4.3.1 *Consider again a Bayesian network $\mathcal{B}$ from previous example. Recall that it has a cutset $C = \{C_1, ..., C_4\}$ with domains $\mathcal{D}(C_1) = \mathcal{D}(C_3) = \mathcal{D}(C_4) = \{0, 1\}$ and $\mathcal{D}(C_2) = \{0, 1, 2\}$. The total number of cutset tuples is $M = 24$. Let X_l be a variable in $\mathcal{B}$. We will compute bounds on $P(x'_l|e)$. Assume we generated the same four cutset tuples ($h = 4$) as before:*

$$
\begin{aligned}
c^1 &= \{0, 1, 0, 0\}, \\
c^2 &= \{0, 1, 0, 1\}, \\
c^3 &= \{0, 2, 1, 0\}, \\
c^4 &= \{0, 2, 1, 1\}
\end{aligned}
$$

The corresponding truncated search tree is shown in Figure 4.2. For the tuple $\{0, 1, 0, 0\}$, we compute exactly the probabilities $P(x'_l, 0, 1, 0, 0, e)$ and $P(0, 1, 0, 0)$. Similarly, we obtain exact probabilities $P(x'_l, 0, 1, 0, 1)$ and $P(0, 1, 0, 1)$ for the second cutset instance $\{0, 1, 0, 1\}$. Since $h = 4$, $\sum_{i=1}^{h} P(x'_l, c^i, e)$ and $\sum_{i=1}^{h} P(c^i, e)$ are:

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

The remaining partial tuples are: $c^1_{1:2} = \{0, 0\}$, $c^2_{1:3} = \{0, 1, 1\}$, $c^3_{1:3} = \{0, 2, 0\}$, and $c^4_{1:1} = \{1\}$. Since these 4 tuples are not full cutsets, we compute bounds on their joint probabilities.

Using the same notation as in Figure 4.3, the sums over the partially instantiated tuples will have the form:

$$UB_A(x_l) \triangleq P_A^U(x_l, c_{1:2}^1, e) + P_A^U(x_l, c_{1:3}^2, e) + P_A^U(x_l, c_{1:3}^3, e) + P_A^U(x_l, c_{1:1}^4, e)$$

$$LB_A(x_l) \triangleq P_A^L(x_l, c_{1:2}^1, e) + P_A^L(x_l, c_{1:3}^2, e) + P_A^L(x_l, c_{1:3}^3, e) + P_A^L(x_l, c_{1:1}^4, e)$$

From Eq. (4.28) we get:

$$P^{U_1}(x_l'|e) = \frac{\sum_{i=1}^4 P(x_l', c^i, e) + UB_A(x_l')}{\sum_{i=1}^4 P(c^i, e) + UB_A(x_l') + \sum_{x_l \neq x_l'} LB_A(x_l)}$$

From Eq. (4.22 and (4.26) we get:

$$P^{L_1}(x_l'|e) = \frac{\sum_{i=1}^4 P(x_l, c^i, e) + LB_A(x_l')}{\sum_{i=1}^4 P(c^i, e) + \sum_{i=1}^4 P_A^U(c_{1:q_j}^j, e)}$$

$$P^{L_2}(x_l'|e) = \frac{\sum_{i=1}^4 P(x_l, c^i, e) + LB_A(x_l')}{\sum_{i=1}^4 P(c^i, e) + LB_A(x_l') + \sum_{x_l \neq x_l'} UB_A(x_l)}$$

The total number of tuples processed is $M' = 4 + 4 = 8 << 24$.

If C is a loop-cutset, then computing the exact probabilities $P(c^i, e)$ and $P(x_l, c^i, e)$ takes $O(N)$, where N is the size of the input, and the total complexity of computing the exact probabilities for h tuples is $O(N \cdot h)$. We can conclude:

THEOREM **4.3.2 (Complexity as a function of loop-cutset)** *Given an algorithm $\mathcal{A}$ that computes lower and upper bounds on the joint probabilities $P(c_{1:q_i}, e)$ and $P(x_l, c_{1:q_i}, e)$ in time $O(T)$, if C is a loop-cutset then ATB expressions (4.22), (4.26), and (4.28) can be computed in $O(h \cdot N + T \cdot h \cdot (d-1) \cdot |C|)$ where d is the maximum domain size and N is the problem input size.*

Theorem 4.3.2 follows immediately from Proposition 4.3.1. It shows that since the number of partially observed cutset tuples grows polynomially with h, the ATB scheme is polynomial if the plug-in bounding algorithm $\mathcal{A}$ is polynomial.

4.3.3 Comparison of Bounding Schemes

Next, we compare the lower bound expressions (4.22) and (4.26) and provide conditions under which the lower bound P^{L_2} is tighter than P^{L_1}.

Comparing Expressions for Lower Bounds

We will show that under certain conditions the lower bound $P_{\mathcal{A}}^{L_2}$ value given in Eq. (4.26) is larger than the value of the brute force lower bound $P_{\mathcal{A}}^{L_1}$ given in Eq. (4.22). In particular, $P_{\mathcal{A}}^{L_2}$ is guaranteed to dominate $P_{\mathcal{A}}^{L_1}$ when node X_l has domain of size 2.

THEOREM **4.3.3 (Lower Bound Dominance1)** *Assume an algorithm $\mathcal{A}$ computes bounds $P^L(c^j_{1:q_j}, e)$ and $P^U(c^j_{1:q_j}, e)$ on $P(c^j_{1:q_j}, e)$ and bounds $P^L(x_l|c^j_{1:q_j}, e)$ and $P^U(x_l|c^j_{1:q_j}, e)$ on $P(x_l|c^j_{1:q_j}, e)$ for $1 \leq j \leq M'$ and $\forall x_l \in \mathcal{D}(X_l)$. Let:*

$$P^L(x_l, c^j_{1:q_j}, e) = P^L(x_l|c^j_{1:q_j}, e)P^L(c^j_{1:q_j}, e)$$

$$P^U(x_l, c^j_{1:q_j}, e) = P^U(x_l|c^j_{1:q_j}, e)P^U(c^j_{1:q_j}, e)$$

If

$$P^L(x'_l|c^j_{1:q_j}, e) \leq 1 - \sum_{x_l \neq x'_l} P^U(x_l|c^j_{1:q_i}, e) \tag{4.34}$$

then

$$P^{L_1}(x'_l|e) \leq P^{L_2}(x'_l|e)$$

where $P^{L_1}(x'_l|e)$ and $P^{L_2}(x'_l|e)$ are defined in Eq. (4.22) and Eq. (4.26) respectively.

The proof is provided in Appendix D.

COROLLARY **4.3.1 (Lower Bound Dominance2)** *Assume an algorithm $\mathcal{A}$ computes bounds $P^L(c^j_{1:q_j}, e)$ and $P^U(c^j_{1:q_j}, e)$ on $P(c^j_{1:q_j}, e)$ and bounds $P^L(x_l|c^j_{1:q_j}, e)$ and $P^U(x_l|c^j_{1:q_j}, e)$ on $P(x_l|c^j_{1:q_j}, e)$ for $1 \leq j \leq M'$ and $\forall x_l \in \mathcal{D}(X_l)$. Let:*

$$P^L(x_l, c^j_{1:q_j}, e) = P^L(x_l|c^j_{1:q_j}, e)P^L(c^j_{1:q_j}, e)$$

$$P^U(x_l, c^j_{1:q_j}, e) = P^U(x_l|c^j_{1:q_j}, e)P^U(c^j_{1:q_j}, e)$$

If $\mathcal{D}(X_l) = 2$ *then* $P^{L_1}(x'_l|e) \leq P^{L_2}(x'_l|e)$ *where* $P^{L_1}(x'_l|e)$ *and* $P^{L_2}(x'_l|e)$ *are defined in Eq. (4.22) and (4.26) respectively.*

The corollary follows from observation that the condition expressed in Eq. (4.34) in Theorem 4.3.3 can be always enforced in nodes with domains of size 2. Namely, for any $X_l \in X$ with domain $\mathcal{D}(X_l) = \{x'_l, x''_l\}$, if $P^L(x'_l|c^i_{1:q_i}, e) > 1 - P^U(x''_l|c^i_{1:q_i}, e)$, then we can adjust upper bound value $P^U(x''_l|c^i_{1:q_i}, e)$ to $1 - P^L(x'_l|c^i_{1:q_i}, e)$. For nodes with domains of size > 2, we should compute both bounds and pick the highest value.

Next, we continue investigating the properties of ATB bounds by comparing them to the bounds obtained by bounded conditioning [56].

Comparing ATB framework with Bounded Conditioning

In ATB framework, the lower bounds are defined by expressions (4.22) and (4.26). Let us denote by BF a brute-force algorithm that trivially instantiates $P^L(x_l, c^j_{1:q_j}, e) = 0$ and $P^U(x_l, c^j_{1:q_j}, e) = 1$ for $\forall x_l \in \mathcal{D}(X_l)$ and also sets $P^U(c^j_{1:q_j}, e) = P(c^j_{1:q_j})$. Then, from Eq. (4.22), we get:

$$P^{L_1}_{BF}(x'_l|e) = \frac{\sum_{i=1}^{h} P(x'_l, c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P(c^j_{1:q_j})} = \frac{\sum_{i=1}^{h} P(x'_l, c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=h+1}^{M} P(c^j)} \quad (4.35)$$

Using algorithm BF with lower bound P^{L_2} in Eq. (4.26), we get:

$$P^{L_2}_{BF}(x'_l|e) = \frac{\sum_{i=1}^{h} P(x'_l, c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P(c^j_{1:q_j})} = \frac{\sum_{i=1}^{h} P(x'_l, c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=h+1}^{M} P(c^j)} \quad (4.36)$$

The right hand side in the equations (4.35) and (4.36) equals the expression for the bounded conditioning lower bound in Eq. (4.6). Namely, $P^{L_1}_{BF}(x'_l|e) = P^{L_1}_{BF}(x'_l|e) = P^L(x'_l|e)$ where

$P^L(x'_l|e)$ is obtained via Eq. (4.6). Since $P^{L_1}_{\mathcal{A}} \geq P^{L_1}_{BF}(x'_l|e)$ and $P^{L_2}_{\mathcal{A}} \geq P^{L_2}_{BF}(x'_l|e)$, then $P^{L_1}_{\mathcal{A}}, P^{L_2}_{\mathcal{A}} \geq P^L(x'_l|e)$.

We can prove in a similar manner that the ATB upper bound $P^{U_1}(x'_l|e)$ is as good or better than the upper bound obtained by bounded conditioning. Applying the brute-force algorithm BF, defined above, to bound $P(x_l, c^j_{1:q_j}, e)$, and $P(c^j_{1:q_j}, e)$, we get from Eq. (4.28):

$$P^{U_1}_{BF}(x'_l|e) = \frac{\sum_{i=1}^{h} P(x'_l, c^i, e) + + \sum_{j=1}^{M'} P(c^j_{1:q_j})}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P(c^j_{1:q_j})} = \frac{\sum_{i=1}^{h} P(x'_l, c^i, e) + \sum_{j=h+1}^{M} P(c^j)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=h+1}^{M} P(c^j)} \tag{4.37}$$

The expression for $P^{U_1}_{BF}$ gives us the worst-case upper bound that can be obtained by ATB from Eq. (4.28). In the next theorem, we prove that the upper bound $P^{U_1}_{BF}(x'_l|e)$ is as good or better than the bounded conditioning upper bound. Namely, $P^{U_1}_{A}$ dominates bounded conditioning as long as $P^U_A(x'_l, c^j_{1:q_j}, e) \leq P^U(c^j_{1:q_j})$:

THEOREM **4.3.4** *Given an algorithm $\mathcal{A}$ that computes lower and upper bounds $P^L_A(x_l, c^j_{1:q_j}, e)$ and $P^U_A(x_l, c^j_{1:q_j}, e)$ such that $\forall j$, $P^U_A(x_l, c^j_{1:q_j}, e) \leq P(c^j_{1:q_j})$ then $P^{U_1}_{\mathcal{A}}(x_l|e) \leq P^U(x_l|e)$ where $P^{U_1}_{\mathcal{A}}(x_l|e)$ is given in Eq. (4.28) and $P^U(x_l|e)$ is the bounded conditioning expression given in Eq. (4.10).*

The proof is given in Appendix D.

Weak ATB Bounds Framework (ATB^w)

In this section, we derive bounds expressions for a *weak* form of ATB, denoted ATB^w. We assume 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

$P(x_l, c_{1:q}, e)$ 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).

Following our assumption that $\mathcal{A}$ cannot produce non-trivial lower bounds for $P(x_l, c_{1:q}, e)$, it instantiates $P^L_{\mathcal{A}}(x_l, c_{1:q}, e) = 0$ and, since $P(x_l, c_{1:q}, e) \leq P(c_{1:q}, e)$, it instantiates $P^U_{\mathcal{A}}(x_l, c_{1:q}, e) = P^U_{\mathcal{A}}(c_{1:q}, e)$. Plugging in lower bound 0 and upper bound $P^U_{\mathcal{A}}(c_{1:q}, e)$ for $P(x_l, c_{1:q}, e)$ in Eq. (4.22), we get:

$$P^{L_3}_{\mathcal{A}}(x'_l|e) \triangleq \frac{\sum_{i=1}^{h} P(x'_l, c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=h+1}^{M'} P^U_{\mathcal{A}}(c^j_{1:q_j}, e)} \tag{4.38}$$

Similarly, replacing the lower and upper bounds on $P(x_l, c_{1:q}, e)$ with 0 and $P^U_{\mathcal{A}}(c_{1:q}, e)$ in the upper bound expression in Eq. (4.28), we obtain:

$$P^{U_3}_{\mathcal{A}}(x'_l|e) \triangleq \frac{\sum_{i=1}^{h} P(x'_l, c^i, e) + \sum_{j=1}^{M'} P^U_{\mathcal{A}}(c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^U_{\mathcal{A}}(c^j_{1:q_j}, e)} \tag{4.39}$$

Note that the expressions for lower and upper bounds in Eq. (4.38) and (4.39) above depend only on the values $P(c^i, e)$ and $P(x_l, c^i, e)$ 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. The penalty for using these more relaxed plug-in bounds is that there exists a lower bound on the computed bounds interval length:

THEOREM **4.3.5** *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, and 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. (4.38) and Eq. (4.39) respectively.

Despite these weakening simplifications, ATB^w is guaranteed to produce as good or better bounds as those obtained by bounded conditioning. The proof is obtained by plugging into ATB^w the brute-force bounding scheme BF described in Section 4.3.3.

We will investigate empirically the trade-offs between plugging in tighter bounds into ATB framework vs. computing more cutset tuples using the ATB^w framework.

4.4 Incorporating Bound Propagation into ATB

In Section 4.3 we defined the Any-Time Bounds framework. Founded on the conditioning principles, the framework computes exactly $P(x_l, c, e)$ and $P(c, e)$ for h cutset tuples and then uses precomputed bounds on $P(x_l, c_{1:q}, e)$ and $P(c_{1:q}, e)$ for the remaining partially instantiated cutset tuples which will be plugged into the corresponding expressions. In Section 4.4.1, we present a method for bounding probabilities $P(x_l, c_{1:q}, e)$ and $P(c_{1:q}, e)$ that is based on bound propagation. Then, in Section 4.4.3, we describe a simple improvement to the bound propagation algorithm BdP. Section 4.4.4 describes an approximate algorithm for solving the linear programming subproblems in the context of bound propagation, without using the simplex solver. As we will show, while the resulting bounds are not as tight, the scheme is an order of magnitude faster and, therefore, may present the right trade-off between time and accuracy when used within ATB. Finally, Section 4.4.5 presents an additional improvement using the ATB bounds as inputs to bound propagation.

4.4.1 Bounding $P(c_{1:q}, e)$ and $P(x_l, c_{1:q}, e)$ using Bound Propagation

We cannot use the bare outputs of bound propagation directly in ATB because BdP computes bounds on the conditional probabilities $P(x_l|e)$ rather than on the joint. In order to use BdP, we denote $Z = C_{1:q} \cup E$ and factorize the joint probability $P(c_{1:q}, e) = P(z)$ as follows:

$$P(z) = \prod_{z_j \in z} P(z_j|z_1, ..., z_{j-1})$$

Each factor $P(z_j|z_1, ..., z_{j-1})$ can be bounded by BdP, yielding:

$$P(z) \geq \prod_{z_j \in z} P^L_{BdP}(z_j|z_1, ..., z_{j-1}) \triangleq P^L_{BdP}(z) \tag{4.40}$$

$$P(z) \leq \prod_{z_j \in z} P^U_{BdP}(z_j|z_1, ..., z_{j-1}) \triangleq P^U_{BdP}(z) \tag{4.41}$$

Processing variables in some order $o = \{Z_1, ..., Z_n\}$, we first apply algorithm BdP to the network without any observations and bound $P(z_1)$. Then, we assign $Z_1 = z_1$ and apply algorithm BdP again to compute lower and upper bounds for $P(z_2|z_1)$, and so on. For each $Z_j \in Z$, we run BdP and obtain bounds $P^L_{BdP}(z_j|z_1, ..., z_{j-1})$. Depending on the ordering of the variables, some of the factors $P(z_j|z_1, ..., z_{j-1})$ may be computed exactly if the relevant subnetwork of $z_1, ..., z_j$ is singly-connected.

Similarly, the joint probability $P(x_l, c_{1:q}, e)$ can be factorized as:

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

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

well:

$$P^L_{BdP}(x_l, c_{1:q}, e) = P^L_{BdP}(x_l|c_{1:q}, e)P^L_{BdP}(c_{1:q}, e) \tag{4.42}$$

$$P^U_{BdP}(x_l, c_{1:q}, e) = P^U_{BdP}(x_l|c_{1:q}, e)P^U_{BdP}(c_{1:q}, e) \tag{4.43}$$

where $P^L_{BdP}(x_l|c_{1:q}, e)$ and $P^U_{BdP}(x_l|c_{1:q}, e)$ are obtained by algorithm BdP directly and $P^L_{BdP}(c_{1:q}, e)$ and $P^U_{BdP}(c_{1:q}, e)$ are obtained from Eq. (4.40) and Eq. (4.41).

4.4.2 Optimizing Variable Processing Order

The factorization order of the variables in Z may affect the efficiency of computation, i.e., BdP computation time and the tightness of the resulting bounds. It may also affect the number of factors $P(z_j|z_1, ..., z_{j-1})$ that can be computed exactly. Here, we analyze two factorization strategies.

One possible strategy is to process evidence variables first. Namely, compute:

$$P(e)^L_{BdP} = \prod_{e_j \in e} P^L_{BdP}(e_j|e_1, ..., e_{j-1}) \tag{4.44}$$

$$P(e)^U_{BdP} = \prod_{e_j \in e} P^U_{BdP}(e_j|e_1, ..., e_{j-1}) \tag{4.45}$$

Then, the lower and upper bound on $P(c_{1:q}, e)$ can be obtained by:

$$P^L_{BdP}(c_{1:q}, e) = \prod_{j=1}^{q} P^L_{BdP}(c_j|c_1, ..., c_{j-1}, e)P^L_{BdP}(e) \tag{4.46}$$

$$P^U_{BdP}(c_{1:q}, e) = \prod_{j=1}^{q} P^U_{BdP}(c_j|c_1, ..., c_{j-1}, e)P^U_{BdP}(e) \tag{4.47}$$

where bounds on $P(c_j|c_1, ..., c_{j-1}, e)$ are computed directly by BdP and $P^L_{BdP}(e)$ and $P^U_{BdP}(e)$ are obtained from Eq. (4.44) and (4.45). In the above scenario, we can precom-

pute $P^L_{BdP}(e)$ and $P^U_{BdP}(e)$ before we start bounding truncated tuples. Then, for each truncated tuple $c_{1:q}$, we will only need to run BdP algorithm $q+1$ times, one time for each $P(c_j|c_1,...,c_{j-1},e)$, and one more time to bound $P(x_l|c_{1:q},e)$. The complexity of the above computation scheme is as follows:

THEOREM **4.4.1 (complexity 1)** *Given a Bayesian network over a set of variables X with the maximum domain size d and loop-cutset C, the complexity of ATB with bounds precomputed by BdP using the variable factorization order $o = \{E, C\}$ is $O(|E| \cdot T_{BdP} + |C|^2 \cdot h \cdot (d-1) \cdot T_{BdP})$ when the time complexity of BdP is $O(T_{BdP})$.*

Proof. The complexity of precomputing bounds on $P(e)$ using BdP is $O(|E| \cdot T_{BdP})$ since we execute algorithm once for each evidence variable. The complexity of bounding one truncated tuple is at most $O(|C| \cdot T_{BdP})$ since a truncated tuple can contain up to $|C|-1$ cutset variables. Since there are $O(|C| \cdot h \cdot (d-1))$ truncated tuples (Proposition 4.3.1), the result follows. ■

An alternative strategy is to process variables in topological order. Let $o = \{Z_1, ..., Z_n\}$ be a topological order of variables in Z. Let $Z^- = \{Z_1, ..., Z_m | Z_m = C_q\}$, namely, let Z^- denote a subset of variables in Z that includes all cutset variables C_1 through C_q and evidence variables that precede C_q in topological order. Let $E^- = E \cap Z^-$ and $E^+ = E \backslash Z^-$. That is, the subset E^- contains the evidence variables preceding C_q in topological order (included in Z^-) and E^+ contains the remaining evidence variables (not included in Z^-). We factorize the joint probability $P(c_{1:q}, e)$ as follows:

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

We know that it is easy to compute $P(c, e)$ if $C \cup E$ form a loop-cutset. It turns out that

computing probability $P(z^-) = P(c_{1:q}, e^-)$ is easy too. The result follows from Theorem 2.3.1 (proved in section 2.3 of chapter 2) which states: *Given Bayesian network over X, evidence $E \subset X$, and cutset $C \subset X \backslash E$, let $Z = C \cup E$ be a loop-cutset. If Z is topologically ordered, then $\forall Z_j \in Z$ the relevant subnetwork of $Z_1, ..., Z_j$ is singly-connected when $Z_1, ..., Z_j$ are observed.*

By definition, subset Z^- satisfies the conditions of Theorem 2.3.1. Therefore:

COROLLARY **4.4.1** *Since the relevant subnetwork over $c_{1:q}$ and e^- is singly-connected, we can compute joint probability $P(c_{1:q}, e^-)$ in linear time.*

We can now apply algorithm BdP to the network conditioned on $c_{1:q}$ and e^- and obtain bounds on $P(e_1|c_{1:q}, e^-)$ for $e_1 \in e^+$, then on $P(e_2|e_1, c_{1:q})$ for $e_2 \in e^+$, and in sequence get bounds $P^L_{BdP}(e_j|e_1, ..., e_{j-1}, c_{1:q}, e^-)$ and $P^U_{BdP}(e_j|e_1, ..., e_{j-1}, c_{1:q}, e^-)$ for each $e_j \in e^+$. Thus:

$$P(c_{1:q}, e) \geq \prod_{e_j \in e^+} P^L_{BdP}(e_j|e_1, ..., e_{j-1}, c_{1:q}, e^-) P(c_{1:q}, e^-) \triangleq P^L_{BdP}(c_{1:q}, e) \quad (4.48)$$

$$P(c_{1:q}, e) \leq \prod_{e_j \in e^+} P^U_{BdP}(e_j|e_1, ..., e_{j-1}, c_{1:q}, e^-) P(c_{1:q}, e^-) \triangleq P^U_{BdP}(c_{1:q}, e) \quad (4.49)$$

The complexity of ATB when using BdP as described above to precompute the input bounds is given next:

THEOREM **4.4.2 (Complexity 2)** *Given a Bayesian network over a set of variables X with maximum domain size d and loop-cutset C, the complexity of ATB with bounds precomputed by BdP using the topological variable factorization order is $O((N + T_{BdP} \cdot (1 + |E|)) \cdot h \cdot (d-1) \cdot |C|)$ when N bounds the size of the input network and the time complexity of BdP is $O(T_{BdP})$.*

Proof. For each tuple $c_{1:q}$, we compute $P(c_{1:q}, e^-)$ in $O(N)$ and apply BdP $|E^+|$ times to

compute bounds on $P(e_j|c_{1:q}, e^-, e_1, ..., e_{j-1})$ for each evidence variable in E^+. We need to apply BdP one more time to compute bounds on $P(x_l|c_{1:q}, e)$. In the worst case, $|E^+| = |E|$ and we execute BdP a total of $(1+|E|)$ times for each truncated tuple. Hence, the total complexity of bounding one partially-instantiated cutset tuple is $O(N + (1 + |E|) \cdot T_{BdP})$. Since the total number of partially-instantiated cutset tuples is bounded by $O(|C| \cdot h \cdot (d-1))$ (Proposition 4.3.1), the total cost of bounding the probability mass of the truncated cutset tuples is $O((N + T_{BdP} \cdot (1 + |E|)) \cdot h \cdot (d-1) \cdot |C|)$. ■

In our experiments, we bound probabilities of the truncated tuples using topological variable order in the factorization of $P(c_{1:q}, e)$. Although a better ordering, producing tighter bounds on $P(c_{1:q}, e)$ in less time, may exist, the topological ordering of the variables guarantees that the upper bound on the probability mass of the truncated tuples does not exceed the prior probability mass of those tuples.

4.4.3 Improving Bound Propagation

Here, we describe how we can improve the accuracy and time of BdP by taking into account the network structure.

Since we plan to plug-in BdP into our ATB scheme, we must pay a special attention to its performance. As shown, BdP's complexity is exponential in the size of Markov blanket.

Restricting Markov blankets

We will control the computation time of bound propagation by restricting the Markov blanket space (or table size). The bounds of variables whose Markov blanket size exceeds maximum will not be updated and, thus, will remain equal to their input values (usually 0

and 1). In turn, it will affect the quality of the bounds for neighboring nodes. As explained, each node Y_i in the Markov blanket of X induces a linear constraint of the form:

$$P^L(y_j|e) \leq \sum_{y \setminus y_j, Y_j = y_j} P(y_1, ..., y_k|e) \leq P^U(y_j|e) \tag{4.50}$$

When the lower and upper bounds $P^L(y_j|e)$ and $P^U(y_j|e)$ are 0 and 1 respectively, then the constraint expressed in Eq. (4.50) is redundant and always holds. We can control the trade-offs between computation time and tightness of the bounds using a parameter k to specify the maximum Markov blanket size, thus, obtaining a parametrized version of bound propagation $BdP(k)$.

Exploiting Relevant Subnetwork Properties

Next, we show how we can reduce the effective Markov blanket size by restricting the Markov blanket of a node to its relevant subnetwork (see Definition 1.2.5 in Section 1.2.1).

It is easy to see that the set of linear inequalities in Eq. (4.15) can be restricted only to the "relevant" portion of the Markov blanket of node X_l. Therefore, for every node X_l, the application of BdP can be made more efficient and will often include only the variable's parent set.

Removing irrelevant nodes (and their parents) from the Markov blanket whenever possible results in a smaller Markov blanket size, shorter computation time, and more accurate bounds as we will demonstrate empirically. In particular, if the relevant subnetwork of node X_l is singly-connected then its posteriors can be computed exactly. We denote by $BdP+$ the bound propagation algorithm that exploits the idea of relevant subnetwork

structure.

In the next section, we present a greedy algorithm for solving the linear programming subproblems in bound propagation which computes suboptimal minimum and maximum of the objective function but substantially reduces the computation time. By replacing the simplex solver with the proposed fast greedy algorithm, we obtain an approximate bouvvnd propagation scheme.

4.4.4 Approximating the LP in Bound Propagation

When $BdP+$ scheme is plugged into the ATB framework, we need to bound a large number of truncated tuples. We need to invoke bound propagation algorithm $1 + |E|$ times to bound one tuple (as we showed in the proof of Theorem 4.4.2). In a single iteration of $BdP+$, we need to solve $O(|X \backslash E| \cdot d)$, where d is the maximum domain size, linear optimization problems (a different optimization problem for each value of each variable). Thus, the ATB framework requires to solve thousands if not hundreds of thousands of linear optimization problems implied by $BdP+$. Therefore, using the simplex method for each problem becomes impractical.

It turns out that the linear optimization problems formulated by bound propagation fall into a class of linear packing and covering problems. The standard fractional packing and covering problem can be defined as follows:

$$\begin{aligned} \min c^T \overline{x} & & (4.51)\\ s.t. & & (4.52)\\ A\overline{x} & \geq l & (4.53)\\ B\overline{x} & \leq m & (4.54)\\ x_i \geq 0, \forall i & & (4.55) \end{aligned}$$

The problem without Eq. (4.54) is called a **fractional covering** problem. The problem without Eq. (4.53) is called a **fractional packing** problem. As the constraints expressed in Eq. (4.15) have both lower and upper bounds, the linear optimization problems in $BdP+$ contain both covering and packing constraints where $A = B$ and A is a 0/1 matrix. Therefore, the bound propagation LP minimization problem can be described as follows:

$$\begin{aligned} \min c^T \overline{x} & & (4.56)\\ s.t. \sum_j x_j = 1 & & (4.57)\\ l \leq A\overline{x} \geq m & & (4.58)\\ 0 \leq x_j \leq 1, \forall j & & (4.59) \end{aligned}$$

where each pair of values l_i and m_i corresponds to lower and upper bound values on some $P(y_k|e)$ and $\forall i, j,\ a_{ij} \in \{0, 1\}$.

Fractional packing and covering problems often rise as a result of solving a relaxation of combinatorial packing and covering problems. The fraction solution is then used as a starting point for finding an integer solution. In the above application, the precision of the fractional solution is often less important then the speed of computation. Subsequently, a number of approximation methods have been developed for solving those classes of prob-

lems [17]. However, those algorithms solve either packing or covering problems, but not both and not with the additional sum-to-1 constraint. We, therefore, resort to solving a relaxed version of the bound propagation LP problems.

There are many possibilities for relaxing the constraints of the original problem. We looked at two alternative relaxations. One is a relaxation to *fractional knapsack packing* and the other is a relaxation to what could be viewed as *fractional multiple knapsack packing with lower bounds*.

Fractional Knapsack Packing

Knapsack Packing is a well known problem. Given a set of items $\overline{x}$ such that each item $x_i \in \overline{x}$ has an associated profit c_i and a size u_i, the objective is to select a subset of items that maximizes the profit such that the total size of the selected items does not exceed the knapsack capacity. In the fractional version of the problem, we can select a portion of the item x_i; in this case, u_i becomes an upper bound on how much of x_i we can take. For fractional packing, we can always fill the knapsack to full capacity and therefore the packing problem can be solved exactly in a greedy fashion.

In the bound propagation LP problem, the sum-to-1 constraint can be interpreted as specifying the "knapsack" capacity. Hence, we maintain the sum-to-1 constraint in the relaxed version of the problem. We drop the lower bound constraints completely. The upper bound on a sum of variables is replaced with the upper bound on individual variables, i.e., for each variable x_i we specify an upper bound $u_j = \min_{i,a_{ij}=1} m_i$ which is the minimum

upper bound value of all the constraints in which variable x_i participates. We obtain the following relaxed LP problem, which is an instance of fractional knapsack packing:

$$\max c^T \overline{x} \tag{4.60}$$

$$s.t. \sum_i x_i = 1 \tag{4.61}$$

$$0 \leq x_j \leq u_j \tag{4.62}$$

The problem can be solved exactly by ordering nodes by their profit and then assigning each node its maximum value until the sum of all node values equals 1. The complexity of the algorithm is $O(n \log n)$, where n is the number of variables, due to the complexity of sorting.

Fractional Knapsack Packing with Multiple Knapsacks

The *Multiple Knapsack* problem (MKP) is a natural generalization of the single knapsack problem defined as follows. The main difference is that we now have not one knapsack, but a set of bins (knapsacks) of varying capacity.

In the relaxation of the bound propagation LP problem, we maintain the sum-to-1 constraint and the constraints induced by a single node in the Markov blanket. The rest of the constraints are dropped after computing individual upper bound u_i for each variable x_i as we did before for fractional packing with single knapsack. Without loss of generality, we can assume that the selected Markov blanket node is Y_q. Then, the relaxed version of the problem can be formulated as follows:

$$\max c^T \overline{x} \tag{4.63}$$

$$s.t. \sum_i x_i = 1 \tag{4.64}$$

$$LB(y_q) \leq \sum_i \delta(y_q, x_i) x_i \leq UB(y_q), \forall y_q \in \mathcal{D}(Y_q) \tag{4.65}$$

$$0 \leq x_j \leq u_j \tag{4.66}$$

where $\delta(y_k, x_i) = 1$ if the value y_k of the variable Y_k matches the instantiation of Y_k in x_i (recall that $x_i = P(y_1, ..., y_q|e)$) and $\delta(y_k, x_i) = 0$ otherwise. We can view the problem as the one of packing $|\mathcal{D}(Y_q)|$ knapsacks, where each knapsack corresponds to some value y_q of variable Y_q and has capacity $UB(y_q)$.

Our problem has additional special properties though. Each knapsack has not only upper bound, but also a lower bound on the size of the load. The sum-to-1 constraint specifies the total size of the load. Further, the domains of the constraints induced by variable Y_q are disjoint. Namely, each variable x_i will participate in only one constraint of the type expressed in Eq. (4.65). The latter can be interpreted as each knapsack having a separate list of items that can be packed in it.

We can obtain a more general formulation of the problem if we use a double index ij with each variable x_{ij} to indicate that it is the quantity of jth item that is placed in knapsack i. Then, we can re-formulate the problem as follows:

$$\max \sum_{i,j} c_{ij} x_{ij} \tag{4.67}$$

$$s.t. \sum_{i,j} x_{ij} = 1 \tag{4.68}$$

$$l_i \leq \sum_{j} x_{ij} \leq u_i, \forall i \tag{4.69}$$

$$0 \leq x_{ij} \leq u_{ij} \tag{4.70}$$

If it was not for the sum-to-1 constraint in Eq. (4.64), we could solve each knapsack-packing problem independently. Still, the greedy approach works. Figure 4.4 defines an algorithm for solving the LP described in Eq. (4.67)-(4.70).

For maximization problem, first, we order variables by their objective function coefficient value from largest to smallest (step 1 in Figure 4.4) and initialize all variable values to 0 (step 2). Then, we make two passes. On the first pass, we assign each node the minimum value necessary to satisfy lower bounds (step 3). We make a second pass incrementing each node value to the maximum, within the constraint bounds, until the sum of all variables equals 1 (step 4). The solution to the minimization problem is the same except variables are ordered by their objective function coefficient value from smallest to largest. We prove the correctness of the algorithm in Appendix E. The complexity of the algorithm is $O(n \log n)$ where $n = |\overline{x}|$. In the contex of bound propagation, $n = |\mathcal{D}(Y)|$, namely, the size of the Markov conditional probability table.

Since we cannot predict which node $Y_j \in Y$ will yield the LP problem with the smallest maximum of the objective function, we solve separately a fractional MKP problem for each $Y_j \in Y$ and pick the smallest value. Then, the total complexity of finding lower and

Greedy Multiple Knapsack With Lower Bounds
Input:

$$\max \sum_{i,j} c_{ij} x_{ij} \tag{4.71}$$

$$s.t. \sum_{i,j} x_{ij} = 1 \tag{4.72}$$

$$l_i \leq \sum_{j} x_{ij} \leq u_i, \forall i \tag{4.73}$$

$$0 \leq x_{ij} \leq u_{ij} \tag{4.74}$$

Output: $f = \max \sum_{i,j} c_{ij} x_{ij}$
1. Sort x_i by coefficients c_i (profit) from largest to smallest.
2. Initialize: $\forall i$, $x_i \leftarrow 0$
3. *Satisfy lower bounds*:
For $i \leftarrow 1$ to $|\mathcal{D}(\mathbf{Y})|$ do:
    $x_{ij} \leftarrow \min\{l_i, u_{ij}\}$
    $u_{ij} \leftarrow u_{ij} - x_{ij}$
    $l_i \leftarrow l_i - x_{ij}$
    $u_i \leftarrow u_i - x_{ij}$
    $s \leftarrow s - x_{ij}$
    $f \leftarrow f + c_{ij} \cdot x_{ij}$
End For
4.4. *Maximize*:
For $i = 1$ to $|\mathcal{D}(\mathbf{Y})|$ do:
    if (s = 0) break
    $\delta = \min\{u_j, u_{ij}, s\}$
    $x_{ij} \leftarrow x_{ij} + \delta$
    $u_{ij} \leftarrow u_{ij} - \delta$
    $u_i \leftarrow u_i - \delta$
    $s \leftarrow s - \delta$
    $f_j \leftarrow f_j + c_{ij} \cdot \delta$
End For

Figure 4.4: Greedy algorithm for fractional MKP problem.

upper bounds for a single variable value is $O(|Y| \cdot n \log n)$. We denote a bound propagation scheme which solves the linear optimization problems approximately by $ABdP+$ for approximate $BdP+$. We will compare the performance of $ABdP+$ using fractional packing with single knapsack and multiple knapsacks in the empirical section.

4.4.5 Algorithm $BBdP+$

$ABdP+$ and $BdP+$ performance depends on the initial values of the lower and upper bounds, usually set to 0 and 1. We can boost the performance of $BdP+$ by using the bounds computed by ATB, instead of 0 and 1, to initialize its lower and upper bounds. As we mentioned earlier, the tightness of the bounds on the posterior marginals of the nodes in the Markov blanket of variable X_i affects the tightness of the constraints in the LP optimization problems for X_i. If ATB computes tighter bounds than $BdP+$ (starting with 0/1 initial bounds) for some variables, then bound propagation may be able to compute tighter bounds on the neighboring nodes. An analogy can be made where tightening some of the bolts holding together the elements of a complex structure restricts the movement of the remaining elements. We can think of using ATB bounds as the input to $BdP+$ or $ABdP+$ as "boosting" bound propagation. We will show results with boosted $BdP+$, denoted $BBdP+$, in the empirical section.

4.5 Searching for High-Probability Tuples

We can expect a better performance from ATB (faster convergence with h) when the selected h cutset tuples are the h highest probability tuples. The task of finding the h highest probability tuples is related to the MAP problem of finding the maximum probability instantiation of a subset of variables. For $h = 1$, the two tasks are equivalent. For $h > 1$, MAP is a subproblem since the maximum probability tuple is the member of the set of the h highest probability tuples. MAP problem is $\#P$-hard [94] (the decision version is NP^{PP}-hard). In fact, it remains NP-hard even in polytrees where computing the posterior marginals is easy. Therefore, finding the h highest probability tuples is also hard.

We can search for the h highest probability cutset tuples using, for example, local greedy search [62, 93, 94].

In [62], local search algorithm is used to find an approximate solution to the MPE problem. In [93] and [94], the same idea is applied to the MAP problem. Another option is to use stochastic simulation such as Gibbs sampling. This is the approach we take here.

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 $S = \{c^{(j)}\}$ such that the frequency of a tuple $c^i \in S$ 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 instances.

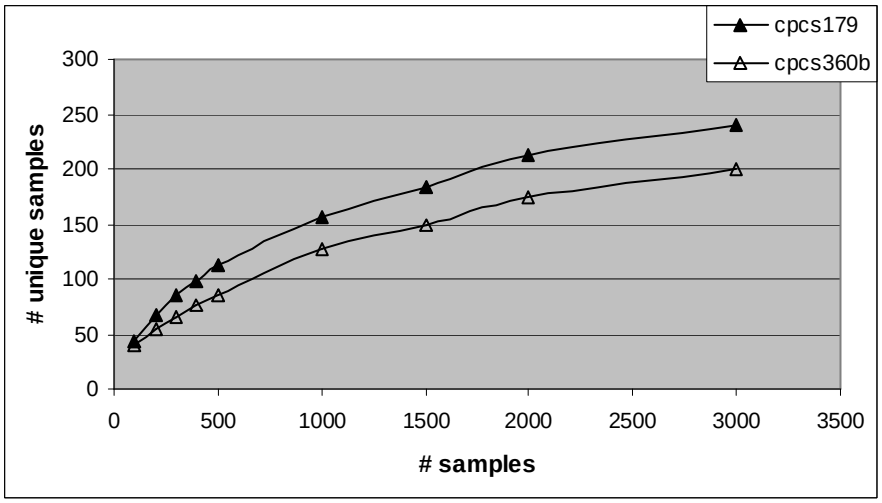


Figure 4.5: The number of unique samples generated by a Gibbs sampler sampling over a loop-cutset is plotted against the total number of samples for two benchmarks, cpcs179 and cpcs360b. The results are averaged over 20 instances of each benchmark with different evidence values.

We propose to generate the cutset tuples using cutset sampling [14, 15] that applies Gibbs sampling over a subset of the variables. Specifically, we focus on the loop-cutset sampling. We used first an ordered Gibbs sampler. 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 4.5 shows the number of unique tuples as a function of the number of samples in cpcs179 and cpcs360b networks. The results are averaged over 20 instances for each benchmark. As we can see, the curves are logarithmic. Namely, the algorithm found most of the high probability tuples quickly and then mostly revisited previously observed tuples.

However, in other benchmarks, cutset sampling often required too long to find enough "heavy" tuples. This was observed previously in [62] where the effectiveness of Gibbs sampler and local greedy search in finding the MPE solution were compared. The main

difference between Gibbs sampleing and greedy local search is that, given distribution $P(X_i|x_{-i})$, Gibbs sampling draws the new value for a variable from this distribution while greedy local search picks the most likely value of X_i. In [62] it was shown that a combination of Gibbs sampling and greedy local search is more effective than either method alone. In our implementation, without modifying Gibbs sampler, we maximize the value of the output it produced by maintaining a list of the h highest probability tuples encountered. We elaborate on the options in the next two paragraphs.

In order to obtain the distribution $P(C_i|c_{-i})$ for a discrete cutset variable C_i, Gibbs-based cutset sampling normally computes probability $P(c_i, c_{-i})$, for each $c_i \in \mathcal{D}(C_i)$. Once a new value for variable C_i is sampled from $P(C_i|c_{-i})$, all those probabilites are discarded. However, the sampled value is not always the most probable one and the tuples with higher probability $P(c_i, c_{-i})$ may be discarded. Maintaining a list of h highest probability tuples computed during sampling (even if the Markov chain did not actually visit them), we optimize Gibbs sampling for search. We denote resulting scheme as $Gopt$.

Figure 4.6 presents the ordered Gibbs sampling scheme that maintains the list of h highest probability tuples. We use $c^{(t)} = \{c_1^{(t)}, c_2^{(t)}, ..., c_k^{(t)}\}$ to denote sample t. Each sample $c^{(t)}$ is some instantiation $c^i \in \mathcal{D}(C)$. A list L stores pairs $< c, P(c, e) >$ where c is the cutset tuple and $P(c, e)$ is its corresponding probability. The scheme in Figure 4.6 only differs from regular Gibbs sampler in that we add step 1.2 where we update list L. If L contains tuples c, we do nothing. Otherwise, if the size of L is less than h and $c \notin L$, we add c to L. If the size of L equals h and $c \notin L$, we find a tuple $c' \in L$ whose probability

Algorithm $Gopt$
Input: Bayesian network $\mathcal{B}$ over X, evidence $E \subset X$, cutset $C \subset X \backslash E$, and integer $h < |\mathcal{D}(C)|$, integer T.
Output: list L of h high probability tuples.
$L \leftarrow \{\}$
Initialize: Assign random value c_i^0 to each $C_i \in C$ and assign e.
Generate samples:
For t = 1 to T, generate a new sample $c^{(t+1)}$ as follows:
For Each $C_i \in C$, compute a new value $c_i^{(t)}$ for variable C_i as follows:
1. For Each $c_i \in \mathcal{D}(C_i)$ do:
1.1. Compute $P(c_i, c_{-i}^{(t)}, e)$.
1.2. If $c' = \{c_i, c_{-i}^{(t)}\} \notin L$ Then
If $|L| < h$ Then
$L \leftarrow L \bigcup <c', P(c', e)>$
Else
$c^* \leftarrow \arg\min_{c \in L} P(c, e)$
If $P(c^*, e) < P(c', e)$ Then
$L \leftarrow L \setminus <c^*, P(c^*, e)>$
$L \leftarrow L \bigcup <c', P(c', e)>$
End If
End If
1.3 End If
2. End For Each c_i
3. $P(C_i | c_{-i}^{(t)}, e) \leftarrow \alpha P(C_i, c_{-i}^{(t)}, e)$.
4. Sample new value:

$$c_i^{(t+1)} \leftarrow P(C_i | c_{-i}^{(t)}, e) \tag{4.75}$$

End For C_i
End For t

Figure 4.6: w-*Cutset sampling* Algorithm

$P(c', e)$ is the smallest. If $P(c', e) < P(c, e)$, then we replace c' with c.

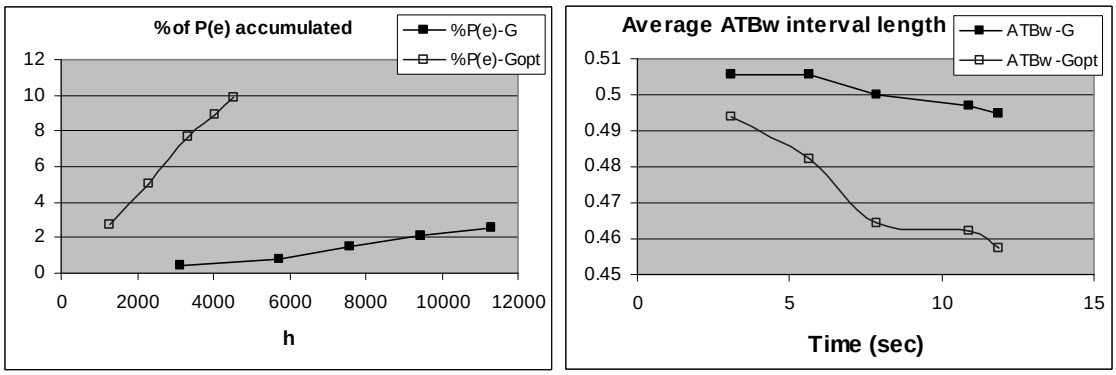


Figure 4.7: Performance of regular Gibbs cutset sampling (G), selecting h cutset tuples with highest probabilities among generated samples, and optimized for search (Gopt), selecting h tuples among all computed tuples, in an instance of Barley network. The left chart shows the percent of $P(e)$ covered by the h tuples as a function of h. The right chart shows the average bounds interval length computed by $ATBw$ as a function of time when using G ($ATBw$-G) and $Gopt$ ($ATBw$-$Gopt$) to find h highest probability tuples.

Figure 4.7 demonstrates empirically a typical improvement in the performance of $Gopt$ over regular Gibbs sampling, denoted G, on an instance of Barley network with 7 evidence nodes and $P(e) = 3E-06$. The chart on the left measures the efficiency of the search process by the percent of the probability mass $P(e)$ covered by the generated tuples, namely:

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

You may recall it provides a lower bound on the bounds interval length for ATB^w. As we can see, the percent of $P(e)$ covered grows much faster when h is selected among all computed tuples using $Gopt$. The chart on the right in Figure 4.7 shows the average bounds interval length computed by ATB^w using the h highest probability tuples generated by Gibbs sampling ($ATBw$-G) and by $Gopt$ ($ATBw$-$Gopt$).

In other experiments, using $Gopt$, we accumulated over 90% of the weight of $P(e)$ in a few thousand tuples in Munin3, cpcs179, and cpcs360b, and 20-30% in cpcs422b and Munin4. In the case of Barley network, we generated up to $h = 2000$ cutset tuples, that is less than 0.0002% of over 2 million tuples, that covered $\approx 1\%$ of the weight of $P(e)$.

4.6 Experiments

In this section, we compare empirically the performance of several bounding schemes discussed in this paper.

4.6.1 Methodology and Algorithms

Algorithms

We evaluate empirically the quality of the bounds obtained by different variants of bound propagation and ATB framework with different plug-in bounding schemes. Our algorithms are:

- BdP - an original bound propagation algorithm that initializes all variables' lower and upper bounds to 0 and 1 and then iteratively updates the bounds using the simplex solver to find the minimum and maximum of the objective function of the linear optimization problem over the Markov blanket of each variable;
- $BdP+$ - an improved bound propagation algorithm that restricts the Markov blanket of a variable to a relevant subnetwork w.r.t. the set containing this variable and evidence

and pre-computes the posterior marginals of nodes whose relevant subnetwork is singly-connected;

- $ABdP$+$FP1$ - an approximate bound propagation algorithm that uses a greedy algorithm to solve the relaxation of the LP problem to fractional packing with a single knapsack;
- $ABdP$+FPM - same as $ABdP$+$FP1$ except it uses an LP relaxation with multiple knapsacks;
- ATB-$FP1$ - ATB framework using $ABdP$+$FP1$ plug-in;
- ATB-FPM or ATB - ATB framework using $ABdP$+FPM plug-in; unless stated otherwise, we reserve the name ATB for ATB-FPM;
- ATB^w - a weak form of ATB using $ABdP$+FPM plug-in; recall that, given h fully-instantiated tuples, ATB^w bounds the remaining probability mass faster than ATB because it only computes an upper bound on $P(c_{1:q}, e)$;
- $BBdP+$ - boosted $BdP+$ that uses the ATB bounds to initialize the starting bounds values for all variables.

The results are organized as follows. First, we compare the performance of the two bound propagation algorithms, BdP and $BdP+$. Next, we compare bound propagation using the simplex solver, $BdP+$, and the two variants of the approximate bound propagating, $ABdP$+$FP1$ and $ABdP$+FPM. Subsequently, we also compare the performance of ATB using the two variants of $ABdP+$ as plug-ins, i.e., $ABdP$+$FP1$ and $ABdP$+FPM.

Finally, we compare side-by-side the performance of the best bound propagation schemes $BdP+$, ATB using the best of the two approximate bound propagation schemes $ABdP+$-FPM as a plug-in, ATB^w using the same plug-in, and boosted $BdP+$. Where applicable, we also compare our bounds with those in [73].

4.6.2 Measures of Performance

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

$$\overline{I} = \frac{\sum_i \sum_{\mathcal{D}(x_i)} (P^U(x_i|e) - P^L(x_i|e))}{\sum_i |\mathcal{D}(x_i)|} \tag{4.76}$$

We 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:

$$\hat{P}(x|e) = \frac{P^U(x|e) + P^L(x|e)}{2} \tag{4.77}$$

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

$$\Delta = \frac{\sum_i \sum_{\mathcal{D}(x_i)} |P(x_i|e) - \hat{P}(x_i|e)|}{\sum_i |\mathcal{D}(x_i)|} \tag{4.78}$$

All exact posterior marginals, used to evaluate the precision of the estimate expressed in Eq. (4.77), were obtained by bucket elimination [28] using the min-fill heuristics for ordering variables.

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

4.6.3 Reporting of the Results

In all benchmarks, ATB and ATB^w enumerate the tuples of a loop cutset C that is generated by the mga algorithm [10]. The results for ATB and ATB^w schemes are reported as a function of h, the number of fully instantiated cutset tuples, and time. Recall that the upper bound on the number of partial cutset tuples is proportional to h (see Theorem 4.3.1). Consequently, the worst case computation time of ATB and ATB^w grows as h increases (see Theorem 4.4.2). Of course, as h becomes close to the total number of cutset tuples, M, the number of truncated tuples decreases. However, in our experiments, h is small compared to the total number of cutset tuples. Hence, computation time increases monotonously with h.

As described, we control the time and memory of bound propagation by restricting the maximum size k of the conditional probability table for a variable and its Markov blanket. Algorithms BdP, $BdP+$, and $ABdP+$ are parametrized by k. For the plug-ins used with ATB and ATB^w, the maximum Markov CPT size was fixed at $k = 1025$.

We also fixed the parameter $k = 1025$ for $BBdP+$. The computation time of $BBdP+$ includes bound propagation time and ATB time. For a constant parameter k, the computation time of bound propagation remains nearly constant for a given network instance and only the computation time of ATB changes. Since ATB time is a function of h, $BBdP+$ time is a function of h as well.

4.6.4 Benchmarks

Table 4.1: Complexity characteristics of the benchmarks 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|**	$\|\mathcal{D}(\mathbf{LC})\|$	**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[1]
cpcs54	54	15	6	32768	1 sec	22 sec
cpcs179	179	8	8	49152	2 sec	37 sec
cpcs360b	360	21	26	2^{26}	20 min	> 8 hrs[1]
cpcs422b	422	22	47	2^{47}	50 min	> 2E+9 hrs[1]
Munin3	1044	7	30	$> 2^{30}$	8 sec	> 1700 hrs[1]
Munin4	1041	8	49	$> 2^{49}$	70 sec	> 1E+8 hrs[1]

Our benchmarks are Alarm network, 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 4.6.4. For each network, the table specifies the number of variables N, the induced width $w*$, the size of loop cutset $|LC|$, the number of loop-cutset tuples $|\mathcal{D}(LC)|$, and the time needed to compute the exact posterior marginals by bucket-tree elimination (exponential in the induced width $w*$), and by cutset conditioning (exponential in the size of loop-cutset).

The Alarm network models the monitoring of patients in intensive care [11]. The Barley network is a part of the decision-support system for growing malting barley developed in [70]. We have performed experiments with four CPCS networks: cpcs54, cpcs179, cpcs360b, and cpcs422b. CPCS networks are derived from the Computer-Based Patient

[1]Times are extrapolated.

Care Simulation system and based on INTERNIST-1 and Quick Medical Reference Expert systems [100]. The last two benchmarks are Munin3 and Munin4, the subsets of the Munin network which is a part of the expert system for computer-aided electromyography [2].

Computing posterior marginals is easy in Alarm network, cpcs54, and cpcs179 using either bucket elimination or cutset conditioning since they have small induced and a small loop-cutset. We include those benchmarks as a proof of concept only. Several other networks, Barley, Munin3, and Munin4, also have small induced width and, hence, their exact posterior marginals can be obtained by bucket elimination.

However, for a fair comparison, ATB should be compared against linear-space schemes such as cutset-conditioning. From this perspective, Barley, Munin3, and Munin4 are hard. For example, Barley network has only 48 variables, its induced width is $w^* = 7$, and exact inference by bucket elimination takes only 30 seconds. Its loop-cutset contains only 12 variables, but the number of loop-cutset tuples exceeds 2 million because some variables have large domain sizes (up to 67 values). Enumerating and computing all cutset tuples, at a rate of about 1000 tuples per second, would take over 22 hours. Similar considerations apply in case of Munin3 and Munin4 networks.

For most benchmarks, the results for each network are averaged over 20 instances instantiated with different evidence.

4.6.5 Results with BdP Variants

Here, we report the bounds intervals obtained by $BdP(k)$ and $BdP+(k)$ as a function of k, the maximum Markov CPT length, for $k \in [2^{10}, 2^{19}]$ on various benchmarks with and without evidence. For $k > 2^{19}$, the computation demands exceeded available memory. The computation time of BdP and $BdP+$ is a function of k and the number of iterations needed to converge. Since the algorithms usually converged in less than 20 iterations, we fixed the maximum number of iterations at 20.

In Table 4.2 and Table 4.3 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 (see Eq. (4.76-(4.78)), while columns 6-7 specify the accuracy and computation time for $BdP+$. We explore the range of values of $k = 2^m$ for $m \in [10, 11, ..., 20]$. The results are separated for networks without evidence (Table 4.2) and networks with evidence (Table 4.2).

Analysis: $BdP+$ always computes tighter bounds and requires less computation time than BdP. The performance gap is wider in the networks without evidence (Table 4.2) where the 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. For the largest benchmark, 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,

Table 4.2: 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 networks without evidence.

		BdP(k)			BdP+(k)		
	k	$\overline{I}$	Δ	**time**	$\overline{I}$	Δ	**time**
Alarm	16384	0.637	0.1677	14	0.075	0.0076	0.1
cpcs54	16384	0.425	0.0229	24	0.091	0.0049	0.1
	32768	0.417	0.0224	72	0.091	0.0049	0.1
	65536	0.417	0.0224	72	0.091	0.0049	0.1
	131072	0.417	0.0224	72	0.091	0.0049	0.1
	262145	0.415	0.0221	265	0.091	0.0049	0.1
cpcs179	16384	0.576	0.2213	30	0.0006	0.00002	0.3
	32768	0.576	0.2213	30	0.0006	0.00002	0.3
	65536	0.576	0.2213	30	0.0006	0.00002	0.3
cpcs360b	16384	0.151	0.0649	64	0.0006	0.0002	1.2
	32768	0.149	0.0641	80	0.0006	0.0002	1.2
cpcs422b	16384	0.237	0.0756	28	0.008	0.0008	8
	262145	0.236	0.0751	33	0.008	0.0008	8

BdP computations take 190 sec while $BdP+$ only takes 16 sec.

Both BdP and $BdP+$ bounds interval becomes larger and computation takes longer when some nodes are observed, as shown in Table 4.3. 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 0.338 and requires 68 seconds. $BdP+$ computes an average bounds interval of 0.064 and requires only 15 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.

Table 4.3: 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	1024	0.828	0.2661	13	0.611	0.2151	4.2
$\|E\|$=3-6	524288	0.828	0.2661	13	0.611	0.2151	4.2
cpcs54	1024	0.616	0.0469	6.6	0.367	0.0232	2.1
$\|E\|$=2-6	4096	0.616	0.0469	6.6	0.360	0.0219	3.4
	16384	0.595	0.0458	30	0.353	0.0210	7.9
	32768	0.591	0.0451	49	0.353	0.0210	9.3
	65536	0.589	0.0450	66	0.353	0.0210	13
	131072	0.588	0.0449	88	0.353	0.0209	23
	262144	0.587	0.0447	166	0.352	0.0207	52
	524288	0.587	0.0447	260	0.352	0.0207	54
cpcs179	1024	0.604	0.2228	30	0.230	0.0783	5.3
$\|E\|$=12-24	4096	0.603	0.2227	31	0.160	0.0520	9.3
	16384	0.603	0.2227	30	0.119	0.0366	19
	32768	0.603	0.2227	30	0.111	0.0331	28
	65536	0.598	0.2214	90	0.073	0.0203	80
	524288	0.592	0.2113	260	0.055	0.0149	413
cpcs360b	1024	0.361	0.1532	15	0.124	0.0516	4.2
$\|E\|$=11-23	2048	0.357	0.1528	18	0.113	0.0468	4.7
	4096	0.356	0.1522	19	0.099	0.0408	6
	8192	0.352	0.1504	23	0.083	0.0330	8
	16384	0.338	0.1423	68	0.064	0.0247	15
	32768	0.337	0.1419	85	0.055	0.0215	24
	65536	0.143	0.3367	120	0.050	0.0192	36
	131072	0.143	0.3366	128	0.043	0.0160	80
	262144	0.143	0.3364	190	0.038	0.0137	130
cpcs422b	1025	0.351	0.1195	16	0.231	0.0740	7
$\|E\|$=6-11	8192	0.337	0.1175	34	0.217	0.0677	18
	16384	0.337	0.1175	34	0.214	0.0665	24
	32768	0.329	0.1081	80	0.203	0.0617	74
	65536	0.317	0.1023	195	0.182	0.0467	150
	131072	0.289	0.0881	1192	0.155	0.0401	1097

Results for $ABdP+$ using two variants of Fractional Packing

Figure 4.8 presents the average bounds interval length obtained by $BdP+$ algorithm and $ABdP+$ with two different approximation schemes, denoted $ABdP$+$FP1$ and $ABdP$+FPM, in **cpcs54**, **cpcs360b**, and **cpcs422b** networks. The charts on the left show average bounds length as a function of m, where $k = 10^m$ determines the maximum Markov blanket size. As we can see from the charts on the left, for the same m (same k), average bounds interval of $BdP+$ is smaller than the bounds interval of BdP+FPM, which, in turn, is smaller than that of BdP+$FP1$, as expected. The charts on the right show average bounds length as a function of time. The computation time of all three schemes grows as m increases. Thus, each point on the right-hand chart corresponds to a particular m. The charts show that $ABdP$+FPM incurs a negligible amount of overhead compared to $ABdP$+$FP1$, but computes considerably tighter bounds. $BdP+$ outperforms $ABdP$+FPM when given enough time. However, for the same k, $BdP+$ requires an order of magnitude more time than $ABdP+$. For example, in cpcs54, for $k = 2^{10}$, $ABdP$+FPM computes bounds in $<$0.01 seconds while $BdP+$ requires a few seconds. Roughly, $BdP+$ requires as much time to compute bounds for $m = 10$ (i.e., $k = 2^{10}$) as $ABdP$+FPM for $k = 2^{17} - 2^{19}$. As a result, $BdP+$ begins to outperform $ABdP$+FPM only after $\approx$2 seconds in cpcs54 and only after 10 seconds in cpcs360b. Hence, in short term, we can obtain more accurate bounds using BdP+FPM.

Comparing the smallest bounds intervals obtained by $BdP+$ and $ABdP$+FPM (for

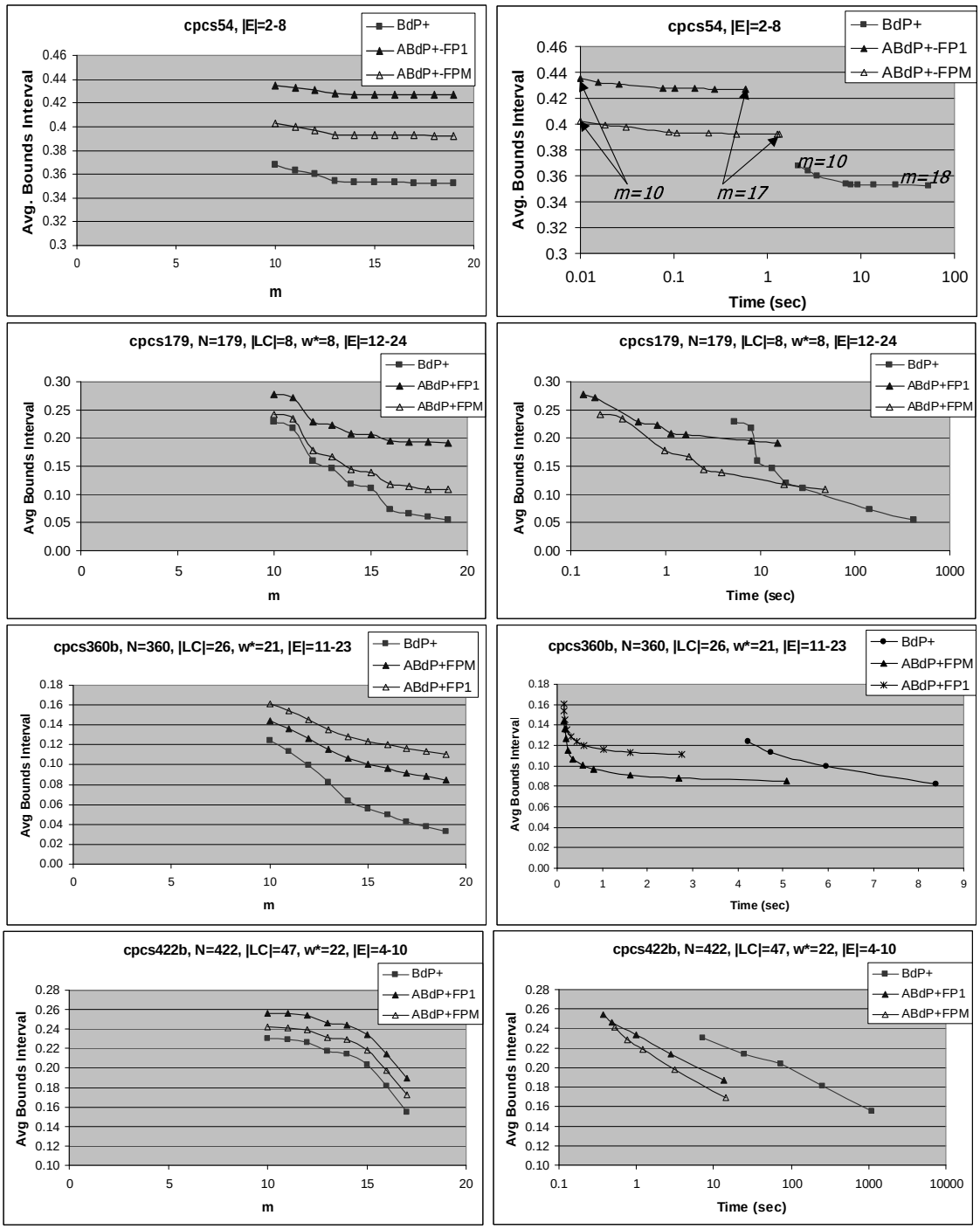


Figure 4.8: Bounds interval length for $BdP+$, $BdP+FP1$, and $BdP+FPM$, averaged over 20 instances of cpcs54, cpcs179, cpcs360b, and cpcs422b, as a function of m and time, where m bounds the maximum Markov table size k=2^m.

maximum m), we see that the difference is small in most networks. In cpcs54, the difference is about 0.04 which is $\approx 1\%$ of the interval length. We observed the largest difference in the case of cpcs360b, where the smallest $BdP+$ bounds interval length is a factor of 2 smaller than $ABdP$+FPM.

Table 4.4: Average bounds interval for $BdP+$, $ABdP$+$FP1$, and $ABdP$+FPM for the maximum value of $k = 2^m$ tried, $m \in [10, 19]$, averaged over 20 instances of each benchmark.

		$BdP+$		$BdP+$-FPM		$BdP+$-$FP1$	
network	m	$\overline{I}$	Time	$\overline{I}$	Time	$\overline{I}$	Time
Alarm	$\geq$10	0.611	4.2	0.611	0.08	0.67	0.02
Barley	$\geq$10	0.231	1.5	0.251	0.05	0.29	0.03
cpcs54	19	0.352	53	0.392	1.3	0.42	0.60
cpcs179	19	0.055	413	0.109	51	0.19	15.3
cpcs360b	19	0.033	270	0.085	5	0.11	2.7
cpcs422b	17	0.155	1027	0.173	9.4	0.19	8.9
Munin3	19	0.255	9	0.258	0.7	0.32	0.2
Munin4	19	0.228	13	0.232	1.0	0.31	0.2

We observe similar results in other networks. Table 4.4 summarizes the average bounds interval results for the maximum k for each benchmark. In **Barley**, $ABdP+$-FPM computes average bounds interval of 0.25 in 0.05 sec, while $BdP+$ computes only slightly smaller average bounds interval of 0.23 in 1.7 sec. In cpcs422b, for $k = 2^{17}$, $\overline{I}_{BdP+-FPM} = 0.17$ and takes 23 seconds to compute while $\overline{I}_{BdP+} = 0.15$ and takes 1025 seconds.

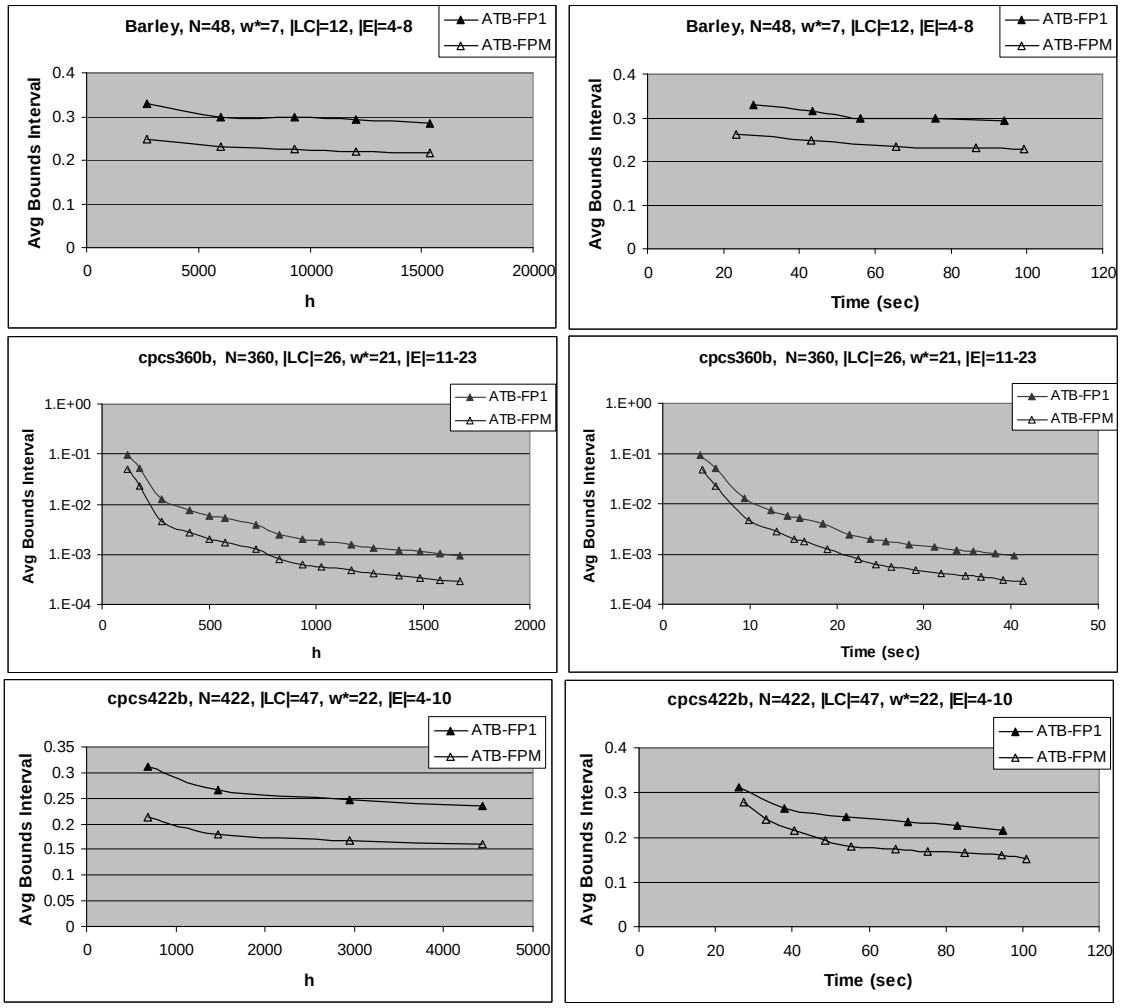


Figure 4.9: Average bounds length for ATB with a bound propagation plugin optimizing LP by fractional packing with 1 (ATB-$FP1$) and many knapsacks (ATB-FPM).

Experimenting with ATB Variants

In Figure 4.9 we compare the performance of ATB framework with two plug-ins, $ABdP$+$FP1$ and $ABdP$+FPM. The resulting algorithms are denoted ATB-$FP1$ and ATB-FPM. We use three representative benchmarks, Barley, cpcs360b, and cpcs422b, which are also the largest of our benchmarks. For each algorithm, the results are averaged over 20 instances of each network. We see that the ATB-FPM line is consistently lower than ATB-$FP1$. Hence, ATB-FPM not only computes tighter bounds, but it is also timewise more efficient than ATB-$FP1$. We also observe that the difference between the results of the two schemes becomes larger with time. This is more noticeable in the case of cpcs360b (Figure 4.9, middle) and cpcs422b (Figure 4.9, bottom). This observation points out that the tightness of the bounds on the unexplored tuples strongly affects the ATB convergence speed.

Results for ATB^w, ATB, $BdP+$ and $BBdP+$

This section provides our main evaluation of the proposed ATB scheme against bound propagation. We use the superior variants of those schemes. Namely, we have showed that $BdP+$ always outperfroms BdP and that $ABdP$+FPM is the superior variant of approximate bound propagation, used either stand-alone or as a plug-in in ATB framework. We compare ATB and ATB^w with $ABdP$+FPM plug-in against $BdP+$. In addition, we show the results of boosted bound propagation $BBdP+$ using ATB results as input.

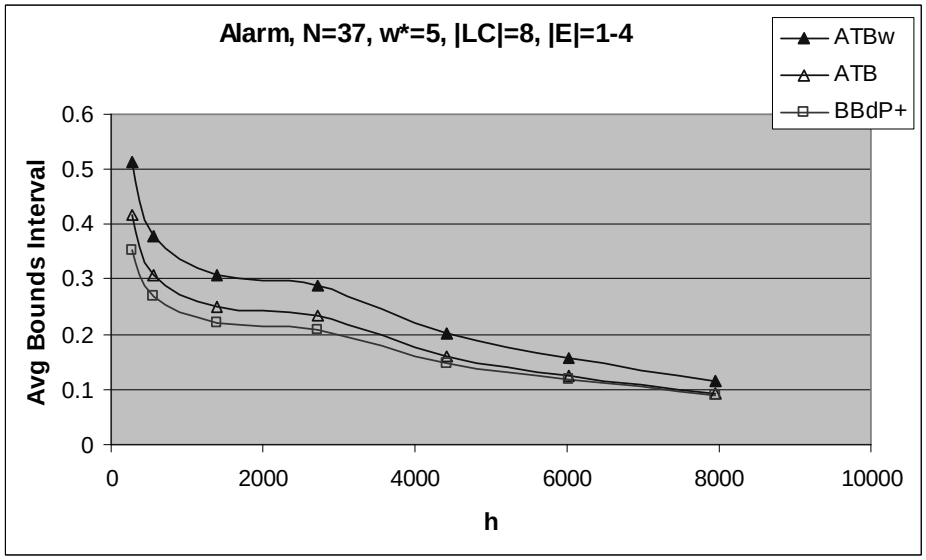
For reference, we also computed the minimum length of bounded conditioning bounds

interval by plugging into the framework the brute force 0 lower bounds and prior $P(c)$ upper bounds. The computed bounds interval length remained close to 0.75 for Munin benchmarks and 0.95 for the others; hence, those results are omitted in the remainder of the section.

We summarize the results for each benchmarks in a tabular format and charts. The tables report the average bounds interval $\overline{I}$, average error Δ, computation time (in seconds), and percent of probability of evidence $P(e)$ covered by the fully-instantiated cutset tuples generated by ATB^w, ATB, and $BBdP+$ algorithms as a function of h. Since $BdP+$ results do not depend on h, they are not included in the tables here. The results for $BdP+$ were reported in Table 4.3. We highlight in bold face the first ATB data point where the average bounds interval is as good or better than $BdP+$. We use charts to show the convergence of bounds interval length as a function of h and time.

Results for Alarm network. The Alarm network has $N = 37$ nodes and a loop-cutset of size $|LC| = 5$ with the cutset state-space of size $|LC| = 108$. The exact posterior marginals in the Alarm network can be obtained using bucket elimination or exact cutset conditioning in less than a second. We present the results in order to relate to the bounds reported previously for bounded conditioning [56] and for bound propagation [76]. The results are reported in Table 4.5 and Figure 4.10.

As expected, the average bounds interval generated by ATB^w and ATB decreases as h increases, demonstrating the any-time property of ATB with respect to h. For a fixed



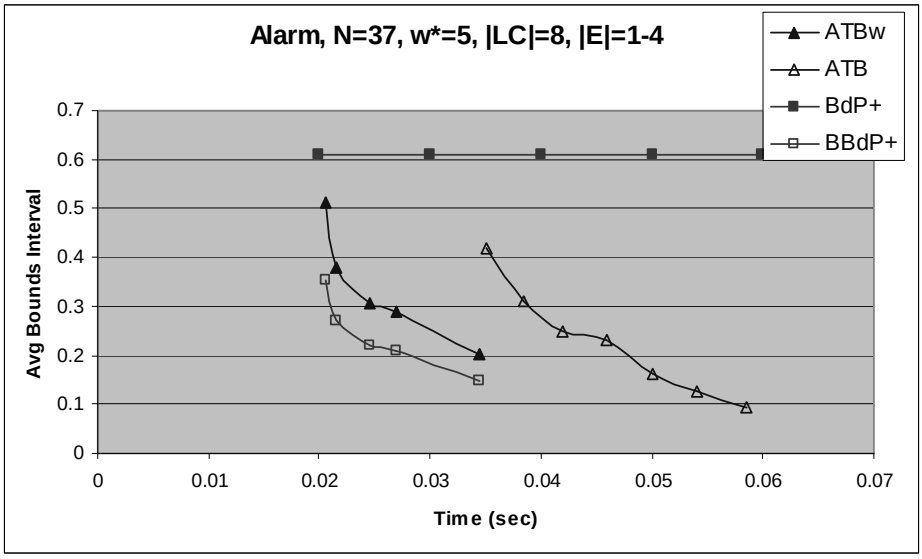


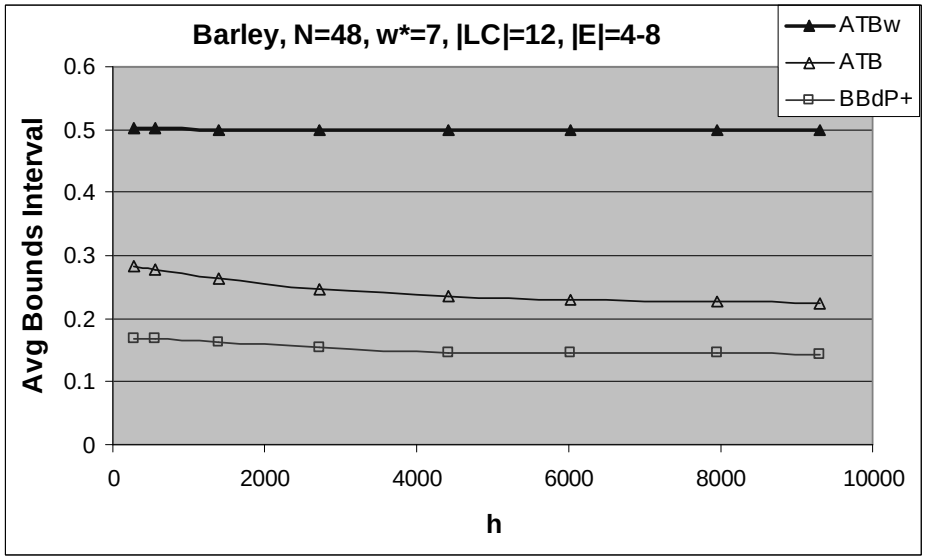
Figure 4.10: Results for Alarm network as a function of h (top) and a function of time (bottom), averaged over 20 instances. Exact inference using bucket elimination is 30 seconds. Exact inference using cutset conditioning is > 22 hours.

Table 4.5: Average error Δ, bounds interval $\overline{I}$, and computation time t for $BdP+$, ATB, and $BBdP+$ over 20 instances of Alarm network as function of h.

Alarm, N=37, w^*=5, $\|LC\|$=8, $\|D_{LC}\|$=108, $\|E\|$=1-4										
		ATB^w			ATB			$BBdP+$		
h	%P(e)	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*
25	86	0.51	0.16	*0.021*	**0.41**	0.12	**0.038**	0.35	0.10	*3.4*
34	93	0.38	0.12	*0.022*	0.31	0.09	*0.039*	0.27	0.08	*2.3*
40	96	0.31	0.10	*0.025*	0.25	0.07	*0.044*	0.22	0.06	*2.1*
48	97	0.20	0.09	*0.035*	0.24	0.05	*0.051*	0.15	0.04	*1.5*
50	98	0.16	0.06	*0.036*	0.16	0.04	*0.052*	0.12	0.03	*1.2*
54	99	0.12	0.05	*0.044*	0.13	0.03	*0.059*	0.09	0.02	*0.86*

h, ATB^w bounds interval is always larger than that of ATB, as predicted. However, the picture is different time-wise. For example, ATB^w computes bounds interval of $\overline{I}_{ATB^w} = 0.12$ within 0.044 seconds ($h = 54$), while ATB only computes $\overline{I}_{ATB} = 0.25$ within the same time ($h = 40$).

Both any-time schemes perform better than $BdP+$. Recall that for $k \geq 65536$ $BdP+$ obtained an average interval length of 0.65 within 4 seconds (Table 4.3). Both ATB^w and ATB compute more accurate bounds starting with the first data point of $h = 25$. In the second row of Table 4.5, $\overline{I}_{ATB^w} = 0.41$ and $\overline{I}_{ATB} = 0.51$, and require respectively 0.021 and 0.038 seconds, an order of magnitude less than $BdP+$. Using ATB bounds as input to bound propagation in $BBdP+$ considerably improves the ATB results. For the same $h = 25$, $\overline{I}_{BBdP+} = 0.35$. However, $BBdP+$ is not efficient time wise compared to ATB since the bound propagation time is considerably larger than that of ATB. Yet, since $BBdP+$ computation time on average is less than $BdP+$, it outperforms $BdP+$ starting with $h = 25$, similar to ATB and ATB^w.



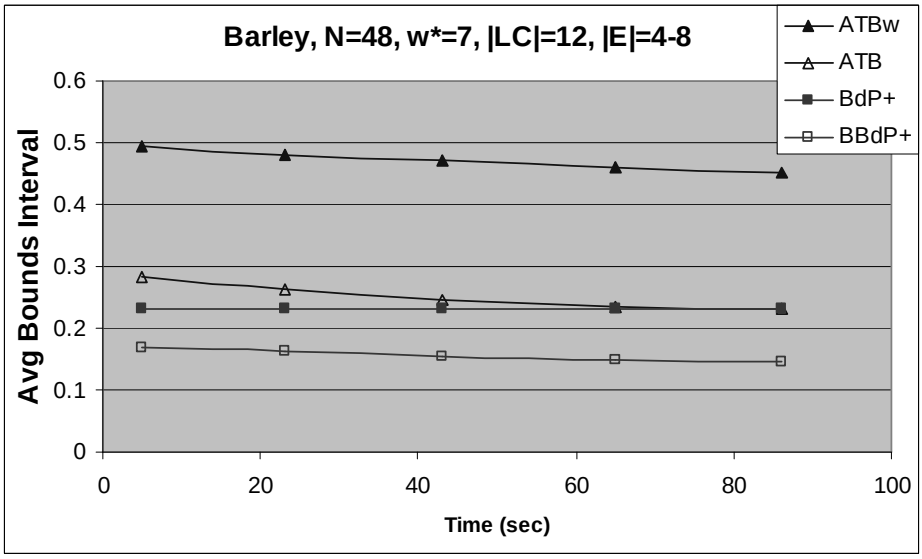


Figure 4.11: Results for Barley network as a function of h (top) and a function of time (bottom), averaged over 20 instances. Exact inference using bucket elimination is 30 seconds. Exact inference using cutset conditioning is > 22 hours.

Table 4.6: Results for Barley network.

Barley, N=48, w^*=7, $\|LC\|$=12, $\|D_{LC}\|$ >2E+6, $\|E\|$=4-8										
		ATB^w			ATB			BBdP+		
h	%P(e)	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*
278	0.005	0.501	0.194	*0.4*	0.282	0.099	*5*	0.168	0.047	*7*
562	0.01	0.501	0.194	*0.8*	0.279	0.097	*9*	0.167	0.047	*10*
1394	0.03	0.500	0.194	*2.0*	0.263	0.090	*23*	0.162	0.045	*25*
2722	0.06	0.500	0.194	*4.0*	0.247	0.084	*43*	0.154	0.042	*45*
4429	0.14	0.500	0.193	*6.3*	0.235	0.079	*65*	0.147	0.040	*67*
6016	0.22	0.499	0.193	*8.6*	**0.230**	0.078	**86**	0.145	0.040	*88*
7950	0.33	0.499	0.193	*11.0*	0.228	0.077	*99*	0.145	0.040	*101*
9297	0.40	0.499	0.193	*12.8*	0.224	0.075	*111*	0.143	0.039	*113*
12478	0.52	0.498	0.193	*17.3*	0.219	0.073	*139*	0.142	0.038	*141*

Results for Barley network. We applied the bounding algorithms to 20 instances of Barley network with different evidence picked at random among the input nodes as defined in [70]. We compare the performance of different bounding schemes within 100 seconds time interval. The results are reported in Table 4.6 and Figure 4.11. The top chart shows the convergence of ATB^w, ATB, and $BBdP+$ bounds interval with h (since $BBdP+$ bounds are never worse than ATB, the bounds converge with h). The bottom chart shows the average interval of ATB^w, ATB, $BdP+$, and $BBdP+$ as a function of time. We see that $BdP+$ converges quickly yielding an average bounds length of 0.23 in less than 2 seconds but does not improve any more with time. ATB^w, ATB, and $BBdP+$ clearly improve as h increases. However, the convergence is slow with h and with time. For example, average ATB^w bounds interval length remains close to 0.5 while ATB bounds interval decreases from 0.28, obtained in 5 seconds, to 0.22 after 139 seconds. It takes ATB about 86 seconds to achieve the same accuracy as $BdP+$. Overall, the winning scheme time-wise is $BBdP+$. The bound propagation time of $BBdP+$ is negligible compared to

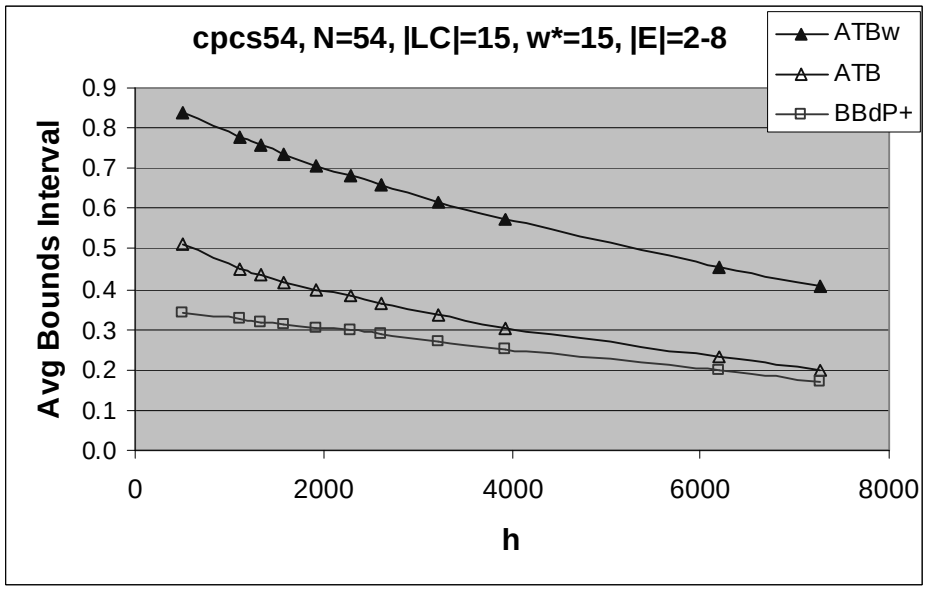
ATB computation time. Using ATB results to jump-start bound propagation substantially improves ATB bounds, computing $\overline{I} = 0.168$ in 7 seconds and $\overline{I} = 0.142$ in 141 seconds.

Table 4.7: Results for cpcs54.

cpcs54, N=54, $\|LC\|$=15, w^*=15, $\|D_{LC}\|$=32678, $\|E\|$=2-8										
		ATB^w			ATB			BBdP+		
h	%P(e)	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*
513	10	0.84	0.062	*0.4*	0.51	0.027	*0.9*	0.34	0.011	*3.1*
1114	19	0.78	0.055	*0.7*	0.45	0.023	*1.5*	0.32	0.010	*3.1*
1343	25	0.76	0.054	*0.9*	0.44	0.023	*1.7*	0.32	0.010	*3.3*
1581	29	0.74	0.052	*1.0*	0.42	0.021	*1.9*	0.31	0.009	*3.4*
1933	34	0.71	0.049	*1.3*	0.40	0.020	*2.2*	0.30	0.009	*3.6*
2290	40	0.68	0.047	*1.5*	0.38	0.019	*2.4*	0.30	0.008	*3.9*
2609	46	0.66	0.045	*1.8*	0.37	0.018	*2.7*	0.29	0.007	*4.0*
3219	53	0.62	0.041	*2.1*	**0.34**	0.016	**3.2**	0.27	0.007	*4.5*
3926	59	0.57	0.038	*2.7*	0.31	0.014	*3.8*	0.25	0.006	*5.2*
6199	63	0.46	0.029	*4.5*	0.23	0.010	*5.9*	0.20	0.006	*6.6*
7274	68	0.41	0.026	*5.4*	0.20	0.008	*6.9*	0.17	0.006	*7.3*

Results for CPCS networks. Results for cpcs54 are given in Table 4.7 and Figure 4.12, for cpcs179 in Table 4.8 and Figure 4.13, for cpcs360b in Table 4.9 and Figure 4.14, for cpcs422b in Table 4.10 and Figure 4.15.

cpcs54. 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 $BdP+$ scheme obtains the bounds interval of 0.35 with the maximum Markov table size of $k = 4096$ and hardly changes at all as k increases up to the memory limit (see Figure 4.3). ATB outperforms $BdP+$ within 3 seconds. ATB also outperforms ATB^w by a wide margin with respect to h (Figure 4.12, top) and time-



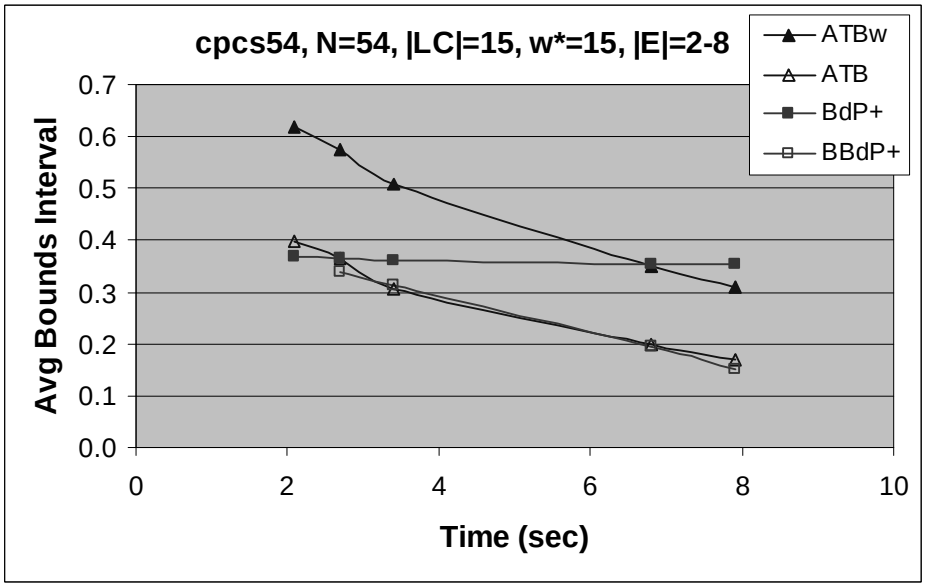


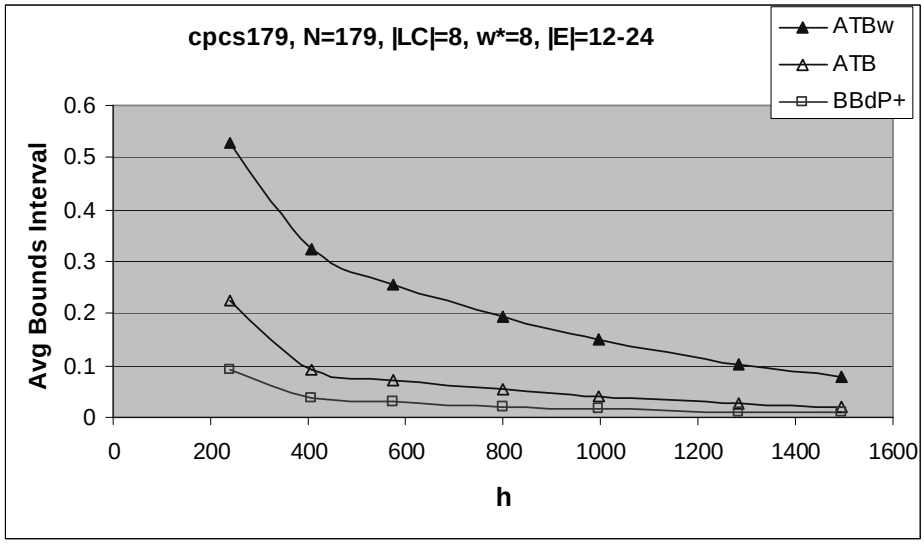
Figure 4.12: Results for cpcs54 network as a function of h (top) and a function of time (bottom), averaged over 20 instances. Exact inference using bucket elimination is 1 second. Exact inference using cutset conditioning is 15 seconds.

wise (Figure 4.12, bottom). $BBdP+$ improves the ATB result as a function of h, as shown in Table 4.7 and Figure 4.12, top. For example, for $h = 513$, $I_{ATB} = 0.51$ while $\overline{I}_{BBdP+} = 0.34$. However, the improvement is not enough to compensate for additional computation time as we see in Figure 4.12, bottom.

Table 4.8: Results for cpcs179.

cpcs179, N=179, w^*=8, $\|LC\|$=8, $\|D_{LC}\|$=49152, $\|E\|$=12-24										
		ATB^w			ATB			BBdP+		
h	%P(e)	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*
242	70	0.533	0.205	*2.5*	**0.224**	0.067	**4**	0.092	0.029	*11*
334	75	0.392	0.151	*3.5*	0.123	0.033	*6*	0.054	0.016	*13*
406	78	0.323	0.124	*3.8*	0.092	0.024	*7*	0.037	0.010	*13*
483	80	0.286	0.110	*4.3*	0.080	0.021	*8*	0.034	0.009	*15*
574	82	0.256	0.099	*4.8*	0.070	0.018	*9*	0.029	0.008	*15*
683	84	0.219	0.084	*5.4*	0.061	0.015	*10*	0.024	0.007	*17*
801	85	0.195	0.075	*5.8*	0.054	0.014	*10*	0.022	0.006	*17*
908	86	0.167	0.064	*6.3*	0.044	0.011	*11*	0.019	0.005	*18*
996	87	0.151	0.058	*6.6*	0.040	0.010	*12*	0.017	0.005	*18*
1130	88	0.124	0.048	*6.8*	0.032	0.008	*12*	0.014	0.004	*19*
1285	88	0.104	0.040	*7.8*	0.026	0.006	*13*	0.012	0.003	*20*
1493	89	0.078	0.030	*9.0*	0.019	0.004	*15*	0.009	0.003	*21*
1669	90	0.064	0.024	*9.7*	0.015	0.003	*16*	0.007	0.002	*22*

cpcs179. The results for cpcs179 network are shown in Table 4.8 and Figure 4.13. The results for ATB, ATB^w, and $BBdP+$ as a function of time are similar and speak for themselves. The first data point for $BdP+$ in Figure 4.13, corresponding to $k = 1024$, is $\overline{I} = 0.23$ and takes 5.3 seconds to compute. As k increases, the average bounds interval of $BdP+$ decreases slowly. Within 20 seconds, the best result of $BdP+$ is $\overline{I} = 0.12$. ATB and ATB^w computer tighter bounds than $BdP+$ after the first 5 seconds. Comparing the two any-time schemes, we see that ATB's curve is considerably lower than ATB^w



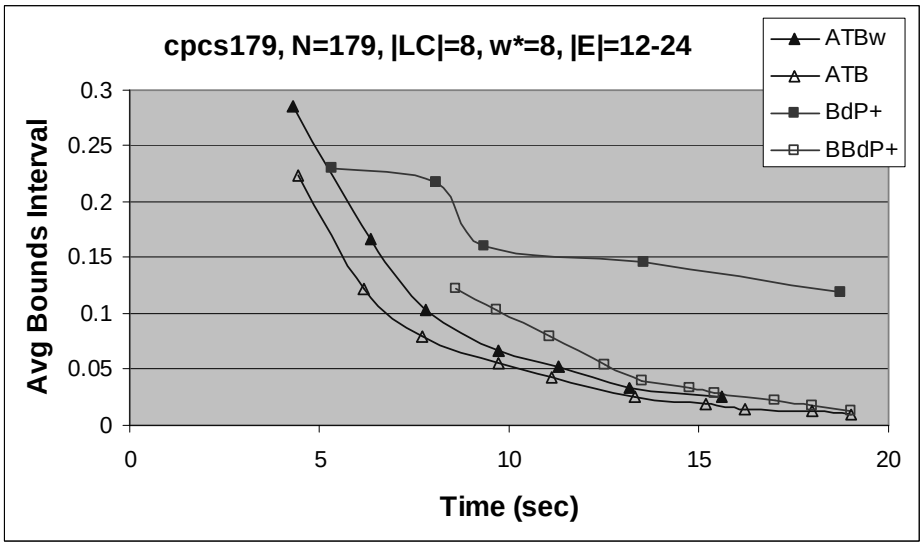


Figure 4.13: Results for cpcs179 network as a function of h (top) and a function of time (bottom), averaged over 20 instances. Exact inference using bucket elimination is 2 seconds. Exact inference using cutset conditioning is 37 seconds.

at first. However, at the end of 20 second interval, their curves becomes very close. The performance of $BBdP+$ as a function of time is lagging behind both ATB^w and ATB. The average $BBdP+$ computation time per network instance is 8 seconds. Subsequently, although $BBdP+$ improves the result of ATB as shown in the Figure 4.13, top, $BBdP+$ is not cost-effective here.

Table 4.9: Results for cpcs360b.

cpcs360b, N=360, $w^* = 21$, $\|LC\| = 26$, $\|E\|$=11-23										
		ATB^w			ATB			BBdP+		
h	%P(e)	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*
121	83	0.235	0.098	*3*	**0.0486**	1.6E-2	**5**	0.0274	1.0E-2	*7*
282	92	0.086	0.035	*7*	0.0046	9.0E-4	*10*	0.0032	8.5E-4	*12*
409	95	0.057	0.023	*9*	0.0028	5.6E-4	*13*	0.0020	5.3E-4	*15*
501	96	0.044	0.018	*10*	0.0020	3.6E-4	*15*	0.0014	3.5E-4	*17*
722	97	0.029	0.012	*13*	0.0012	2.4E-4	*19*	0.0009	2.3E-4	*21*
831	98	0.024	0.010	*16*	0.0008	1.2E-4	*22*	0.0006	1.2E-4	*25*
938	98	0.020	0.008	*18*	0.0006	8.4E-5	*25*	0.0004	7.8E-5	*27*
1027	98	0.018	0.007	*19*	0.0006	8.1E-5	*26*	0.0004	7.5E-5	*29*
1168	98	0.015	0.006	*22*	0.0005	7.5E-5	*29*	0.0004	6.9E-5	*31*
1271	99	0.013	0.005	*24*	0.0004	6.7E-5	*32*	0.0003	6.1E-5	*34*
1388	99	0.012	0.005	*26*	0.0004	5.9E-5	*35*	0.0003	5.4E-5	*37*
1486	99	0.011	0.004	*28*	0.0003	5.8E-5	*37*	0.0003	5.3E-5	*39*
1582	99	0.010	0.004	*30*	0.0003	5.3E-5	*39*	0.0002	4.8E-5	*41*
1674	99	0.009	0.004	*32*	0.0003	5.0E-5	*41*	0.0002	4.6E-5	*43*
1757	99	0.008	0.003	*34*	0.0003	4.7E-5	*43*	0.0002	4.4E-5	*46*

cpcs360b loop-cutset has 2^{26} cutset tuples, prohibitively many for complete enumeration. The exact computation time for cpcs360b by bucket elimination is about 20 minutes. We experimented with 20 instances of the network with the number of evidence nodes ranging from 11 to 23. We have compared the performance of all four bounding algorithms within 40 seconds, that is only a fraction of time necessary to compute posterior marginals

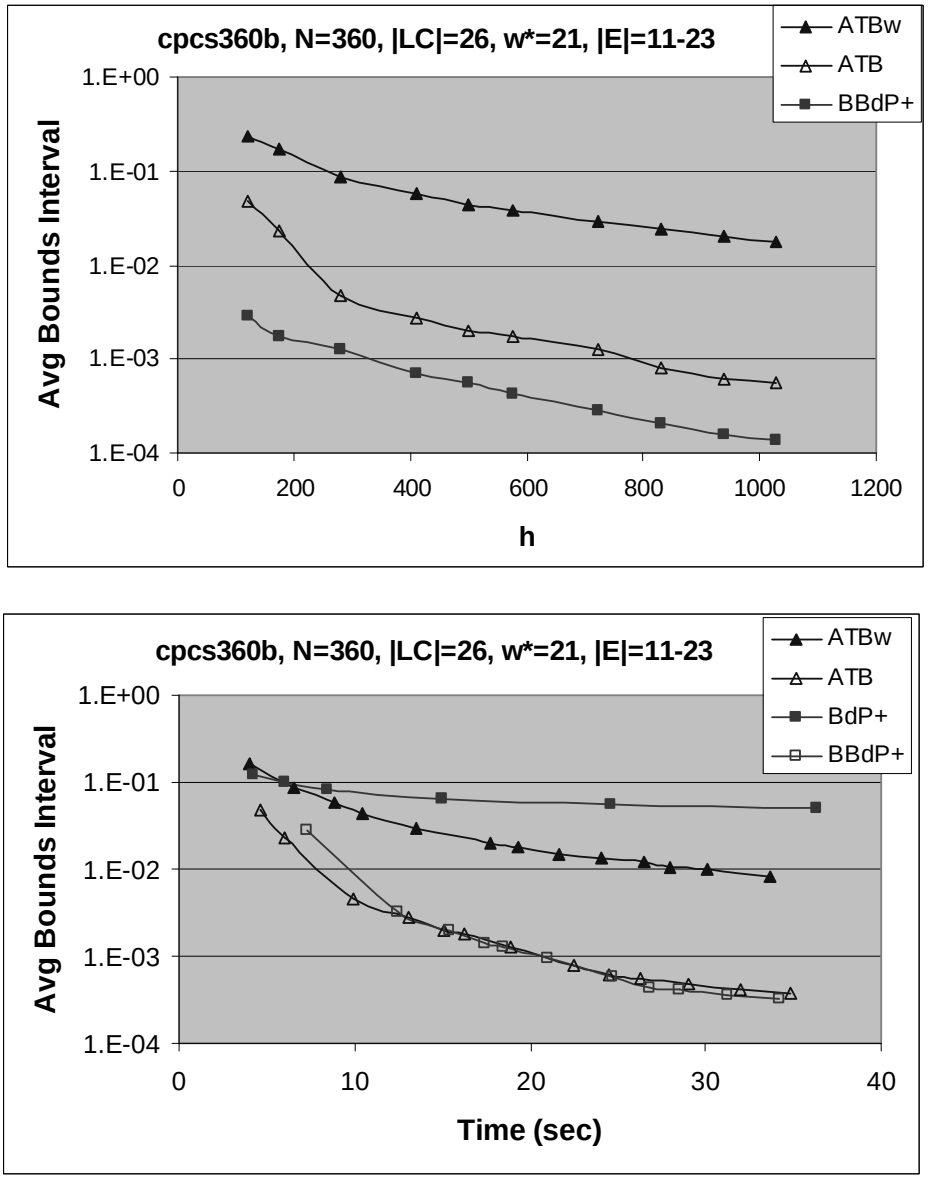


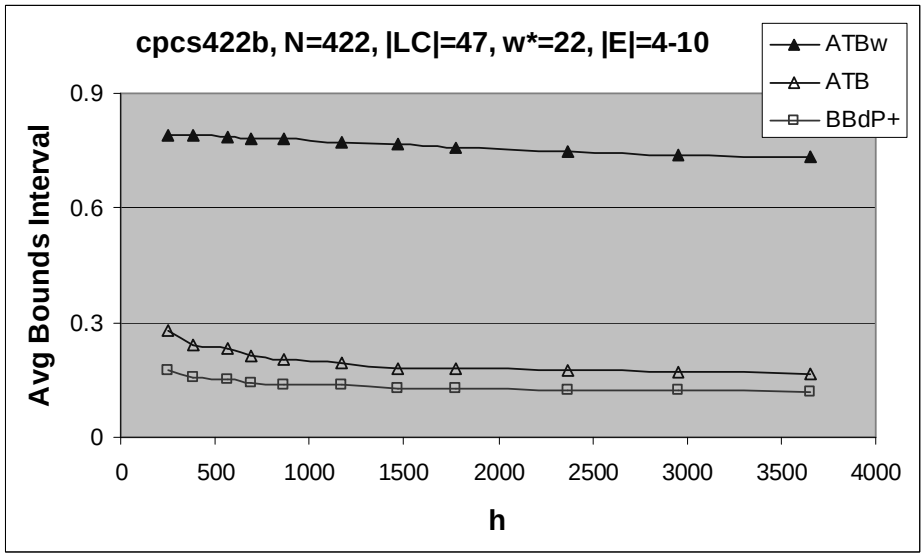
Figure 4.14: Results for cpcs360b as a function of h (top) averaged over 20 instances. Evidence is chosen randomly among leaf nodes only. Exact inference using bucket elimination is 20 minutes. Exact inference using cutset conditioning is > 8 hours.

exactly. The results for cpcs360b are summarized in Table 4.9 and Figure 4.14.

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 $\overline{I} = 0.24$ for $h = 121$ (3 seconds) to $\overline{I} = 0.008$ for $h = 1757$ (33 seconds). However, ATB and, consequently, $BBdP+$ converge considerably faster. In 30 seconds, their average bounds interval decreases to 0.003. Although the overhead of bound propagation time in $BBdP+$ is relatively small, ≈2 seconds, the performance of ATB and $BBdP+$ is very similar timewise. As we see, ATB^w, ATB, and $BBdP+$ schemes outperformed $BdP+$. Larkin's algorithm [73], 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 ≈ 0.005. However, the comparison is not on the same instances since the evidence nodes are not the same.

cpcs422b. The result for the fourth and the largest cpcs network, cpcs422b, are shown in Table 4.10 and Figure 4.15. Cpcs422b is challenging for any inference scheme as it has large induced width of $w^* = 22$ and 2^{47} loop-cutset tuples. Exact inference by bucket elimination requires about 50 minutes. The estimated cutset conditioning time is over $2E + 9$ hours.

From Table 4.10, we see that ATB outperforms ATB^w by a wide margin and $BBdP+$ improves a little over ATB. ATB outperforms ATB^w as a function of h and time as shown



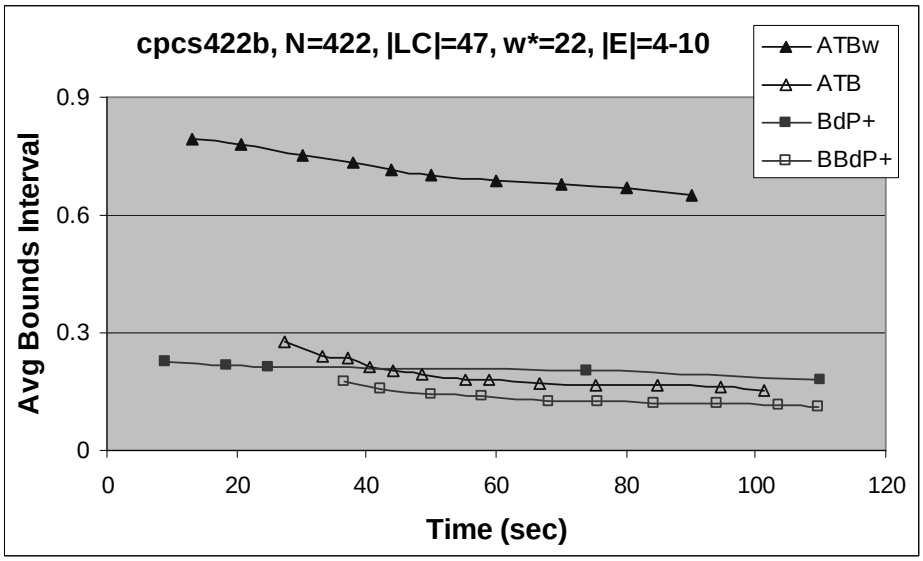


Figure 4.15: cpcs422b average bounds length as a function of h (top) and time (bottom), averaged over 20 instances. Exact inference using bucket elimination is 50 minutes. Exact inference using cutset conditioning is $> 2E + 9$ hours.

Table 4.10: Results for cpcs422b.

cpcs422b, $\|E\|$=6-11										
		ATB^w			ATB			BBdP+		
h	%P(e)	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*
253	1.7	0.79	0.326	*14*	0.28	0.090	*27*	0.17	0.056	*37*
381	2.0	0.79	0.325	*16*	0.24	0.076	*33*	0.16	0.050	*42*
563	2.6	0.79	0.323	*17*	0.23	0.073	*37*	0.15	0.049	*47*
688	2.9	0.78	0.322	*19*	0.21	0.066	*40*	0.14	0.046	*50*
867	3.4	0.78	0.321	*21*	**0.20**	0.062	**44**	0.14	0.044	*53*
1171	4.5	0.77	0.318	*22*	0.19	0.059	*49*	0.14	0.043	*58*
1472	5.4	0.77	0.315	*26*	0.18	0.054	*55*	0.13	0.040	*64*
1779	6.5	0.76	0.312	*27*	0.18	0.054	*59*	0.13	0.040	*68*
2368	8.0	0.75	0.307	*30*	0.17	0.052	*67*	0.12	0.039	*76*
2954	9.5	0.74	0.303	*34*	0.17	0.050	*75*	0.12	0.038	*84*
3654	10.8	0.73	0.300	*38*	0.16	0.049	*85*	0.12	0.038	*94*
4429	12.2	0.72	0.296	*42*	0.16	0.047	*95*	0.12	0.037	*104*
5120	13.7	0.72	0.292	*44*	0.15	0.044	*101*	0.11	0.035	*110*

in Figure 4.15. It outperforms $BdP+$ after 40 seconds. Overall, $BBdP+$ is the best algorithm. The $BBdP+$ result for h=5120, $\overline{I} = 0.1124$, is the best of all algorithms for the 2 minute time interval.

Larkin [73] reports an average bounds interval of 0.15, obtained within 30 seconds. ATB and $BdP+$ obtain comparable results. Within 30 seconds, both ATB ($h = 379$ in Table 4.10) and $BdP+$ ($k = 16384$ in Table 4.3) compute average bounds interval of length ≈ 0.21. ATB's bounds interval is reduced to 0.15 after 100 seconds (see Table 4.10, $h = 4598$). Note also that $BBdP+$ computes $\overline{I} = 0.15$ in 47 seconds.

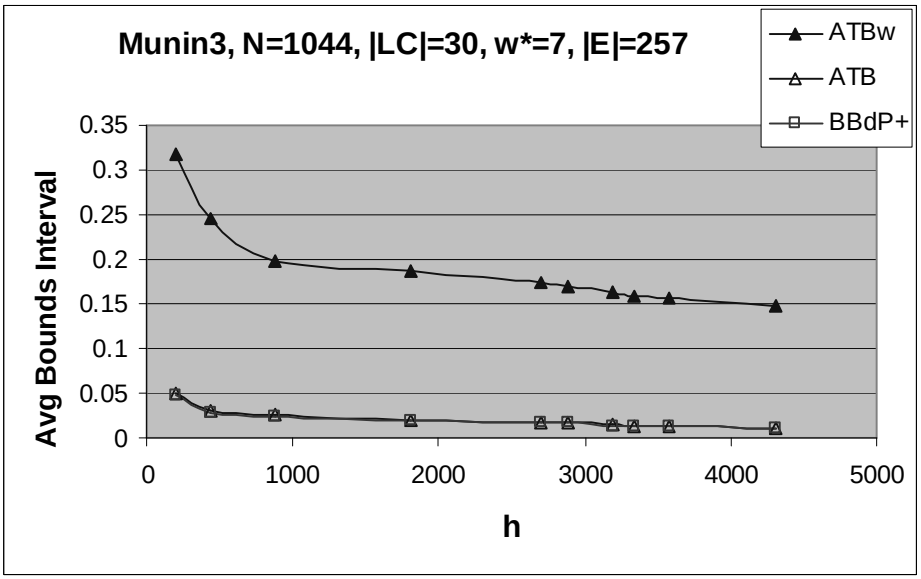
Munin's benchmarks. Our last two benchmarks are Munin3 and Munin4. The evidence in each network instance has been pre-defined. Both networks are large, with

Table 4.11: Results for Munin3.

Munin3, N=1044, w^*=7, $\|LC\|$=30, $\|E\|$=257										
		ATB^w			ATB			BBdP+		
h	%P(e)	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*
196	64	0.32	0.133	*4*	**0.050**	0.020	**8**	0.048	0.020	*16*
441	72	0.25	0.100	*6*	0.030	0.011	*12*	0.029	0.012	*20*
882	78	0.20	0.078	*10*	0.025	0.009	*18*	0.025	0.009	*26*
1813	79	0.19	0.073	*19*	0.020	0.007	*32*	0.019	0.007	*40*
2695	80	0.17	0.068	*28*	0.018	0.006	*46*	0.017	0.007	*54*
2891	81	0.17	0.065	*30*	0.017	0.006	*49*	0.016	0.006	*57*
3185	82	0.16	0.062	*34*	0.014	0.005	*54*	0.014	0.005	*62*
3577	82	0.16	0.060	*44*	0.013	0.004	*68*	0.012	0.004	*76*
4312	83	0.15	0.056	*52*	0.011	0.004	*80*	0.010	0.004	*88*

Table 4.12: Results for Munin4.

Munin4, N=1041, w^*=8, $\|LC\|$=49, $\|E\|$=235										
		ATB^w			ATB			BBdP+		
h	%P(e)	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*	$\overline{I}$	Δ	*time*
245	1	0.87	0.396	*3*	0.39	0.16	*14*	0.24	0.102	*21*
441	7	0.82	0.372	*4*	0.32	0.13	*17*	0.22	0.095	*24*
1029	11	0.78	0.352	*7*	0.28	0.12	*34*	0.21	0.089	*44*
2058	17	0.73	0.329	*12*	**0.25**	0.11	**54**	0.19	0.082	*65*
3087	20	0.70	0.316	*17*	0.22	0.11	*83*	0.18	0.077	*91*
5194	24	0.67	0.301	*27*	0.21	0.09	*134*	0.17	0.072	*145*



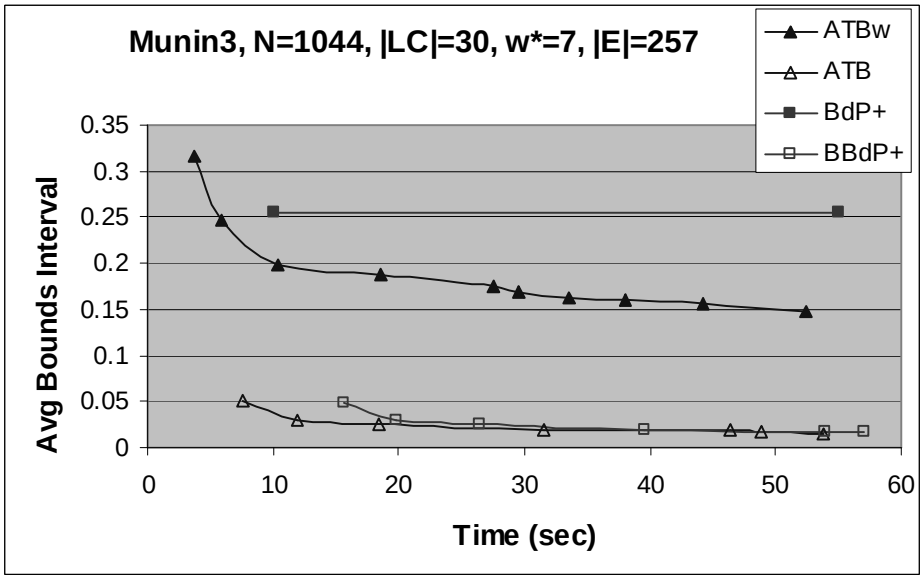


Figure 4.16: Munin3 average bounds length as a function of h (top) and time (bottom). Exact inference using bucket elimination is 8 seconds. Exact inference using cutset conditioning is > 1700 hours.

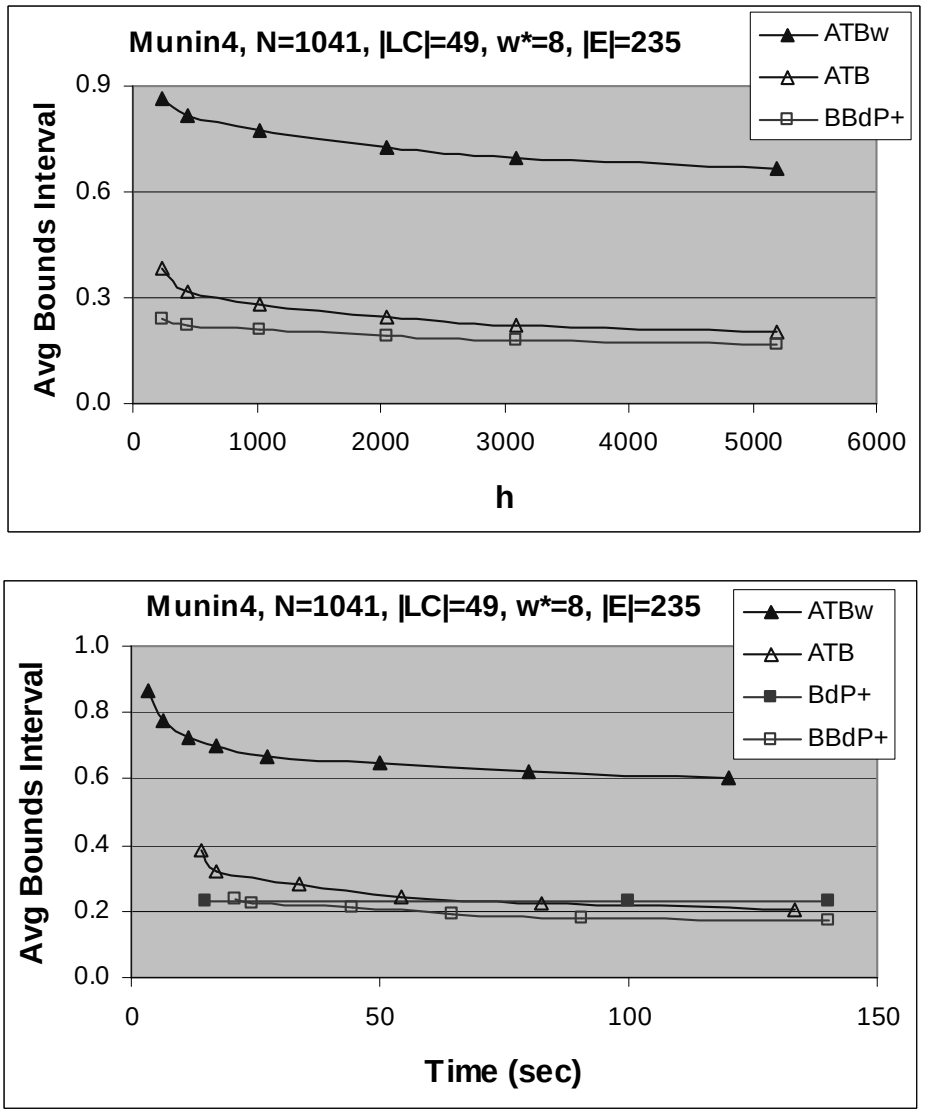


Figure 4.17: Munin4 average bounds length as a function of h (top) and time (bottom). Exact inference using bucket elimination is 70 seconds. Exact inference using cutset conditioning is $> 1E + 8$ hours.

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 4.6.4, the exact computation time of Munin3 is 8 seconds and Munin4 is 70 seconds. The empirical results for each network are summarized in Tables 4.11 and 4.12 in Figures 4.16 and 4.17.

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 4.16, top, and Figure 4.17, 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 length is an order of magnitude smaller than ATB^w for Munin3 and a factor of 2 smaller in Munin4. $BBdP+$ improvement over ATB is very small in Munin3. Consequently, the two curves are very close and they are hard to distinguish on the charts. The improvement is more noticeable in Munin4.

Now, we look at the performance of ATB^w, ATB, $BBdP+$, and $BdP+$ as a function of time. $BdP+$ algorithm computes bounds interval of 0.25 for Munin3 and 0.23 for Munin4 within 10 and 15 seconds respectively and does not improve any more. In Munin3, ATB outperforms $BdP+$ by a wide margin yielding a bounds interval of 0.05 in 8 seconds. In Munin4, the loop-cutset size is larger and, thus, convergence of ATB is slower; ATB outperforms $BdP+$ after ≈70 seconds. $BBdP+$ performance is very close to ATB in Munin3 after 20 seconds. In Munin4, Figure 4.17, we observe that $BBdP$ consistently

outperforms ATB.

4.6.6 Discussion

We demonstrated that in all benchmarks, except Alarm network, ATB converges faster with time than ATB^w. Hence, it is usually cost-effective to invest time into computing tighter bounds on $P(x, c_{1:q}, e)$. The fast convergence of ATB^w bounds in the case of Alarm network may be attributed to the small number of loop-cutset tuples; there are only 108 tuples to explore.

Comparing the average bounds length to the average absolute error in the estimator, we observe, across all benchmarks, that the error is usually smaller than half of the interval length. This indicates that bounds are usually well centered around posterior marginals. In most cases, the relative error in ATB^w (absolute error relative to the average bounds interval) appears larger than in ATB which indicates that computing tighter bounds on $P(x, c_{1:q}, e)$ produces bounds on posterior marginals that are not only tighter but also better centered around the exact values.

4.7 Conclusions and Future Work

In this chapter we presented an anytime framework for bounding the posterior beliefs of every variable. The scheme is parametrized by a fixed number of cutset tuples h over which it applies exact computation using cutset conditioning. We used a modified Gibbs sampling scheme to find h high probability tuples. We developed expressions to bound

the rest of the probability mass. Those expressions facilitate the use of any off-the-shelve bounding algorithms (worst-case we can plug in 0 and priors). If we denote this scheme by $ATB_h(\mathcal{A})$ (using h cutset tuples and plug-in algorithm $\mathcal{A}$), $ATB_h(\mathcal{A})$ can be viewed as a boosting scheme for $\mathcal{A}$.

ATB can make any existing bounding scheme anytime and improve it. In this paper we focused on a specific algorithm, $\mathcal{A}$=$ABdP+$, which is a variant of bound propagation [76]. $ABdP+$ is based on the improved bound propagation scheme, $BdP+$, that exploits the directionality of the network to its advantage, restricting variable's Markov blanket to its relevant subnetwork. Consequently, the $BdP+$ computation time is reduced and tighter bounds are obtained for the same input parameters which we confirmed empirically on several benchmarks. However, using $BdP+$ as a plug-in in the proposed any-time framework was infeasible time-wise. Instead, we chose algorithm $ABdP+$ which incorporates the improvements in $BdP+$ scheme but uses an approximate greedy solution to the LP optimization problems rather than the simplex solver. Although the resulting bounds are less accurate, we reduce computation time by more than an order of magnitude. Since the framework focuses on enumerating high-probability cutset-tuples and only uses the plug-in to bound the remaining probability mass of $P(e)$, we can compenstate for the loss of accuracy due to using $ABdP+$ plug-in instead of $BdP+$ by enumarating more cutset tuples.

We showed that the any-time framework with $ABdP+$ plug-in outperformed $BdP+$ in all benchmarks after exploring a few hundred to a few thousand cutset tuples. In larger networks, such as cpcs360b and cpcs422b, ATB computed a small bounds interval in a

fraction of the time needed to compute exact posterior marginals by bucket elimination or cutset conditioning.

Finally, we showed that for iterative bounding algorithms, such as $BdP+$, another boosting step is feasible by taking the results of ATB and plugging them back into $BdP+$. The resulting algorithm $BBdP+$ improves results further over ATB as we showed.

The main improvements in ATB framework compared to bounded conditioning are 1) the tighter bound formulation and 2) tighter bounds on the probability mass of the unexplored cutset tuples. ATB approach is also related to the algorithm for estimating and bounding posterior probabilities proposed by David Poole in [98]. In fact, the bounds expression in [98] is similar to ATB bounds derived in this paper where the summation over the truncated cutset tuples in ATB expressions represents the Poole's bounding function. The main difference is that in [98] the enumeration is over the network tuples as opposed to a cutset. Consequently, the approach is more similar to search than conditioning. Also, in [98], the bounding function is updated via conflict counting, while ATB refines the bounding function and bounds probabilities on individual cutset tuples.

Another approach for computing bounds was proposed in [99] where "context-specific" bounds were obtained by 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 [73], 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 problem to find a table decomposition that minimizes the error. A specialized *large deviation bounds* approach for layered networks is proposed in [65, 64] and an elaborate bounding scheme with non-linear objective function was proposed in [87].

Of all the methods mentioned, only bounded conditioning and ATB offer any-time properties, namely, improve the bounds given more time to explore more cutset instances and converge to exact posterior marginals. 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.

Chapter 5

Conclusions

When it is not in our power to determine what is true,
we ought to follow what is most probable.
-Rene Descartes

In this final chapter, we conclude the dissertation with a discussion of contributions and possible directions for future work.

Our research addresses the problem of answering Bayesian queries in networks whose induced width is so large that using exact algorithms is infeasible. In such cases, we resort to using approximation and bounding methods. In this dissertation, we showed how existing approximation and bounding algorithms can be improved by combining them with exact inference.

5.1 Contributions

Our main contributions to AI research are three novel schemes for automated reasoning in Bayesian networks:

- *Cutset sampling*. In Chapter 2 we proposed a general scheme for sampling on a subset of variables using Gibbs sampling and likelihood weighting in Bayesian networks resulting in a faster convergence of the estimates. The proposed algorithms remain time efficient by exploiting the network structure to bound the complexity of exact inference used to

computed sampling probabilities.

- *Minimum w-cutset Algorithm.* The efficiency of any scheme utilizing computation on a cutset can be improved by minimizing the size of the cutset for a fixed induced width bound w. The problem of finding minimal loop-cutset has been addressed previously. We contribute an algorithm for finding the minimum cost w-cutset.

- *Any-Time Bounding Framework.* In Chapter 4, we extended the ideas explored in bounded conditioning resulting in an any-time bounding framework that computes exactly a subset of cutset tuples and uses any off-the-shelf method to bound the probability mass spread over the remaining cutset tuples. Plugging bound propagation algorithm into proposed framework, we obtained a hybrid scheme that is superior to both bounded conditioning and bound propagation scheme.

The contributions of this dissertation to practical applications are in the area of planning and on-line decision support systems. Both of the proposed schemes for approximating and bounding posterior marginals are any-time. and converge to the correct posterior marignals. Sampling estimates improve as the number of generated samples increases. ATB bounds improve as the number of generated cutset tuples increases. The any-time property is important in on-line applications, where the least cost-effective action is "no action" and we cannot afford to wait idly for answers to our queries. We can take initial action based on the rough estimates and return later for more accurate results. Consider an autonomous system on board a satellite that has to make decisions on what information to send back to the ground. Typical constraints are communication bandwidth and commu-

nication time. The on-board computer can start transmitting data based on the preliminary estimates of their importance while refining the importance measures of the remaining data.

5.2 Future Work

The future work will focus on improving the efficiency of the proposed schemes and extending their applications to real-time on-line planning systems where new observations become available during processing. We have already discussed in previous chapters the possible improvements to the proposed approximation and bounding schemes. Incorporating new observation into our approximation and bounding schemes is motivated by scenarios where new information is gained during computation process, possibly as a result of some action. For example, if a set of actions leads to a dead end, it is only reasonable to incorporate this result into our computation so that we do not repeat the same mistake again. Of the previously proposed schemes, cutset conditioning addressed the problem of incorporating new evidence but the solution implied updating all previously visited tuples. Our objective is to incorporate new observations into the proposed sampling and bounding schemes while minimizing the amount of "recomputation" required.

Bibliography

[1] *COmputational INfrastructure for Operations Research*. http://www.coin-or.org.

[2] "Munin - an expert EMG assistant," in *Computer-Aided Electromyography and Expert Systems, ch. 21* (DESMEDT, J. E., ed.), Elsevier Science Publishers, Amsterdam, 1990.

[3] A. BECKER, R. BAR-YEHUDA, D. G., "Random algorithms for the loop cutset problem," *Journal of Artificial Intelligence Research*, pp. 219–234, 2000.

[4] ABDELBAR, A. M. and HEDETNIEMI, S. M., "Approximating maps for belief networks is NP-hard and other theorems," *Artificial Intelligence*, vol. 102, pp. 21–38, 1998.

[5] ANDRIEU, C., DE FREITAS, N., and DOUCET, A., "Rao-Blackwellised particle filtering via data augmentation," in *Advances in Neural Information Processing Systems*, MIT Press, 2002.

[6] ARNBORG, S. A., "Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey," *BIT*, vol. 25, pp. 2–23, 1985.

[7] BAR-YEHUDA, R., GEIGER, D., NAOR, J., and ROTH, R., "Approximation algorithms for the vertex feedback set problems with applications to constraint satsifaction and bayesian inference," in *Proceedings of the 5th Annual ACM-Siam Symposium On Discrete Algorithms, Arlington, Virginia*, 1994.

[8] BECKER, A., BAR-YEHUDA, R., and GEIGER, D., "Random algorithms for the loop cutset problem," in *Proceedings of Uncertainty in AI*, 1999.

[9] BECKER, A. and GEIGER, D., "Approximation algorithms for the loop cutset problem," in *Proceedings of Uncertainty in AI (UAI'94)*, pp. 60–68, 1994.

[10] BECKER, A. and GEIGER, D., "A sufficiently fast algorithm for finding close to optimal junction trees," in *Proceedings of Uncertainty in AI*, pp. 81–89, 1996.

[11] BEINLICH, I., SUERMONDT, G., CHAVEZ, R., and COOPER, G., "The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks," in *Second European Conference on AI and Medicine, Berlin, 1989*, Springer–Verlag, 1989.

[12] BELLARE, M., HELVIG, C., ROBINS, G., and ZELIKOVSKY, A., "Provably good routing tree construction with multi-port terminals," in *Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pp. 294–304, 1993.

[13] BERTELE, U. and BRIOSCHI, F., *Nonserial Dynamic Programming*. Academic Press, 1972.

[14] BIDYUK, B. and DECHTER, R., "Cycle-cutset sampling for Bayesian networks," in *16th Canadian Conference on AI*, pp. 297–312, 2003.

[15] BIDYUK, B. and DECHTER, R., "Empirical study of w-cutset sampling for Bayesian networks," in *UAI*, pp. 37–46, 2003.

[16] BIDYUK, B. and DECHTER, R., "On finding minimal w-cutset problem," in *Uncertainty in AI*, pp. 43–50, Morgan Kaufmann, 2004.

[17] BIENSTOCK, D., *Potential function methods for approximately solving linear programming problems: theory and practice*. Kluwer Academic Publishers, 2002.

[18] BILLINGSLEY, P., *Convergence of Probability Measures*. John Wiley & Sons, New York, 1968.

[19] CANO, J. E., HERNANDEZ, L. D., and MORAL, S., "Importance sampling algorithms for the propagation of probabilities on belief networks," *International Journal of Approximate Reasoning*, vol. 15, pp. 77–92, 1996.

[20] CASELLA, G. and ROBERT, C. P., "Rao-Blackwellisation of sampling schemes," *Biometrika*, vol. 83, no. 1, pp. 81–94, 1996.

[21] CHENG, J. and DRUZDZEL, M. J., "AIS-BN: An adaptive importance sampling algorithm for evidenctial reasoning in large baysian networks," *Journal of Aritificial Intelligence Research*, vol. 13, pp. 155–188, 2000.

[22] COOPER, G., "The computational complexity of probabilistic inferences," *Artificial Intelligence*, vol. 42, pp. 393–405, 1990.

[23] COZMAN, F. G., "Generalizing variable-elimination in Bayesian networks," in *Workshop on Probabilistic reasoning in Bayesian networks at SBIA/Iberamia 2000*, pp. 21–26, 2000.

[24] DAGUM, P. and LUBY, M., "Approximating probabilistic inference in Bayesian belief networks is NP-hard," *Artificial Intelligence*, vol. 60, no. 1, pp. 141–153, 1993.

[25] DAGUM, P. and LUBY, M., "An optimal algorithm formonte carlo estimation (extended abstract)," in *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science, Portland, Oregon*, pp. 142–149, 1995.

[26] DAGUM, P. and LUBY, M., "An optimal approximation algorithm for Bayesian inference.," *Artificial Intelligence*, vol. 93, pp. 1–27, 1997.

[27] DECHTER, R., "Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition," *Artificial Intelligence*, vol. 41, pp. 273–312, 1990.

[28] DECHTER, R., "Bucket elimination: A unifying framework for reasoning," *Artificial Intelligence*, vol. 113, pp. 41–85, 1999.

[29] DECHTER, R., *Constraint Processing*. Morgan Kaufmann, 2003.

[30] DECHTER, R. and FATTAH, Y. E., "Topological parameters for time-space tradeoff," *Artificial Intelligence*, vol. 125, no. 1-2, pp. 93–118, 2001.

[31] DECHTER, R., KASK, K., and MATEESCU, R., "Iterative join-graph propagation," in *Uncertainty in AI*, pp. 128–136, 2002.

[32] DOUCET, A. and ANDRIEU, C., "Iterative algorithms for state estimation of jump markov linear systems," *IEEE Transactions on Signal Processing*, vol. 49, no. 6, pp. 1216–1227, 2001.

[33] DOUCET, A., ANDRIEU, C., and GODSILL, S., "On sequential Monte Carlo sampling methods for Bayesian filtering," *Statistics and Computing*, vol. 10, no. 3, pp. 197–208, 2000.

[34] DOUCET, A., DE FREITAS, N., MURPHY, K., and RUSSELL, S., "Rao-Blackwellised particle filtering for dynamic Bayesian networks," in *Uncertainty in AI*, pp. 176–183, 2000.

[35] DOUCET, A., DEFREITAS, N., and GORDON, N., *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, New York, Inc., 2001.

[36] DOUCET, A., GORDON, N., and KRISHNAMURTHY, V., "Particle filters for state estimation of jump markov linear systems," tech. rep., Cambridge University Engineering Department, 1999.

[37] EDWARDS, R. and SOKAL, A., "Generalization of the fortuin-kasteleyn-swendsen-wang representation and monte carlo algorithm," *Physical Review D*, vol. 38, no. 6, pp. 2009–12, 1988.

[38] ESCOBAR, M. D., "Estimating normal means iwth a dirichlet process prior," *Journal of the American Statistical Aasociation*, vol. 89, pp. 268–277, 1994.

[39] FREY, B. J. and MACKAY, D. J. C., "A revolution: Belief propagation in graphs with cycles," in *Neural Information Processing Systems*, vol. 10, 1997.

[40] FUNG, R. and CHANG, K.-C., "Weighing and integrating evidence for stochastic simulation in Bayesian networks," in *Uncertainty in AI*, pp. 209–219, 1989.

[41] FUNG, R. and DEL FAVERO, B., "Backward simulation in Bayesian networks," in *Uncertainty in AI*, pp. 227–234, Morgan Kaufmann Publishers, 1994.

[42] GAREY, M. R. and JOHNSON, D. S., "Computers and intractability: A guide to the theory of NP-completeness," in *W. H. Freeman and Company, San Francisco*, 1979.

[43] GEIGER, D. and FISHELSON, M., "Optimizing exact genetic linkage computations," in *7th Annual International Conf. on Computational Molecular Biology*, pp. 114–121, 2003.

[44] GELFAND, A. and SMITH, A., "Sampling-based approaches to calculating marginal densities," *Journal of the American Statistical Association*, vol. 85, pp. 398–409, 1990.

[45] GEMAN, S. and GEMAN, D., "Stochastic relaxations, Gibbs distributions and the Bayesian restoration of images," *IEEE Transaction on Pattern analysis and Machine Intelligence*, vol. 6, pp. 721–742, 1984.

[46] GEYER, C. J., "Practical markov chain monte carlo," *Statistical Science*, vol. 7, pp. 473–483, 1992.

[47] GILKS, W., RICHARDSON, S., and SPIEGELHALTER, D., *Markov chain Monte Carlo in practice*. Chapman and Hall, 1996.

[48] GILKS, W. R. and WILD, P., "Adaptive rejection sampling for Gibbs sampling," *Applied Statistics*, vol. 41, pp. 337–348, 1992.

[49] GOGATE, V. and DECHTER, R., "A complete anytime algorithm for treewidth," in *Uncertainty in AI*, 2004.

[50] GOTTLOB, G., LEONE, N., and SCARELLO, F., "A comparison of structural CSP decomposition methods," in *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 394–399, Morgan Kaufmann, 1999.

[51] H. J. SUERMONDT, G. F. C. and HECKERMAN, D. E., "A combination of cutset conditioning with clique-tree propagation in the Path-finder system," in *Uncertainty in Artificial Intelligence (UAI'91)*, pp. 245–253, 1991.

[52] HASTINGS, W. K., "Monte carlo sampling using markov chains and their applications," *Biometrica*, vol. 57, no. 1, pp. 97–109, 1970.

[53] HECKERMAN, D., HORVITZ, E., and NATHAWANI, B., "Towards normative expert systems: Part i. the pathfinder project," *Methods of Information in Medicine*, vol. 31, no. 2, pp. 90–105, 1992.

[54] HENRION, M., "Propagating uncertainty in Bayesian networks by probabilistic logic sampling," in *Uncertainty in AI*, pp. 149–163, 1988.

[55] HIGDON, D., "Auxiliary variable methods for Markov Chain Monte Carlo with applications," *American Statistical Association*, vol. 93, pp. 585–595.

[56] HORVITZ, E., SUERMONDT, H., and COOPER, G., "Bounded conditioning: Flexible inference for decisions under scarce resources," in *Workshop on Uncertainty in Artificial Intelligence*, pp. 181–193, 1989.

[57] JENSEN, C., KONG, A., and KJÆRULFF, U., "Blocking Gibbs sampling in very large probabilistic expert systems," *International Journal of Human Computer Studies. Special Issue on Real-World Applications of Uncertain Reasoning.*, vol. 42, no. 6, pp. 647–666, 1995.

[58] JENSEN, C. and KONG, A., "Blocking Gibbs sampling for linkage analysis in large pedigrees with many loops," Research Report R-96-2048, Aalborg University, Denmark, 1996.

[59] JENSEN, F. V., LAURITZEN, S. L., and OLESEN, K. G., "Bayesian updating in causal probabilistic networks by local computation," *Computational Statistics Quarterly*, vol. 4, pp. 269–282, 1990.

[60] JONES, G. and HOBERT, J. P., "Honest exploration of intractable probability distributions via markov chain monte carlo," *Statist. Sci.*, vol. 16, no. 4, pp. 312–334, 2001.

[61] KARP, R., "Reducibility among combinatorial problems," in *Complexity of Computer Computations* (MILLER, R. and THATCHER, J., eds.), pp. 85–103, Plenum Press, New York, 1972.

[62] KASK, K. and DECHTER, R., "Stochastic local search for Bayesian networks," in *Workshop on AI and Statistics* (HECKERMAN, D. and WHITTAKER, J., eds.), pp. 113–122, Morgan Kaufmann, 1999.

[63] KASK, K., DECHTER, R., LARROSA, J., and DECHTER, A., "Unifying cluster-tree decompositions for reasoning in graphical models," *Artificial Intelligence*, vol. 166, pp. 165–193, 2005.

[64] KEARNS, M. and SAUL, L., "Large deviation methods for approximate probabilistic inference, with rates of convergence," in *Proc. of Uncertainty in AI*, pp. 311–319, Morgan Kaufmann, 1998.

[65] KEARNS, M. and SAUL, L., "Inference in multilayer networks via large deviation bounds," *Advances in Neural Information Processing Systems*, vol. 11, pp. 260–266, 1999.

[66] KJÆRULFF, U., *Triangulation of graphs - algorithms giving small total space*. No. R 90-09, 1990.

[67] KJÆRULFF, U., "HUGS: Combining exact inference and Gibbs sampling in junction trees," in *Uncertainty in AI*, pp. 368–375, Morgan Kaufmann, 1995.

[68] KOLLER, D., LERNER, U., and ANGELOV, D., "A general algorithm for approximate inference and its application to hybrid bayes nets," in *Uncertainty in AI*, pp. 324–333, 1998.

[69] KONG, A., LIU, J. S., and WONG, W., "Sequential imputations and Bayesian missing data problems," *Journal of the American Statistical Association*, vol. 89, no. 425, pp. 278–288, 1994.

[70] KRISTENSEN, K. and RASMUSSEN, I., "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*, vol. 33, pp. 197–217, 2002.

[71] KSCHISCHANG, F. R. and FREY, B. J., "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 219–230, 1998.

[72] KULLBACK, S., *Information Theory and Statistics*. Wiley, New York, 1959.

[73] LARKIN, D., "Approximate decomposition: A method for bounding and estimating probabilistic and deterministic queries," in *Proceedings of UAI*, pp. 346–353, 2003.

[74] LARROSA, J. and DECHTER, R., "Boosting search with variable elimination in constraint optimization and constraint satisfaction problems," *Constraints*, vol. 8, no. 3, pp. 303–326, 2003.

[75] LAURITZEN, S. and SPIEGELHALTER, D., "Local computation with probabilities on graphical structures and their application to expert systems," *Journal of the Royal Statistical Society, Series B*, vol. 50(2), pp. 157–224, 1988.

[76] LEISINK, M. A. R. and KAPPEN, H. J., "Bound propagation," *Journal of Artificial Intelligence Research*, vol. 19, pp. 139–154, 2003.

[77] LIU, J., *Correlation Structure and Convergence Rate of the Gibbs Sampler*. 1991.

[78] LIU, J., "The collapsed Gibbs sampler in Bayesian computations with applications to a gene regulation problem," *Journal of the American Statistical Association*, vol. 89, no. 427, pp. 958–966, 1994.

[79] LIU, J., WONG, W., and KONG, A., "Covariance structure of the Gibbs sampler with applications to the comparison of estimators and augmentation schemes," *Biometrika*, vol. 81, no. 1, pp. 27–40, 1994.

[80] LIU, J. S., "Nonparametric hierarchical bayes via sequential imputations," *Annals of Statistics*, vol. 24, no. 3, pp. 911–930, 1996.

[81] LIU, J. S., *Monte Carlo Strategies in Scientific Computing*. Springer-Verlag, New York, Inc., 2001.

[82] LUND, C. and YANNAKAKIS, M., "On the hardness of approximating minimization problems," *J. of ACM*, vol. 41, pp. 960–981, September 1994.

[83] MACEACHERN, S., CLYDE, M., and LIU, J., "Sequential importance sampling for nonparametric bayes models: The next generation," *The Canadian Journal of Statistics*, vol. 27, pp. 251–267, 1998.

[84] MACEACHERN, S. N., "Estimating normal means with a conjugate style dirichlet process prior," *Communications in Statistics-Simulation and Computation*, vol. 23, no. 3, pp. 727–741, 1994.

[85] MACKAY, D., "Introduction to monte carlo methods," in *Proceedings of NATO Advanced Study Institute on Learning in Graphical Models. Sept 27-Oct 7*, pp. 175–204, 1996.

[86] MAIER, D., "The theory of relational databases," in *Computer Science Press, Rockville, MD*, 1983.

[87] MANNINO, M. V. and MOOKERJEE, V. S., "Probability bounds for goal directed queries in Bayesian networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, pp. 1196–1200, September/October 2002.

[88] MCELIECE, R., MACKAY, D., and CHENG, J.-F., "Turbo decoding as an instance of pearl's belief propagation algorithm," *IEEE J. Selected Areas in Communication*, 1997.

[89] METROPOLIS, N., ROSENBLUTH, A. W., ROSENBLUTH, M. N., TELLER, A. H., and TELLER, E., "Equations of state calculations by fast computing," *Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1091, 1953.

[90] MORAL, S. and SALMERON, A., "Dynamic importance sampling in bayesian networks based on probability trees," *International Journal of Approximate Reasoning*, vol. 38, pp. 245–261, 2005.

[91] MURPHY, K. P., WEISS, Y., and JORDAN, M. I., "Loopy belief propagation for approximate inference: An empirical study," in *Uncertainty in AI*, pp. 467–475, 1999.

[92] PARK, J., "Map complexity results and approximation methods," in *UAI*, pp. 388–396, Morgan Kaufmann Publishes, Inc., 2002.

[93] PARK, J. and DARWICHE, A., "Approximating map using local search," in *UAI*, pp. 388–396, Morgan Kaufmann Publishes, Inc., 2001.

[94] PARK, J. and DARWICHE, A., "Complexity results and approximation strategies for map explanations," *JAIR*, 2003.

[95] PARKER, R. and MILLER, R., "Using causal knowledge to create simulated patient cases: the CPCS project as an extension of INTERNIST-1," in *Proc. 11th Symp. Comp. Appl. in Medical Care*, pp. 473–480, 1987.

[96] PEARL, J., *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[97] PEOT, M. A. and SHACHTER, R. D., "Fusion and propagation with multiple observations in belief networks," *Artificial Intelligence*, vol. 48, pp. 299–318, 1992.

[98] POOLE, D., "Probabilistic conflicts in a search algorithm for estimating posterior probabilities in Bayesian networks," *Artificial Intelligence*, vol. 88, no. 1–2, pp. 69–100, 1996.

[99] POOLE, D., "Context-specific approximation in probabilistic inference," in *Proc. of Uncertainty in Artificial Intelligence (UAI)*, pp. 447–454, 1998.

[100] PRADHAN, M., PROVAN, G., MIDDLETON, B., and HENRION, M., "Knowledge engineering for large belief networks," in *Proceedings of Tenth Conf. on Uncertainty in Artificial Intelligence, Seattle, WA*, pp. 484–490, 1994.

[101] RAJAGOPALAN, S. and VAZIRANI, V., "Primal-dual RNC approximation algorithms for (multi)set (multi)cover and covering integer programs," *SIAM J. of Computing*, vol. 28, no. 2, pp. 525–540, 1998.

[102] RISH, I. and DECHTER, R., "To guess or to think? Hybrid algorithms for SAT," in *Principles of Constraint Programming (CP-96)*, pp. 555–556, 1996.

[103] RISH, I. and DECHTER, R., "Resolution vs. search; two strategies for sat," *J. of Automated Reasoning*, vol. 24(1/2), pp. 225–275, 2000.

[104] RISH, I., KASK, K., and DECHTER, R., "Empirical evaluation of approximation algorithms for probabilistic decoding," in *Uncertainty in AI*, 1998.

[105] ROBERTS, G. O. and SAHU, S. K., "Updating schemes; correlation structure; blocking and parameterization for the Gibbs sampler," *Journal of the Royal Statistical Society, Series B*, vol. 59, no. 2, pp. 291–317, 1997.

[106] ROSTI, A.-V. and GALES, M., "Rao-Blackwellised Gibbs sampling for switching linear dynamical systems," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2004)*, pp. 809–812, 2004.

[107] ROTH, D., "On the hardness of approximate reasoning," *Artificial Intelligence*, vol. 82, pp. 273–302, Apr. 1996.

[108] SCHERVISH, M. and CARLIN, B., "On the convergence of successive substitution sampling," *Journal of Computational and Graphical Statistics*, vol. 1, pp. 111–127, 1992.

[109] SHACHTER, R. D., ANDERSON, S. K., and SOLOVITZ, P., "Global conditioning for probabilistic inference in belief networks," in *Uncertainty in Artificial Intelligence (UAI'94)*, pp. 514–522, 1994.

[110] SHACHTER, R. D. and PEOT, M. A., "Simulation approaches to general probabilistic inference on belief networks," in *Uncertainty in AI*, pp. 221—231, 1989.

[111] SHIMONY, S. E., "Finding MAPs for belief networks is NP-hard," *Artificial Intelligence*, vol. 68, pp. 399–410, 1994.

[112] SHOIKET, K. and GEIGER, D., "A practical algorithm for finding optimal triangulations," in *Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pp. 185–190, 1997.

[113] STEIGER, N. M. and WILSON, J. R., "Convergence properties of the batch means method for simulation output analysis," *INFORMS Journal on Computing*, vol. 13, no. 4, pp. 277–293, 2001.

[114] SUERMONDT, H. J. and COOPER, G. F., "Probabilistic inference in multiply connected belief network using loop cutsets," *International Journal of Approximate Reasoning*, vol. 4, pp. 283–306, 1990.

[115] SWENDSEN, R. and WANG, J., "The calculation of posterior distributions by data augmentation (with discussion)," *American Statistical Association*, vol. 82, no. 382, pp. 528–540, 1987.

[116] TANNER, M. A. and WONG, W. H., "The calculation of posterior distributions by data augmentation," *Journal of the American Statistical Association*, vol. 82, pp. 528–550, 1987.

[117] TIERNEY, L., "Markov chains for exploring posterior distributions," *Annals of Statistics*, vol. 22, no. 4, pp. 1701–1728, 1994.

[118] TREVISAN, L., "Non-approximability results for optimization problems on bounded degree instances," *In proceedings of 33rd ACM STOC*, 2001.

[119] VAZIRANI, V. V., *Approximation Algorithms*. Springer, 2001.

[120] YUAN, C. and DRUZDZEL, M., "An importance sampling algorithm based on evidence pre-propagation," in *Uncertainty in AI*, pp. 624–631, 2003.

Appendix A

KL-distance between target and sampling distribution

LEMMA **A.0.1** *Given non-negative real weights $w_1, ..., w_n$ s.t. $\sum_i w_i = 1$, and given two set of non-negative values $f_1, ..., f_n$ and $g_1, ..., g_n$ s.t. $\forall i$, $g_i = f_i \epsilon_i$, where $0 \leq \epsilon_i \leq 1$, then:*

$$R = \frac{\sum_i w_i f_i g_i}{(\sum_i w_i f_i)(\sum_i w_i g_i)} \geq 1$$

Proof. The proof is based on power means inequality. Using the terms defined here, the r-th weighted power mean of the f_i is defined as:

$$M_w^r(f_1, ..., f_n) = (w_1 f_1^r + w_2 f_2^r + ... + w_n f_n^r)^{1/r}$$

The power means inequality states that if $r < s$, then:

$$M_w^r \leq M_w^s$$

Let $r = 1$ and $s = 2$. Then we get:

$$w_1 f_1 + w_2 f_2 + ... + w_n f_n \leq (w_1 f_1^2 + w_2 f_2^2 + ... + w_n f_n^2)^{1/2}$$

which is equivalent to:

$$(w_1 f_1 + w_2 f_2 + ... + w_n f_n)^2 \leq w_1 f_1^2 + w_2 f_2^2 + ... + w_n f_n^2 \tag{A.1}$$

Deviding both sides of inequality by $(w_1 f_1 + w_2 f_2 + ... + w_n f_n)^2$, we get:

$$1 \leq \frac{w_1 f_1^2 + w_2 f_2^2 + ... + w_n f_n^2}{(w_1 f_1 + w_2 f_2 + ... + w_n f_n)^2} = \frac{\sum_i w_i f_i^2}{(\sum_i w_i f_i)^2} \tag{A.2}$$

Let $w_i' = \frac{\epsilon_i w_i}{\sum_j \epsilon_j w_j}$. Write power means inequality for f_i using weights $w_1', ..., w_n'$:

$$1 \leq \frac{\sum_i w_i' f_i^2}{(\sum_i w_i' f_i)^2}$$

Substituting $\frac{\epsilon_i w_i}{\sum_j \epsilon_j w_j}$ for w_i', we get:

$$1 \leq \frac{\sum_i \frac{\epsilon_i w_i}{\sum_j \epsilon_j w_j} f_i^2}{(\sum_i \frac{\epsilon_i w_i}{\sum_j \epsilon_j w_j} f_i)^2} = \frac{\frac{\sum_i \epsilon_i w_i f_i^2}{\sum_j \epsilon_j w_j}}{(\frac{\sum_i \epsilon_i w_i f_i}{\sum_i \epsilon_i w_i})^2} = \frac{(\sum_i \epsilon_i w_i f_i^2)(\sum_i \epsilon_i w_i)^2}{(\sum_i \epsilon_i w_i)(\sum_i \epsilon_i w_i f_i)^2} = \frac{(\sum_i \epsilon_i w_i f_i^2)(\sum_i \epsilon_i w_i)}{(\sum_i \epsilon_i w_i f_i)^2}$$

By decomposing the fraction on the right-hand side into a product of two fractions, we get:

$$1 \leq \frac{\sum_i \epsilon_i w_i f_i^2}{\sum_i \epsilon_i w_i f_i} \frac{\sum_i \epsilon_i w_i}{\sum_i \epsilon_i w_i f_i} = \frac{\sum_i w_i g_i f_i}{\sum_i w_i g_i} \frac{\sum_i \epsilon_i w_i}{\sum_i \epsilon_i w_i f_i} \tag{A.3}$$

We focus on analyzing the factor $\frac{\sum_i \epsilon_i w_i}{\sum_i \epsilon_i w_i f_i}$. Since $\frac{a}{b} \leq \frac{a+\delta}{b+\delta}$, then:

$$\frac{\sum_i \epsilon_i w_i}{\sum_i \epsilon_i w_i f_i} \leq \frac{\sum_i \epsilon_i w_i + \sum_i w_i f_i (1-\epsilon_i)}{\sum_i \epsilon_i w_i f_i + \sum_i w_i f_i (1-\epsilon_i)} = \frac{\sum_i \epsilon_i w_i + \sum_i w_i f_i - \sum_i w_i f_i \epsilon_i}{\sum_i w_i f_i} \tag{A.4}$$

$$= \frac{\sum_i w_i f_i + \sum_i \epsilon_i w_i (1-f_i)}{\sum_i w_i f_i} \leq \frac{\sum_i w_i f_i + \sum_i w_i (1-f_i)}{\sum_i w_i f_i} \tag{A.5}$$

$$= \frac{\sum_i w_i}{\sum_i w_i f_i} = \frac{1}{\sum_i w_i f_i} \tag{A.6}$$

Substituting right-hand side of Eq. A.6 for $\frac{\sum_i \epsilon_i w_i}{\sum_i \epsilon_i w_i f_i}$ in Eq. A.3, we get:

$$1 \leq \frac{\sum_i w_i g_i f_i}{\sum_i w_i g_i} \frac{\sum_i \epsilon_i w_i}{\sum_i \epsilon_i w_i f_i} \leq \frac{\sum_i w_i g_i f_i}{\sum_i w_i g_i} \frac{1}{\sum_i w_i f_i} = \frac{\sum_i w_i g_i f_i}{(\sum_i w_i g_i)(\sum_i w_i f_i)} = R$$

■

THEOREM **2.3.3 (Reduced Information Distance)**

Proof. Assume we have a Bayesian network $\mathcal{B}$ over $X = \{X_1, ..., X_n\}$. Let lower case x denote an instantiation of all variables. Let $E \subset X$ be a subset of evidence variables, $|E| = m$. Let $C = \{C_1, ..., C_m\} \subset X \backslash E$ denote a subset of variables.The target distribution of Likelihood Weighting is denoted $P(X|e)$. The sampling distribution of Likelihood Weighting is denoted $Q(X)$. When sampling over cutset C, the target distribution is $P(C|e)$ and sampling distribution is $Q(C)$.

First, we evaluate $KL(P(C|e), Q(C))$, denoted KL_c. By definition:

$$KL_c = \sum_c P(c|e) \log \frac{P(c|e)}{Q(c)} \tag{A.7}$$

$$= \sum_c P(c|e) \log \frac{P(c,e)}{Q(c)P(e)} \tag{A.8}$$

$$= \sum_c P(c|e) \log \frac{P(c,e)}{Q(c)} - \sum_c P(c|e) \lg P(e) \tag{A.9}$$

$$= \sum_c P(c|e) \log \frac{P(c,e)}{Q(c)} - \log P(e) \tag{A.10}$$

To simplify analysis, we focus on evaluating KL_c without constant $\log P(e)$. We denote:

$$KL'_c = KL_c + \log P(e) = \sum_c P(c|e) \log \frac{P(c,e)}{Q(c)} \tag{A.11}$$

For each variable $E_i \in E$, let $c_{1:k_i}$ denote a subset of cutset variables that precede E_i in the sampling order. Let $c_{k_i+} = c \backslash c_{1:k_i}$. Note, that $\frac{P(c,e)}{Q(c)} = \prod_{E_i \in E} P(e_i|c_{1:k_i}, e_{1:i-1})$. Therefore:

$$KL'_c = \sum_c P(c|e) \log \prod_{E_i \in E} P(e_i|c_{1:k_i}, e_{1:i-1}) \tag{A.12}$$

Since $\log$ of a product equals the sum of $\log$s of factors, then we can transform the expression above into:

$$KL'_c = \sum_c P(c|e) \sum_{E_i \in E} \log P(e_i|c_{1:k_i}, e_{1:i-1}) = \sum_{E_i \in E} \sum_c P(c|e) \log P(e_i|c_{1:k_i}, e_{1:i-1}) \tag{A.13}$$

$$= \sum_{E_i \in E} \sum_c P(c_{1:k_i}, c_{k_i+}|e) \log P(e_i|c_{1:k_i}, e_{1:i-1}) \tag{A.14}$$

Clearly, we can sum out variables in c_{k_i+}.

$$KL'_c = \sum_{E_i \in E} \sum_{c_{1:k_i}} P(c_{1:k_i}|e) \log P(e_i|c_{1:k_i}, e_{1:i-1}) \tag{A.15}$$

Next, we evaluate $KL(P(X|e), Q(X))$, denoted KL_x. Let $Y = X \backslash C, E$. By definition:

$$KL_x = \sum_{c,y} P(y, c|e) \log \frac{P(y, c|e)}{Q(c, y)} \tag{A.16}$$

Replacing $P(y, c|e)$ with $P(y, c, e)/P(e)$, similar to KL_c, we get:

$$KL_x = \sum_{c,y} P(y, c|e) \log \frac{P(y, c, e)}{Q(c, y)} - \log P(e) \tag{A.17}$$

Again, to simplify notation, we denote:

$$KL'_x = KL_x + \log P(e) = \sum_{c,y} P(y, c|e) \log \frac{P(y, c, e)}{Q(c, y)} \tag{A.18}$$

Using equality $\frac{P(y,c,e)}{Q(y,c)} = \prod_{E_i \in E} P(e_i|pa_i)$ yields:

$$KL'_x = \sum_{c,y} P(y, c|e) \log \prod_{E_i \in E} P(e_i|pa_i) \tag{A.19}$$

Since $\log$ of a product equals the sum of $\log$s of factors, then we get:

$$KL'_x = \sum_{c,y} P(y, c|e) \sum_{E_i \in E} \log P(e_i|pa_i) = \sum_{E_i \in E} \sum_{c,y} P(y, c|e) \log P(e_i|pa_i) \tag{A.20}$$

Without loss of generality, we can assume that $c_{1:k_i}$ and pa_i are disjoint. Let $y_i = (c \cup y) \backslash c_{1:k_i}, pa_i$. Then:

$$KL'_x = \sum_{E_i \in E} \sum_{c_{1:k_i}, y_i} \sum_{pa_i} P(c_{1:k_i}, pa_i, y_i|e) \log P(e_i|pa_i) \tag{A.21}$$

Clearly, we can sum out variables in y_i:

$$KL'_x = \sum_{E_i} \sum_{c_{1:k_i}} \sum_{pa_i} P(c_{1:k_i}, pa_i|e) \log P(e_i|pa_i) \tag{A.22}$$

$$= \sum_{E_i} \sum_{c_{1:k_i}} \sum_{pa_i} P(pa_i|c_{1:k_i}, e) P(c_{1:k_i}|e) \log P(e_i|pa_i) \tag{A.23}$$

$$= \sum_{E_i} \sum_{c_{1:k_i}} P(c_{1:k_i}|e) \sum_{pa_i} P(pa_i|c_{1:k_i}, e) \log P(e_i|pa_i) \tag{A.24}$$

Due to Jensen's inequality:

$$\sum_{pa_i} P(pa_i|c_{1:k_i}, e) \log P(e_i|pa_i) \geq \log \sum_{pa_i} P(pa_i|c_{1:k_i}, e) P(e_i|pa_i) \tag{A.25}$$

Consequently:

$$KL'_x \geq \sum_{E_i} \sum_{c_{1:k_i}} P(c_{1:k_i}|e) \log \sum_{pa_i} P(pa_i|c_{1:k_i}, e) P(pa_i|c_{1:k_i}, e) P(e_i|pa_i) \tag{A.26}$$

Thus:

$$KL_x - KL_c = KL'_x - \log P(e) - KL'_c + \log P(e) = KL'_x - KL'_c \tag{A.27}$$

$$\geq \sum_{E_i \in E} \sum_{c_{1:k_i}} P(c_{1:k_i}|e) \log \sum_{pa_i} P(pa_i|c_{1:k_i}, e) P(e_i|pa_i) \tag{A.28}$$

$$- \sum_{E_i \in E} \sum_{c_{1:k_i}} P(c_{1:k_i}|e) \log P(e_i|c_{1:k_i}, e_{1:i-1}) \tag{A.29}$$

$$= \sum_{E_i \in E} \sum_{c_{1:k_i}} P(c_{1:k_i}|e) [\log \sum_{pa_i} P(pa_i|c_{1:k_i}, e) P(e_i|pa_i) - \log P(e_i|c_{1:k_i}, e_{1:i-1})] \tag{A.30}$$

Sing $\log a - \log b = \log \frac{a}{b}$, then we get :

$$KL_x - KL_c \geq \sum_{E_i} \sum_{c_{1:k_i}} P(c_{1:k_i}|e) \log \frac{\sum_{pa_i} P(e_i|pa_i) P(pa_i|c_{1:k_i}, e)}{P(e_i|c_{1:k_i}, e_{1:i-1})} \tag{A.31}$$

Let us evaluate the $\log$:

$$l = \frac{\sum_{pa_i} P(e_i|pa_i) P(pa_i|c_{1:k_i}, e)}{P(e_i|c_{1:k_i}, e_{1:i-1})}$$

Multiplying the numerator and denominator of the fraction by $P(c_{1:k_i}, e)$ yields:

$$l = \frac{\sum_{pa_i} P(e_i|pa_i)P(pa_i, c_{1:k_i}, e)}{P(e_i|c_{1:k_i}, e_{1:i-1})P(c_{1:k_i}, e)}$$

Denoting $e_{1:i-1} = \{e_1, ..., e_{i-1}\}$ and $e_{i:m} = \{e_i, ..., e_m\}$, we can rewrite:

$$l = \frac{\sum_{pa_i} P(e_i|pa_i)P(pa_i, c_{1:k_i}, e_{1:i-1}, e_{i:m})}{P(e_i|c_{1:k_i}, e_{1:i-1})P(c_{1:k_i}, e_{1:i-1}, e_{i:m})} \tag{A.32}$$

$$= \frac{\sum_{pa_i} P(e_i|pa_i)P(e_{i:m}|pa_i, c_{1:k_i}, e_{1:i-1})P(pa_i|c_{1:k_i}, e_{1:i-1})P(c_{1:k_i}, e_{1:i-1})}{P(e_i|c_{1:k_i}, e_{1:i-1})P(e_{i:m}|c_{1:k_i}, e_{1:i-1})P(c_{1:k_i}, e_{1:i-1})} \tag{A.33}$$

The terms $P(e_i|c_{1:k_i}, e_{1:i-1})$ in the denominator and nominator cancel-out, yielding:

$$l = \frac{\sum_{pa_i} P(e_i|pa_i)P(e_{i:m}|pa_i, c_{1:k_i}, e_{1:i-1})P(pa_i|c_{1:k_i}, e_{1:i-1})}{P(e_i|c_{1:k_i}, e_{1:i-1})P(e_{i:m}|c_{1:k_i}, e_{1:i-1})}$$

We can apply Lemma A.0.1 to l by letting:

$$\begin{aligned} f_i &= P(e_i|pa_i, c_{1:k_i}, e_{1:i-1}) = P(e_i|pa_i) \\ g_i &= P(e_{i:m}|pa_i, c_{1:k_i}, e_{1:i-1}) \\ \epsilon_i &= P(e_{i+1:m}|pa_i, c_{1:k_i}, e_{1:i}) \\ w_i &= P(pa_i|c_{1:k_i}, e_{1:i-1}) \end{aligned}$$

and also observing that:

$$\sum_i w_i f_i = \sum_{pa_i} P(e_i|pa_i, c_{1:k_i}, e_{1:i-1})P(pa_i|c_{1:k_i}, e_{1:i-1}) = P(e_i|c_{1:k_i}, e_{1:i-1})$$

$$\sum_i w_i g_i = \sum_{pa_i} P(e_{i:m}|pa_i, c_{1:k_i}, e_{1:i-1})P(pa_i|c_{1:k_i}, e_{1:i-1}) = P(e_{i:m}|c_{1:k_i}, e_{1:i-1})$$

Therefore, due to Lemma A.0.1, we get:

$$l \geq 1$$

Therefore:

$$\lg l \geq 1$$

Subsequently:

$$KL_x - KL_c \geq 0$$

Proof is complete. ■

Appendix B

Analysis of Bounded Conditioning

In [56], 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 [56]. 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 [56] while disregarding evidence e in [56], we have following lower and upper bounds:

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

$$P^U(x|e) = \sum_{i=1}^{h} P(x|c^i,e)w_i^U + \sum_{i=h+1}^{j} w_i^U + \sum_{i=j+1}^{M} w_i^{U'} \tag{B.2}$$

Since the sum $\sum_{i=h+1}^{j} w_i^U$ in $P^U(x|e)$ corresponds in [56] 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:

$$P^L(x|e) = \sum_{i=1}^{h} P(x|c^i,e)w_i^L \tag{B.3}$$

$$P^U(x|e) = \sum_{i=1}^{h} P(x|c^i,e)w_i^U + \sum_{i=h+1}^{M} w_i^{U'} \tag{B.4}$$

The weights in the above expressions are defined as follows:

$$w_i^L = \frac{P(c^i|e)}{\sum_{k=1}^{h} P(c^k|e) + \sum_{k=h+1}^{M} P(c^k)} = \frac{P(c^i,e)}{\sum_{k=1}^{h} P(c^k,e) + \sum_{k=h+1}^{M} P(c^k)} \tag{B.5}$$

$$w_i^U = \frac{P(c^i|e)}{\sum_{k=1}^{h} P(c^k|e)} = \frac{P(c^i,e)}{\sum_{k=1}^{h} P(c^k,e)} \tag{B.6}$$

$$w_i^{U'} = \frac{P(c^i)}{\sum_{k=1}^{h} w_k^L + (1 - \sum_{k=h+1}^{M} w_k^U)} \tag{B.7}$$

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:

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

Appendix C

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:

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

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:

$$P(c'_k|e) = \frac{\sum_{i=1}^{M_k} P(c'_k, z^i, e)}{\sum_{c_k \in \mathcal{D}(C_k)} \sum_{i=1}^{M_k} P(c_k, z^i, e)} \tag{C.2}$$

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. (C.2) as follows:

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

Similarly, 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. (C.2) takes on the form:

$$P(c'_k|e) = \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{i=h_{c'_k}+1}^{M_k} P(c'_k, z^i, e)}{\sum_{c_k \in \mathcal{D}(C_k)} \sum_{i=1}^{h_{c_k}} P(c_k, z^i, e) + \sum_{c_k \in \mathcal{D}(C_k)} \sum_{i=h_{c_k}+1}^{M_k} P(c_k, z^i, e)} \tag{C.3}$$

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. (C.3) transforms into:

$$P(c'_k|e) = \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{i=h_{c'_k}+1}^{M_k} P(c'_k, z^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{c_k \in \mathcal{D}(C_k)} \sum_{i=h_{c_k}+1}^{M_k} P(c_k, z^i, e)} \tag{C.4}$$

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:

$$P(c'_k|e) = \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{c_k \in \mathcal{D}(C_k)} \sum_{j=1}^{M'_{c_k}} P(c_k, z^j_{1:q_j}, e)} \tag{C.5}$$

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:

$$P(c'_k|e) = \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'_{c'_k}} P(c'_k, z^j_{1:q_j}, e) + \sum_{c_k \neq c'_k} \sum_{j=1}^{M'_{c_k}} P(c_k, z^j_{1:q_j}, e)} \tag{C.6}$$

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 4.3.2:

$$P(c'_k|e) \leq \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P^U(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'_{c'_k}} P^U(c'_k, z^j_{1:q_j}, e) + \sum_{c_k \neq c'_k} \sum_{j=1}^{M'_{c_k}} P(c_k, z^j_{1:q_j}, e)} \tag{C.7}$$

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:

$$P(c'_k|e) \leq \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c_k}} P^U(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'_{c_k}} P^U(c'_k, z^j_{1:q_j}, e) + \sum_{c_k \neq c'_k} \sum_{j=1}^{M'_{c_k}} P^L(c_k, z^j_{1:q_j}, e)} = P_c^{U_1} \tag{C.8}$$

The lower bound derivation is similar. Taking Eq. (C.4) 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. (C.4) into:

$$P(c'_k|e) = \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{i=h_{c'_k}+1}^{M_{c_k}} P(c'_k, z^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M} P(c^i, e)} \tag{C.9}$$

Now, replacing the summation of unexplored fully-instantiated tuples in Eq. (C.7) with summation over corresponding partially-instantiated tuples, we obtain:

$$P(c'_k|e) = \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P(c^j_{1:q_j}, e)} \tag{C.10}$$

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:

$$P(c'_k|e) \geq \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P^L(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^U(c^j_{1:q_j}, e)} = P_c^{L_1} \tag{C.11}$$

We can obtain a different lower bound if we start with Eq. (C.6) and replace $P(c'_k, z^i_{1:q_i}, e)$ in numerator and denominator with a lower bound. Lemma 4.3.2 guarantees that the resulting fraction will be a lower bound on $P(c'_k|e)$:

$$P(c'_k|e) \geq \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P^L(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'_{c'_k}} P^L(c'_k, z^j_{1:q_j}, e) + \sum_{c_k \neq c'_k} \sum_{j=1}^{M'_{c_k}} P(c_k, z^j_{1:q_j}, e)} \tag{C.12}$$

Finally, replacing $P(c_k, z^j_{1:q_j}, e)$ in Eq. (C.12) with a corresponding upper bound, we obtain the second lower bound $P_c^{L_2}$:

$$P(c'_k|e) \geq \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P^L(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'_{c'_k}} P^L(c'_k, z^j_{1:q_j}, e) + \sum_{c_k \neq c'_k} \sum_{j=1}^{M'_{c_k}} P^U(c_k, z^j_{1:q_j}, e)} = P_c^{L_2} \tag{C.13}$$

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. (4.22) and (4.26). Hence, the result of Theorem 4.3.3 and Corollary 4.3.3 apply.

With respect to computing bounds on $P(c'_k, z_{1:q}, e)$ in Eq. (C.8) and (C.13) 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 D

Proofs for Chapter 4, Section 4.3

THEOREM **4.3.3 (Lower Bound Dominance1)** 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. Assume an algorithm $\mathcal{A}$ computes bounds $P^L(c^j_{1:q_j}, e)$ and $P^U(c^j_{1:q_j}, e)$ on $P(c^j_{1:q_j}, e)$ and bounds $P^L(x|c^j_{1:q_j}, e)$ and $P^U(x|c^j_{1:q_j}, e)$ on $P(x|c^j_{1:q_j}, e)$ for $1 \leq j \leq M'$ and $\forall x \in \mathcal{D}(X)$. Let:

$$P^L(x, c^j_{1:q_j}, e) = P^L(x|c^j_{1:q_j}, e)P^L(c^j_{1:q_j}, e)$$

$$P^U(x, c^j_{1:q_j}, e) = P^U(x|c^j_{1:q_j}, e)P^U(c^j_{1:q_j}, e)$$

If $P^L(x'|c^j_{1:q_j}, e) \leq 1 - \sum_{x \neq x'} P^U(x|c^j_{1:q_i}, e)$ then $P^{L_1}(x'|e) \leq P^{L_2}(x'|e)$ where P^{L_1} and P^{L_2} are defined in Eq. (4.22) and Eq. (4.26) respectively.

Proof. The numerators in Eq. (4.22) and Eq. (4.26) 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 $D_2 - D_1$. Therefore, the difference is:

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

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

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

Thus, $D_2 \leq D_1$. Therefore, $P^{L_1} \leq P^{L_2}$. ■

THEOREM **4.3.4** 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 $\forall j$, $P^U(x, c^j_{1:q_j}, e) \leq P(c^j_{1:q_j})$ then $P^{U_1}(x|e) \leq P^U(x|e)$ where $P^{U_1}(x|e)$ is given in Eq. (4.28) and $P^U(x|e)$ is the bounded conditioning upper bound given in Eq. (4.10).

Proof. Setting $P^L(x, c^j_{1:q_j}, e) = 0$, $x \neq x'$, in Eq. (4.28) and, hence, reducing the denominator, 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:

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

$$= \frac{\sum_{i=1}^{h} P(x, c^i, e) + \sum_{i=1}^{M} P(c^j)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=1}^{M} P(c^j)} \triangleq P^{U_3}(x'|e) \quad \text{(D.2)}$$

The bound $P^{U_3}(x'|e)$ in Eq. (D.2) 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 addend in $P^{U_3}(x|e)$ is smaller than the first addend in Eq. (4.10):

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

The second addend in $\hat{P}^U(x|e)$ is smaller then the second addend in Eq. (4.10):

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

The theorem follows. ■

THEOREM **2.3.1** 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 preceding 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 preceding 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 **4.3.5** 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:

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

where $P_{\mathcal{A}}^{L_3}$ and $P_{\mathcal{A}}^{U_3}$ are expressed in Eq. (4.38) and Eq. (4.39) 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 $P_{\mathcal{A}}^{U} - P_{\mathcal{A}}^{L}$ is always lower bounded by $1 - q$. We begin by computing the bounds interval:

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

We replace $\sum_{j=1}^{M'} P_{\mathcal{A}}^{U}(c^i, e)$ 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:

$$P_{\mathcal{A}}^{U} - P_{\mathcal{A}}^{L} \geq \frac{\sum_{j=1}^{M'} P(c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P(c^i, e)} \tag{D.3}$$

Since, the $\sum_{j=1}^{M'} P(c^j, e) = \sum_{i=h+1}^{M} P(c^i, e)$, then the Eq. (D.3) transforms into:

$$P_{\mathcal{A}}^{U} - P_{\mathcal{A}}^{L} \geq \frac{\sum_{i=h+1}^{M} P(c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M} P(c^i, e)} \tag{D.4}$$

Replacing the sums in denominator in Eq. (D.3) 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:

$$P_{\mathcal{A}}^{U} - P_{\mathcal{A}}^{L} \geq \frac{P(e) - \sum_{i=1}^{h} P(c^i, e)}{P(e)} = \frac{P(e) - qP(e)}{P(e)} = 1 - q \tag{D.5}$$

■

Appendix E

Proof of Optimality of Greedy Algorithm

Greedy Algorithm For Multiple-Knapsack Packing and Covering with Sum-to-1 Constraint

The inputs to the problem are a set of variables $x = \{x_1, ..., x_n\}$, representing some type of commodities, and a set of knapsacks $K = \{K_1, ..., K_m\}$ with minimum required fill capacity L_i and maximum capacity U_i. Each variable X_i is assigned to some knapsack K_j and cannot be placed in any other knapsack. Let $Q_j \subset x$, $j \in [1, m]$, denote a set of commodities that can be loaded into knapsack K_j. All subsets Q_j are mutually exclusive. Each variable x_i has an associated payoff (reward) c_i. The objective is to pack m knapsacks so as to maximize payoff. The linear program is defined as follows:

$$\max f = \sum_i c_i X_i \tag{E.1}$$

$$s.t. \quad \sum_i x_i = 1 \tag{E.2}$$

$$L_j \leq \sum_{i, x_i \in Q_j} x_i \leq U_j, \forall j \tag{E.3}$$

$$0 \leq x_i \leq x_i^{max}, \forall x_i \in x \tag{E.4}$$

A problem is feasible if and only if:
1) $\sum_j L_j \leq 1$
2) $\sum_{x_k \in Q_j} x_k^{max} \geq L_j$
3) $\sum_{x_k \in X} x_k^{max} \geq 1$.
Therefore, we assume that these conditions always hold.

Sort variables by their coefficients from largest to smallest:

$$c_1 \leq c_2 \leq ... \leq c_n$$

First, we satisfy the lower bound constraints. Initialize each variable value to 0. Assign each variable X_i in order:

$$x_i \leftarrow \min\{x_i^{max}, L_j | x_i \in Q_j\}$$

and update corresponding lower and upper bound:

$$L_j \leftarrow L_j - x_i$$

$$U_j \leftarrow U_j - x_i$$

$$x_i^{max} \leftarrow x_i^{max} - x_i$$

We stop when all lower bounds are zero or we reach the end of the list. Denote the assignment to all variables after the first pass as $x^0 = \{x_1^0, ..., x_n^0\}$. All lower bounds will be satisfied after processing all variables as long as $\sum_j L_j \leq 1$ and $\forall j$, $\sum_{x_k \in Q_j} x_k^{max} \geq L_j$.

Let $X_i = Y_i + x_i'$ where $0 \leq Y_i \leq x_i^{max} - x_i'$. Let Y denote a set of variables Y_i. Then the new objective function will take a form:

$$\max f = \sum_i c_i(Y_i + x_i') = \sum_i c_i x_i' + \sum_i c_i Y_i$$

Since $\sum_i c_i x_i'$ is constant, it is sufficient to maximize now the sum $\sum_i c_i Y_i$. Denoting $T = T^{(1)} = 1 - \sum_i x_i'$, $U_i' = U_j - \sum_{i, y_i \in Q_j} x_i'$, and $y_i^{max} = x_i^{max} - x_i'$, we obtain a simpler optimization problem 2 over Y:

$$\max f = \sum_i c_i Y_i \tag{E.5}$$

$$s.t. \quad \sum_i Y_i = T \tag{E.6}$$

$$0 \leq \sum_{i, y_i \in Q_j} Y_i \leq U_j', \forall K_j \tag{E.7}$$

$$0 \leq Y_i \leq y_i', \forall y_i \in Y \tag{E.8}$$

We process the variables in the same order:

$$y_i \leftarrow \min\{y_i^{max}, U_j, T^{(i)} | y_i \in Q_j\}$$

and update:

$$U_j \leftarrow U_j - y_i \tag{E.9}$$

$$y_i^{max} \leftarrow y_i^{max} - y_i \tag{E.10}$$

$$T^{(i+1)} = T^{(i)} - y_i \tag{E.11}$$

We stop when $T^{(i+1)} = 0$ or after all variables are processed. We can guarantee that $T^{(n+1)} = 0$ (which means that sum-to-1 constraint is satisfied) at the end of processing as long as $\sum_{x_k \in X} x_k^{max} \geq 1$. Otherwise, feasible solution does not exist. Denote final solution as $x^1 = \{x_1^1, ..., x_n^1\}$.

THEOREM **E.0.1** *Algorithm computes a maximum value of the objective f-n* f.

Proof.

1. The obtained solution is feasible. As we already mentioned, the lower bounds are satisfied as long as $\sum_j L_j \leq 1$ and $\forall j$, $\sum_{x_k \in Q_j} x_k^{max} \geq L_j$. The upper bounds are not violated by construction. The sum-to-1 is also satisfied by construction as long as $\sum_{x_k \in X} x_k^{max} \geq 1$.

2. Now we prove that the solution is optimal.

Note: we can assume without loss of generality that all variables' coefficients in any one knapsack K_j are different. If not, we can group several variables x_i into one variable whose upper bound equals the sum of the upper bounds of the constituents. We are only interested in the total mass assigned to this subset of variables. How the mass is really distributed among those x_i's does not affect neither the value of the objective function nor the total load in knapsack K_j.

2.1. We prove that any optimal solution will have an assignment of values as defined above.

Let $x' = \{x'_1, ..., x'_n\}$ denote some optimal solution. Assume $\exists x'_i \in x'$ s.t. $x'_i < x^0_i$. Assume x_i is assigned to knapsack L_j. Note that $\sum_{x_k \in Q_j} x'_k = L_j$. Since $x'_i < x^0_i$ but $\sum_{x_k \in L_j} x'_k \geq L_j$ (the solution is feasible), then there must exist variable $x_l \in Q_j$ s.t. $x'_l > x^0_l$.

Case 1: $c_l > c_i$. This is impossible. Since we assign values to variables in order from largest coefficients to smallest, variable x_l has been assigned value x^0_l prior to x_i. By definition, variable x_l is assigned either value L_j or its maximum value. If variable x_l was assigned a maximum value L_j, then the value x^0_i would be equal to 0 which contradicts our assumption. If variable x_l was assigned its maximum value x^{max}_l, then solution x' is not feasible.

Case 2: $c_l < c_i$. We can reduce the value c_l down to c^0_l and increase value of c_i by the same amount. Let $\delta = c'_l - c^0_l$. Then, we set $c_l = c'_l - \delta$ and $c_i = c'_i + \delta$. By doing so, we preserve the feasibility of solution. We also increase the value of the objective function by $(c_i - c_l)\delta$ which means that x' is not an optimal solution. Hence, this scenario is also impossible.

Thus, any optimal feasible solution x' guarantees for all $x_i \in X$ that $x'_i \geq x^0_i$. Thus, the partial assignment of values obtained after first pass always can be extended to an optimal solution. Then, $\max f(x) = \sum_i c_i x^0_i + \max \sum_i c_i y_i$. Thus, as long as we can prove that y^1 is optimal, then algorithm A finds an optimal solution.

2.2. We prove that the solution to the optimization problem 2 is optimal.

Let y' be some optimal solution. We need to show that sum $y'_i > y^0_i$.

First, consider variable y_1 in knapsack K_1. Assume $y'_1 > y^1_1$. Recall how we picked the value of y_1.

If $y^0_1 = T$, then it is impossible for $y'_1 > y^1_1$ because solution x' would violate sum-to-1 constraint.

If $y^0_1 = U_1$, then $y'_1 > y^1_1$ is also impossible because solution x' would violate sum-to-1 constraint.

If $y^0_1 = y^{max}_i$ then $y'_1 > y^1_1$ is also impossible because solution x' would violate sum-to-1 constraint.

Thus, consider case of $y'_i < y^0_k$.

If $y^1_1 = T$, then since coefficient c_1 is the largest, then $c_1 y^0_1 \geq sum_i c_i y'_i$. Thus, either $y^0_1 = U_1$ or $y^0_1 = y^{max}_i$.

Since the sum of all variables equals to 1, then there exists a variable y_k s.t. $y'_k > y^0_k$.

If y_k is in the same knapsack as y_1, then we can reduce y_k from y'_k down to y^0_k and increase y_1 by the same amount. Since the coefficient of y_1 is the largest, that would increase the value of the objective function which contradicts assumption that x' is optimal. This means that for all y_j in knapsack K_1, $y'_j \leq y^0_k$. Hence, y_k must belong to a different knapsack. Next we analyze the relationship between coefficients c_1 and c_k of x_1 and x_k.

Case 1. $c_k > c_1$. Impossible. Coefficient c_1 is the largest.

Case 2. $c_k < c_1$. Impossible. If that was true, we could shift some weight from c_k to c_1 in solution x' without violating constraints and yield a solution with larger objective f-n value contradicting assumption that x' is optimal.

Thus, the only possibility is that $c_k = c_1$. But this is impossible too. We use the same line of reasoning as we used to prove that y'_1 cannot be greater than y^0_1. We consider the possibilities of how we picked the value of y_k.

If $y^0_k = U_j$, then $y'_k > y^1_k$ is also impossible because solution x' would violate sum-to-1 constraint.

If $y^0_k = y^{max}_i$ then $y'_k > y^1_k$ is also impossible because solution x' would violate sum-to-1 constraint.

If $y^0_k = T^{(k)}$, then x_k was the last variable processed. The rest of the variables were assigned value 0. Then, in solution x^0, the maximum total load T is distributed over only variables with maximal coefficients equals to c_1. Then, the value of x^0 is maximal and it is indeed an optimal solution.

Thus, we proved that either $x^0_1 = x'_1$ or the solution x^0 with assignment $x_1 = x^0_1$ is optimal. Either case implies that assignment $x_1 = x^0_1$ extends to an optimal solution. Namely, we can re-write our objective f-n as:

$$f = c_1 x^0_1 + \max \sum_{i>1} c_i x_i$$

Thus, the problem exhibits optimal substructure. We can prove in a similar manner that the solution to the subproblem over $n-1$ variables has the same properties. Namely, we can show for variable x_2 that either $x^0_2 = x'_2$ or the solution $x^0_2, ..., x^0_n$ with assignment $x_2 = x^0_2$ is optimal. Continuing recursively in similar manner we prove that either solutions x^0 and x' are identical or x' is optimal. Proof is complete. ■