UNIVERSITY OF CALIFORNIA,
IRVINE

Sampling Algorithms for Probabilistic Graphical Models with Determinism

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Sciences

by

Vibhav Giridhar Gogate

2009

DEDICATION

To my lovely wife Tripti,
my son Neeraj,
my parents,
my sister Kirti, her husband Ajay,
and my niece Anisha.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First I would like to thank Rina Dechter, my advisor through most of my graduate career. She has shown me a vision of what Artificial Intelligence in general, and what probabilistic and constraint reasoning in particular, can and should be. I am graduating with a clear idea of what a scientific researcher should be because of Rina's example. Her high standards, hard working practices and unfettered commitment to excellence have made a profound impact on me and have challenged me to do the same. Needless to say that most of the work presented in the thesis benefited greatly from her comments. Without her, this thesis and most of my papers would have been utterly incomprehensible. I would like to thank other members of my committee – Alex Ihler, Padhraic Smyth and Max Welling – for their time and comments, for their constant encouragement and for their enormous help on job search.

I would also like to thank Professor Sharad Mehrotra who has served as my mentor ever since I have met him. When funding was tight, he 'rescued' my Ph.D. by supporting me through a large NSF grant titled RESCUE (award numbers IIS-0412854, IIS-0331707) which stands for "Responding to Crisis and Unexpected Events". Sharad always took a keen interest in my research and his insights and broad knowledge helped me enormously.

My graduate study was helped supported by the NSF grants IIS-0086529, IIS-0412854, IIS-0331707 and IIS-0412854, by the NIH grant R01-HG004175-02, and by the Donald Bren School of Information and Computer Science at University of California, Irvine.

I also wish to express my gratitude to our graduate student advisors Kris Bolcer, Milena Wypchlak and Gina Anzivino and to all ICS faculty for creating an inspiring environment in our department. Many thanks to current and former members of Rina's group: Dan Frost, Kalev Kask, Bozhena Bidyuk, Robert Mateescu, Radu Marinescu, Lars Otten, Emma Rollon, Tuan Nguyen and Natasha Flerova for many countless interesting conversations. Fellow graduate students in our department, both past and present, have been a great help to me: Jonathan Hutchins, Ramaswamy Hariharan, Guy Yosiphon, Srikanth Agaram, Bijit Hore, Ravi Chandra Jammalamadaka, Vidhya Balasubramanian, Chaitanya Desai, Chaitanya Chemudugunta, Arthur Asuncion and Ian Porteous.

Next, I would like to thank Professor Elise Turner, my Master's advisor from University of Maine, Orono and her husband Professor Roy Turner. Elise and Roy introduced me to Artificial Intelligence and taught me "how to think like a computer scientist". Elise passed away in August, 2008 and she will be greatly missed.

This thesis (obviously) would not have been possible without the endless love and support of my wife, Tripti. She stood firmly with me through the highs and lows of working on a Ph.D. She always patiently listened to all my ideas, wrote some GUI interfaces in Java and C++ to help sell my research and more importantly made sure that I'm at least 30 lb heavier than what I was when we got married.

Finally, I would like to thank my parents, my sister Kirti and her husband Ajay for their support. They always encouraged me to do what I loved to do.

CURRICULUM VITAE

Vibhav Giridhar Gogate

EDUCATION

Ph.D. Information and Computer Science, 2009

Donald Bren School of Information and Computer Science

University of California, Irvine

Dissertation: Sampling Algorithms for Probabilistic Graphical models with Determinism

Advisor: Rina Dechter

M.S. Computer Science, 2002

University of Maine, Orono

B.E. Computer Engineering, 1999

Mumbai University, Maharashtra, India.

PUBLICATIONS

[1] Vibhav Gogate and Rina Dechter: *Approximate Solution Sampling (and Counting) on AND/OR Spaces*. In 14th International Conference on Principles and Practice of Constraint Programming (CP) 2008. Pages 534-538.

[2] Vibhav Gogate and Rina Dechter: *AND/OR Importance Sampling*. In 24th Conference on Uncertainty in Artificial Intelligence (UAI) 2008: 212-219.

[3] Vibhav Gogate and Rina Dechter: *Studies in Solution Sampling*. In 23rd Conference on Artificial Intelligence (AAAI) 2008. Pages 271-276.

[4] Vibhav Gogate, Bozhena Bidyuk and Rina Dechter: *Studies in Lower Bounding Probability of Evidence using the Markov Inequality*. In 23rd Conference on Uncertainty in Artificial Intelligence (UAI) 2007. Pages 141-148.

[5] Vibhav Gogate and Rina Dechter: *Approximate Counting by Sampling the Backtrack-free Search Space*. In 22nd Conference on Artificial Intelligence (AAAI) 2007. Pages 198-203.

[6] Vibhav Gogate and Rina Dechter: *SampleSearch: A Scheme that Searches for Consistent Samples*. In Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS) 2007. Pages 147-154.

[7] Vibhav Gogate and Rina Dechter: *A New Algorithm for Sampling CSP Solutions Uniformly at Random*. In 12th International Conference on Principles and Practice of Constraint Programming (CP) 2006. Pages 711-715.

[8] Vibhav Gogate and Rina Dechter: *Approximate Inference Algorithms for Hybrid Bayesian Networks with Discrete Constraints*. In 21st Conference on Uncertainty in Artificial Intelligence (UAI) 2005. Pages 209-216.

[9] Vibhav Gogate, Rina Dechter, Bozhena Bidyuk, Craig Rindt and James Marca: *Modeling Transportation Routines using Hybrid Dynamic Mixed Networks*. In 21st Conference on Uncertainty in Artificial Intelligence (UAI) 2005. Pages 217-224.

[10] Vibhav Gogate and Rina Dechter: *A Complete Anytime Algorithm for Treewidth*. In 20th Conference on Uncertainty in Artificial Intelligence (UAI) 2004. Pages 201-208.

[11] Kalev Kask, Rina Dechter and Vibhav Gogate: *Counting-Based Look-Ahead Schemes for Constraint Satisfaction*. In 10th International Conference on Principles and Practice of Constraint Programming (CP) 2004. Pages 317-331.

ABSTRACT OF THE DISSERTATION

Sampling Algorithms for Probabilistic Graphical Models with Determinism

By

Vibhav Giridhar Gogate

Doctor of Philosophy in Information and Computer Sciences

University of California, Irvine, 2009

Professor Rina Dechter, Chair

Mixed constraint and probabilistic graphical models occur quite frequently in many real world applications. Examples include: genetic linkage analysis, functional/software verification, target tracking and activity modeling. Query answering and in particular probabilistic inference on such graphical models is computationally hard often requiring exponential time in the worst case. Therefore in practice sampling algorithms are widely used for providing an approximate answer. In presence of deterministic dependencies or hard constraints, however, sampling has to overcome some principal challenges. In particular, importance sampling type schemes suffer from what is known as the rejection problem in that samples having zero weight may be generated with probability arbitrarily close to one yielding useless results. On the other hand, Markov Chain Monte Carlo techniques do not converge at all often yielding highly inaccurate estimates.

In this thesis, we address these problems in a two fold manner. First, we utilize research done in constraint satisfaction and satisfiability communities for processing constraints to reduce or eliminate rejection. Second, mindful of the time overhead in sample generation due to determinism, we both make and utilize advances in statistical estimation theory to make the “most” out of the generated samples.

Utilizing constraint satisfaction and satisfiability research, we propose two classes of sampling algorithms - one based on consistency enforcement and the other based on systematic search. The consistency enforcement class of algorithms work by shrinking the domains of random variables, by strengthening constraints, or by creating new ones, so that some or all zeros in the problem space can be removed. This improves convergence because of dimensionality reduction and also reduces rejection because many zero weight samples will not be generated. Our systematic search based techniques called SampleSearch manage the rejection problem by interleaving sampling with backtracking search. In this scheme, when a sample is supposed to be rejected, the algorithm continues instead with systematic backtracking search until a strictly positive-weight sample is generated. The strength of this scheme is that any state-of-the-art constraint satisfaction or propositional satisfiability search algorithm can be used with minor modifications. Through large scale experimental evaluation, we show that SampleSearch outperforms all state-of-the-art schemes when a significant amount of determinism is present in the graphical model.

Subsequently, we combine SampleSearch with known statistical techniques such as Sampling Importance Resampling and Metropolis Hastings yielding efficient algorithms for sampling solutions from a uniform distribution over the solutions of a Boolean satisfiability formula. Unlike state-of-the-art algorithms, our SampleSearch-based algorithms guarantee convergence in the limit.

As to statistical estimation, we make two distinct contributions. First, we propose several new statistical inequalities extending the one-sample Markov inequality to multiple samples which can be used in conjunction with SampleSearch to probabilistically lower bound likelihood tasks over mixed networks. Second, we present a novel framework called “AND/OR importance sampling” which generalizes the process of computing sample mean by exploiting AND/OR search spaces for graphical models. Specifically we provide a spectrum of AND/OR sample means which are defined on the same set of samples but derive different estimates trading variance with time. At one end is the AND/OR sample tree mean which has smaller variance than the conventional OR sample tree mean and has the same time complexity. At the other end is the AND/OR graph sample mean which has even lower variance but has higher time and space complexity. We demonstrate empirically that AND/OR sample means are far closer to the exact answer than the conventional OR sample mean.

Chapter 1

Introduction

Representation and reasoning are fundamental computer science concepts. Representation involves translating real world knowledge into a model intended for computer processing while reasoning involves solving problems that arise in the real world by querying the computer model. In this thesis, we will focus on a widely used graph based knowledge representation and reasoning framework for dealing with uncertainty called *Probabilistic Graphical models*.

Probabilistic graphical models such as Bayesian and Markov networks use the well studied language of probability theory for modeling real world uncertain phenomenon. Since its advent in the late 17^{th} century, probability theory has been widely used in many fields such as biology, statistics and physics; to name a few. The semantics of probability theory are quite natural: the world consists of random variables and the joint distribution over the variables represents complete knowledge about them. Unfortunately, storing the joint distribution is exponential in the number of random variables and therefore clearly infeasible for any realistic application.

Probabilistic graphical models such as Bayesian and Markov networks address these concerns by representing the joint distribution compactly using a graph-based representation. In a Bayesian network, the joint distribution is represented by a Directed Acyclic Graph (DAG) where each node in the DAG corresponds to a random variable and is associated with a Conditional Probability Distribution (CPTs) of the variable given its parents in the DAG. The joint distribution of a Bayesian network is a product of its CPTs. In a Markov network, the joint distribution is represented using an undirected graph, with potential functions defined on the cliques in the graph. The joint distribution represented by a Markov network is the normalized product of the potential functions.

In this thesis, we will focus on Bayesian and Markov networks which may contain also significant deterministic relationships. Deterministic relationships can occur either in the form of zeros inside the CPTs or in the potential functions, or may come in the form of explicit external constraints that restrict the values that the random variables can take. Throughout the thesis, we will use the framework of mixed networks [92] to present our new algorithmic contributions. The mixed network framework was introduced for the purpose of addressing the representational and computational aspects of representing deterministic and probabilistic information in graphical models. The central aim is to allow exploiting the power of constraint processing for efficient probabilistic inference.

Constraint networks and propositional theories are common frameworks for modeling deterministic information. A constraint network uses the declarative language of constraints for representing knowledge. In a constraint network, we have a set of variables which can be assigned values from a finite domain and a set of constraints which restrict these assignments. The main task is to find a solution or a model i.e. assign values to all variables without violating any of the constraints. Another is to count all such assignments referred to as model or solution counting. The constraints can be associated with an undirected graph called the constraint graph in which nodes represent variables and edges connect any two variables involved in a constraint. Thus, a constraint network is a deterministic graphical model. One example of a constraint network is graph coloring. Here, the variables are vertices, the values are the available colors and the constraints enforce the condition that no two adjacent vertices can be assigned the same color. The constraint graph in this case is the graph to be colored.

It should be noted however that the use of constraints within the confines of a mixed network adds nothing new to the representation power of a Bayesian or a Markov network. In principle, constraints and in general a constraint network can be easily incorporated into a Bayesian or Markov network. In case of a Markov network, one can specify each constraint as a $0/1$ potential function in which a 1 indicates allowed tuples while a 0 indicates otherwise (and which may add additional edges in the undirected graph). In a Bayesian network, we can embed a constraint by modeling it as a Conditional Probability Table (CPT). In this approach, one has to add a new variable for each constraint that is perceived as its effect (child node) in the corresponding causal relationship and then to clamp its value to true i.e. set it as evidence.

The main shortcoming, however, of any of the above approaches is computational. Constraints have special properties that render them computationally attractive. When constraints are disguised as probabilistic relationships, their computational benefits may be hard to exploit. In particular, the power of constraint inference and constraint propagation may not be brought to bear. Therefore in mixed networks, the identity of the respective relationships, as constraints or probabilities are maintained explicitly, so that their respective computational power and semantic differences can be made vivid and easy to exploit.

Perhaps the most fundamental issue for any graphical model is the problem of *inference*. Given a (mixed) graphical model that represents some joint probability distribution $\mathcal{P}(\mathbf{X})$ over a set of variables $\mathbf{X}$, we are required to answer a specific query relative to an event $\mathbf{E} = \mathbf{e}$, which is an assignment of values to a subset of variables; also called as evidence. Examples of such queries are computing the probability of evidence $P(\mathbf{E} = \mathbf{e})$ (also called as likelihood computation) or computing the posterior marginal distribution $P(X|\mathbf{E} = \mathbf{e})$ (also called as belief updating). For example, in medical diagnosis [110] one is interested in finding the probability that a person has a disease given a collection of symptoms; or in activity modeling [86] one is interested in assessing the likelihood that a person is going home given his current location and the time of the day.

Inference algorithms are developed to answer queries over graphical models. The algorithms can be exact or approximate. Exact algorithms however are applicable to networks which admit a relatively simple graph representation having low treewidth. When the graph structure is too complex, approximate algorithms must be used.

Approximate algorithms for graphical models are of two primary types: (i) bounded message passing based or (ii) sampling based. They approximate the two types of exact algorithms which are tree-clustering based or search based. Examples of message passing approximate schemes are mini bucket elimination [39], loopy belief propagation [106] and generalized belief propagation [132] while the main types of sampling schemes are importance sampling or Markov Chain Monte Carlo sampling. Since the focus of this thesis is on sampling schemes, we will provide some general description in the following paragraphs. A detailed overview of all the schemes is given in section 1.4.

The main idea in importance sampling type algorithms is to express the likelihood and marginal estimation tasks as the expected value of some random variable whose probability distribution is selected by the algorithm designer. The only requirement is that the special distribution called the proposal (or importance or trial) distribution should be easy to sample from (typically in a sequential variable by variable manner). Importance sampling then generates samples from the proposal distribution and aims to approximate the true expected value (also called the true average) by a weighted average over the samples (also called sample average). The sample average has several desirable properties like: (a) unbiasedness, namely that the expected value of the sample average equals the true average, and (b) the mean squared error between the sample average and the true average decreases with the number of samples.

Markov Chain Monte Carlo (MCMC) sampling schemes are based on an utterly different principle. If we are able to generate samples from the posterior distribution $\mathcal{P}(\mathbf{X}|\mathbf{e})$, then we can estimate the marginal probability $P(X = x|\mathbf{e})$ for any variable $X \in \mathbf{X}$ as the number of times $X = x$ appears in the generated samples. Unfortunately, $\mathcal{P}(\mathbf{X}|\mathbf{e})$ is typically very difficult to sample from and therefore MCMC schemes construct a Markov chain that has $\mathcal{P}(\mathbf{X}|\mathbf{e})$ as its equilibrium or stationary distribution. A Markov chain consists of a sequence of states and a transition-rule for moving from state $\mathbf{x}$ to state $\mathbf{y}$. The state of the chain after a large number of steps is then used as a sample from the desired posterior distribution. The quality of the samples improves as a function of the number of steps. It is usually not hard to construct Markov chains for a graphical model (see Section 1.4.3). However, convergence to the posterior distribution depends upon the Markov chain being *ergodic* requiring a positive probability of moving from a state $\mathbf{x}$ to a state $\mathbf{y}$ in a finite number of steps.

In presence of hard constraints, however, sampling becomes much harder, in the sense that some guarantees that hold for a strictly positive probability space break down. Specifically, given a bounded number of evidence nodes, a small real number $\epsilon < 1$, and a strictly positive Bayesian network, importance sampling yields a randomized polynomial time approximation (polynomial in the number of evidence nodes and ϵ) of the posterior marginal distribution with relative error ϵ [22]. However, in presence of determinism guaranteeing an approximation with relative error ϵ is NP-hard.

Indeed, it is well known empirically that importance sampling and MCMC schemes perform quite poorly in presence of determinism. In particular, importance sampling suffers from the so-called *rejection problem*. The rejection problem occurs when the proposal distribution does not faithfully capture all the zeros or constraints in the probability distribution of interest. Consequently, samples generated from the proposal distribution may have zero weight and will be essentially rejected because they contribute nothing to the sample average. In case of Markov Chain Monte Carlo techniques like Gibbs sampling, guarantees like ergodicity break down due to determinism rendering the space disconnected, meaning that not all states are reachable from a given current state. In such cases, MCMC techniques may not even converge.

The contribution of this thesis is in investigating and addressing these deficiencies of sampling based techniques in presence of determinism. Over the course of the next five chapters, we will present a range of sampling techniques which will mitigate the rejection and address convergence issues under different sets of input assumptions and which will be specialized for different queries such as computing the posterior marginals, computing the probability of evidence and generating samples from the posterior distribution.

1.1 Thesis Outline and Contributions

Our approach to managing sampling in presence of determinism has two components. First, we utilize research done in the constraint satisfaction and satisfiability communities for processing constraints to reduce or eliminate rejection and to improve convergence. Second, mindful of the time overhead in sample generation due to determinism, we both make and utilize advances in statistical estimation theory to make the "most" out of the generated samples.

We propose two classes of sampling algorithms for reducing or eliminating rejection - one based on consistency enforcement and the other based on systematic search.

The consistency enforcement approach presented in Chapter 2 works by tightening the domains of the random variables, by strengthening the constraints, or by creating new ones, so that some or all zeros in the problem space can be removed. This reduces the problem space on which sampling is carried out which not only improves convergence but also reduces or altogether eliminates rejection because either some or all zeros, which would otherwise be sampled, are removed. We applied our new importance sampling technique to learn a dynamic Bayesian network which models car travel activities of individuals using GPS data. Given a time of day and current location via the persons' GPS device, the Bayesian network can be used to predict the person's destination and the route to the destination given his current location.

In Chapter 3, we present a family of algorithms called SampleSearch which combine systematic backtracking search with sampling. Unlike consistency enforcement schemes which reduce rejection in a pre-processing manner, SampleSearch handles the rejection problem during the sampling process itself. In this scheme, when a sample is about to be rejected due to inconsistency, the algorithm treats it as a dead-end during search, and instead of aborting, it backtracks as is common in systematic backtracking search, until a strictly positive weight sample is generated. The strength of this scheme is that any state-of-the-art constraint satisfaction or propositional satisfiability search algorithm can be used with minor modifications. However, SampleSearch by itself cannot facilitate unbiased or asymptotically unbiased estimators because search introduces a bias which is not articulated by the underlying proposal distribution. We therefore characterize this bias, showing that the sampling distribution of SampleSearch is the backtrack-free distribution of the original proposal distribution which helps derive unbiased or asymptotically unbiased estimators. Through large scale experimental evaluation, we show that SampleSearch outperforms all state-of-the-art schemes when a substantial amount of determinism is present in the graphical model.

In Chapter 4, we focus on the solution sampling task which requires generating solutions of a constraint satisfaction problem or a satisfiability formula, such that each solution is equally likely. While we can use SampleSearch to generate random solution samples, the distribution over the generated solution samples would converge to the backtrack free distribution which is biased away from the uniform distribution over the solutions. We correct this bias using well known statistical techniques such as Sampling/Importance Resampling and Metropolis-Hastings which guarantee convergence to the uniform distribution in the limit. Such guarantees are not available for other state-of-the-art schemes. Our extensive experimental evaluation demonstrates that our solution sampling techniques substantially outperform alternative approaches both in terms of speed of sample generation and in terms of accuracy.

Chapter 5 presents lower bounding schemes for desired queries such as probability of evidence and solution counting based on the well known Markov inequality. A straight forward application of Markov inequality, however, yields bad estimates - a fact which is well known in statistics. We show that Markov inequality is weak because its estimates are based only on one sample. We derive several new statistical inequalities based on multiple samples which guarantee that the lower bound will likely increase as more samples are drawn. Our empirical results capitalize on the strength of SampleSearch demonstrating the virtue of our lower bounding schemes in the context of mixed networks.

In Chapter 6, we introduce a family of alternative statistics for estimating the sample average based on the AND/OR search spaces for graphical models. AND/OR search spaces capture problem decomposition in search using AND nodes. We show that problem decomposition uncovered through the AND/OR search space yields a family of sample means having smaller variance. The new family is defined on the same set of samples but yields different estimates that trade time with variance. At one end of the spectrum we have the AND/OR sample tree mean which has the same time complexity as conventional importance sampling but has lower mean squared error or variance. At the other end is AND/OR sample graph mean which requires more time and space but has the lowest mean squared error or variance. Via large scale experimental evaluation, we demonstrate that given the same amount of time the AND/OR sample means are far closer to the true mean than conventional sample mean (output by importance sampling).

Finally in Chapter 7, we conclude by surveying the contributions of this thesis, and outline directions for future research.

1.2 Overview of Graphical Models

Many practical applications such as genetic linkage analysis [46, 1], car travel activity modeling [86, 53] functional verification [4, 31], target tracking [105], machine vision [45, 85], medical diagnosis [94, 110] and music parsing [111] typically involve a large collection of random variables, which interact with each other in a non trivial manner. Consequently, representing these applications using a joint probability distribution is infeasible because of exponential memory requirements. For example, a joint distribution over 1000 random binary variables requires $2^{1000} - 1$ entries.

Graphical models such as Bayesian and Markov networks address these concerns by representing the joint distribution compactly in a factored form using a graph-based representation. In a Bayesian network, the joint distribution is represented by a Directed acyclic graph (DAG) with every node in the DAG corresponding to a random variable that is associated with a conditional probability distribution of the variable given its parents in the DAG. In a Markov network, the joint distribution is compactly represented using an undirected graph, with potential functions defined on the cliques in the graph. The joint distribution represented by a Markov network is defined as the normalized product of the potential functions.

In this section, we introduce and compare several different families of graphical models, such as Bayesian networks, Markov networks, constraint networks and the mixed networks framework which contains both probabilistic information as well as constraints.

1.2.1 Notation

A discrete graphical model consists of a set of variables, each of which can take a value from a finite domain and a set of functions defined over the variables. We denote variables by upper case letters (e.g. $X, Y, \ldots$) and values of variables by the corresponding lower case letters (e.g. $x, y, \ldots$). Sets of variables are denoted by bold upper case letters, (e.g. $\mathbf{X} = \{X_1, \ldots, X_n\}$) while the corresponding values are denoted by bold lower case letters (e.g. $\mathbf{x} = \{x_1, \ldots, x_n\}$). $X = x$ denotes an assignment of value to a variable while $\mathbf{X} = \mathbf{x}$ denotes an assignment of values to all variables in the set. We denote by $\mathbf{D}_i$ the set of possible values of X_i (also called as the domain of X_i) and by $\mathbf{D}_{\mathbf{X}}$ the Cartesian product $\mathbf{D}_1 \times \mathbf{D}_2 \times \ldots \mathbf{D}_n$ of the domains of all variables in the set $\mathbf{X}$.

$\sum_{\mathbf{x} \in \mathbf{X}}$ denotes the sum over the possible values of variables in $\mathbf{X}$, namely, $\sum_{x_1 \in X_1} \sum_{x_2 \in X_2} \ldots \sum_{x_n \in X_n}$. The expected value $\mathbb{E}_Q[X]$ of a random variable X with respect to a distribution Q is defined as: $\mathbb{E}_Q[X] = \sum_{x \in X} xQ(x)$. The variance $V_Q[X]$ of X is defined as: $V_Q[X] = \sum_{x \in X}(x - \mathbb{E}_Q[X])^2 Q(x)$. Often for clarity, we will write $\mathbb{E}_Q[X]$ as $\mathbb{E}[X]$ and $V_Q[X]$ as $V[X]$.

We denote the projection of an assignment $\mathbf{x}$ to a set $\mathbf{S} \subseteq \mathbf{X}$ by $\mathbf{x}_{\mathbf{S}}$. Given an assignment $\mathbf{y}$ and $\mathbf{z}$ to the sub-sets $\mathbf{Y}$ and $\mathbf{Z}$ of $\mathbf{X}$, such that $\mathbf{X} = \mathbf{Y} \cup \mathbf{Z}$, $\mathbf{x} = (\mathbf{y}, \mathbf{z})$ denotes the composition of $\mathbf{y}$ and $\mathbf{z}$.

1.2.2 Graph concepts

We describe some common notations, definitions and results from graph theory. For more information see [44, 20, 106, 29].

DEFINITION **1** (**Directed and Undirected Graph**). *A directed graph is a pair $G = (\boldsymbol{X}, \boldsymbol{E})$, where $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ is a set of vertices, and $\boldsymbol{E} = \{(X_i, X_j) | X_i, X_j \in \boldsymbol{X}\}$ is the set of directed edges. An undirected graph is defined similarly to a directed graph, but there is no directionality associated with the edges i.e (X_i, X_j) is equal to (X_j, X_i).*

A clique is a set of nodes in G for which all pairs are connected by an edge. If the entire graph forms a clique, it is said to be complete. A path between nodes X_0, X_d is a sequence of distinct nodes $(X_0, X_1, \ldots, X_{d-1}, X_d)$ such that for all $l = 1, \ldots, d$, $(X_{l-1}, X_l) \in \mathbf{E}$. A cycle, is a path which starts and ends with the same node i.e. $X_0 = X_d$. If there is a path between every pair of nodes in G, it is connected. If an edge joins two non-consecutive vertices within some cycle, then it is called a chord.

A graph that has no undirected cycles is called a tree. A chordal graph is an undirected graph which has no chord less cycle.

Given a directed edge $(X_i, X_j) \in \mathbf{E}$, X_i is called the parent of X_j and X_j is called the child of X_i. The set of parents of a node X_i is denoted by either $pa(X_i)$ or $\mathbf{pa}_i$ while the set of children are denoted by $chi(X_i)$ or $\mathbf{chi}_i$. A node X_j is called the descendant of X_i iff there is a directed path starting at X_i and ending at X_j. Similarly, a node X_j is called the ancestor of X_i iff X_i is the descendant of X_j. A leaf node has no descendants while a root node has no ancestors. A directed graph is acyclic (also called as Directed Acyclic Graph or a DAG) if it has no directed cycles.

DEFINITION **2** (**Moral Graph**). *A moral graph of a directed graph is an undirected graph obtained by connecting all parents of a node to each other and removing direction.*

DEFINITION **3** (**Induced Width**). *An ordered graph is a pair (G, o) where G is an undirected graph, and $o = (X_1, ..., X_n)$ is an ordering of the nodes. The width of a node X is the number of the neighbors of X that precede it in the ordering. The width of an ordering o, is the maximum width over all nodes. The induced width of an ordered graph, $w_G(o)$,*

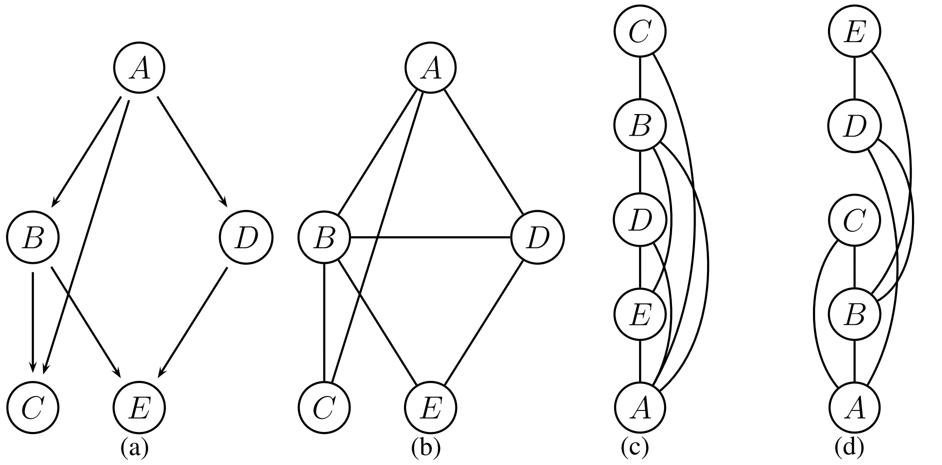


Figure 1.1: Figure showing (a) Directed Acyclic Graph (DAG), (b) Moral graph of DAG in (a), (c) Induced graph along the ordering (A, E, D, B, C), (d) Induced graph along the ordering (A, B, C, D, E)

is the width of the induced ordered graph obtained as follows: nodes are processed from last to first; when node X_i is processed, all its preceding neighbors are connected. The induced width of a graph, denoted by t^* (or w^*), is the minimum induced width over all its orderings.

EXAMPLE **1.** *Figure 1.1 (a) shows a directed acyclic graph with 5 vertices. Figure 1.1 (b) shows the moral graph of the directed acyclic graph of Figure 1.1 (a). Figure 1.1 (c) shows the induced graph along the ordering (A, E, D, B, C), whose induced width is 3, while Figure 1.1 (d) shows the induced graph along the ordering (A, B, C, D, E) whose induced width is 2.*

DEFINITION **4** (**Tree Decomposition and Treewidth**). *A tree decomposition of an undirected graph $G(\boldsymbol{X}, \boldsymbol{E})$ is a pair (T, Ψ) where $T = (\boldsymbol{V}, \boldsymbol{F})$ is a tree and $\Psi = \{\psi_V | V \in \boldsymbol{V}\}$ is a family of subsets of $\boldsymbol{X}$ such that the following conditions are satisfied:*

1. $\cup_{\psi_V \in \Psi} \psi_V = \boldsymbol{X}$.
2. *For each edge $(X_i, X_j) \in \boldsymbol{E}$, there exists a $\psi_V \in \Psi$ such that both X_i and X_j belong to ψ_V.*

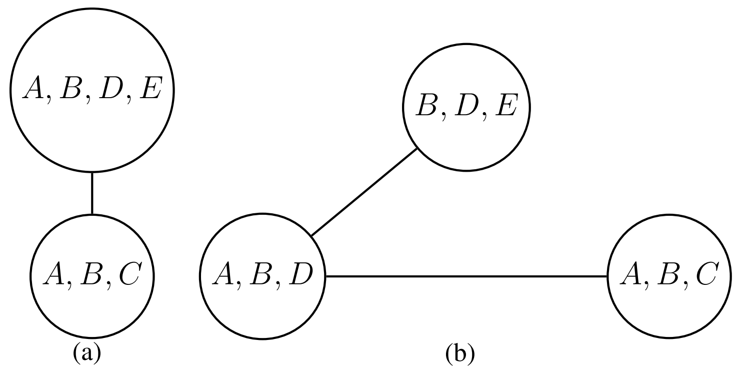


Figure 1.2: Figure showing two valid tree decompositions for the moral graph in Figure 1.1(b)

3. *For all $X_j \in \boldsymbol{X}$, there is a set of nodes $\{V \in \boldsymbol{V} | X_j \in \psi_V\}$ that forms the connected sub-tree of T (running intersection property).*

The treewidth of a tree-decomposition is given by: $max_{V \in \boldsymbol{V}}(|\psi_V| - 1)$. The treewidth of a graph G denoted by t^* equals the minimum treewidth over all possible tree decompositions of G.

Tree decomposition and induced graphs are related as follows. If we connect the maximal cliques of the induced graph in a tree structure that obeys the running intersection property, we get a tree decomposition. In this case, the induced width of the induced graph is equal to the treewidth of the tree decomposition [29]. The task of finding the treewidth of a graph is NP-complete [2]. In the past two decades, substantial research has focused on designing exact and approximate algorithms for finding the treewidth [74, 76, 79, 52, 42].

EXAMPLE **2.** *Figure 1.2 (a) and (b) show two valid tree decompositions for the moral undirected graph in Figure 1.1 (b). The treewidth of the two tree decompositions is* 3 *and* 2 *respectively. The tree decompositions in Figure 1.2 (a) and (b) are obtained from the induced graphs given in Figures 1.1(c) and (d) respectively by connecting the maximum cliques in a tree structure such that they satisfy the running intersection property.*

1.2.3 Bayesian Networks

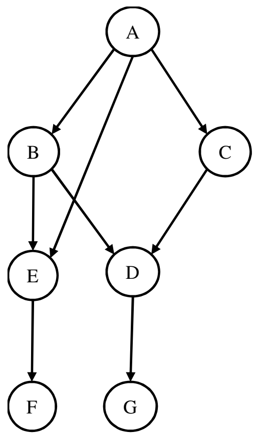
DEFINITION 5 **(Graphical Models).** *A discrete graphical model $\mathcal{G}$ is a 3-tuple $\langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$ where $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ is a finite set of variables, $\boldsymbol{D} = \{\boldsymbol{D}_1, \ldots, \boldsymbol{D}_n\}$ is a finite set of domains where $\boldsymbol{D}_i$ is the domain of variable X_i and $\boldsymbol{F} = \{F_1, \ldots, F_n\}$ is a finite set of discrete-valued functions. Each function F_i is defined over a subset of variables $\boldsymbol{S}_i$ called its scope and is denoted by $scope(F_i)$. The graphical model represents a product of all of its functions.*

Each graphical model is associated with a primal graph which captures the dependencies present in the model.

DEFINITION 6 **(Primal Graph).** *The primal graph of a graphical model $\mathcal{G} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$ is an undirected graph $G(\boldsymbol{X}, \boldsymbol{E})$ which has variables of $\mathcal{G}$ as its vertices and an edge between two variables that appear in the scope of a function.*

DEFINITION 7 **(Bayesian or Belief Networks).** *A Bayesian network is a graphical model $\mathcal{B} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{G}, \boldsymbol{P} \rangle$ where $G = (\boldsymbol{X}, \boldsymbol{E})$ is a directed acyclic graph over the set of variables $\boldsymbol{X}$. The functions $\boldsymbol{P} = \{P_1, \ldots, P_n\}$ are conditional probability tables $P_i = P(X_i|\boldsymbol{pa}_i)$, where $\boldsymbol{pa}_i = scope(P_i) \setminus \{X_i\}$ is the set of parents of X_i in G. The primal graph of a Bayesian network is same as the moral graph of G. When the entries of the CPTs are 0 and 1 only, they are called* deterministic or functional *CPTs. An evidence $\mathbf{E} = \boldsymbol{e}$ is an instantiated subset of variables.*

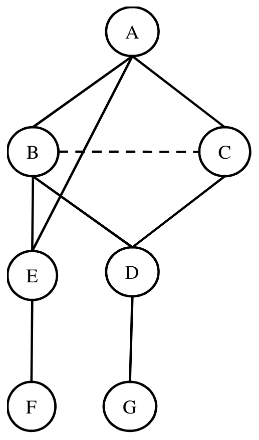
A Bayesian network represents the joint probability distribution given by $P_{\mathcal{B}}(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|\mathbf{pa}_i)$ and therefore can be used to answer any query defined over the joint distribution. In this thesis, we consider two queries: (a) computing the probability of evidence $P(\mathbf{E} = \mathbf{e})$ and (b) computing the posterior marginal distribution $P(X_i|\mathbf{E} = \mathbf{e})$ for each variable $X_i \in \mathbf{X}$.



(a)

D	B	C	P(D|B,C)
0	0	0	0.2
1	0	0	0.8
0	0	1	0.7
1	0	1	0.3
0	1	0	0.1
1	1	0	0.9
0	1	1	0.6
1	1	1	0.4

(b)



(c)

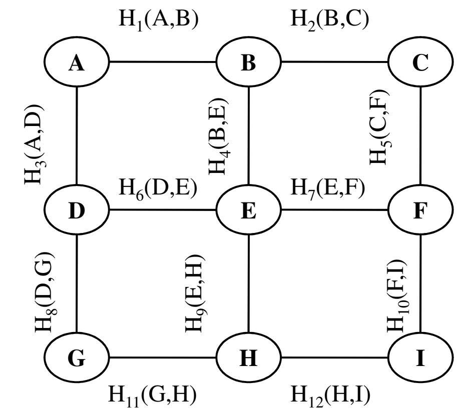
Figure 1.3: (a) An example Bayesian network, (b) An example CPT $P(D|B, C)$ and (C) Moral graph of the Bayesian network shown in (a)

EXAMPLE **3.** *Figure 1.3 (a) shows an example Bayesian network over seven variables* $\{A, B, C, D, E, F, G\}$*. The network depicts structural relationships between different variables. The conditional probability tables (CPTs) associated with variables* A*,* B*,* C*,* D*,* E*,* F *and* G *are* $P(A)$*,* $P(B|A)$*,* $P(C|A)$*,* $P(D|B, C)$*,* $P(E|A, B)$*,* $P(F|E)$ *and* $P(G|D)$ *respectively. An example CPT for* $P(D|B, C)$ *is given in Figure 1.3(b). Figure 1.3 (c) shows the network's moral graph which coincides with the primal graph. The Bayesian network represents the joint distribution:*

$$P(A, B, C, D, E, F, G) = P(A)P(B|A)P(C|A)P(D|B, C)P(E|A, B)P(F|E)P(G|D)$$

1.2.4 Markov Networks

DEFINITION **8** (**Markov Networks**)**.** *A Markov network is a graphical model* $\mathcal{T} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{H}\rangle$ *where* $\boldsymbol{H}= \{H_1, \ldots, H_m\}$ *is a set of potential functions where each potential* H_i *is a non-negative real-valued function defined over variables* $\mathbf{S}_i$*. The Markov network*



(a)

D	E	H_6(D,E)
0	0	20.2
0	1	12
1	0	23.4
1	1	11.7

(b)

Figure 1.4: (a) An example 3×3 square Grid Markov network (ising model) and (b) An example potential $H_6(D, E)$

represents a joint distribution over the variables ***X*** *given by:*

$$P(\boldsymbol{x}) = \frac{1}{Z}\prod_{i=1}^{m} H_i(\boldsymbol{x}) \quad where \;\; Z = \sum_{\boldsymbol{x}\in\boldsymbol{X}}\prod_{i=1}^{m} H_i(\boldsymbol{x})$$

where the normalizing constant Z is often referred to as the partition function.

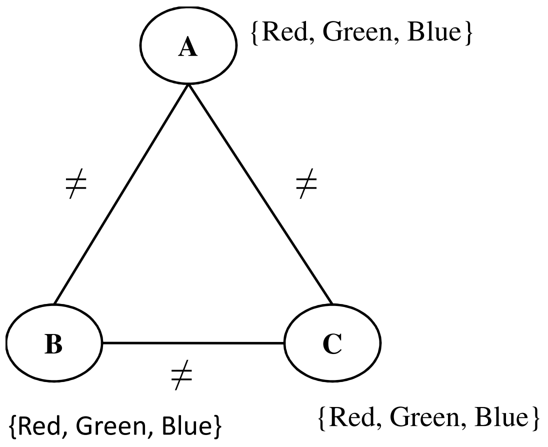
The primary queries over Markov networks are computing the posterior marginal distribution over all variables $X_i \in \mathbf{X}$ and finding the partition function.

EXAMPLE **4.** *Figure 1.4 shows a 3×3 square grid Markov network with 9 variables $\{A, B, C, D, E, F, G, H, I\}$. The twelve potentials are: $H_1(A, B)$, $H_2(B, C)$, $H_3(A, D)$, $H_4(B, E)$, $H_5(C, F)$, $H_6(C, D)$, $H_7(D, E)$, $H_8(D, G)$, $H_9(E, H)$, $H_{10}(F, I)$, $H_{11}(G, H)$ and*

$H_{12}(H, I)$. The Markov network represents the probability distribution formed by taking a product of these twelve functions and then normalizing. Namely,

$$P(A, B, \ldots, I) = \frac{1}{Z}\prod_{i=1}^{12} H_i$$

where Z is the partition function.



(a)

A	**B**	**C**
Red	**Blue**	**Green**
Red	**Green**	**Blue**
Blue	**Green**	**Red**
Blue	**Red**	**Green**
Green	**Red**	**Blue**
Green	**Blue**	**Red**

(b)

Figure 1.5: (a) An example constraint network showing a three coloring problem (b) The set of solutions of the constraint network

1.2.5 Constraint Networks

DEFINITION **9** (**Constraint Networks).** *A constraint network is a graphical model $\mathcal{R} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C} \rangle$ where $\boldsymbol{C} = \{C_1, \ldots, C_m\}$ is a set of constraints. Each constraint C_i is a 0/1 function defined over a subset of variables $\boldsymbol{S}_i$, called the scopes of the constraints. Given an assignment $\boldsymbol{S}_i = \boldsymbol{s}_i$, a constraint is satisfied if $C_i(\boldsymbol{s}_i) = 1$. A constraint can also be expressed by a pair $\langle R_i, \boldsymbol{S}_i \rangle$ where R_i is a relation defined over the variables $\boldsymbol{S}_i$ and contains all tuples $\boldsymbol{S}_i = \boldsymbol{s}_i$ for which $C_i(\boldsymbol{s}_i) = 1$. The primal graph of a constraint network is called the constraint graph.*

The primary query over a constraint network is to decide whether it has a solution i.e. to find an assignment $\mathbf{X} = \mathbf{x}$ to all variables such that all constraints are satisfied or to prove that no such assignment exists. The constraint network represents its set of solutions. Another important query is that of counting the number of solutions of the constraint network.

EXAMPLE **5.** *Figure 1.5(a) shows a graph coloring problem that can be modeled by a constraint network. The task is to color each vertex in such a way that no two vertices that share an edge have the same color. To encode the problem as a constraint network, we de-*

fine three variables $\{A, B, C\}$*. The domain of each variable is* $\{Red, Green, Blue\}$*. The constraints require variables sharing an edge to have different colors and can be modeled as binary "not-equal constraints". Figure 1.5(b) shows the set of solutions of the constraint network.*

1.2.6 Propositional Satisfiability

A special case of a constraint satisfaction problem is a *propositional formula* in conjunctive normal form (CNF). A formula F in *conjunctive normal form* (CNF) is a conjunction of *clauses* $Cl_1, \ldots, Cl_t$ (denoted as a set $\{Cl_1, \ldots, Cl_t\}$) where a clause is a disjunction of *literals* (propositions or their negations). For example, $Cl = (P \vee \neg Q \vee \neg R)$ is a clause, where P, Q and R are propositions, and P, $\neg Q$ and $\neg R$ are literals. The set of models or solutions of a formula F is the set of all truth assignments to all its symbols that do not violate any clause. Common queries in SAT are *satisfiability* i.e. finding a model or proving that none exists; also called the SAT problem, and *model counting* i.e. counting the number of models or counts.

Propositional satisfiability can be defined as a constraint network, where propositions correspond to variables with bi-valued domains {0, 1}, and constraints are represented by clauses (for instance, clause $(\neg A \vee B)$ allows all tuples over (A, B) except $(A = 1, B = 0)$).

1.2.7 Mixed Networks

Throughout the thesis, we will use the framework of mixed networks defined in [36, 92]. Mixed networks represent all the deterministic information explicitly in the form of constraints facilitating the use of constraint processing techniques developed over the past three decades for efficient probabilistic inference. This framework includes Bayesian, Markov and constraint networks as a special case. Therefore, many inference tasks become equivalent when we consider a mixed network view allowing a unifying treatment of all these problems within a single framework. For example, problems such as computing the probability of evidence in a Bayesian network, the partition function in a Markov network and counting solutions of a constraint network can be expressed as weighted counting over mixed networks.

DEFINITION **10** (**Mixed Network**). *[36, 92] A mixed network is a four-tuple $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C}\rangle$ where $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ is a set of random variables, $\boldsymbol{D} = \{\boldsymbol{D}_1, \ldots, \boldsymbol{D}_n\}$ is a set of domains where $\boldsymbol{D}_i$ is the domain of X_i, $\boldsymbol{F} = \{F_1, , \ldots, F_m\}$ is a set of non-negative real valued functions where each F_i is defined over a subset of variables $\mathbf{S}_i \subset \boldsymbol{X}$ (its scope) and $\boldsymbol{C} = \{C_1, \ldots, C_p\}$ is a set of constraints (or $0/1$ functions).*

A mixed network represents a joint distribution over $\boldsymbol{X}$ given by:

$$P_{\mathcal{M}}(\mathbf{x}) = \begin{cases} \frac{1}{Z}\prod_{i=1}^{m} F_i(\mathbf{x}) & \text{if } \boldsymbol{x} \in sol(\boldsymbol{C}) \\ 0 & \text{otherwise} \end{cases}$$

where $sol(\boldsymbol{C})$ is the set of solutions of $\boldsymbol{C}$ and $Z = \sum_{\boldsymbol{x}\in sol(\boldsymbol{C})}\prod_{i=1}^{m} F_i(\mathbf{x})$ is the normalizing constant.

The primal graph of a mixed network has variables as its vertices and an edge between any two variables than appear in the scope of a function $F \in \boldsymbol{F}$ or a constraint $C \in \boldsymbol{C}$.

We can define several queries over the mixed network. In this thesis, however we will focus on the following two queries:

DEFINITION **11** (**The Weighted Counting Task**). *Given a mixed network $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F},$*

$\boldsymbol{C}\rangle$, the weighted counting task is to compute the normalization constant given by:

$$Z = \sum_{\mathbf{x} \in Sol(\mathbf{C})} \prod_{i=1}^{m} F_i(\mathbf{x}) \tag{1.1}$$

where $sol(\boldsymbol{C})$ is the set of solutions of the constraint portion $\boldsymbol{C}$. Equivalently, if we represent the constraints in $\boldsymbol{C}$ as $0/1$ functions, we can rewrite Z as:

$$Z = \sum_{\mathbf{x} \in \mathbf{X}} \prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x}) \tag{1.2}$$

We will refer to Z as **weighted counts**.

DEFINITION **12** (**Marginal task**). *Given a mixed network $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$, the marginal task is to compute the marginal distribution at each variable. Namely, for each variable X_i and $x_i \in \boldsymbol{D}_i$, compute:*

$$P(x_i) = \sum_{\mathbf{x} \in \mathbf{X}} \delta_{x_i}(\mathbf{x}) P_{\mathcal{M}}(\mathbf{x}), \text{ where } \delta_{x_i}(\mathbf{x}) = \begin{cases} 1 & \textit{if } X_i \textit{ is assigned the value } x_i \textit{ in } \boldsymbol{x} \\ 0 & \textit{otherwise} \end{cases}$$

To be able to use the constraint portion of the mixed network more effectively, for the remainder of the thesis, we require that all zero probabilities in the mixed network are also represented as constraints. It is easy to define such a network as we show below. The new constraints are redundant though.

DEFINITION **13** (**Modified Mixed network**). *Given a mixed network $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$, a modified mixed network is a four-tuple $\mathcal{M}' = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C}' \rangle$ where $\boldsymbol{C}' = \boldsymbol{C} \cup \{FC_i\}_{i=1}^{m}$ where*

$$FC_i(\mathbf{S}_i = \mathbf{s}_i) = \begin{cases} 0 & \textit{if } F_i(\boldsymbol{s}_i) = 0 \\ 1 & \textit{Otherwise} \end{cases} \tag{1.3}$$

FC_i can be expressed as a relation. It is sometimes called the flat constraints of the prob-

ability function.

Clearly, the modified mixed network M' and the original mixed network M are equivalent in that $P_{\mathcal{M}'} = P_{\mathcal{M}}$.

It is easy to see that the weighted counts over a mixed network specialize to (a) the probability of evidence in a Bayesian network, (b) the partition function in a Markov network and (c) the number of solutions of a constraint network. The marginal problem can express the posterior marginals in a Bayesian or Markov network.

ENCODING **1** (**Bayesian network with evidence as a mixed network**). *A Bayesian network* $\mathcal{B} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{G}, \boldsymbol{P} \rangle$ *and evidence* $(E_1 = e_1, \ldots, E_k = e_k)$ *is a mixed network* $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$ *where* $\boldsymbol{F} = \{P_1, \ldots, P_n\}$ *are the conditional probability tables (CPTs) and* $\boldsymbol{C} = \{C_i\}$ *where*

$$C_i(e'_i) = \begin{cases} 1 & e'_i = e_i \\ 0 & \textit{otherwise} \end{cases}$$

It is trivial to prove that:

PROPOSITION **1.** *Given a Bayesian network* $\mathcal{B} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{G}, \boldsymbol{P} \rangle$*, evidence* $(E_1 = e_1, \ldots, E_k = e_k)$ *and a mixed network* $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$ *encoded according to Encoding 1, weighted counts computed over* $\mathcal{M}$ *is equal to computing the probability of evidence in* $\mathcal{B}$ *while the posterior marginals of* $\mathcal{M}$ *equal the posterior marginals over each variable* $X_i \in \boldsymbol{X}$ *of* $\mathcal{B}$ *given the evidence.*

ENCODING **2** (**Markov network as a mixed network**). *A Markov network* $\mathcal{T} = \langle \boldsymbol{X}, \boldsymbol{D}, G, \boldsymbol{H} \rangle$ *is a mixed network* $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$ *where* $F = \{H_1, \ldots, H_m\}$ *is a set of potential functions and* ***C*** *contains a single global constraint* GC *in which all variable value combinations are allowed. Namely,* $GC(\mathbf{x}) = 1$

It is trivial to prove that:

PROPOSITION **2.** *Given a Markov network $\mathcal{T} = \langle \boldsymbol{X}, \boldsymbol{D}, G, \boldsymbol{H} \rangle$ and a mixed network $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$ encoded according to Encoding 2, the weighted counts computed over $\mathcal{M}$ are equal to the partition function of $\mathcal{T}$ and the posterior marginals over $\mathcal{M}$ are equal to the posterior marginals over $\mathcal{T}$.*

ENCODING **3** (**Constraint network as a mixed network).** *A constraint network $\mathcal{R} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C} \rangle$ is a mixed network $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$ where F contains a single unit function F_1 whose scope is* $\boldsymbol{X}$*, namely $F_1(\boldsymbol{x}) = 1$.*

It is trivial to prove that:

PROPOSITION **3.** *Given a constraint network $\mathcal{R} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C} \rangle$ and a mixed network $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$ encoded according to Encoding 3, the weighted counts of $\mathcal{M}$ are equal the number of solutions of $\mathcal{R}$.*

1.3 Exact Inference

A mixed network represents knowledge about the world as a joint distribution. The process of answering queries posed on this knowledge base is called inference. Unfortunately, it turns out that the inference in a graphical model is NP-hard.

THEOREM **1.** *Computing a solution to the weighted counting and marginal problems is NP-hard.*

Proof. See [19] for a proof. □

In spite of this worst case hardness result, in some cases, we can take advantage of some structural graph based properties like treewidth of the graphical model to perform exact inference efficiently. In this section, we review some of these schemes.

1.3.1 Cluster Tree Elimination

DEFINITION **14** (**Cluster-tree or Join-tree**). *Let $\mathcal{M} = \langle$**X, D, C, F**$\rangle$ be a mixed network. A* cluster-tree *or a* join-tree *of $\mathcal{M}$ is a triple $JT =< T, \chi, \psi >$, where $T = ($**V, E**$)$ is a tree, and χ and ψ are labeling functions which associate with each vertex $V \in$ **V** two sets, variable label $\chi(V) \subseteq$ **X** and function label $\psi(V) \subseteq$ **C** $\cup$ **F**.*

1. *For each function $H \in$ **C**$\cup$**F** , there is* exactly *one vertex $V \in$ **V** such that $H \in \psi(V)$ and $scope(H) \subseteq \chi(V)$.*
2. *For each variable $X_i \in$ **X**, the set $\{V \in$ **V**$|X_i \in \chi(V)\}$ induces a connected sub-tree of T. The connectedness requirement is also called the running intersection property.*

*Let $(U, V) \in$ **E** be an edge of T. The* separator *of U and V is defined as $sep(U, V) = \chi(U) \cap \chi(V)$. The* eliminator *of U and V is defined as $elim(U, V) = \chi(U) - sep(U, V)$.*

Cluster-tree elimination (CTE) [32, 72] is a message-passing algorithm on a cluster tree, the nodes of which are called clusters, each associated with variable and function subsets (their labels). CTE computes two messages for each edge (one in each direction), from each node to its neighbors, in two passes, from the leaves to the root and from the root to the leaves. The message that cluster U sends to cluster V is as follows. The cluster takes a product of all its own functions (in its label), with all the messages received from its neighbors excluding V and then marginalizes the resulting function relative to the eliminator between U and V. This yields a message defined on the separator between U and V.

The complexity of CTE is time exponential in the maximum size of variable subsets and space exponential in the maximum size of the separators. Join-tree and junction-tree algorithms for constraint and Bayesian networks are instances of CTE. For more details see [32, 72, 33].

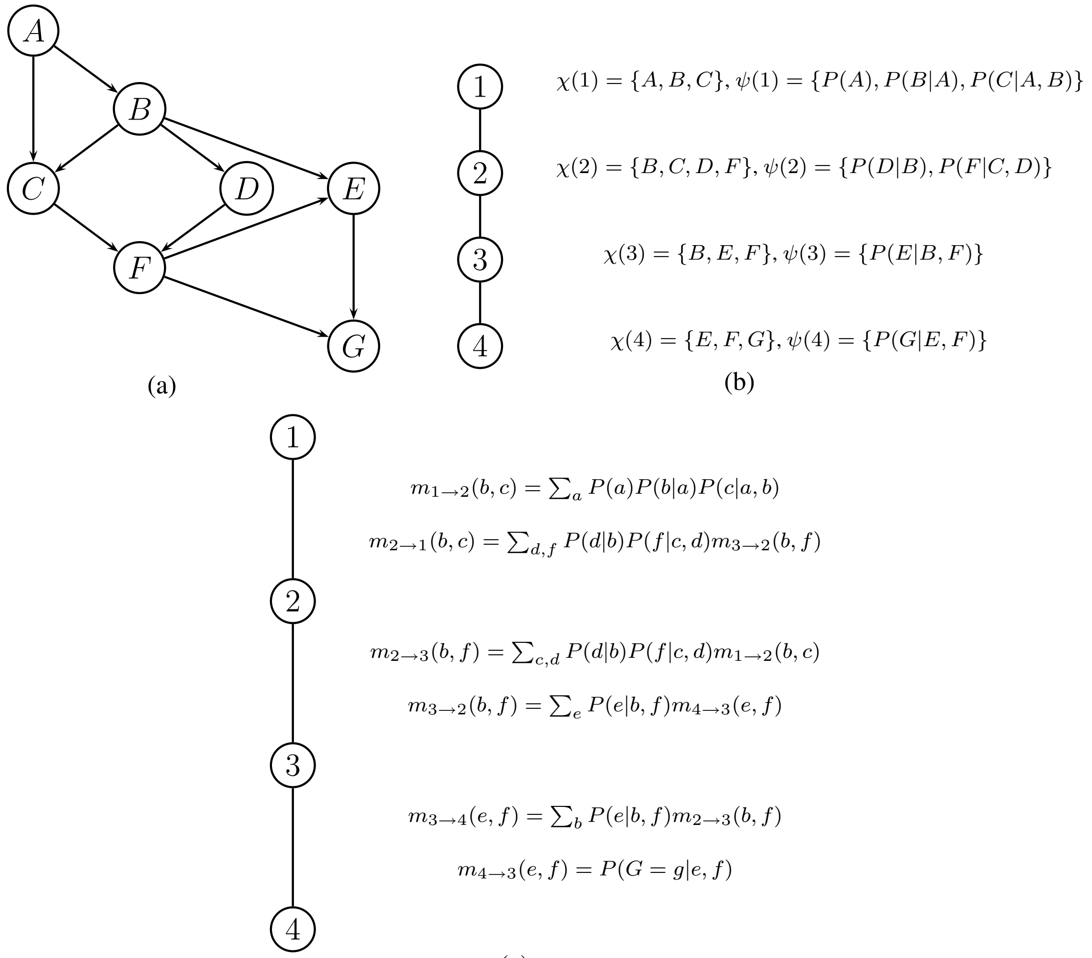

A
B
C
D
E
F
G
(a)
1
2
3
4
$\chi(1) = \{A, B, C\}, \psi(1) = \{P(A), P(B|A), P(C|A, B)\}$
$\chi(2) = \{B, C, D, F\}, \psi(2) = \{P(D|B), P(F|C, D)\}$
$\chi(3) = \{B, E, F\}, \psi(3) = \{P(E|B, F)\}$
$\chi(4) = \{E, F, G\}, \psi(4) = \{P(G|E, F)\}$
(b)
$m_{1\to2}(b, c) = \sum_a P(a)P(b|a)P(c|a, b)$
$m_{2\to1}(b, c) = \sum_{d,f} P(d|b)P(f|c, d)m_{3\to2}(b, f)$
$m_{2\to3}(b, f) = \sum_{c,d} P(d|b)P(f|c, d)m_{1\to2}(b, c)$
$m_{3\to2}(b, f) = \sum_e P(e|b, f)m_{4\to3}(e, f)$
$m_{3\to4}(e, f) = \sum_b P(e|b, f)m_{2\to3}(b, f)$
$m_{4\to3}(e, f) = P(G = g|e, f)$
(c)

Figure 1.6: $a)$ A Bayesian network; $b)$ A join-tree decomposition; $c)$Execution of CTE.

Formally, given a cluster tree 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, where a message is defined as follows:

DEFINITION **15** (**Message**). *Given a join tree $JT =< G(\boldsymbol{V}, \boldsymbol{E}), \chi, \psi >$, a message from a vertex $A \in \boldsymbol{V}$ to a vertex $B \in \boldsymbol{V}$ such that $(A, B) \in \boldsymbol{E}$, denoted by $m_{A\to B}$ is defined as follows. Let $\boldsymbol{Y} = sep(A, B)$, $\boldsymbol{Z} = elim(A, B)$ and $N_G(A)$ be the set of vertices adjacent to A in G.*

$$m_{A\to B}(\boldsymbol{y}) = \sum_{\boldsymbol{z}\in\boldsymbol{Z}} \left(\prod_{F\in\psi(A)} F(\boldsymbol{y}, \boldsymbol{z}) \prod_{D\in N_G(A)\setminus\{B\}} m_{D\to A}(\boldsymbol{y}, \boldsymbol{z}) \right)$$

EXAMPLE **6.** *Figure 1.6 (a) describes a Bayesian network and a join-tree decomposition* *for it (b). Figure 1.6(c) shows the trace of running CTE.* $m_{U \to V}$ *is a message that cluster* U *sends to cluster* V*.*

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

We can compute the posterior marginals for any variable $X_i \in \chi(A)$ by taking a product of all functions and messages in the cluster A and marginalizing out all other variables:

$$P(x_i) = \sum_{\mathbf{z} \in \chi(A) \setminus \{X_i\}} \left(\prod_{F \in \psi(A)} F(\mathbf{z}, x_i) \prod_{D \in N_G(A)} m_{D \to A}(\mathbf{z}, x_i) \right) \tag{1.4}$$

Similarly, we can compute the weighted counts by simply selecting any cluster A and marginalizing out all the variables:

$$Z = \sum_{\mathbf{z} \in \chi(A)} \left(\prod_{F \in \psi(A)} F(\mathbf{z}) \prod_{D \in N_G(A)} m_{D \to A}(\mathbf{z}) \right) \tag{1.5}$$

Bucket elimination is a special case of cluster-tree elimination where messages are passed from leaves to root along a bucket-tree [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 [28]. This scheme is similar to CTE.

1.3.2 Cutset Conditioning

When the treewidth of the mixed network is large, cluster tree elimination is infeasible because of its excessive memory requirements. In general, we can usually wait for some extra time for an algorithm to terminate but can do nothing when space limits are exceeded, unless we buy more memory. Therefore, schemes which trade time with memory known as conditioning were introduced in the graphical models literature. The main idea in this scheme is to select a subset of variables $\mathbf{K} \subseteq \mathbf{X}$, called a cutset or the conditioning set, and obtain posterior marginals for any node $X_i \in \mathbf{X} \setminus \mathbf{K}$ by:

$$P(x_i) = \sum_{\mathbf{k} \in \mathbf{K}} P(x_i|\mathbf{k})P(\mathbf{k}) \tag{1.6}$$

where $P(x_i|\mathbf{k})$ is the marginal probability of x_i given $\mathbf{K} = \mathbf{k}$ and $P(\mathbf{k})$ is the probability of the assignment $\mathbf{K} = \mathbf{k}$. The probabilities $P(x_i|\mathbf{k})$ and $P(\mathbf{k})$ can be computed using the cluster tree elimination algorithm described in the previous section. Equation 1.6 implies that we can compute $P(x_i)$ by enumerating all instantiations over $\mathbf{K}$ then summing up over the results.

The main idea in cutset conditioning schemes is to select $\mathbf{K}$ in such a way that the treewidth of the remaining graph is bounded by a constant and can be formalized using the notion of w-cutset defined below:

DEFINITION **16 (w-cutset).** *Given a graph $G(\boldsymbol{X}, \boldsymbol{E})$, a w-cutset is a subset of variables $\boldsymbol{K} \subseteq \boldsymbol{X}$ such that the treewidth of the graph G' obtained by removing all the vertices in $\boldsymbol{K}$ from G is bounded by w.*

Given a w-cutset of size c, each instantiation of cluster tree elimination runs in $O((n - c) \times exp(w))$ time and space and therefore the overall time and space requirements of the scheme are $O(exp(|\mathbf{D_K}|) \times (n - c) \times exp(w))$ and $O((n - c) \times exp(w))$ respectively.

It is well-known that the minimum induced width t^* of the network is always less than or equal to the size of the smallest w-cutset . Namely, $t^* + 1 \leq |\mathbf{C}|$ for any $\mathbf{C}$. Thus, inference algorithms (e.g., cluster tree elimination) are never worse and are often better than cutset conditioning time-wise. However, when t^* 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 w-cutset **K** such that the induced width w of the graph conditioned on **K** is small enough to permit exact inference.

1.4 Approximate Inference

1.4.1 Iterative Join Graph Propagation

Iterative Join-Graph Propagation (IJGP) [33] can be perceived as an iterative version of cluster tree elimination. It applies the same message-passing to join-graphs rather than join-trees, iteratively. A join graph is a decomposition of functions into a graph of clusters (rather than a tree) that satisfies the running intersection property. It can thus be thought of as a relaxation of the join tree (or a tree decomposition) in which the requirement that the clusters should form a tree is relaxed. The IJGP class of algorithms generalizes loopy belief propagation [106, 99]. These algorithms are not guaranteed to converge, nor have bounded accuracies, however they have been demonstrated to be useful approximation methods [33, 132]. While CTE is exact, and requires only 2 iterations, IJGP tends to improve its performance with additional iterations. The size of clusters in a join-graph can be far smaller than the treewidth and is only restricted by the function's scopes. IJGP can be parameterized by i which controls the cluster size in the join-graph, yielding a class of algorithms denoted by IJGP(i) whose complexity is exponential in i, that allow a trade-off between accuracy and complexity. As i increases, accuracy generally increases. When i is big enough to allow a tree-structure, IJGP(i) coincides with CTE and becomes exact.

Next, we formally define join graphs along with an algorithmic description of IJGP. For more details, see [33].

DEFINITION **17** (**(minimal) Edge-labeled Join-Graph**). *[33] Given a mixed network $\mathcal{M}$ = $\langle$ **X**, **D**, **F**, **C**$\rangle$, an edge-labeled join-graph is a four-tuple $JG =< G, \chi, \psi, \theta >$, where $G = (\mathbf{V}, \mathbf{E})$ is a graph, and χ and ψ are labeling functions which associate with each vertex $U \in \mathbf{V}$ two sets, variable label $\chi(U) \subseteq \mathbf{X}$ and function label $\psi(U) \subseteq \mathbf{F} \cup \boldsymbol{C}$ and θ associates with each edge $(A, B) \in \boldsymbol{E}$, the set $\theta(A, B) \subseteq \boldsymbol{X}$ such that:*

1. *For each function $H \in \mathbf{F} \cup \boldsymbol{C}$, there is exactly one vertex $U \in \mathbf{V}$ such that $H \in \psi(U)$, and the $scope(H) \subseteq \chi(U)$ and*
2. *(edge-connectedness) For each edge (A, B), $\theta(A, B) \subseteq \chi(A) \cap \chi(B)$, such that $\forall X_i \in \boldsymbol{X}$, any two clusters containing X_i can be connected by a path whose every edge label includes X_i.*

Finally, an edge-labeled join-graph is minimal *if no variable can be deleted from any label while still satisfying the edge-connectedness property.*

DEFINITION **18** (**separator and eliminator of edge-labeled join-graphs**). *Given two adjacent vertices A and B of JG, the* separator *of A and B is defined as $sep(A, B) = \theta(A, B)$, and the* eliminator *of A with respect to B is $elim(A, B) = \chi(A) - \theta(A, B)$. The separator width is* $\max_{(A,B)} |sep(A, B)|$.

Hence forth, for brevity, we will refer to edge-labeled join graphs simply as join-graphs.

DEFINITION **19** (**Message**). *Given a join graph $JG =< G(\boldsymbol{V}, \boldsymbol{E}), \chi, \psi, \theta >$, a message $m_{A \rightarrow B}$ from a vertex $A \in \boldsymbol{V}$ to a vertex $B \in \boldsymbol{V}$ such that $(A, B) \in \boldsymbol{E}$ is defined over*

Algorithm 1: Iterative Join Graph Propagation

```
Input: A mixed network $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C})$ and a partial assignment $\mathbf{S} = \mathbf{s}$ to a
       subset of variables, an integer $i$, a tolerated error $\epsilon$ and integer $max\_iter$
Output: A Join Graph with clusters containing original functions and constraints
        from $\mathcal{M}$ along with messages received from neighbors.
1  Construct a join graph $JG = (G(\mathbf{V}, \mathbf{E}), \chi, \psi, \theta)$ from the $\mathcal{M}$ such that each cluster
   of JG has at most $i$ variables;
2  Select an Activation schedule which contains all (two-way) edges
   $d = (A_1, B_1), \ldots, (A_{2*|E|}, B_{2*|E|})$.;
   // Let $m^{(j)}_{A \to B}$ be the message during the j-th iteration of
      IJGP
3  j=0, KLD=0 ;
4  Initialize all messages $m^0_{A \to B}$ to the uniform distribution.;
5  repeat
6      j=j+1, KLD=0 ;
7      for each $(A, B)$ in $d$ do
           // Let $\mathbf{Y} = \theta(A, B)$
8          Compute the message $m^{(j)}_{A \to B}$.
               $m^{(j)}_{A \to B}(\mathbf{y}) = \alpha \sum_{\mathbf{z} \in elim(A,B)} \left( \prod_{F \in \psi(A)} F(\mathbf{y}, \mathbf{z}) \prod_{D \in N_G(A) \setminus \{B\}} m^{(j-1)}_{D \to A}(\mathbf{y}, \mathbf{z}) \right)$
9          $KLD$=$KLD$+KL-Distance($m^{(j)}_{A \to B}, m^{(j-1)}_{A \to B}$);
10 until $KLD < \epsilon$ OR $j \geq max\_iter$ ;
```

variables $\mathbf{Y} = \theta(A, B)$ *as follows.*

$$m_{A \to B}(\boldsymbol{y}) = \alpha \sum_{\boldsymbol{z} \in elim(A,B)} \left(\prod_{F \in \psi(A)} F(\boldsymbol{y}, \boldsymbol{z}) \prod_{D \in N_G(A) \setminus \{B\}} m_{D \to A}(\boldsymbol{y}, \boldsymbol{z}) \right)$$

where $N_G(A)$ *is set of vertices adjacent to* A *in* G *and* α *is the normalization constant which ensures that each message is a proper probability distribution.*

The pseudo-code of IJGP is given in Algorithm 1. It takes as input a mixed network, an i-bound i, a tolerated message error ϵ and an integer max_iter. The algorithm first constructs a join graph having at most i variables in each cluster in step 1 using Algorithm IJGP-structuring and then iteratively passes messages between adjacent clusters in steps

2-10. The iterative process is stopped either when the sum of the KL distance between the messages in two consecutive iterations is less than some threshold ϵ (indicating that the process has converged to a fixed-point [132]) or when the maximum number of iterations have been reached. The output of IJGP(i) is a collection of functions and messages, each bounded exponentially by i.

Constructing Join Graphs bounded by i

Given a join-graph $JG = \langle G(\mathbf{V}, \mathbf{E}, \chi, \psi, \theta \rangle$, the accuracy and complexity of the (iterative) join-graph propagation algorithm depends on two different widths: joinwidth of JG (defined as $max_{A \in \mathbf{V}} |\chi(A)|$) (this determines the complexity of processing one cluster) and treewidth of G (which may affect the speed of convergence of iterative join-graph propagation) as defined next.

Algorithm 2: Join-Graph Structuring

```
  Input: A mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$, an Integer $i$
  Output: A Join graph in which each cluster has at most $i$ variables.
1 Apply procedure schematic mini-bucket($i$);
2 Associate each resulting mini-bucket with a node in the join-graph, the variables of
  the nodes are those appearing in the mini-bucket, the original functions are those in
  the mini-bucket ;
3 Keep the edges created by the procedure (called out-edges) and label them by the
  regular separator;
4 Connect the mini-bucket clusters belonging to the same bucket in a chain by
  in-edges labeled by the single variable of the bucket;
```

DEFINITION **20** (**external and internal widths**). *Given an edge-labeled join-graph* $JG = \langle G(\boldsymbol{V}, \boldsymbol{E}), \chi, \psi, \theta \rangle$ *of a mixed network* $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$*, the internal width of* JG *is* $max_{V \in \boldsymbol{V}} |\chi(V)|$*, while the external width of* JG *is the treewidth of* G*.*

Intuitively, the external width measures how close the join-graph is to a join-tree and therefore minimizing it would improve accuracy and speed of convergence. Algorithm *Join-*

Procedure 3 : `Schematic Mini-Bucket(`i`)`

```
Order the variables from $X_1$ to $X_n$ minimizing (heuristically) induced-width, and
associate a bucket for each variable ;
Place each CPT in the bucket of the highest index variable in its scope;
for $j = n$ to $1$ do
    Partition the functions in $bucket(X_j)$ into mini-buckets having at most $i$
    variables;
    For each mini-bucket $mb$ create a new scope-function (message) $f$ where
    $scope(f) = \{X|X \in mb\} - \{X_i\}$ and place scope(f) in the bucket of its highest
    variable;
    Maintain an edge between $mb$ and the mini-bucket (created later) of $f$;
```

Graph Structuring describes a heuristic scheme for constructing join-graphs having minimal external width. Given a mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$, and a bounding parameter i the algorithm returns a join-graph JG of $\mathcal{M}$ whose internal width is bounded by i. The algorithm applies the procedure *Schematic Mini-Bucket(i)* (see Procedure 3). The procedure only traces the scopes of the functions that would be generated by the full mini-bucket procedure, avoiding actual computation. The procedure ends with a collection of mini-bucket trees, each rooted in the mini-bucket of the first variable. Each of these trees is minimally edge-labeled. Then, *in-edges* labeled with only one variable are introduced, and they are added only to obtain the running intersection property between branches of these trees.

PROPOSITION **4.** *[33] Algorithm Join-Graph Structuring(i) generates a minimal edge-labeled join-graph having bound i.*

EXAMPLE **7.** *Figure 1.7(a) shows the trace of procedure schematic mini-buckets with i-bound of* 3 *and Figure 1.7(b) shows a join graph created from this trace. The only cluster partitioned is that of F into two scopes (FCD) and (BF), connected by an in-edge labeled with F.*

EXAMPLE **8.** *Figure 1.8 shows a range of edge-labeled join-graphs. On the left extreme we have a graph with smaller clusters, but more cycles. This is the type of graph Iterative Belief Propagation (IBP) [99, 106] works on. On the right extreme we have a tree decomposition,*

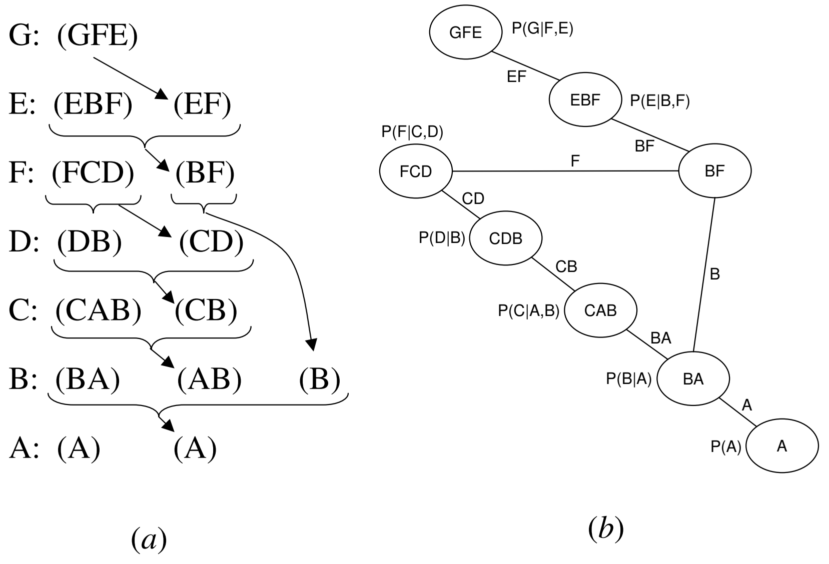


Figure 1.7: Constructing a Join-graph decomposition using schematic mini-buckets

which has no cycles but has bigger clusters. In between, there could be a number of join-graphs where maximum cluster size can be traded for number of cycles. Intuitively, the graphs on the left present less complexity for join-graph algorithms because the cluster size is small, but they are also likely to be less accurate. The graphs on the right side are computationally more complex, because of the larger cluster size, but are likely to be more accurate.

We can use the output of IJGP to estimate the posterior marginals $P_i(x_i)$ (the estimate is denoted by $Q_i(x_i)$) as follows. Let A be the cluster in the join graph such that $X_i \in \chi(A)$.

$$Q_i(x_i) = \alpha \sum_{\mathbf{z}\in\chi(A)\setminus\{X_i\}} \left(\prod_{F\in\psi(A)} F(\mathbf{z}, x_i) \prod_{B\in N_G(A)} m_{B\to A}(\mathbf{z}, x_i) \right) \tag{1.7}$$

where α is the normalization constant which ensures that $Q_i(X_i)$ is a proper probability distribution.

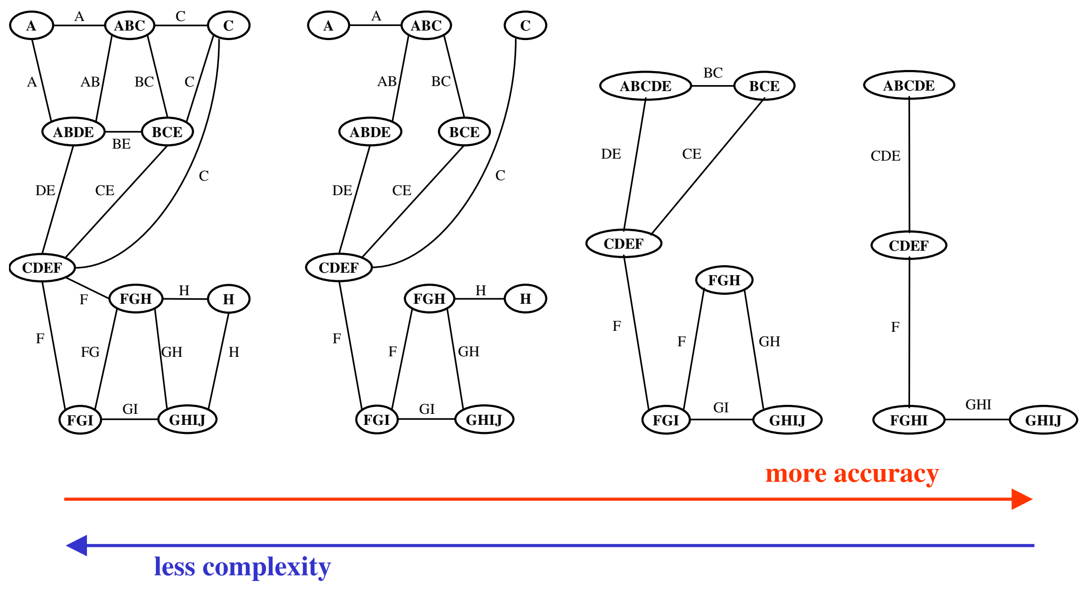


Figure 1.8: Join-graphs

1.4.2 Importance Sampling

Importance sampling [90, 50] is a general Monte Carlo simulation technique which can be used for estimating various statistics of a given target distribution. Since it is often hard to sample from the target distribution, the main idea is to generate samples from another easy-to-simulate distribution Q called the proposal (or trial or importance) distribution and then (as was shown) estimate various statistics over the target distribution by a weighted sum over the samples. The weight of a sample is the ratio between the probability of generating the sample from the target distribution and its probability based on the proposal distribution. In this subsection, we describe how the weighted counts and posterior marginals can be approximated via importance sampling. We first describe how to generate samples from Q.

If Q is given in a product form [1]: $Q(\mathbf{X}) = \prod_{i=1}^{n} Q_i(X_i|X_1,\ldots,X_{i-1})$, we can generate a full sample from Q easily as follows. For i = 1 to n, sample $X_i = x_i$ from the conditional distribution $Q(X_i|X_1 = x_1,\ldots,X_{i-1} = x_{i-1})$ and set $X_i = x_i$.This is often referred to as

[1]Through out the thesis, we assume that the proposal distribution is specified in a product form.

an *ordered Monte Carlo sampler*.

Thus, when we say that Q is easy to sample from, we assume that Q can be expressed in a product form and can be specified in polynomial space, namely,

$$Q(\mathbf{X}) = \prod_{i=1}^{n} Q_i(X_i|X_1, \ldots, X_{i-1}) = \prod_{i=1}^{n} Q_i(X_i|\mathbf{Y_i}) \tag{1.8}$$

where $\mathbf{Y}_i \subseteq \{X_1, \ldots, X_{i-1}\}$. The size of the set $\mathbf{Y}_i$ is assumed to be bounded by a constant.

Estimating the weighted counts by importance sampling

We start with some required definitions:

DEFINITION **21** (**An estimator**). *An estimator is a function of data (or samples) that produces an estimate for an unknown parameter or statistics of the distribution that produced the data (or samples).*

DEFINITION **22** (**Unbiased Estimator**). *Given a probability distribution Q and a statistics θ of Q, an estimator $\widehat{\theta_N}$ which is based on N random samples drawn from Q, is an unbiased estimator of θ if $\mathbb{E}_Q[\widehat{\theta_N}] = \theta$. Similarly, an estimator $\widetilde{\theta_N}$ which is based on N random samples drawn from Q, is an asymptotically unbiased estimator of θ if $\lim_{N\to\infty} \mathbb{E}_Q[\widetilde{\theta_N}] = \theta$. Clearly, all unbiased estimators are asymptotically unbiased.*

Note that, we will denote an unbiased estimator of a statistics θ by $\widehat{\theta}$, an asymptotically unbiased estimator by $\widetilde{\theta}$ and an arbitrary estimator by $\overline{\theta}$.

The notion of unbiasedness and asymptotic unbiasedness is important because it helps to characterize the performance of an estimator which we explain briefly below (for more details see [118]). The mean-squared error of an estimator $\overline{\theta}$ is given by:

$$\begin{aligned} MSE(\overline{\theta}) &= \mathbb{E}_Q[(\overline{\theta}-\theta)^2] && (1.9)\\ &= \mathbb{E}_Q[\overline{\theta}^2] - 2\mathbb{E}_Q[\overline{\theta}]\theta + \theta^2 && (1.10)\\ &= \left[\mathbb{E}_Q[\overline{\theta}^2] - \mathbb{E}_Q[\overline{\theta}]^2\right] + \left[\mathbb{E}_Q[\overline{\theta}]^2 - 2\mathbb{E}_Q[\overline{\theta}]\theta + \theta^2\right] && (1.11)\end{aligned}$$

The bias of $\overline{\theta}$ is given by:

$$B_Q[\overline{\theta}] = \mathbb{E}_Q[\overline{\theta}] - \theta$$

The variance of $\overline{\theta}$ is given by:

$$V_Q[\overline{\theta}] = \mathbb{E}_Q[\overline{\theta}^2] - \mathbb{E}_Q[\overline{\theta}]^2$$

From the definitions of bias, variance and mean-squared error, we have:

$$MSE(\overline{\theta}) = \left[\mathbb{E}_Q[\overline{\theta}^2] - \mathbb{E}[\overline{\theta}]^2\right] + \left[\mathbb{E}[\overline{\theta}]^2 - 2\mathbb{E}_Q[\overline{\theta}]\theta + \theta^2\right] = V_Q[\overline{\theta}] + \left[B_Q[\overline{\theta}]\right]^2 \quad (1.12)$$

In other words, the mean squared error of an estimator is equal to bias squared plus variance [118]. In case of an unbiased estimator, the bias equals zero and therefore one can reduce the mean squared error of an unbiased estimator by just reducing its variance. In case of an asymptotically unbiased estimator, the bias of the estimator goes to zero as the number of samples tend to infinity. However, for a finite sample size it may have a non-zero bias. Although, in principle an unbiased estimator seems to be better than an asymptotically unbiased estimator, the latter may have lower MSE than the former because it may have lower variance. In general, one would prefer an unbiased estimator over an asymptotically unbiased estimator because its MSE can be controlled by just controlling the variance.

Next, we show how the weighted counts can be estimated using importance sampling. Consider the expression for weighted counts (see Definition 1.13).

$$Z = \sum_{\mathbf{x} \in \mathbf{X}} \prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x}) \tag{1.13}$$

If we have a proposal distribution $Q(\mathbf{X})$ such that $\prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x}) > 0 \rightarrow Q(\mathbf{x}) > 0$, we can rewrite Equation 1.13 as follows:

$$Z = \sum_{\mathbf{x} \in \mathbf{X}} \frac{\prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x})}{Q(\mathbf{x})} Q(\mathbf{x}) = \mathbb{E}_Q \left[\frac{\prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x})}{Q(\mathbf{x})} \right] \tag{1.14}$$

Given independent and identically distributed (i.i.d.) samples $(\mathbf{x}^1, \ldots, \mathbf{x}^N)$ generated from Q, we can estimate Z by:

$$\widehat{Z}_N = \frac{1}{N} \sum_{k=1}^{N} \frac{\prod_{i=1}^{m} F_i(\mathbf{x}^k) \prod_{j=1}^{p} C_j(\mathbf{x}^k)}{Q(\mathbf{x}^k)} = \frac{1}{N} \sum_{k=1}^{N} w(\mathbf{x}^k) \tag{1.15}$$

where

$$w(\mathbf{x}^i) = \frac{\prod_{i=1}^{m} F_i(\mathbf{x}^i) \prod_{j=1}^{p} C_j(\mathbf{x}^i)}{Q(\mathbf{x}^i)}$$

is the weight of sample $\mathbf{x}^i$. By definition, the variance of the weights is:

$$V_Q[w(\mathbf{x})] = \sum_{\mathbf{x} \in \mathbf{X}} (w(\mathbf{x}) - Z)^2 Q(\mathbf{x}) \tag{1.16}$$

We can estimate the variance of $\widehat{Z}_N$ by (see for example [118]):

$$\widehat{V_Q}[\widehat{Z}_N] = \frac{1}{N(N-1)} \sum_{k=1}^{N} \left(w(\mathbf{x}^k) - \widehat{Z}_N \right)^2 \tag{1.17}$$

and it can be shown that $\widehat{V_Q}[\widehat{Z}_N]$ is an unbiased estimator of $V_Q[\widehat{Z}_N]$, namely,

$$\mathbb{E}_Q[\widehat{V_Q}[\widehat{Z}_N]] = V_Q[\widehat{Z}_N]$$

1. $\mathbb{E}_Q[\widehat{Z}_N] = Z$ i.e. $\widehat{Z}_N$ is *unbiased.*

2. $\lim_{N\to\infty} \widehat{Z}_N = Z$, with probability 1 (follows from the central limit theorem).

3. $\mathbb{E}_Q\left[\widehat{V_Q}[\widehat{Z}_N]\right] = V_Q[\widehat{Z}_N] = V_Q[w(\mathbf{x})]/N$

Therefore, $V_Q[\widehat{Z}_N]$ can be reduced by either increasing the number of samples N or by reducing the variance of the weights. From [118] it follows that that if $Q \propto \prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x})$, then $\forall\ N\ \widehat{Z}_N = Z$, thereby qualitatively identifying an optimal variance estimator. However, making $Q \propto \prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x})$ is NP-hard and therefore in order to have a small MSE in practice, it is recommended [88] that Q must be as "close" as possible to the function it tries to approximate which in our case is $\prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x})$.

Estimating the marginals by importance sampling

The posterior marginals are defined as:

$$P(x_i) = \sum_{\mathbf{x}\in\mathbf{X}} \delta_{x_i}(\mathbf{x}) P_{\mathcal{M}}(\mathbf{x}) \tag{1.18}$$

where $P_{\mathcal{M}}$ is defined by:

$$P_{\mathcal{M}}(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x}) \tag{1.19}$$

Given a proposal distribution $Q(\mathbf{x})$ satisfying $P_{\mathcal{M}}(\mathbf{x}) > 0 \rightarrow Q(\mathbf{x}) > 0$, we can rewrite

Equation 1.18 as follows:

$$P(x_i) = \sum_{\mathbf{x} \in \mathbf{X}} \frac{\delta_{x_i}(\mathbf{x}) P_{\mathcal{M}}(\mathbf{x})}{Q(\mathbf{x})} Q(\mathbf{x}) = \mathbb{E}_Q \left[\frac{\delta_{x_i}(\mathbf{x}) P_{\mathcal{M}}(\mathbf{x})}{Q(\mathbf{x})} \right] \tag{1.20}$$

Given independent and identically distributed (i.i.d.) samples $(\mathbf{x}^1, \ldots, \mathbf{x}^N)$ generated from Q, we can estimate $P(x_i)$ by:

$$\widehat{P_N}(x_i) = \frac{1}{N} \sum_{k=1}^{N} \frac{\delta_{x_i}(\mathbf{x}^k) P_{\mathcal{M}}(\mathbf{x}^k)}{Q(\mathbf{x}^k)} = \frac{1}{N} \sum_{k=1}^{N} \frac{\delta_{x_i}(\mathbf{x}^k) \prod_{i=1}^{m} F_i(\mathbf{x}^k) \prod_{j=1}^{p} C_j(\mathbf{x}^k)}{Z Q(\mathbf{x}^k)} \tag{1.21}$$

Unfortunately, Equation 1.21, while an unbiased estimator of $P(x_i)$ cannot be evaluated because Z is not known. We can however sacrifice unbiasedness and estimate $P(x_i)$ by using properly weighted samples [88].

DEFINITION **23** (**Properly weighted samples).** *A set of weighted samples $\{\boldsymbol{x}^k, w(\boldsymbol{x}^k)\}_{k=1}^{N}$ drawn from a distribution G are said to be properly weighted with respect to a distribution π if for any discrete function H,*

$$\mathbb{E}_G[H(\mathbf{x}^k) w(\mathbf{x}^k)] = c \mathbb{E}_\pi[H(\mathbf{x})]$$

where c is a normalization constant common to all samples.

Given the weighted set of samples, we can estimate $\mathbb{E}_\pi[H(\mathbf{x})]$ as:

$$\widetilde{\mathbb{E}}_\pi[H(\mathbf{x})] = \frac{\sum_{k=1}^{N} H(\mathbf{x}^k) w(\mathbf{x}^k)}{\sum_{k=1}^{N} w(\mathbf{x}^k)}$$

Substituting Equation 1.19 in Equation 1.20, we have:

$$P(x_i) = \frac{1}{Z} \mathbb{E}_Q \left[\frac{\delta_{x_i}(\mathbf{x}) \prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x})}{Q(\mathbf{x})} \right] \tag{1.22}$$

It is easy to prove that [88]:

PROPOSITION **5.** *Given* $w(\boldsymbol{x}) = \frac{\delta_{x_i}(\boldsymbol{x}) \prod_{i=1}^{m} F_i(\boldsymbol{x}) \prod_{j=1}^{p} C_j(\boldsymbol{x})}{Q(\boldsymbol{x})}$*, the set of samples* $\{\boldsymbol{x}^k, w(\boldsymbol{x}^k)\}_{k=1}^{N}$ *are properly weighted with respect to* $P_{\mathcal{M}}$.

Therefore, we can estimate $P(x_i)$ as follows:

$$\widetilde{P_N}(x_i) = \frac{\sum_{k=1}^{N} w(\mathbf{x}^k)\delta_{x_i}(\mathbf{x}^k)}{\sum_{k=1}^{N} w(\mathbf{x}^k)} \tag{1.23}$$

It is easy to prove that $\lim_{N\to\infty} \mathbb{E}[\widetilde{P_N}(x_i)] = P(x_i)$ i.e. it is *asymptotically unbiased.* Therefore, by weak law of large numbers the sample average $\widetilde{P_N}(x_i)$ converges almost surely to $P(x_i)$ as $N \to \infty$. Namely,

$$\lim_{N\to\infty} \widetilde{P_N}(x_i) = P(x_i) \quad \text{, with probability 1}$$

Also it can be shown that in order to have small estimation error, the proposal distribution Q should be as close as possible to the target distribution $P_{\mathcal{M}}$ [88].

The Algorithm

Algorithm 4 presents an importance sampling scheme for estimating the weighted counts and marginals. The algorithm maintains two mutable variables Z and $P[j][k]$ which store the estimate of weighted counts and posterior marginals respectively. Note that j indexes the variables $j = 1, \ldots, n$ and k indexes the values in the domains of variable X_j, $k = 1, \ldots, |D_j|$. In steps 1-4, we initialize Z and all $P[j][k]$ to zero. Steps 5-9 contain the main loop of the algorithm. In this loop, we first generate a sample $(x_1^i, \ldots, x_n^i)$ from the proposal distribution Q (step 6) and then compute its weight w^i in step 7. Then, we update Z and the appropriate marginal probability $P[j]$ by adding the weight w^i of the current sample to their current value. Finally, after the required N samples are generated

Algorithm 4: Importance Sampling

```
Input: A mixed network $\mathcal{M} = \langle$X,D,C,F$\rangle$, a proposal distribution $Q$, and number of samples $N$
Output: Estimate of posterior marginals and weighted counts.
1  $Z = 0$ ;
2  for $j = 1$ to $n$ do
3      for $k = 1$ to $D_i$ do
4          $P[j][k] = 0.0$ ;
5  for $i = 1$ to $N$ do
6      Generate a sample $\mathbf{x}^i = (X_1 = x_1^i, \ldots, X_n = x_n^i)$ from $Q$ using the ordered Monte Carlo
       sampler.;
7      Calculate the importance weight
           $$w^i = \frac{\prod_{j=1}^{m} F_j(\mathbf{x}^i) \prod_{k=1}^{p} C_k(\mathbf{x}^i)}{Q(\mathbf{x}^i)}$$
       $Z = Z + w^i$ ;
8      for $j = 1$ to $n$ do
9          $P[j][x_j^i] = P[j][x_j^i] + w^i$ ;
10 $Z = Z / N$;
11 for $j = 1$ to $n$ do
12     Normalize P[j] ;
13 Return $Z$ as a estimate of the counting problem and $P$ as a estimate of the marginal problem;
```

the algorithm returns Z/N as an estimate of the weighted counts and normalized value of $P[j]$ for $j = 1, \ldots, n$ as an estimate of the marginals.

Importance sampling schemes in Literature

The obvious question one has to answer in order to use importance sampling is how to choose the proposal distribution Q.

Likelihood weighting (LW) is the most simplest form of importance sampling in Bayesian networks which uses a slight modification of the prior distribution (distribution without evidence) as the proposal distribution. Given a Bayesian network $\mathcal{B} = \langle$**X,D,P**$\rangle$ with evidence $\mathbf{E} = \mathbf{e}$, the proposal distribution of LW defined over the non-evidence variables $\mathbf{Y} = \mathbf{X} \setminus \mathbf{E}$ is given by:

$$Q(\mathbf{y}) = \prod_{X_i \in \mathbf{Y}} P(X_i = \mathbf{y}_{\{X_i\}} | \mathbf{y}_{pa(X_i)}, \mathbf{e}_{pa(X_i)}) \tag{1.24}$$

where, as mentioned earlier, the notation $\mathbf{y}_{\{X_i\}}$ denotes the restriction of the assignment $\mathbf{Y} = \mathbf{y}$ to the set $\{X_i\}$.

We can generate samples from $Q(\mathbf{Y})$ by processing the nodes in topological order. If the node is not an evidence node, we sample a value for it based on its CPT and the value already sampled for its parents. If the node is observed we set it to its observed value and continue. The process just described generates one sample. Repeating it N times generates the required N samples.

Recall that the efficiency of importance sampling depends on how close the proposal distribution is to the posterior distribution. When all the evidence is in the root, Equation 1.24 represents exactly the posterior distribution and LW yields an optimal sampling scheme. On the other hand, if all the evidence is in the leaf nodes, LW samples from the prior distribution. If the evidence is very unlikely, the prior distribution is a very bad approximation of the posterior distribution. In this case, LW may yield very bad estimates and show poor convergence.

To deal with the case of unlikely evidence in the leaf nodes, backward sampling [48] attempts to change the sampling order so that evidence variables are sampled earlier. In [97], 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 variables bucket and summing out all other variables. However, bucket elimination is not possible for larger functions and therefore [97] suggest to approximate large functions by a probability tree. In [133], the output of loopy belief propagation is used to construct the proposal distribution in topological order.

Another area of research, which is orthogonal to selecting the best proposal distribution Q is adaptive importance sampling which tries to dynamically update Q based on the generated samples [16, 101]. The updating step is performed every l samples where l is usually selected heuristically or empirically. Since initial proposal distribution, no matter how well chosen, is often very different from the posterior distribution, dynamic updating can substantially improve the convergence of importance sampling. Examples of adaptive importance sampling include methods such as self-importance sampling and heuristic importance sampling [121], and, more recently, AIS-BN [16], and EPIS [133] and dynamic importance sampling [97]. The main challenge here is in selecting the update step, so that as more and more samples are drawn, the updated proposal distribution gets closer and closer to the posterior distribution. The procedures for updating the sampling probabilities vary. In the following, we describe the AIS-BN scheme in more detail.

AIS-BN algorithm is based on the observation that if we could sample each node in topological order from distribution $P(X_i|pa(X_i), \mathbf{e})$, then the resulting sample would be drawn from the posterior distribution $P(\mathbf{X}|\mathbf{e})$. Since this distribution is unknown for any variable that has observed descendants, AIS-BN initializes the proposal distribution Q to some $Q^0(\mathbf{X})$ defined by a collection of sampling distributions $Q_i^0(X_i|pa(X_i), \mathbf{e})$.

The objective of AIS-BN is to update each distribution $Q_i^k(X_i|pa(X_i), \mathbf{e})$ so that the next sampling distribution $Q_i^{k+1}(X_i|pa(X_i), \mathbf{e})$ will be closer to the required posterior distribution $\mathcal{P}(X_i|pa(X_i), \mathbf{e})$ than $Q_i^k(X_i|pa(X_i), \mathbf{e})$.

The updating formula, applied after generating every l samples, is as follows:

$$Q_i^{k+1}(x_i|pa(X_i), \mathbf{e}) = Q_i^k(x_i|pa(X_i), \mathbf{e}) + \alpha(k)(Pr'(x_i|pa(X_i), \mathbf{e}) - Q_i^k(x_i|pa(X_i), \mathbf{e})) \tag{1.25}$$

where $\alpha(k)$ is a positive function that determines the learning rate and $Pr'(x_i|pa(X_i), \mathbf{e})$ is an estimate of $\mathcal{P}(x_i|pa(X_i), \mathbf{e})$ based on the last l samples. When $\alpha(k) = 0$ (lower bound), the importance function is not updated. When $\alpha(k) = 1$ (upper bound), the old function is discarded so that $Q_i^{k+1}(x_i|pa(X_i), \mathbf{e}) = Pr'(x_i|pa(X_i), \mathbf{e})$. The convergence speed is directly related to $\alpha(k)$ and there are various trade-offs in choosing an appropriate $\alpha(k)$. If $\alpha(k)$ is small, the convergence will be slow due to the large number of updating steps needed to reach a local minimum. On the other hand, if it is large, convergence rate will be initially very fast, but the algorithm will eventually start to oscillate and thus may not reach a minimum.

1.4.3 Markov Chain Monte Carlo schemes

Markov Chains

A Markov chain is a stochastic process which evolves over time. We will denote by $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots$, the states of the system at discrete time steps starting at 0 and extending to infinity. The distribution from which $\mathbf{x}^{(0)}$ is drawn is called the *initial distribution* denoted by $Q(\mathbf{x}^{(0)})$. The infinite sequence $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots$ is called a Markov Chain if it satisfies the following Markov property:

$$A(\mathbf{x}^{(t+1)} = \mathbf{y} | \mathbf{x}^{(t)} = \mathbf{x}, \ldots, \mathbf{x}^{(0)} = \mathbf{z}) = A(\mathbf{x}^{(t+1)} = \mathbf{y} | \mathbf{x}^{(t)} = \mathbf{x}) \tag{1.26}$$

In other words, the distribution of $\mathbf{x}^{(t+1)}$ given all states up to t only depends on $\mathbf{x}^{(t)}$. If the transition probability $A(\mathbf{x}^{(t+1)} = \mathbf{y} | \mathbf{x}^{(t)} = \mathbf{x})$ does not change over time, then it is often expressed as the transition function $T(\mathbf{y}|\mathbf{x})$. Because $T(\mathbf{y}|\mathbf{x})$ is a conditional distribution, it has to satisfy the following property:

$$\sum_{\mathbf{y}} T(\mathbf{y}|\mathbf{x}) = 1 \tag{1.27}$$

The most important property of a Markov chain that we rely upon is its forgetfulness property. Namely, after a Markov chain has evolved for a period of time, the current state is nearly independent of the starting state. This "fixed point" of a Markov chain is a distribution called the stationary or invariant distribution.

DEFINITION **24** (**Stationary distribution**). *An invariant or stationary distribution of a Markov Chain is a distribution that once reached remains forever. Namely, $\mathcal{P}(\boldsymbol{x})$ is an invariant distribution iff for every state* **x***, the following condition is satisfied:*

$$\mathcal{P}(\boldsymbol{y}) = \sum_{\boldsymbol{x}} \mathcal{P}(\boldsymbol{x}) A(\boldsymbol{y}|\boldsymbol{x})$$

In this thesis, we are interested in Markov chains that converge eventually to the stationary distribution $\mathcal{P}$ regardless of the initial distribution A. The property of ergodicity forms the backbone of such chains, which we define below:

DEFINITION **25** (**Ergodicity**). *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 start state is equal to one. 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. An aperiodic, irreducible Markov chain is called* ergodic.

Aperiodicity means that the chain does not have regular loops where after every k steps we return to the same state. Irreducibility guarantees that we can get to a state **y** from another state **x** with non-zero probability and thus, will be able to visit (in the limit of infinite samples) all statistically important regions of the state space. The conditions are almost always satisfied as long as all probabilities are positive. As already mentioned, ergodicity is important because it ensures convergence to the stationary distribution. Formally,

THEOREM **2** (**Convergence**). *An ergodic Markov Chain converges to its unique stationary distribution regardless of the initial distribution.*

Proof. See [88] for a proof. □

Constructing MCMC techniques

The main idea in MCMC type of simulation algorithms in the context of graphical models is as follows. We start from some initial full assignment (or state according to Markov chain terminology) $\mathbf{x}^{(0)}$ to all (non-evidence) variables, and then use the Markov transition function to change this current state. The Markov transition function is selected in such a way that it satisfies the following properties to ensure convergence to the posterior distribution $\mathcal{P}$ of the graphical model:

1. the stationary distribution of the Markov Chain is the required posterior distribution $\mathcal{P}$ of the graphical model
2. the resulting Markov chain is ergodic.

The question is how to ensure that the two aforementioned properties are satisfied. The task may sound daunting but in the context of graphical models is quite easy to achieve. In the next two sub subsections, we overview two alternative constructions.

Metropolis Hastings Method

The main idea in the Metropolis Hastings method is to generate a large number of samples from a proposal function $T(\mathbf{y}|\mathbf{x})$ and then employ some acceptance/rejection criteria to "thin down" the samples, namely, only accept a subset of "good" samples which guarantee convergence. The only restriction imposed on $T(\mathbf{y}|\mathbf{x})$ is that $T(\mathbf{y}|\mathbf{x}) > 0$ implies that $T(\mathbf{x}|\mathbf{y}) > 0$ (which is similar to the importance sampling criteria described Section 1.4.2). The Metropolis algorithm then implements the following iteration given a current state $\mathbf{x}^{(t)}$:

- Draw $\mathbf{y}$ from the proposal distribution $T(\mathbf{y}|\mathbf{x}^{(t)})$.

- Draw a real number p from the uniform distribution $[0, 1]$ and update:

$$\mathbf{x}^{(t+1)} = \begin{cases} \mathbf{y} & \text{if } p \leq r(\mathbf{y}|\mathbf{x}^{(t)}) \\ \mathbf{x}^{(t)} & \text{otherwise} \end{cases} \tag{1.28}$$

  where $r(\mathbf{y}|\mathbf{x})$ is given by:

$$r(\mathbf{y}|\mathbf{x}) = min\left\{1, \frac{\mathcal{P}(\mathbf{y})T(\mathbf{x}|\mathbf{y})}{\mathcal{P}(\mathbf{x})T(\mathbf{y}|\mathbf{x})}\right\} \tag{1.29}$$

Next, we show that the Metropolis Hastings method prescribes a Markov transition rule whose stationary distribution equals the posterior distribution $\mathcal{P}$. Let $A(\mathbf{y}|\mathbf{x})$ be the transition function of the algorithm. To prove that the distribution is stationary (see Definition 24), we have to prove that:

$$\mathcal{P}(\mathbf{y}) = \sum_{\mathbf{x}} \mathcal{P}(\mathbf{x})A(\mathbf{y}|\mathbf{x}) \tag{1.30}$$

Fortunately, we can check a rather easier condition called *detailed balance* to prove that $\mathcal{P}$ is the stationary distribution, which is given by:

$$\mathcal{P}(\mathbf{x})A(\mathbf{y}|\mathbf{x}) = \mathcal{P}(\mathbf{y})A(\mathbf{x}|\mathbf{y}) \tag{1.31}$$

It should be clear that the detailed balance condition ensures invariance or convergence to the stationary distribution as shown below:

$$\sum_{\mathbf{x}} \mathcal{P}(\mathbf{x})A(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{x}} \mathcal{P}(\mathbf{y})A(\mathbf{y}|\mathbf{x}) = \mathcal{P}(\mathbf{y}) \sum_{\mathbf{x}} A(\mathbf{y}|\mathbf{x}) = \mathcal{P}(\mathbf{y}) \tag{1.32}$$

The Markov transition probability of the Metropolis Hastings method is given by:

$$A(\mathbf{y}|\mathbf{x}) = T(\mathbf{y}|\mathbf{x})r(\mathbf{y}|\mathbf{x}) = T(\mathbf{y}|\mathbf{x})min\left\{1, \frac{\mathcal{P}(\mathbf{y})T(\mathbf{y}|\mathbf{x})}{\mathcal{P}(\mathbf{x})T(\mathbf{y}|\mathbf{x})}\right\} \tag{1.33}$$

THEOREM **3.** *The Metropolis Hastings method satisfies the detailed balance condition:*

$$\mathcal{P}(\boldsymbol{x})A(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{P}(\boldsymbol{y})A(\boldsymbol{x}|\boldsymbol{y})$$

Proof.

$$\begin{aligned}
\mathcal{P}(\mathbf{x})A(\mathbf{y}|\mathbf{x}) &= \mathcal{P}(\mathbf{x})\left[T(\mathbf{y}|\mathbf{x})min\left\{1, \frac{\mathcal{P}(\mathbf{y})T(\mathbf{y}|\mathbf{x})}{\mathcal{P}(\mathbf{x})T(\mathbf{y}|\mathbf{x})}\right\}\right] && (1.34)\\
&= min\{\mathcal{P}(\mathbf{x})T(\mathbf{y}|\mathbf{x}), \mathcal{P}(\mathbf{y})T(\mathbf{x}|\mathbf{y})\} && (1.35)\\
&= \mathcal{P}(\mathbf{y})\left[T(\mathbf{x}|\mathbf{y})min\left\{1, \frac{\mathcal{P}(\mathbf{x})T(\mathbf{x}|\mathbf{y})}{\mathcal{P}(\mathbf{y})T(\mathbf{x}|\mathbf{y})}\right\}\right] && (1.36)\\
&= \mathcal{P}(\mathbf{y})A(\mathbf{x}|\mathbf{y}) && (1.37)
\end{aligned}$$

□

The Metropolis Hastings method is so general that one can use any positive function as $T(\mathbf{y}|\mathbf{x})$. However, as in importance sampling, the rate of convergence to the posterior distribution will depend upon how close $T(\mathbf{y}|\mathbf{x})$ is to the posterior distribution.

When the graphical model has zero probabilities or in general for a mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$, however, there is a small caveat in generating the initial sample $\mathbf{x}^{(0)}$. As before, one can use any positive function as $T(\mathbf{y}|\mathbf{x})$ but we have to ensure that $\mathcal{P}(\mathbf{x}^{(0)}) > 0$ to ensure that transitions are made only in the space of solutions of $\mathbf{C}$.

Gibbs Sampling

In this subsection, we present the Gibbs sampling algorithm specialized for Bayesian networks. Gibbs sampling starts with generating some random assignment to all variables (the assignment is typically generated using importance sampling). Then at each step j, we randomly choose X_i and sample a value for it, given the current assignment to the other variables. Formally, the sample is generated from:

$$\mathcal{P}(X_i | x_1^{(j)}, \ldots, x_{i-1}^{(j)}, x_{i+1}^{(j)}, \ldots, x_n^{(j)})$$

For simplicity, we introduce some new notation. Let $\mathbf{x}_{-i}^{(j)} = (x_1^{(j)}, \ldots, x_{i-1}^{(j)}, x_{i+1}^{(j)}, \ldots, x_n^{(j)})$.

In Bayesian networks, the conditional distribution $\mathcal{P}(X_i|\mathbf{x}_{-i}^{(j)})$ is dependent only on the assignment to the Markov blanket $\mathbf{ma}_i$ of variable X_i, where Markov blanket is defined as the set containing X_i, its parents, its children, and parents of children. Let $\mathbf{x}_{-i,\mathbf{ma}_i}$ denote the restriction of the assignment $\mathbf{x}_{-i}$ to $\mathbf{ma}_i$. Thus,

$$\mathcal{P}(X_i|\mathbf{x}_{-i}^{(j)}) = \mathcal{P}(X_i|\mathbf{x}_{-i,\mathbf{ma}_i}^{(j)})$$

Given an assignment to the Markov blanket of X_i, we can obtain the sampling probability distribution $\mathcal{P}(X_i|\mathbf{x}_{-i,\mathbf{ma}_i}^{(j)})$ using the following equation [106]:

$$\mathcal{P}(X_i|\mathbf{x}_{-i,\mathbf{ma}_i}^{(j)}) \propto P(X_i|\mathbf{x}_{-i,pa(X_i)}^{(j)}) \prod_{X_k \in chi(X_i)} P(\mathbf{x}_{-i,\{X_k\}}^{(j)}|\mathbf{x}_{-i,pa(X_k)}^{(j)}) \tag{1.38}$$

Algorithm 5 describes the Gibbs sampling scheme. We can see that generating a full new sample requires $O(n \times r)$ multiplication steps where r is the maximum family size and n is the number of variables.

Assuming it takes approximately 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 $\mathcal{P}(\mathbf{X}|\mathbf{e})$. The time spent computing K discarded samples is referred to as *burn-in* time. However, determining K is hard [88]. In general, the "burn in time " is optional in the sense that the convergence of the estimates to the correct posterior marginals does not depend on it.

Algorithm 5: Gibbs sampling

```
  Input: A belief network $\mathcal{B} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P} \rangle$, and evidence $\mathbf{E} = \mathbf{e}$.
  Output: A set of samples $\{\mathbf{x}^{(t)}\}_{t=1}^{T}$
  // Initialize:
1 Assign random value $X_i = x_i^{(1)}$ to each variable $X_i$. Assign evidence variables their
  observed values.;
  // Generate samples
2 for t = 1 to T do
      // Generate a new sample $\mathbf{x}^{(t)}$.
3     for each $X_i \in \boldsymbol{X} \setminus \boldsymbol{E}$ do
          // compute a new value $x_i^{(t)}$
4         Compute distribution $\mathcal{P}(X_i|\mathbf{x}_{-i,\mathbf{ma}_i}^{(t)})$ using Equation 1.38;
5         Sample and set $X_i = x_i^{(t)}$ from $\mathcal{P}(X_i|\mathbf{x}_{-i,\mathbf{ma}_i}^{(t)})$;
```

A sufficient condition to ensure that Gibbs sampling has an ergodic Markov chain is that whenever we sample X_i there is a non-zero probability for it to be assigned to any one of its possible values. This ensures that we have a positive probability to get from any state to any state in some number of moves. In particular, if there are no zero entries in the CPTs the chain is ergodic. Furthermore, if the chain is ergodic it is trivial to verify that the stationary distribution of the chain is the posterior distribution $\mathcal{P}$.

The crucial factor in analyzing Markov Chains is the *mixing rate* , i.e., the rate with which a chain converges to the stationary distribution. Recall that we start the chain with the burn in phase. If the mixing rate is very low the burn in phase must be very long to avoid generating samples from the wrong distribution. Thus, we might need to generate a very large number of samples without being able to use them. Furthermore, when the mixing rate is low, the samples we generate after the burn in phase are very correlated and we might need a very large number of them in order to get a reliable estimate.

Unfortunately, the task of determining the mixing rate of a Gibbs sampler is quite difficult. Thus, the questions of when we can start using the samples and how many samples are needed for a reliable estimate are hard and make the use of Gibbs sampling a non-trivial task. Also, another problem with Gibbs sampling is that it cannot be used to generate samples from mixed networks. This is because the sampling space of a mixed network is not ergodic.

1.4.4 Rao-Blackwellised sampling

Recall that the mean squared error and therefore the quality of an unbiased estimator is directly proportional to the variance of the weights. Therefore, a vast amount of literature has been devoted to reducing variance (see for example [118, 88] and the references there in). In this subsection, we will review the Rao-Blackwellised importance sampling scheme which based on the Rao-Blackwell theorem reduces the variance by combining sampling with exact inference.

THEOREM **4 (Rao-Blackwell Theorem).** *Let $F(\boldsymbol{Y},\boldsymbol{Z})$ be a function and $Q(\boldsymbol{Y},\boldsymbol{Z})$ be a probability distributions which satisfies the importance sampling property that for any assignment $(\boldsymbol{Y}=\boldsymbol{y},\boldsymbol{Z}=\boldsymbol{z})$, $F(\boldsymbol{y},\boldsymbol{z})>0 \Rightarrow Q(\boldsymbol{y},\boldsymbol{z})>0$. Then,*

$$V_Q\left[\frac{F(\boldsymbol{y},\boldsymbol{z})}{Q(\boldsymbol{y},\boldsymbol{z})}\right] \geq V_Q\left[\frac{F(\boldsymbol{y})}{Q(\boldsymbol{y})}\right]$$

where $Q(\boldsymbol{y})=\sum_{\boldsymbol{z}} Q(\boldsymbol{y},\boldsymbol{z})$ and $F(\boldsymbol{y})=\sum_{\boldsymbol{z}} F(\boldsymbol{y},\boldsymbol{z})$.

Proof. See [88] for a proof. □

Rao-Blackwellisation [88, 118] can be used to estimate the weighted counts Z as follows. Assume that the set of variables $\mathbf{X}$ is partitioned into two sets $\mathbf{K}$ and $\mathbf{R}$ and let $Q(\mathbf{K})$ be a proposal distribution defined over **K**. Then,

$$
\begin{aligned}
Z &= \sum_{\mathbf{x}\in\mathbf{X}} \prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x}) && (1.39)\\
&= \sum_{\mathbf{k}\in\mathbf{K}} \sum_{\mathbf{r}\in\mathbf{R}} \prod_{i=1}^{m} F_i(\mathbf{k},\mathbf{r}) \prod_{j=1}^{p} C_j(\mathbf{k},\mathbf{r}) && (1.40)\\
&= \sum_{\mathbf{k}\in\mathbf{K}} \sum_{\mathbf{r}\in\mathbf{R}} \frac{\prod_{i=1}^{m} F_i(\mathbf{k},\mathbf{r}) \prod_{j=1}^{p} C_j(\mathbf{k},\mathbf{r})}{Q(\mathbf{k})} Q(\mathbf{k}) && (1.41)\\
&= \mathbb{E}\left[\sum_{\mathbf{r}\in\mathbf{R}} \frac{\prod_{i=1}^{m} F_i(\mathbf{k},\mathbf{r}) \prod_{j=1}^{p} C_j(\mathbf{k},\mathbf{r})}{Q(\mathbf{k})}\right] && (1.42)
\end{aligned}
$$

Given N independent samples $(\mathbf{k}^1,\ldots,\mathbf{k}^N)$ generated from $Q(\mathbf{K})$, we can estimate Z in an unbiased fashion by replacing the expectation in Equation 1.42 by the sample average as given below:

$$
\widehat{Z}_{rao-blackwell} = \frac{1}{N}\sum_{i=1}^{N} \frac{\sum_{\mathbf{r}\in\mathbf{R}} \prod_{j=1}^{m} F_j(\mathbf{r},\mathbf{K}=\mathbf{k}^i) \prod_{a=1}^{p} C_a(\mathbf{r},\mathbf{K}=\mathbf{k}^i)}{Q(\mathbf{k}^i)} \qquad (1.43)
$$

The assumption here is that $\sum_{\mathbf{r}\in\mathbf{R}} \prod_{j=1}^{m} F_j(\mathbf{r},\mathbf{K}=\mathbf{k}^i) \prod_{a=1}^{p} C_a(\mathbf{r},\mathbf{K}=\mathbf{k}^i)$ can be computed efficiently (using exact inference). It follows from the Rao-Blackwell theorem that the variance of $\widehat{Z}_{rao-blackwell}$ is less than the variance of conventional importance sampling estimator $\widehat{Z}$ given in Equation 1.15 when the two are based on the same set of samples over the sub-set **K** of variables.

w-cutset sampling [7] is an elegant implementation of the Rao-Blackwellisation idea. In this scheme, given a mixed network $\mathcal{M} = \langle \mathbf{X},\mathbf{D},\mathbf{F},\mathbf{C}\rangle$ and a w-cutset **K** (see Definition 16) of $\mathcal{M}$, we sample the variables in **K** and compute using Bucket Elimination [28] the exact value of $\sum_{\mathbf{r}\in\mathbf{R}} \prod_{j=1}^{m} F_j(\mathbf{r},\mathbf{K}=\mathbf{k}^i) \prod_{a=1}^{p} C_a(\mathbf{r},\mathbf{K}=\mathbf{k}^i)$ for each sample $\mathbf{K}=\mathbf{k}^i$. Because the time and space complexity of Bucket Elimination is exponential in the treewidth of the graph, it is obvious that given a w-cutset, Bucket Elimination can be carried out efficiently in polynomial time (exponential in the constant w).

PROPOSITION **6.** *Given a mixed network $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$ having n variables and a w-cutset of $\mathcal{M}$ of size c, the time and space complexity of w-cutset importance sampling is $O((n-c) \times N \times exp(w) + c \times N)$ and $O((n-c) \times exp(w))$ respectively where N is the number of samples.*

Proof. See [7] for a proof. □

It is clear from Proposition 6 that each w-cutset sample is heavy in the sense that we require more time (and space) to compute its weight. Therefore, we expect that the conventional estimator $\widehat{Z}$ given in Equation 1.15 will be based on a larger sample size than $\widehat{Z}_{rao-blackwell}$ and consequently w-cutset (Rao-Blackwellised) sampling may not always have lower variance. Thus, w-cutset sampling presents interesting time versus variance tradeoff. In practice, it seems that w-cutset sampling is well worth the extra computational complexity in that it usually yields more accurate estimates than conventional importance sampling [7].

Chapter 2

Hybrid Dynamic Mixed Networks for modeling Transportation routines

2.1 Introduction

Modeling sequential real-life domains requires the ability to represent both probabilistic and deterministic information. Hybrid Dynamic Bayesian Network (HDBN) were recently proposed for modeling such domains [83]. In essence, these are factored representation of Markov processes that allow discrete and continuous variables. Such probabilistic frameworks which focus on handling uncertain information, represent constraints as probabilistic entities and thus do not advertise their presence. This may have negative computational consequences. In this chapter, we address this issue by extending the mixed networks framework to dynamic environments through the model of dynamic Bayesian networks (DBN) accommodating both discrete and continuous variables. The resulting framework is called Hybrid Dynamic mixed networks (HDMN). We address the algorithmic issues that emerge and demonstrate the potential of our approach on a complex dynamic domain of a person's transportation routines.

Our motivation for developing HDMNs as a modeling framework is a range of problems in the transportation literature that depend upon reliable estimates of the prevailing demand for travel over various time scales. At one end, there is a pressing need for accurate and complete estimation of the global origins and destinations (O-D) matrix at any given time for an entire urban area. Such estimates are used in both urban planning applications [122] and integrated traffic control systems based upon dynamic traffic assignment techniques [107]. Even the most advanced techniques, however, are hamstrung by their reliance upon out-dated, pencil-and-paper travel surveys and sparsely distributed detectors in the transportation system. We view the increasing proliferation of powerful mobile computing devices as an opportunity to remedy this situation. If even a small sample of the traveling public agreed to collect their travel data and make that data publicly available, transportation management systems could significantly improve their operational efficiency. At the other end of the spectrum, *personal traffic assistants* running on the mobile devices could help travelers re-plan their travel when the routes they typically use are impacted by failures in the system arising from accidents or natural disasters. A common starting point for these problems is to develop an efficient formulation for learning and inferring individual traveler routines like traveler's destination and his route to destination from raw data points. This specific application was also considered in Liao et al. [86] and we use their probabilistic model with minor modifications for our purposes. The main focus of this chapter is on the algorithmic aspects.

In this chapter, we extend several algorithmic schemes that were shown to be effective for dynamic Bayesian networks such as generalized belief propagation [132, 33, 9, 67] and Rao-Blackwellised Particle Filtering (RBPF) [40, 47] to accommodate and exploit discrete constraints in the presence of continuous probabilistic functions. The chapter describes the first steps that we took and will serve to provide a base-line starting point for the algorithms that we will develop in the rest of the chapters. As we will show extending generalized belief propagation to handle constraints is easy, extension to continuous variables is somewhat more intricate but still straightforward. However, the primary challenge is to have effective importance sampling based techniques like particle filtering in the presence of constraints; because of the rejection problem.

The chapter starts by describing the framework of HDMN (section 2.2) and the transportation domain modeling within this framework (section 2.3). The algorithms for processing hybrid models are described in three stages. Section 2.4 shows how importance sampling can use a proposal distribution obtained from the output of Iterative Join Graph Propagation (IJGP) [33] within any graphical model. Section 2.5 addresses the rejection problem associated with the proposal distribution of importance sampling and proposes the use of constraint based pre-processing techniques to mitigate rejection. Sections 2.6 and 2.7 extend IJGP and IJGP-based importance sampling respectively to the full framework of HDMNs. Finally, we present experimental results in section 2.8 on our application domain and conclude in section 2.9.

The research presented in this chapter is based in part on [53, 60].

2.2 Hybrid Dynamic Mixed networks

We use the term hybrid for graphical models that have both discrete and continuous variables and mixed for graphical models that have mixed constraint and probabilistic dependencies.

DEFINITION **26** (**Hybrid Bayesian networks (HBN)**). *[81] Hybrid Bayesian networks (HBN) are graphical models defined by a tuple $\mathcal{B} = \langle \boldsymbol{X},\boldsymbol{D},\boldsymbol{G},\boldsymbol{P} \rangle$, where $\boldsymbol{X}$ is the set of variables partitioned into discrete and continuous ones $\boldsymbol{X} = \boldsymbol{\Gamma} \bigcup \boldsymbol{\Delta}$, where $\boldsymbol{\Delta}$ are discrete* *variables and* $\mathbf{\Gamma}$ *are continuous variables.* G *is a directed acyclic graph (DAG) over* $\boldsymbol{X}$*.* $\boldsymbol{P} = \{P_1, ..., P_n\}$ *is a set of conditional probability distributions (CPD). The graph structure* G *is restricted in that continuous variables cannot have discrete variables as their child nodes. The conditional distributions of discrete variables are expressed in a tabular form as usual, whereas the conditional distribution of continuous variables are given by a linear Gaussian model:* $P(X_i|\boldsymbol{I} = \boldsymbol{i}, \boldsymbol{Z} = \boldsymbol{z}) = N(\alpha(\boldsymbol{i}) + \beta(\boldsymbol{i})^T \times \boldsymbol{z}, \gamma(\boldsymbol{i}))$*,* $X_i \in \mathbf{\Gamma}$ *where* $\mathbf{Z}$ *and* $\boldsymbol{I}$ *are the set of continuous and discrete parents of* X_i *respectively,* $N(\mu, \sigma^2)$ *is a normal distribution with mean* μ *and variance* σ^2*,* $\gamma(\boldsymbol{i})$ *and* $\alpha(\boldsymbol{i})$ *are real numbers and* $\beta(\boldsymbol{i})$ *is a vector of the same dimension as* $|\boldsymbol{Z}|$*. The network represents a joint distribution over all its variables given by a product of all its CPDs.*

The joint distribution represented by a Hybrid Bayesian network is a mixture of multivariate Gaussians where every mixture component corresponds to an assignment $\mathbf{x}_{\mathbf{\Delta}}$ to all the discrete variables.

The mixed networks framework [92, 36] for augmenting Bayesian networks with constraints can immediately be applied to HBNs yielding the *hybrid mixed networks (HMNs)*. Formally,

DEFINITION **27 (Hybrid Mixed networks (HMN)).** *Given a HBN* $\mathcal{B} = (\boldsymbol{X}, \boldsymbol{D}, G, \boldsymbol{P})$ *that expresses the joint probability* $P_{\mathcal{B}}$ *and given a constraint network* $\mathcal{R} = (\mathbf{\Delta}, \boldsymbol{D}, \boldsymbol{C})$ *that expresses a set of solutions* $sol(\boldsymbol{C})$*, an HMN is a pair* $\mathcal{M} = (\mathcal{B}, \mathcal{R})$*. The discrete variables and their domains are shared by* $\mathcal{B}$ *and* $\mathcal{R}$*. We assume that* $\mathcal{R}$ *is consistent. The hybrid mixed network* $\mathcal{M} = (\mathcal{B}, \mathcal{R})$ *represents the conditional probability* $P_{\mathcal{M}}(\boldsymbol{x}) = P_{\mathcal{B}}(\boldsymbol{x})\ if\ \boldsymbol{x}_{\mathbf{\Delta}} \in sol(\boldsymbol{C})$ *and* 0 *otherwise.*

Dynamic probabilistic networks [98] are discrete time Markov models whose state-space and transition functions are expressed in a factored form using two-slice probabilistic networks. They are defined by a prior $P(\mathbf{X}_0)$ and a state transition function by the two-slice dynamic network expressing $P(\mathbf{X}_{t+1}|\mathbf{X}_t)$. *Hybrid* dynamic probabilistic networks also include continuous variables, while Hybrid dynamic *mixed* networks accommodate both continuous variables and explicit constraints. Formally,

DEFINITION **28 (Hybrid Dynamic mixed networks (HDMN)).** *A Hybrid Dynamic Mixed Network (HDMN) is a pair* $(\mathcal{M}_0, \mathcal{M}_{\rightarrow})$*, defined over a set of variables* $\boldsymbol{X} = \{X_1, ..., X_n\}$*, where* $\mathcal{M}_0$ *is an HMN defined over* $\boldsymbol{X}$ *representing* $P(\boldsymbol{X}_0)$*.* $\mathcal{M}_{\rightarrow}$ *is a 2-slice network defining the stochastic process* $P(\boldsymbol{X}_{t+1}|\boldsymbol{X}_t)$*. The* 2-time-slice hybrid mixed network (2-THMN) *is an HMN* $\mathcal{M} = (\mathcal{B}, \mathcal{R})$ *defined over* $\boldsymbol{X}' \cup \boldsymbol{X}''$ *such that* $\boldsymbol{X}'$ *and* $\boldsymbol{X}''$ *are identical to* $\boldsymbol{X}$*. The acyclic graph of the probabilistic portion is restricted so that nodes in* $\boldsymbol{X}'$ *are root nodes. The constraints are defined the usual way. The 2-THMN represents the conditional distribution* $P(\boldsymbol{X}''|\boldsymbol{X}')$*.*

The semantics of any dynamic network can be understood by unrolling the network to T time-slices. Namely, $P(\mathbf{X}_{0:t}) = P(\mathbf{X}_0) * \prod_{t=1}^{T} P(\mathbf{X}_t|\mathbf{X}_{t-1})$ where each probabilistic component can be factored in the usual way, yielding a regular HMN over T copies of the state variables.

The most common task over dynamic probabilistic models is filtering and prediction. Filtering is the task of determining the belief state $P(\mathbf{X}_t|\mathbf{e}_{0:t})$ where $\mathbf{X}_t$ is the set of variables at time t and $\mathbf{e}_{0:t}$ are the observations accumulated from time-slices 0 to t. Performing filtering in dynamic networks is often called *online inference* or *sequential inference*.

Filtering can be accomplished in principle by unrolling the dynamic model and using any state-of-the art exact or approximate reasoning algorithm. For example, one could use the join-tree clustering algorithm. When all variables in a dynamic network are discrete, the treewidth of the dynamic model is at most $O(n)$ where n is the number of variables in a time-slice (see [98] for details). However, when continuous Gaussian variables are mixed with discrete ones, the treewidth of HDMN is at least $O(T)$ when T is the number of

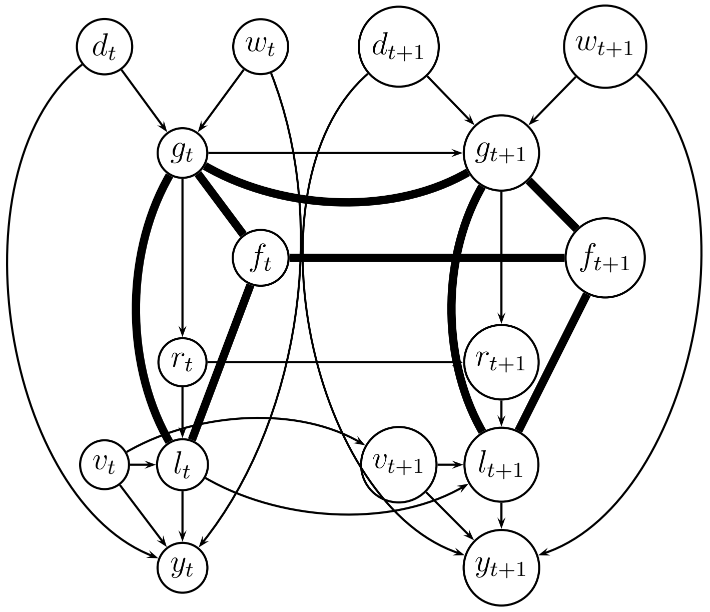


Figure 2.1: Car travel activity model

.

time slices. This is because to maintain correctness of join-tree propagation, we have to eliminate all continuous variables before the discrete ones which creates a clique between discrete variables from time slices $1, \ldots, T$. Thus exact inference is clearly infeasible in dynamic graphical models having both discrete and continuous variables [81, 83]. Therefore the applicable approximate inference algorithms for hybrid dynamic networks are either sampling-based such as Particle Filtering or propagation-based such as generalized belief propagation.

2.3 The transportation model

The primary motivating domain we use is the transportation model. In this section, we describe the application of HDMNs to the problem of inferring car travel activity of individuals. The main query of interest is predicting where a traveler is likely to go and what his/her route to the destination is likely to be, given the current location of the traveler's car.

This application was described in [86] and our goal is extend their work and study algorithms that are relevant to such domain using the general modeling framework we defined and which can be useful for a variety of applications.

Figure 2.1 shows a HDMN model for modeling the car travel activity of individuals. Note that the directed links express the probabilistic relationships while the undirected (bold) edges express the constraints.

Similar to [86], we consider the roads as a Graph $G(\mathbf{V}, \mathbf{E})$ where the vertices $\mathbf{V}$ correspond to intersections while the edges $\mathbf{E}$ correspond to segments of roads between intersections. The variables in the model are all indexed by time index t as follows. The variables d_t and w_t represent the information about time-of-day and day-of-week respectively at time t. The variable d_t, the day at time t, is a discrete variable and has four values $(morning, afternoon, evening, night)$ while the variable w_t, the week at time t, has two values $(weekend, weekday)$. Variable g_t represents the persons next goal at time t (e.g. his office, home etc). We consider a location where the person spends significant amount of time as a proxy for a goal as in [86]. These locations are determined through a preprocessing step by noting the locations in which the dwell-time is greater than a threshold (15 minutes). Once such locations are determined, we cluster those that are in close proximity to simplify the goal set. A goal can be thought of as a set of edges $\mathbf{E}_1 \subset \mathbf{E}$ in our graph representation. The route level r_t represents the route taken by the person moving from one goal to the next. We arbitrarily set the number of values it can take to $|g_t|^2$ i.e., the model can identify $|g_t|^2$ distinct routes that the individual uses to navigate. The person's location l_t and velocity v_t at time t are estimated from the GPS reading y_t at time t. f_t is a counter (essentially goal duration) that governs goal switching. The location l_t is represented in the form of a two-tuple $(A, N(\mu, \sigma^2))$ where $A = (V_1, V_2)$, $A \in \mathbf{E}$ and $V_1, V_2 \in \mathbf{V}$ is an edge of the map $G(\mathbf{V}, \mathbf{E})$ and $N(\mu, \sigma^2)$ is a Gaussian whose mean is equal to the distance between the person's current position on A and one of the intersections, say V_1.

The probabilistic dependencies in the model are same as those used in [86] except for two new variables, time-of-day and day-of-week, which determine the person's goal and the GPS reading. The constraints in the model are as follows. We assume that a person switches his goal from one time slice to another when he is near a goal or moving away from a goal but not when he is on a goal location. In this latter case, goal switching is forced when a specified maximum time at that goal, or duration D, is reached. These assumptions of switching between goals is modeled using the following constraints between the current location, the current goal, the next goal and the switching counters:

1. If $l_{t-1} = g_{t-1}$ and $F_{t-1} = 0$ Then $F_t = D$
2. If $l_{t-1} = g_{t-1}$ and $F_{t-1} > 0$ Then $F_t = F_{t-1} - 1$,
3. If $l_{t-1} \neq g_{t-1}$ and $F_{t-1} = 0$ Then $F_t = 0$,
4. If $l_{t-1} \neq g_{t-1}$ and $F_{t-1} > 0$ Then $F_t = 0$,
5. If $F_{t-1} > 0$ and $F_t = 0$ Then g_t is given by $P(g_t|g_{t-1})$,
6. If $F_{t-1} = 0$and $F_t = 0$ Then g_t is same as g_{t-1},
7. If $F_{t-1} > 0$ and $F_t > 0$ g_t is same as g_{t-1} and
8. If $F_{t-1} = 0$ and $F_t > 0$ g_t is given by $P(g_t|g_{t-1})$.

2.4 Constructing a proposal distribution using Iterative Join Graph Propagation

To perform inference in HDMNs, we will use an importance sampling algorithm for dynamic domains called particle filtering. As noted earlier, one of the main building block of importance sampling is its proposal distribution. In this section we will show how to construct a proposal distribution using Iterative Join Graph Propagation (IJGP) (described in subsection 1.4.1 of Chapter 1).

The proposal distribution is defined in a product form: $Q(\mathbf{X}) = \prod_{i=1}^{n} Q_i(X_i \,| X_1, \ldots, X_n)$ along an ordering $o = (X_1, \ldots, X_n)$ of variables. Recall that (see for example [118, 88]), the optimal proposal distribution is one in which each component $Q_i(X_i|X_1, \ldots, X_{i-1})$ is equal to the target distribution $\mathcal{P}_i(X_i|X_1, \ldots, X_{i-1})$ (namely the posterior distribution in our case). Since $\mathcal{P}_i(X_i|X_1, \ldots, X_{i-1})$ is NP-hard to construct, in practice, we should use a good approximation of $\mathcal{P}_i(X_i|X_1, \ldots, X_{i-1})$ as a substitute. It was shown that IJGP [33] yields a very good approximation of $\mathcal{P}_i(X_i|X_1, \ldots, X_{i-1})$ and therefore it is an ideal candidate for constructing Q.

We now describe a straightforward way of constructing the components $Q_i(X_i|\ X_1, \ldots,$ $X_{i-1})$ from the output of Iterative Join Graph propagation (IJGP). Given an ordering $o =$ $(X_1,\ \ldots,\ X_n)$ of the variables to be sampled, we can use IJGP dynamically during the sampling process to create the conditional distribution $Q_i(X_i|x_1, \ldots, x_{i-1})$ or we can use IJGP once in a pre-processing manner. Clearly, the latter would be more efficient but less accurate. There is also a spectrum of choices in between trading time for accuracy (we will present a scheme that exploits this spectrum in Chapter 4). In order to minimize overhead, we investigate here the choice with the least overhead, namely, constructing the proposal distribution obtained by running IJGP once prior to sampling.

Algorithm IJGP-sampling is given in Algorithm 6. As noted, the proposal distribution is pre-computed before sampling by running IJGP *just once*. Given a mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$, the output of IJGP (see Algorithm 1 in Chapter 1) is a join graph $JG = (G(\mathbf{V}, \mathbf{E}), \chi, \psi, \theta)$ in which each cluster $A \in \mathbf{V}$ contains the original functions $\psi(A) \subseteq \mathbf{F} \cup \mathbf{C}$ and messages $m_{B \to A}$ received from all its neighbors $B \in N_G(A)$. Each component $Q_i(X_i|X_1, \ldots, X_{i-1})$ can be constructed from JG as follows. Let A be the cluster of JG

containing X_i, namely $X_i \in \chi(A)$. Let $\mathbf{Y} = \chi(A) \cap \{X_1, \ldots, X_{i-1}\}$. Then,

$$Q_i(x_i|\mathbf{y}) = \alpha \sum_{\mathbf{z} \in \chi(A) \setminus \{X_1,\ldots,X_i\}} \left(\prod_{F \in \psi(A)} F(\mathbf{z}, x_i, \mathbf{y}) \prod_{B \in N_G(A)} m_{B \rightarrow A}(\mathbf{z}, x_i, \mathbf{y}) \right) \tag{2.1}$$

where α is the normalization constant which ensures that $Q_i(X_i|\mathbf{y})$ is a proper probability distribution. Because there could be many clusters in the join graph that mention X_i, we choose heuristically the cluster which mentions X_i and has the largest intersection with $\{X_1, \ldots, X_{i-1}\}$ (ties are broken randomly). We will refer to this heuristic as the *max heuristic* [1]. We can show that:

PROPOSITION **7.** *The time and space complexity of IJGP-sampling (Algorithm 6) is $O(h \times exp(i) + n \times N)$ and $O(h \times exp(i))$ respectively where h is the number of clusters in the join-graph of IJGP, i is the i-bound of IJGP, N is the number of samples and n is the number of variables.*

Proof. The time complexity of IJGP (step 1) is $O(h \times exp(i))$ [33]. Because $Q_i(X_i|\mathbf{x}_{i-1})$ is computed in step 4 by consulting a cluster whose size is bounded exponentially by i, the time and space complexity of computing $Q_i(X_i|\mathbf{x}_{i-1})$ is $O(exp(i))$. Therefore, to generate all components of Q, we require $O(n \times exp(i))$ time. Given Q, the importance sampling step has a time complexity of $O(nN)$. Therefore, the overall time complexity is $O(h \times exp(i) + n \times exp(i) + n \times N)$. Since $h \geq n$, the overall time complexity is given by $O(h \times exp(i) + n \times N)$. The space required by IJGP and Q is $O(h \times exp(i))$ and $O(n \times exp(i))$ respectively and therefore the overall space complexity is $O(h \times exp(i))$. □

[1]The max heuristic was chosen based on a preliminary empirical study in which we compared it with a random heuristic (in which we randomly select a cluster that mentions X_i) and found that the max heuristic was more accurate.

Algorithm 6: IJGP(i)-sampling

```
  Input: A mixed network $\mathcal{M} = \langle$X,D,F,C$\rangle$ and an ordering of variables
      $o = (X_1, \ldots, X_n)$, Integers $i$ and $N$.
  Output: Estimates of posterior marginals or weighted counts
1 Run Algorithm IJGP(i) (given in Algorithm 1) on the mixed network $\mathcal{M}$;
  // The output of IJGP is a join graph $JG = (G(\mathbf{V}, \mathbf{E}), \chi, \psi, \theta)$
     in which each cluster $A \in \mathbf{V}$ contains the original
     functions $\psi(A) \subseteq \mathbf{F} \cup \mathbf{C}$ and messages $m_{B \rightarrow A}$ received from
     all its neighbors $B \in N_G(A)$.
  // Proposal distribution construction
2 for i=1 to n do
3     Select a cluster $A$ in $JG$ that mentions $X_i$ and has the largest number of
      variables common with $\{X_1, \ldots, X_{i-1}\}$;
4     Compute $Q_i(X_i|X_1, \ldots, X_{i-1})$ from $A$ as follows. Let
      $\mathbf{Y} = \chi(A) \cap \{X_1, \ldots, X_{i-1}\}$

      $$Q_i(x_i|\mathbf{y}) = \alpha \sum_{\mathbf{z} \in \chi(A) \setminus \{X_1, \ldots, X_i\}} \left( \prod_{F \in \psi(A)} F(\mathbf{z}, x_i, \mathbf{y}) \prod_{B \in N_G(A)} m_{B \rightarrow A}(\mathbf{z}, x_i, \mathbf{y}) \right)$$

  // Sampling phase
5 Run Importance sampling with $(\mathcal{M}, Q, N)$ as input (see Algorithm 4).
```

2.5 Eliminating and Reducing Rejection

Given a mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$, a proposal distribution Q defined over **X** suffers from the rejection problem if the probability of generating a sample from Q that violates constraints of $P_{\mathcal{M}}$ expressed in **C** is relatively high. When a sample **x** violates some constraints in **C**, its weight $w(\mathbf{x})$ is zero and it is effectively rejected from the sample average estimate given by Equation 1.15. In an extreme case, if the probability of generating a rejected sample is arbitrarily close to one, then even after generating a huge number of samples, the estimate of weighted counts (given by Equation 1.15) would be zero (and the estimate of marginals given by Equation 1.23 would be an ill-defined ratio of 0/0); making importance sampling impractical.

Obviously, if Q encodes all the zeros in $\mathcal{M}$, then we would have no rejection.

DEFINITION **29** (**Zero Equivalence**). *A distribution P is zero equivalent with a distribution P', iff their flat constraint networks are equivalent. Namely, they have the same set of consistent solutions.*

Clearly then, given a mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$ representing $P_{\mathcal{M}}$ and a proposal distribution $Q = \{Q_1, \ldots, Q_n\}$ which is zero equivalent to $P_{\mathcal{M}}$, every sample **x** generated from Q satisfies $P_{\mathcal{M}}(\mathbf{x}) > 0$ (i.e. no sample generated from Q is rejected).

Because a proposal distribution Q is expressed in a product form: $Q(\mathbf{X}) = \prod_{i=1}^{n} Q_i(X_i| X_1, \ldots, X_{i-1})$ along $o = \langle X_1, \ldots, X_n \rangle$, we can make Q zero equivalent to $P_{\mathcal{M}}$ by modifying its components $Q_i(X_i|X_1, \ldots, X_{i-1})$ along o. To accomplish that, we have to make the set $\{Q_1, \ldots, Q_n\}$ backtrack-free along o w.r.t. **C**. The following definitions formalize this notion.

DEFINITION **30** (**consistent and globally consistent partial sample**). *Given a set of constraints **C** defined over a set of variables **X** $= \{X_1, \ldots, X_n\}$, a partial sample $(x_1, \ldots, x_i)$*

*is said to be consistent if it does not violate any relevant constraint in **C**. A partial sample* $(x_1, \ldots, x_i)$ *is said to be globally consistent if it can be extended to a solution of **C** (i.e. it can be extended to a full assignment to all* n *variables that satisfies all constraints in **C**).*

Note that a consistent partial sample may not be globally consistent.

DEFINITION **31** (**Backtrack-free distribution of** Q **w.r.t. C**). *Given a mixed network* $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$ *and a proposal distribution* $Q = \{Q_1, \ldots, Q_n\}$ *representing* $Q(\boldsymbol{X}) = \prod_{i=1}^{n} Q_i(X_i|X_1, \ldots, X_{i-1})$ *along an ordering* o*, the backtrack-free distribution* $Q^F = \{Q_1^F, \ldots, Q_n^F\}$ *of* Q *along* o *w.r.t.* ***C*** *where* $Q^F(\boldsymbol{X}) = \prod_{i=1}^{n} Q_i^F(X_i|X_1, \ldots, X_{i-1})$ *is defined by:*

$$Q_i^F(x_i|x_1, \ldots, x_{i-1}) \begin{cases} = \alpha Q_i(x_i|x_1, \ldots, x_{i-1}) & \textit{if } (x_1, \ldots, x_i) \textit{ is globally consistent w.r.t } \boldsymbol{C} \\ = 0 & \textit{otherwise}. \end{cases}$$

where α *is a normalization constant.*

Let $\mathbf{x}_{i-1} = (x_1, \ldots, x_{i-1})$ and let $\mathbf{B}_i^{\mathbf{x}_{i-1}} = \{x'_i \in \mathbf{D}_i | (x_1, \ldots, x_{i-1}, x'_i)$ is not globally consistent w.r.t. $\mathbf{C}$ $\}$. Then, α can be expressed by:

$$\alpha = \frac{1}{1 - \sum_{x'_i \in \mathbf{B}_i^{\mathbf{x}_{i-1}}} Q_i(x'_i|x_1, \ldots, x_{i-1})}$$

By definition, a proposal distribution $\{Q_1, \ldots, Q_n\}$ is backtrack-free w.r.t. itself along o. The modified proposal distribution defined above (Definition 31) takes a proposal distribution that is backtrack-free relative to itself (as a flat constraint network) and modifies its components to yield a distribution that is backtrack-free relative to $P_{\mathcal{M}}$ (as a flat constraint network).

Given a mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$ and a proposal distribution $Q = \{Q_1, \ldots, Q_n\}$ along o, we now show how to generate samples from the backtrack-free distribution $Q^F = \{Q_1^F, \ldots, Q_n^F\}$ of Q w.r.t. $\mathbf{C}$. Algorithm 7 assumes that we have an oracle which takes

Algorithm 7: Sampling from the Backtrack-free distribution

```
Input: A mixed network $\mathcal{M} = \langle$X,D,F,C$\rangle$, a proposal distribution $Q$ along an
       ordering $o$ and an oracle
Output: A full sample $(x_1, \ldots, x_n)$ from the backtrack free distribution $Q^F$ of $Q$
1  $\mathbf{x} = \phi$;
2  for $i$=1 to $n$ do
3      $Q_i^F(X_i|\mathbf{x}) = Q_i(X_i|\mathbf{x})$;
4      for each $x_i \in \mathbf{D}_i$ do
5          $\mathbf{y} = \mathbf{x} \cup x_i$;
6          if oracle says that $\mathbf{y}$ is not globally consistent w.r.t $\mathbf{C}$ then
7              $Q_i^F(x_i|\mathbf{x}) = 0$ ;
8      Normalize $Q_i^F(X_i|\mathbf{x})$ and generate a sample $X_i = x_i$ from it;
9      $\mathbf{x} = \mathbf{x} \cup x_i$;
10 return $\mathbf{x}$
```

a partial assignment $(x_1, \ldots, x_i)$ and a constraint satisfaction problem $\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$ as input and answers "yes" if the assignment is globally consistent and "no" otherwise. Given a partial assignment $(x_1, \ldots, x_{i-1})$, the algorithm constructs $Q_i^F(X_i|x_1, \ldots, x_{i-1})$ and samples a value for X_i as follows. $Q_i^F(X_i|x_1, \ldots, x_{i-1})$ is initialized to $Q_i(X_i|x_1, \ldots, x_{i-1})$. Then, for each extension $(x_1, \ldots, x_{i-1}, x_i)$ to X_i, it checks using the oracle, whether $(x_1, \ldots, x_{i-1}, x_i)$ is globally consistent relative to **C**. If not, it sets $Q_i^F(x_i|x_1, \ldots, x_{i-1})$ to zero, normalizes $Q_i^F(x_i|x_1, \ldots, x_{i-1})$ and generates a sample from it. Repeating this process along the order $(X_1, \ldots, X_n)$ yields a single sample from Q^F. Note that for each sample, the oracle should be invoked a maximum of $O(n \times d)$ times where n is the number of variables and d is the maximum domain size.

Given samples $(\mathbf{x}^1, \ldots, \mathbf{x}^N)$ generated from Q^F, we can unbiasedly estimate Z (defined in Equation 1.2) by replacing Q by Q^F in Equation 1.15. We get:

$$\widehat{Z}_N = \frac{1}{N} \sum_{k=1}^{N} \frac{\prod_{i=1}^{m} F_i(\mathbf{x}^k) \prod_{j=1}^{p} C_j(\mathbf{x}^k)}{Q^F(\mathbf{x}^k)} = \frac{1}{N} \sum_{k=1}^{N} w^F(\mathbf{x}^k) \tag{2.2}$$

where

$$w^F(\mathbf{x}) = \frac{\prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x})}{Q^F(\mathbf{x})} \tag{2.3}$$

is the backtrack-free weight of the sample.

Similarly, we can estimate the posterior marginals by replacing the weight $w(\mathbf{x})$ in Equation 1.23 with the backtrack free weight $w^F(\mathbf{x})$.

$$\widetilde{P_N}(x_i) = \frac{\sum_{k=1}^{N} w^F(\mathbf{x}^k)\delta_{x_i}(\mathbf{x}^k)}{\sum_{k=1}^{N} w^F(\mathbf{x}^k)} \tag{2.4}$$

Clearly, $\widehat{Z}_N$ defined in Equation 2.2 is an unbiased estimate of Z while $\widetilde{P}_N(x_i)$ defined in Equation 2.4 is an asymptotically unbiased estimate of the posterior marginals $P(x_i)$.

2.5.1 Sampling from the backtrack-free distribution using adaptive consistency

One of the methods that can be used to generate the backtrack-free distribution is adaptive consistency [29].

DEFINITION **32** (**Directional i-consistency**). *[29] A constraint network $\mathcal{R} = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C})$ is said to be directional i-consistent along an ordering $o = (X_1, \ldots, X_n)$ if every consistent assignment to $i-1$ variables can be consistently extended to another variable that is higher in the order. A network is strong directional i-consistent if it is directional j-consistent for every $j \leq i$.*

It can be shown that:

PROPOSITION **8.** *[29] If the ordered constraint graph of a constraint network $\mathcal{R}$ has a width (see Definition 3 in Chapter 1) of $i-1$ along o and if $\mathcal{R}$ is also strong directional*

i-consistent, then $\mathcal{R}$ is backtrack-free along o.

Because the original constraint network may not satisfy the desired relationship between width and local consistency, we can increase the level of strong directional consistency until it matches the width of the network. Adaptive consistency [38] implements this idea. Given an ordering o, adaptive consistency establishes directional i-consistency recursively, changing levels for each variable to adapt to the changing width of the currently processed variable. The method works because when a variable is processed, its final induced width is determined and a matching level of consistency can be achieved. After running adaptive consistency, we can assemble every solution in a backtrack-free manner along o in linear time.

We can use adaptive consistency to sample from the backtrack-free distribution as described in Algorithm ADC-sampling given as Algorithm 8. The algorithm first runs adaptive consistency to make the constraint portion **C** strong directional i-consistent along o. Given an ordering o, the algorithm places each constraint into the bucket of the variable that appears latest in its scope. Subsequently, buckets are processed in reverse order. A bucket is processed by solving the subproblem and recording its solutions as a new constraint. The newly generated constraint is placed in the bucket of its latest variable.

In the sampling step, given a partial assignment $(x_1, \ldots, x_{i-1})$, the algorithm constructs $Q_i^F(X_i|x_1, \ldots, x_{i-1})$ as follows. For each extension $(x_1, \ldots, x_{i-1}, x_i)$ to X_i, it checks whether $(x_1, \ldots, x_{i-1}, x_i)$ is consistent w.r.t. to the constraints in the bucket of X_i. If it is not consistent, it sets $Q_i^F(x_i|x_1, \ldots, x_{i-1})$ to zero. Then it normalizes $Q_i^F(x_i|x_1, \ldots, x_{i-1})$ and generates a sample from it. Repeating the process along the order $(X_1, \ldots, X_n)$ yields a single sample.

THEOREM **5.** *[29] Every sample generated by ADC-sampling is globally consistent and is generated from the backtrack-free distribution of Q relative to* ***C****.*

Algorithm 8: ADC-sampling

```
Input: A proposal distribution $Q(\mathbf{X}) = \prod_{i=1}^{n} Q_i(X_i|X_1, \ldots, X_{i-1})$, a mixed
       network $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C})$, an integer $N$
Output: $N$ globally consistent samples from $Q$
// Adaptive Consistency step
1  Partition constraints in $\mathbf{C}$ into $bucket_1 \ldots, bucket_n$ as follows;
2  for i=n down to 1 do
3      Put in $bucket_i$ all unplaced constraints mentioning $X_i$
4  for p down to 1 do
5      for all constraints $C_1, \ldots, C_j$ in $bucket_p$ do do
6          $A = \cup_{i=1}^{j} C_i - \{X_p\}$;
7          $C_A = \pi_A(\bowtie_{i=1}^{j} C_i)$;
8          Add $C_A$ to the bucket of the latest variable in scope $A$. Let $X_i$ be the latest
           variable in A. Update: $bucket_i = bucket_i \cup C_A$;
// Sampling Step
9  for j=1 to N do
10     $\mathbf{x} = \phi$;
11     for i=1 to n do
12         $Q_i^F(X_i) = Q_i(X_i|\mathbf{x})$;
13         for each value $x_i \in \boldsymbol{D}_i$ do
14             If $(\mathbf{x}, x_i)$ violates any constraints in $bucket_i$ $Q_i^F(x_i) = 0$;
15         Normalize $Q_i$ and sample a value $x_i$ from $Q_i^F(X_i)$;
16         $\mathbf{x} = \mathbf{x} \cup x_i$;
17     Output $\mathbf{x}$;
```

THEOREM **6.** *The time and space complexity of ADC sampling is $O(n \times exp(w) + O(nN))$ where n is the number of variables, w is the induced width along o and N is the number of samples.*

2.5.2 Reducing Rejection using IJGP-sampling

Clearly, ADC-sampling is infeasible when the induced width along o is large. In such cases, we could use bounded directional consistency techniques like directional i-consistency, where i is a constant. This will not eliminate rejection but would certainly reduce it.

In principle, we can use mini-bucket elimination [39] to achieve partial directional i-consistency. Interestingly, if the join graph of IJGP is constructed using the schematic mini-buckets scheme (see Procedure 3 in Chapter 1) [33], then IJGP achieves a stronger level of partial directional i-consistency than mini-buckets (see [35]).

We describe a sampling scheme which uses this partial directional i-consistency power of IJGP to reduce rejection in Algorithm 9. Because our primary concern is minimizing rejection, we select an ordering o that minimizes the induced width w.r.t. the constraint portion of the mixed network in Step 1. Then, we create a join graph JG along o using the schematic mini-buckets scheme (see Procedure 3 in Chapter 1), run IJGP on JG and create a proposal distribution from its output using the max heuristic described in Algorithm 6. Finally, in steps 4-13, we generate the required samples. The only difference between ADC-sampling and the current algorithm is that we update the proposal distribution using the zeros derived from the output of IJGP instead of adaptive consistency in steps 9 and 10 (consistency checking step). Formally, as proved in [35], given a partial sample $(x_1, \ldots, x_i)$ if the marginal probability denoted by $F_A(x_1, \ldots, x_i)$ is zero at any cluster A, then $(x_1, \ldots, x_i)$ is inconsistent. We can therefore safely set $Q_i^C(x_i|x_1, \ldots, x_{i-1})$ to zero whenever $F_A(x_1, \ldots, x_i)$ is zero. Obviously. given samples generated using Algorithm 9,

Algorithm 9: IJGP-sampling with consistency checking

```
   Input: A mixed network $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C})$, and integer $N$ and $i$
   Output: $N$ samples from $Q^C$
1  Select an ordering $o = (X_1, \ldots, X_n)$ minimizing heuristically induced width w.r.t.
   $\mathbf{C}$;
2  Create a i-bounded join graph $JG$ using Algorithm Join-Graph structuring (see
   Algorithm 2 in Chapter 1) by running the schematic mini-buckets scheme along $o$;
3  Run IJGP($i$) on $JG$ and create a proposal distribution $Q$ along $o$ from the output of
   IJGP(i) using the max heuristic outlined in Algorithm 6;
   // Sampling Phase
4  for j=1 to N do
5      $\mathbf{x} = \phi$;
6      for i=1 to n do
7          $Q_i^C(X_i) = Q_i(X_i|\mathbf{x})$;
8          for each value $x_i \in \boldsymbol{D}_i$ do
               // Consistency Checking Step
9              for all clusters A corresponding to the mini-buckets of $X_i$ do
                   // Let $F_A(\mathbf{x}, x_i)$ denote the marginal over
                      cluster $A$
10                 If $F_A(\mathbf{x}, x_i) = 0$ set $Q_i^C(X_i) = 0$;
11         Normalize $Q_i$ and sample a value $x_i$ from $Q_i^C(X_i)$;
12         $\mathbf{x} = \mathbf{x} \cup x_i$;
13     Output $\mathbf{x}$;
```

we have to use Q^C instead of Q in Equations 1.15 and 1.23 to maintain unbiasedness and asymptotic unbiasedness respectively.

In the next two sections, we extend the algorithms IJGP and IJGP-sampling to Hybrid Dynamic Mixed Networks (HDMNs).

2.6 Iterative Join Graph Propagation for HDMNs

We first extend Iterative Join Graph Propagation (IJGP) to Hybrid Mixed Networks which contain both discrete and continuous variables yielding hybrid IJGP. Then, we present algorithm IJGP-S which extends hybrid IJGP to dynamic networks. These extensions are straight-forward and can be easily derived by combining results from [81, 33, 67].

2.6.1 Hybrid IJGP(i) for inference in Hybrid Mixed Networks

To extend IJGP(i) to hybrid networks, we have to use the marginalization and product operators specified in [81] for constructing messages. This is because when continuous variables are mixed with discrete ones it yields a conditional Gaussian, which has its own unique properties (see [81] for details). Second, Gaussian nodes can be processed in polynomial time and therefore the complexity of processing each cluster is exponential only in the number of its discrete variables. We capture this property using the notion of adjusted-i-bound.

DEFINITION **33** (**Adjusted i-bound**). *The adjusted i-bound of a join graph $JG = \langle\, G\ (\boldsymbol{V}\ , \boldsymbol{E})\ , \chi\ , \psi\ , \theta\ \rangle$ of a Hybrid Mixed Network $\mathcal{M} = (\mathcal{B} = \langle\ \boldsymbol{X}\ ,\boldsymbol{D}\ , G, \boldsymbol{P}\ \rangle\ , \mathcal{R} = \langle\ \boldsymbol{\Delta}, \boldsymbol{D}\ , \boldsymbol{C}\ \rangle$ is $ai = \max_{U \in \boldsymbol{V}} |\chi(U) \cap \boldsymbol{\Delta}| - 1$.*

Henceforth, we will use i-bound to mean adjusted i-bound.

2.6.2 IJGP(i)-S for inference in sequential domains

Next, we will extend hybrid IJGP for performing filtering in Hybrid *Dynamic* Mixed Networks yielding IJGP(i)-S where "S" denotes that the algorithm performs online or sequential inference.

Constructing the join graph of IJGP(i)-S

Murphy [98] describes a scheme called the *interface algorithm* for constructing join-trees in a dynamic network. Interface is defined as a set of nodes in the current time slice that have an edge with nodes in the next time slice. For example, the shaded nodes in Figure 2.2 show the interface nodes for the given DBN. Murphy [98] showed that we can create a join tree in an online manner by simply pasting together join-trees of a so-called $1-\frac{1}{2}$-slice graph associated with each time slice via the interface nodes. Figure 2.3(a) illustrates this construction. A $1-\frac{1}{2}$-slice graph at time t is an undirected graph constructed from the moral graph over the nodes in time slices $t-1$ and t as follows. We first form a clique of all interface nodes at time $t-1$ and t. Then, we remove all non-interface nodes from time slice $t-1$ yielding a $1-\frac{1}{2}$-slice graph.

We adapt the interface algorithm for constructing i-bounded join graphs as follows. We create join-graphs over the $1-\frac{1}{2}$-slice graph and paste them together using the interface cluster. If the interface cluster has more than $i+1$ variables, then it is split into smaller clusters such that the new clusters have less than $i+1$ variables. This construction procedure is illustrated in Figure 2.3(b).

Message Passing

In IJGP(i)-S, we perform the same message passing as the join-tree algorithm for Dynamic Bayesian networks over a join graph. If all variables are discrete, we can compute $P(\mathbf{X}_T|\mathbf{e}_{0:T})$ (i.e. perform filtering) by passing messages from left to right over the join-tree as follows (this is same as sending messages from leaves to the root in cluster tree elimination where the root is a cluster in time slice T and all leaves are present in time slice 0). At each time-slice t, we are given a distribution over the interface cluster $\mathbf{I}_{t-1}$ denoted by $P(\mathbf{I}_{t-1}|\mathbf{e}_{0:t})$ which is passed from the previous time slice $t-1$. Then, we pass messages from the leaves to the root of the join-tree at time slice t and project a message onto the the interface cluster $\mathbf{I}_t$ which is then passed to the time-slice $t+1$ and so on. To compute messages at time slice t, we do not need the message received by $t-1$ from $t-2$ and therefore we do not need to keep the whole unrolled network in memory yielding a memory efficient online filtering scheme.

In IJGP-S, we pass messages in a join graph in a similar manner. Namely, at each time-slice t, we are given a distribution over all the interface clusters $\mathbf{I}_{t-1,1},\ldots,\mathbf{I}_{t-1,j}$ denoted by $P'(\mathbf{I}_{t-1,1}|\mathbf{e}_{0:t})$, . . . , $P'(\mathbf{I}_{t-1,j}|\mathbf{e}_{0:t})$ which is passed from the previous time slice $t-1$. Then we iteratively pass messages in the join-graph at time slice t until convergence or until a maximum number of iterations have been reached and project a message onto the interface clusters $\mathbf{I}_{t,1},\ldots,\mathbf{I}_{t,j}$ which is then passed to the time-slice $t+1$ and so on.

Hybrid IJGP-S is similarly defined except that we use Lauritzen's operators [81] for message passing and use adjusted i-bound for creating the join-graphs.

We summarize the complexity of IJGP-S in the following theorem:

THEOREM **7.** *The time complexity of IJGP(i)-S is $O(h \times exp(i) \times j^3 \times T)$ where h is the number of clusters of the join graph at each time slice, j is the number of continuous variables in a time slice and T is the number of time slices. The space complexity of*

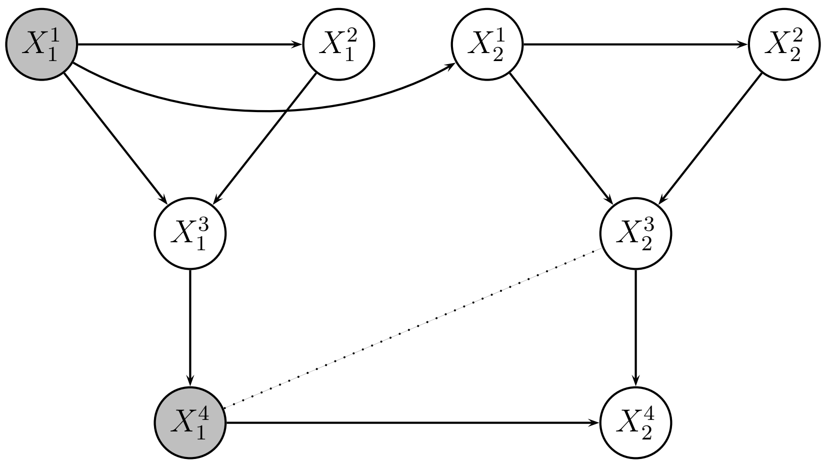
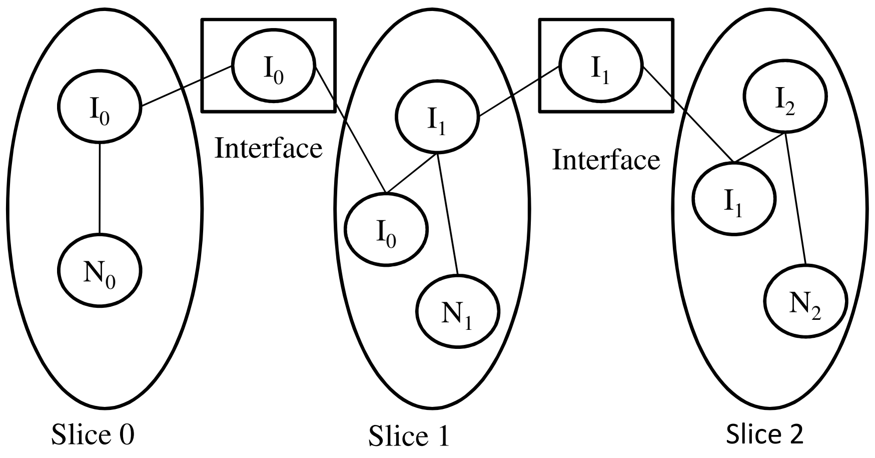


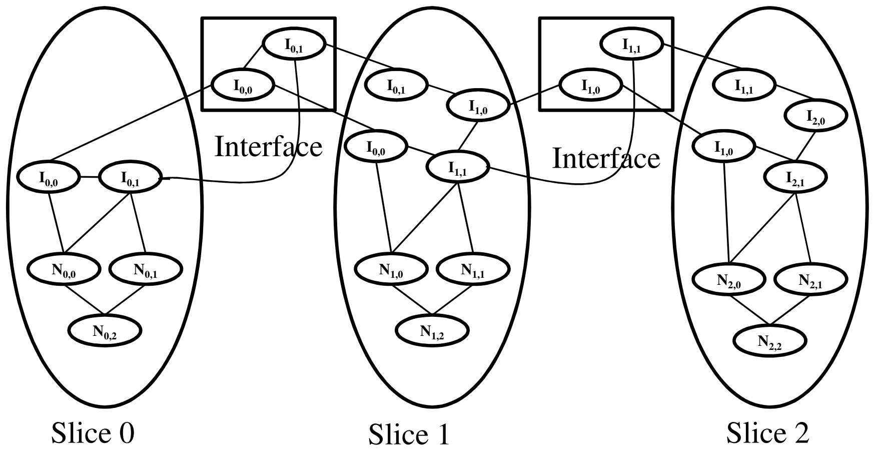
Figure 2.2: Figure showing an example two slice Dynamic Bayesian network. The shaded nodes form the Forward interface (adapted from Murphy [98])
.

IJGP(i)-S is $O(h \times exp(i))$.

Proof. The time complexity of constructing a message using Lauritzen's operators [81] in a cluster containing i discrete variables and j continuous variables is $O(exp(i) \times j^3)$. Since there are T time slices and h clusters in each time slice, the total number of clusters is bounded by $O(h \times T)$. Therefore, the overall time complexity is $O(h \times exp(i) \times j^3 \times T)$. Because, we do not utilize clusters of the join graph from time slice $t-2$ when we perform message passing in time slice t, we only have to keep $O(h)$ clusters in memory. Because each cluster requires only $O(exp(i))$ space, the overall space complexity is $O(h \times exp(i))$. □



(a) Join tree construction



(b) Join graph construction

Figure 2.3: Schematic illustration of the Procedure used for creating join-graphs and join-trees for HDMNs. I indicates a set of interface nodes while N indicates non-interface nodes.

Algorithm 10: IJGP(i)-RBPF

```
   Input: A Hybrid Dynamic Mixed Network $(\mathbf{X}, \mathbf{D}, G, \mathbf{P}, \mathbf{C})_{0:T}$ and a observation
          sequence $\mathbf{e}_{0:T}$ Integers $N$ and $w$
   Output: $P(\mathbf{X}_T|\mathbf{e}_{0:T})$
1  for $t$ = $0$ to $T$ do
       // 1.  Sequential Importance Sampling step
       // 1.1 Rao-Blackwellisation step
2      Partition the Variables $\mathbf{X}_t$ into discrete $\mathbf{\Delta}_t$ and continuous $\mathbf{\Gamma}_t$ variables;
       // 1.2 IJGP step
3      Construct a join graph $JG$ over the variables $\mathbf{X}_t$ in time-slice $t$ using the
       interface method outlined in section 2.6;
4      Run hybrid IJGP($i$) on $JG$;
5      Construct a proposal distribution $Q_t(\mathbf{\Delta}_t|\mathbf{e}_t, \mathbf{\Delta}_{t-1}, \mathbf{\Gamma}_{t-1})$ over the discrete
       variables $\mathbf{\Delta}_t$ from the output of IJGP(i) using the max heuristic outlined in
       Algorithm 6;
       // 1.3 Sampling step
6      for $i$ = 1 to $N$ do
7          Generate a sample $\delta_t^i$ from $Q_t(\mathbf{\Delta}_t|\mathbf{e}_t, \delta_{t-1}^i, \gamma_{t-1}^i)$ using (the sampling phase
           of) Algorithm 9;
           // Exact Step
8          Use join-tree-propagation to compute a distribution over $\mathbf{\Gamma}_t$ given $\delta_t^i$,
           $\delta_{t-1}^i, \gamma_{t-1}^i$ and $\mathbf{e}_t$ denoted by $\gamma_t^i$.;
9          Compute the importance weight $w_t^i$ of $(\delta_t^i, \gamma_t^i)$
10     Normalize the importance weights to form $\widehat{w_t^i}$.;
       // 2.  Resampling or Selection Step
11     Re-sample N samples from $\delta_t^i, \gamma_t^i$ according to the normalized importance
       weights $\widehat{w_t^i}$ to obtain new N random samples.;
```

2.7 Rao-Blackwellised Particle Filtering for HDMNs

In this section, we will present IJGP-based Rao-Blackwellised Particle filtering algorithm (IJGP-RBPF) for Hybrid Dynamic mixed networks. The main idea in IJGP-RBPF is to use IJGP for constructing a proposal distribution within the Rao-Blackwellised Particle filtering scheme [40, 47] and use its partial i-consistency power to reduce rejection (see Algorithm 9). The algorithm is straight-forward to derive and we present it here for completeness sake. We begin with a review of Rao-Blackwellised particle filtering.

Particle filtering uses a weighted set of samples or particles to approximate the filtering distribution $P(\mathbf{X}_t|\mathbf{e}_{0:t})$. The main idea is to approximate the distribution sequentially by first creating a set of weighted samples which approximate the distribution $P(\mathbf{X}_0|\mathbf{e}_0)$ at the first time slice, then using these samples to construct a new set of weighted samples which approximate the distribution $P(\mathbf{X}_1|\mathbf{e}_0, \mathbf{e}_1)$, and iterating this procedure until we reach the required time slice t for which the filtering distribution is desired. Particle filtering uses an appropriate (importance) proposal distribution $Q_t(\mathbf{X}_t|\mathbf{X}_{0:t-1}, \mathbf{e}_{0:t})$ at each time slice to generate the set of weighted samples. Formally, given samples $\mathbf{x}_t^1, \ldots, \mathbf{x}_t^N$ drawn from Q_t, the distribution at $\mathbf{X}_t$ is approximated using $\{w_t^i, \mathbf{x}_t^i\}_{i=1}^N$, where w_t^i is the importance weight of sample $\mathbf{x}_t^i$. It is often desirable to have the samples in unweighted form. For instance if importance weights are close to zero, only a few of them would dominate the sample-based estimates and after a few iterations all but one particle will have negligible weight. This is called the degeneracy or sample depletion problem [41, 73] which yields an inefficient particle filter in which a large computational effort is devoted to updating particles whose contribution to the approximation is almost zero. To mitigate this problem, we can generate unweighted samples using a resampling step in which we generate N samples from $\{w_t^i, \mathbf{x}_t^i\}_{i=1}^N$ by sampling each $\mathbf{x}_t^i$ with probability proportional to w_t^i.

Particle filtering often shows poor performance in high-dimensional spaces and its performance can be improved by sampling from a sub-space using the *Rao-Blackwell (RB) theorem* (and the particle filtering is called Rao-Blackwellised Particle Filtering (RBPF)). Specifically, the state $\mathbf{X}_t$ is divided into two sets: $\mathbf{Y}_t$ and $\mathbf{Z}_t$ such that only variables in set $\mathbf{Y}_t$ are sampled from the proposal distribution $Q_t(\mathbf{Y}_t|\mathbf{Y}_{0:t}, \mathbf{Z}_{0:t}, \mathbf{e}_{0:t})$ while the distribution on $\mathbf{Z}_t$ is computed analytically given a sample $\mathbf{Y}_t = \mathbf{y}_t$. The complexity of RBPF is proportional to the complexity of exact inference step for each sample $\mathbf{y}_t^k$.

Since exact inference can be done in polynomial time if a hybrid dynamic mixed network (HDMN) contains only continuous variables, a straightforward application of RBPF to HDMNs is to sample only the discrete variables in each time slice and apply exact inference to the continuous variables [47, 83]. Given a sampled assignment to the discrete variables, the continuous variables have a normal distribution which can be computed exactly. Each particle is thus *heavy* containing an assignment to the discrete variables and a Gaussian over all the continuous variables.

The IJGP-RBPF (i) scheme is given in Algorithm 10. The algorithm has two main steps: Sequential importance sampling (SIS) step and resampling step. Only the SIS step differs from standard RBPF described in [40]. SIS is divided into three sub-steps: (1) In the Rao-Blackwellisation step, we first partition the variables $\mathbf{X}_t$ in a 2THMN into discrete $\mathbf{\Delta}_t$ and continuous variables $\mathbf{\Gamma}_t$. (2) Then, in the IJGP step, we use hybrid IJGP(i) introduced in the previous section (Section 2.6) to generate a proposal distribution $Q_t(\mathbf{\Delta}_t|\mathbf{e}_t, \mathbf{\Delta}_{t-1}, \mathbf{\Gamma}_{t-1})$. (3) Finally, in the sampling step, we first generate N samples over the discrete variables from Q_t using the sampling phase of IJGP with consistency checking scheme given by Algorithm 9. Then, we compute a distribution over the continuous variables $\mathbf{\Gamma}_t$ exactly using join-tree propagation [81][2].

THEOREM **8.** *The time complexity of IJGP-RBPF(i) is* $O([n \times N \times j^3 + h \times exp(i) \times j^3] \times T)$ *where* n *is the number of discrete variables in each time slice,* h *is the number of clusters*

[2]Note that given an assignment to all the discrete variables, join-tree propagation over the continuous variables is equivalent to a Kalman filter [116].

of the join graph at each time slice, j is the number of continuous variables in a time slice, N is the number of samples or particles generated at each time slice and T is the number of time slices. The space complexity of IJGP-RBPF(i) is $O((n + j^2) \times N + h \times exp(i))$.

Proof. From Theorem 7, the time complexity of running IJGP is $O(h \times exp(i) \times j^3 \times T)$. To generate N samples on n discrete variables, it takes $O(N \times n)$ time at each time slice. To compute the weight of each sample via exact inference on j continuous variables, using Lauritzen's join-tree propagation scheme [81], we need $O(j^3)$ time. Therefore, the time required to generate N samples at each time slice is is $O(n \times N \times j^3)$. Thus, the overall time complexity is $O([n \times N \times j^3 + h \times exp(i) \times j^3] \times T)$.

The space complexity of running IJGP(i) is $O(h \times exp(i))$. Note that each particle consists of a full assignment to all the discrete variables and a multi-variate Gaussian distribution over all the continuous variables. To store each multi-variate Gaussian over j variables, we require $O(j^2)$ space and to store an assignment over n discrete variables, we need $O(n)$ space. Therefore, to store N particles, we need $O((n + j^2) \times N)$ space yielding an overall space complexity of $O((n + j^2) \times N + h \times exp(i))$. □

2.8 Experimental Results

The test data consists of a log of GPS readings collected by a post doctoral researcher in the Institute of Transportation Science at UCI. The test data was collected over a six month period at intervals of 1-5 seconds each. The data consist of the current time, date, location and velocity of the person's travel. The location is given as latitude and longitude pairs. The data was first divided into individual routes taken by the person and the HDMN model was learned using the Monte Carlo version of the EM algorithm [86, 84].

We used the first three months' data as our training set while the remaining data was used as a test set. TIGER/Line files available from the US Census Bureau formed the graph on which the data was snapped. As specified earlier our aim is two-fold: (a) Finding the destination or goal of a person given the current location and (b) Finding the route taken by the person towards the destination or goal.

To evaluate our inference and learning algorithms, we use three HDMN models. Model-1 is the model shown in Figure 2.1. Model-2 is the model given in Figure 2.1 when the variables w_t and d_t are removed from each time-slice. Model-3 is the base-model which tracks the person without any high-level information and is constructed from Figure 2.1 by removing the variables w_t, d_t, g_t and r_t from each time-slice.

We used four inference algorithms within the EM-learning scheme [86, 84]. Since EM-learning uses inference as a sub-step, we have four different learning algorithms. We call these algorithms as IJGP-S(1), IJGP-S(2) and IJGP-RBPF(1,N) and IJGP-RBPF(2,N) respectively. Note that the algorithm IJGP-S(i) (described in Section 2.6) uses i as the i-bound. IJGP-RBPF(i,N) (described in Section 2.7) uses i as the i-bound for IJGP(i) and N is the number of particles at each time slice. Three values of N were used: 100, 200 and 500. For the EM-learning, N was 500. Experiments were run on a Pentium-4 2.4 GHz machine with 2G of RAM. Note that for Model-3, we only use IJGP-RBPF(1) and IJGP(1)-S because the maximum i-bound in this model is bounded by 1 .

2.8.1 Finding destination or goal of a person

The results for goal prediction with various combinations of models, learning and inference algorithms are shown in Tables 2.1, 2.2 and 2.3. We define the prediction accuracy as the number of goals predicted correctly. Learning was performed off-line. Our slowest learning algorithm IJGP-RBPF(2) used almost 5 days of CPU time for Model-1, and almost 4 days for Model-2—significantly less than the period over which the data was collected.

The column 'Time' in Tables 1, 2 and 3 shows the average time in seconds required for performing prediction by the inference algorithms on a given learnt model while the other entries indicate the accuracy for each combination of inference and learning algorithms.

We can see that Model-1 achieves the highest prediction accuracy of 84% while Model-2 and Model-3 achieve prediction accuracies of 77% and 68% respectively or lower.

For Model-1, we see that the learning algorithms that use IJGP-RBPF(2) and IJGP(2)-S yield an average accuracy of 83% and 81% respectively. For Model-2, we see that the average accuracy of learning algorithms that use IJGP-RBPF(2) and IJGP(2)-S is 76% and 75% respectively. Therefore, IJGP-RBPF(2) and IJGP(2)-S are the best performing learning algorithms.

For Model-1 and Model-2, to verify which inference algorithm yields the best accuracy given a learned model, we see that IJGP(2)-S is the most cost-effective alternative in terms time versus accuracy while IJGP-RBPF yields the best accuracy.

			LEARNING			
			IJGP-RBPF		IJGP-S	
N	Inference	Time	(i=1)	(i=2)	(i=1)	(i=2)
100	IJGP-RBPF(1)	12.3	78	80	79	80
100	IJGP-RBPF(2)	15.8	81	84	78	81
200	IJGP-RBPF(1)	33.2	80	84	77	82
200	IJGP-RBPF(2)	60.3	80	84	76	82
500	IJGP-RBPF(1)	123.4	81	84	80	82
500	IJGP-RBPF(2)	200.12	84	84	81	82
	IJGP(1)-S	9	79	79	77	79
	IJGP(2)-S	34.3	74	84	78	82
	Average		79.625	82.875	78.25	81.25

Table 2.1: Goal prediction accuracy for Model-1. Each cell in the last four columns contains the prediction accuracy for the corresponding combination of the inference scheme (row) and the learning algorithm (column). Given a learned model, the column 'Time' reports the average time required by each inference algorithm for predicting the goal.

2.8.2 Finding the route taken by the person

To see how our models predict a person's route, we use the following method. We first run our inference algorithm on the learned model and predict the route that the person is likely to take. Then, we super-impose this route on the actual route taken by the person and count the number of roads that were not taken by the person but were in the predicted route (i.e. the false positives), and the number of roads that were taken by the person but were not in the actual route (i.e. the false negatives). The two measures are reported in Table 2.4 for the best performing learning models in each category: viz IJGP-RBPF(2) for Model-1 and Model-2 and IJGP-RBPF(1) for Model-3. As we can see Model-1 and Model-2 have the best route prediction accuracy (given by low false positives and false negatives).

Finally, in Figure 2.4, we show a typical execution of our system. On the left is the person's current location derived from the GPS device on his car and on the right is the route the person is most likely to take based on his past history (predicted by our model).

			LEARNING			
			IJGP-RBPF		IJGP-S	
N	Inference	Time	(i=1)	(i=2)	(i=1)	(i=2)
100	IJGP-RBPF(1)	8.3	73	73	71	73
100	IJGP-RBPF(2)	14.5	76	76	71	75
200	IJGP-RBPF(1)	23.4	76	77	71	75
200	IJGP-RBPF(2)	31.4	76	77	71	76
500	IJGP-RBPF(1)	40.08	76	77	71	76
500	IJGP-RBPF(2)	51.87	76	77	71	76
	IJGP(1)-S	6.34	71	73	71	74
	IJGP(2)-S	10.78	76	76	72	76
	Average		75	75.75	71.125	75.125

Table 2.2: Goal prediction accuracy for Model-2. Each cell in the last four columns contains the prediction accuracy for the corresponding combination of the inference scheme (row) and the learning algorithm (column). Given a learned model, the column 'Time' reports the average time required by each inference algorithm for predicting the goal.

			LEARNING	
N	Inference	Time	IJGP-RBPF(1)	IJGP(1)-S
100	IJGP-RBPF(1)	2.2	68	61
200	IJGP-RBPF(1)	4.7	67	64
500	IJGP-RBPF(1)	12.45	68	63
	IJGP(1)-S	1.23	66	62
	Average		67.25	62.5

Table 2.3: Goal prediction accuracy for Model-3. Each cell in the last four columns contains the prediction accuracy for the corresponding combination of the inference scheme (row) and the learning algorithm (column). Given a learned model, the column 'Time' reports the average time required by each inference algorithm for predicting the goal.

2.9 Related Work and Conclusion

Lioa et al. [86] and Patterson et al. [104] describe a model based on AHMEM and HMM respectively for inferring high-level behavior from GPS-data. Our model goes beyond their model by representing two new variables day-of-week and time-of-day which improves the accuracy by about 6%.

A mixed network framework for representing deterministic and uncertain information was presented in [34, 80, 36]. These previous works also describe exact inference algorithms for mixed networks with the restriction that all variables should be discrete. Our work goes beyond these previous works in that we describe approximate inference algorithms for the mixed network framework, allow continuous Gaussian nodes with certain restrictions and model discrete-time stochastic processes. The approximate inference algorithms

		Model1	Model2	Model3
N	INFERENCE	FP/FN	FP/FN	FP/FN
	IJGP(1)	33/23	39/34	60/55
	IJGP(2)	31/17	39/33	
100	IJGP-RBPF(1)	33/21	39/33	60/54
200	IJGP-RBPF(1)	33/21	39/33	58/43
100	IJGP-RBPF(2)	32/22	42/33	
200	IJGP-RBPF(2)	31/22	38/33	

Table 2.4: False positives (FP) and False negatives for routes taken by a person (FN)

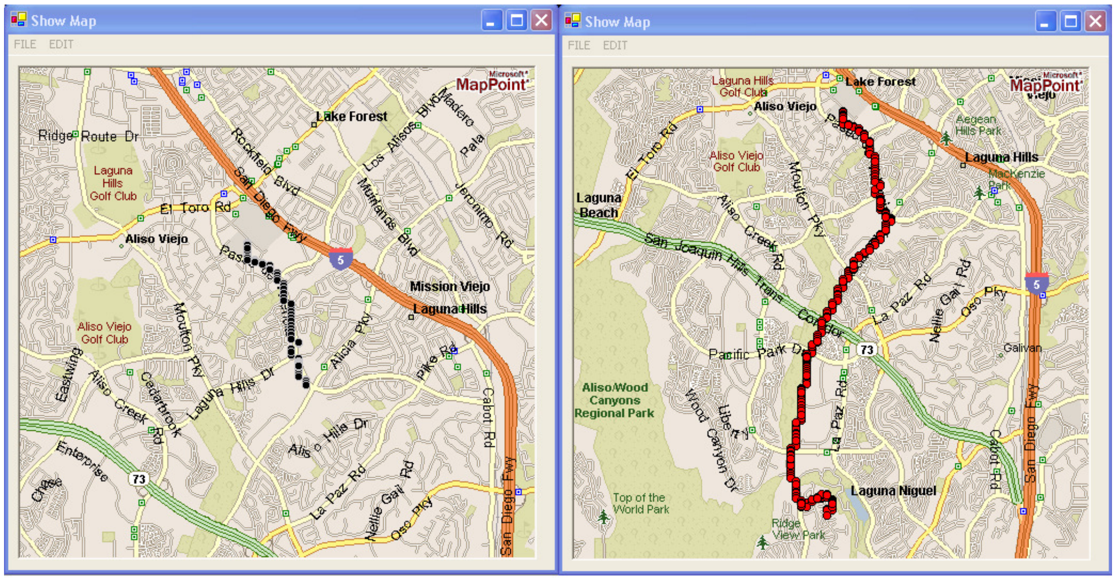


Figure 2.4: Route prediction. On the left is the person's current location derived from the GPS device on his car and on the right is the route the person is most likely to take based on his past history predicted by our model.

called IJGP(i) described in [33] handles only discrete variables. In our work, we extend this algorithm to include Gaussian variables and discrete constraints. We also develop a sequential version of this algorithm for dynamic models.

Particle Filtering is a very attractive research area [40]. Particle Filtering in HDMNs can be inefficient if non-solutions of constraint portion have high probability of being sampled. We show how to alleviate this difficulty by performing IJGP(i) before sampling. This algorithm IJGP-RBPF yields the best performance in our settings and might prove to be useful in applications in which particle filtering is preferred.

To conclude, in this chapter, we introduced a new importance sampling algorithm called IJGP-sampling which uses the output of Iterative Join Graph Propagation (IJGP) to construct a proposal distribution. IJGP is a good choice for constructing the proposal distribution because it not only yields a very good approximation to the posterior distribution but also achieves directional i-consistency which can substantially reduce the amount of rejection.

Subsequently, we introduced a new modeling framework called HDMNs, a representation that handles discrete-time stochastic processes with deterministic and probabilistic information on both continuous and discrete variables in a systematic way. We also proposed a IJGP-based algorithm called IJGP(i)-S for approximate inference in this framework, and proposed a class of Rao-Blackwellised particle filtering algorithm, IJGP-RBPF, which uses IJGP-sampling as a backbone, for effective sampling in HDMNs in the presence of constraints.

The HDMN framework and the two inference schemes were then applied to modeling travel behavior demonstrating their usefulness on a complex and highly relevant real life domain. This domain is attractive because travel behavior is highly routinized, but at the same time is highly variable. People may make daily trips between home and work, but those trips are varied in time and space, and interspersed with other trips. Eventually, the goal of this research is to be able to infer exactly the routine portions of travel behavior, and also to be able to infer exactly when routines have been broken. This goal has applicability to designing travel behavior surveys, as well as to designing in-vehicle or hand-held travel assistance devices.

Chapter 3

SampleSearch: A scheme that searches for consistent samples

3.1 Introduction

The chapter presents a scheme called SampleSearch for performing effective importance sampling based inference over mixed probabilistic and deterministic networks [34, 80, 36, 92]. As mentioned earlier, the mixed networks framework encompasses probabilistic graphical models such as Bayesian and Markov networks [106], and deterministic graphical models such as constraint networks [29]. We focus on weighted counting and posterior marginal queries over mixed networks. Weighted counts express the probability of evidence of a Bayesian network, the partition function of a Markov network and the number of solutions of a constraint network. Marginals seek the marginal distribution of each variable, also called as belief updating or posterior estimation in a Bayesian or Markov network.

It is straightforward to design importance sampling algorithms [90, 118, 50] for approximately answering counting and marginal queries because both are variants of *summation* *problems* for which importance sampling was designed (see Section 1.4.2). In particular, weighted counts is the sum of a function over some domain while a marginal is a ratio between two sums. The main idea in importance sampling is to transform a summation into an expectation using a special distribution called the proposal (or importance or trial) distribution from which it would be easy to sample from. Importance sampling then generates samples from the proposal distribution and approximates the expectation (also called true average or true mean) by a weighted average over the samples (also called sample average or sample mean). The sample average can be shown to be an unbiased estimate of the original summation, and therefore importance sampling yields an unbiased estimate of the weighted counts. In case of marginals, importance sampling has to compute a ratio of two unbiased estimates yielding an asymptotically unbiased estimate only.

In presence of hard constraints or zero probabilities, however, importance sampling may suffer from the *rejection problem* (see Section 2.5). The rejection problem occurs when the proposal distribution does not faithfully capture all the zero probabilities or all the constraints in the given mixed network. Consequently, a number of samples generated from the proposal distribution may have zero weight and do not contribute to the sample mean. In some cases, the probability of generating a rejected sample can be arbitrarily close to 1 yielding completely wrong estimates of both weighted counts and marginals.

Although the pre-processing using consistency enforcement approach presented in Chapter 2 reduces the amount of rejection, for some mixed networks having a hard constraint portion, consistency enforcement may not be powerful enough and may still allow a large number of rejected samples. In this chapter, we propose a more powerful scheme called *SampleSearch* which with some additional computation manages rejection during the sampling process itself.

SampleSearch, as the name suggests combines systematic backtracking search with Monte Carlo sampling. In this scheme, when a sample is supposed to be rejected, the algorithm continues instead with a randomized backtracking search until a sample with non-zero weight is found. In this formulation, the problem of generating a non-zero weight sample is equivalent to the problem of finding a solution to a satisfiability (SAT) or a constraint satisfaction problem (CSP). SAT and CSPs are NP-Complete problems and therefore the idea of generating a sample by solving an NP-Complete problem may seem inefficient. However, recently SAT/CSP solvers have achieved unprecedented success and are able to solve some large industrial problems having as many as a million variables within a few seconds [69, 112]. Therefore, a rather radical idea of solving a constant number of NP-complete problems to approximate a #P-complete problem such as weighted counting is no longer unreasonable.

We show that SampleSearch generates samples from a modification of the proposal distribution which is *backtrack-free*. Recall that (see Section 2.5) the backtrack-free distribution can be obtained by removing all partial assignments which lead to a zero weight sample. Namely, the backtrack-free distribution is zero whenever the exact distribution from which we wish to sample is zero. We propose two schemes to compute the backtrack-free probability of the generated samples which is required for computing the sample weights. The first is a computationally intensive method which involves invoking a CSP or a SAT solver $O(n \times d)$ times where n is the number of variables and d is the maximum domain size. The second scheme approximates the backtrack-free probability by consulting information gathered during SampleSearch's operation. This latter scheme has several desirable properties: (i) it runs in linear time, (ii) it yields an asymptotically unbiased estimate and (iii) it can provide upper and lower bounds on the exact backtrack-free probability.

Finally, we present an extensive empirical evaluation demonstrating the power of SampleSearch. We conducted experiments on three tasks: (a) counting models of a SAT formula (b) computing the probability of evidence in a Bayesian network and the partition function of a Markov network, and (c) computing posterior marginals in Bayesian and Markov networks.

For model counting, we compared against three approximate solution counters: ApproxCount [130], SampleCount [62] and Relsat [115]. Our experiments show that on most instances, given the same time bound SampleSearch yields solution counts which are closer to the true counts by a few orders of magnitude compared with the other schemes.

For the problem of computing the probability of evidence or the partition function, we compared SampleSearch with the any time scheme of Variable Elimination and Conditioning (VEC) [28], and an advanced generalized belief propagation scheme called Edge Deletion Belief Propagation (EDBP) [18]. We show that on most instances the estimates output by SampleSearch were more accurate than those output by EDBP. VEC solved some instances exactly, however on the remaining instances it was substantially inferior.

For the posterior marginal tasks, we experimented with linkage analysis benchmarks, with partially deterministic grid benchmarks, with relational benchmarks and with logistics planning benchmarks. Here, we compared the accuracy of SampleSearch in terms of a standard distance measure called the Hellinger distance against three other schemes: two generalized belief propagation schemes of Iterative Join Graph Propagation [33] and Edge Deletion belief propagation [18] and an adaptive importance sampling scheme called Evidence Pre-propagation Importance sampling (EPIS) [133]. Again, we found that except for the grid instances, SampleSearch consistently yields estimates having smaller error (or Hellinger distance) than the competition.

Thus, via a large scale experimental evaluation, we conclude that SampleSearch consistently yields very good approximations. In particular, on instances which have a substantial amount of determinism SampleSearch yields an order of magnitude improvement over state-of-the-art schemes.

The research presented in this chapter is based in part on [56, 55].

The rest of the chapter is organized as follows. In Section 3.2, we present the SampleSearch scheme. In Section 3.3, we present experimental results and we conclude in Section 3.4.

3.2 The SampleSearch Scheme

In a nutshell, SampleSearch incorporates systematic backtracking search into the ordered Monte Carlo sampler (see Section 1.4.2) so that all full samples are solutions of the constraint portion of the mixed network but it does not insist on backtrack-freeness of the search process itself. We will sketch our ideas using the most basic form of systematic search: chronological backtracking, emphasizing that the scheme can work with any advanced systematic search scheme. In our empirical work, we will indeed use advanced search schemes such as minisat [125].

Given a mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$ and a proposal distribution $Q(\mathbf{X})$, a traditional ordered Monte Carlo sampler samples variables along the order $o = (X_1, \ldots, X_n)$ from Q and rejects a partial sample $(x_1, \ldots, x_i)$ if it violates any constraints in **C**. Upon rejecting a sample, the sampler starts sampling anew from the first variable in the ordering. SampleSearch instead modifies the conditional probability as $Q_i(X_i = x_i | x_1, \ldots, x_{i-1}) = 0$ (to reflect that $(x_1, \ldots, x_i)$ is not consistent), normalizes the distribution $Q_i(X_i | x_1, \ldots, x_{i-1})$ and re-samples X_i from the normalized distribution. The newly sampled value may be consistent in which case the algorithm proceeds to variable X_{i+1} or it may be inconsistent. If we repeat the process we may reach a point where $Q_i(X_i | x_1, \ldots, x_{i-1})$ is 0 for all values of X_i. In this case, $(x_1, \ldots, x_{i-1})$ is inconsistent and therefore SampleSearch revises the distribution at X_{i-1} by setting $Q_{i-1}(X_{i-1} = x_{i-1} | x_1, \ldots, x_{i-2}) = 0$, normalizes Q_{i-1} and re-samples a new value for X_{i-1} and so on. SampleSearch repeats this process until a consistent full sample that satisfies all the constraints in **C** is generated.

Algorithm 11: SampleSearch

```
Input: A mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$, an initial proposal distribution
        $Q(\mathbf{X}) = \prod_{i=1}^{n} Q_i(X_i|X_1, \ldots, X_{i-1})$ along an ordering $o = (X_1, \ldots, X_n)$
Output: A consistent full sample $\mathbf{x} = (x_1, \ldots, x_n)$
1  SET i=1, $D'_i = D_i$ (copy domains), $Q'_1(X_1) = Q_1(X_1)$ (copy distribution), $\mathbf{x} = \emptyset$;
2  while $1 \leq i \leq n$ do
       // Forward phase
3      if $D'_i$ is not empty then
4          Sample $X_i = x_i$ from $Q'_i$ and remove it from $D'_i$;
5          if $(x_1, \ldots, x_i)$ violates any constraint in C then
6              SET $Q'_i(X_i = x_i|x_1, \ldots, x_{i-1}) = 0$ and normalize $Q'_i$;
7              Goto step 3.;
8          $\mathbf{x} = \mathbf{x} \cup x_i$, $i = i + 1$, $D'_i = D_i$, $Q'_i(X_i|x_1, \ldots, x_{i-1}) = Q_i(X_i|x_1, \ldots, x_{i-1})$;
       // Backward phase
9      else
10         $\mathbf{x} = \mathbf{x} \backslash x_{i-1}$.;
11         SET $Q'_{i-1}(X_{i-1} = x_{i-1}|x_1, \ldots, x_{i-2}) = 0$ and normalize
           $Q'_{i-1}(X_{i-1}|x_1, \ldots, x_{i-2})$;
12         SET $i = i - 1$;
13 if $i = 0$ then
14     return inconsistent;
15 else
16     return $\mathbf{x}$;
```

The pseudo-code for SampleSearch is given in Algorithm 11. It is a depth first backtracking search (DFS) over the state space of consistent partial assignments searching for a solution to a constraint satisfaction problem $\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$, whose value ordering is stochastically guided by Q that evolves during search. The updated distribution that guides the search is maintained at Q'. The first phase is a forward phase in which the variables are sampled in sequence and a current partial sample (or assignment) is extended by sampling a value x_i for the next variable X_i using the current distribution Q'_i. If for all values $x_i \in \mathbf{D}_i$, $Q'_i(x_i|x_1, \ldots, x_{i-1}) = 0$, then SampleSearch backtracks to the previous variable X_{i-1} (backward phase) and updates the distribution Q'_{i-1} by setting $Q'_{i-1}(x_{i-1}|x_1, \ldots, x_{i-2}) = 0$ and normalizing Q'_{i-1} and continues.

3.2.1 The Sampling Distribution of SampleSearch

Let us denote by $I = \prod_{i=1}^{n} I_i(X_i|X_1, \ldots, X_{i-1})$ the sampling distribution of SampleSearch along the ordering $o = (X_1, \ldots, X_n)$. We will show that I coincides with the backtrack-free distribution Q^F of Q w.r.t. **C** (see Definition 31 in Chapter 2). To recap, given a mixed network $\mathcal{M} = \langle$**X,D,F,C**$\rangle$ and a proposal distribution $Q = \{Q_1, \ldots, Q_n\}$ along an ordering o, the backtrack-free distribution is defined as $Q^F = \{Q_1^F, \ldots, Q_n^F\}$ where:

$$Q_i^F(x_i|x_1, \ldots, x_{i-1}) \begin{cases} = \alpha Q_i(x_i|x_1, \ldots, x_{i-1}) & \text{if } (x_1, \ldots, x_i) \text{ is globally consistent w.r.t. } \mathbf{C} \\ = 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

Let $\mathbf{x}_{i-1} = (x_1, \ldots, x_{i-1})$ and let $\mathbf{B}_i^{\mathbf{x}_{i-1}} = \{x'_i \in \mathbf{D}_i | (x_1, \ldots, x_{i-1}, x'_i)$ is not globally consistent w.r.t. **C** $\}$. Then, α can be expressed by:

$$\alpha = \frac{1}{1 - \sum_{x'_i \in \mathbf{B}_i^{\mathbf{x}_{i-1}}} Q_i(x'_i|x_1, \ldots, x_{i-1})}$$

THEOREM 9 (**Main Result**). *Given a mixed network $\mathcal{M} = \langle$**X,D,F,C**$\rangle$ and an input proposal distribution $Q = \{Q_1, \ldots, Q_n\}$ along o, SampleSearch generates independent and* *identically distributed samples from the backtrack-free probability distribution Q^F of Q w.r.t. **C**, i.e. $\forall\ i\ Q^F_i = I_i$.*

To prove this theorem, we need the following proposition.

PROPOSITION **9.** *Given a mixed network $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$, an initial proposal distribution $Q = \{Q_1, \ldots, Q_n\}$ and a globally consistent partial assignment $(x_1, \ldots, x_{i-1})$ w.r.t. **C**, SampleSearch samples values without replacement from the domain of variable X_i until a globally consistent sample $(x_1, \ldots, x_{i-1}, x_i)$ is generated.*

Proof. Consider a globally inconsistent extension $(x_1, \ldots, x'_i)$ of $(x_1, \ldots, x_{i-1})$. Because SampleSearch is systematic, if $(x_1, \ldots, x'_i)$ is sampled then SampleSearch would eventually detect its inconsistency by not being able to extend it to a solution. At this point, it will set $Q'_i(x'_i|x_1, \ldots, x_{i-1}) = 0$ either in step 6 or step 11 and normalize Q'_i. In other words, x'_i is sampled just once yielding sampling without replacement from $Q'_i(X_i|x_1, \ldots, x_{i-1})$. On the other hand, if a globally consistent extension $(x_1, \ldots, x_i)$ is sampled, SampleSearch will always extend it to a full sample that is consistent. □

The following example demonstrates how the backtrack free distribution is constructed.

EXAMPLE **9.** *Consider the complete search tree corresponding to the proposal distribution and to the constraints given in Figure 3.1. The inconsistent partial assignments are grounded in the figure. Each arc is labeled with the probability of generating the child node from Q given an assignment from the root node to its parent. Consider the full assignment $(A = 0, B = 2, C = 0)$. The five different ways in which this assignment could be generated by SampleSearch (called as DFS-traces) are shown in Figure 3.2. In the following, we show how to compute the probability $I_B(B = 2|A = 0)$ i.e. the probability of sampling $B = 2$ given $A = 0$. Given $A = 0$, the events that could lead to sampling $B = 2$ are shown in Figure 3.2, (a) $\langle B = 2 \rangle | A = 0$ (b) $\langle B = 0, B = 2 \rangle | A = 0$ (c) $\langle B = 3, B = 0 \rangle | A = 0$*

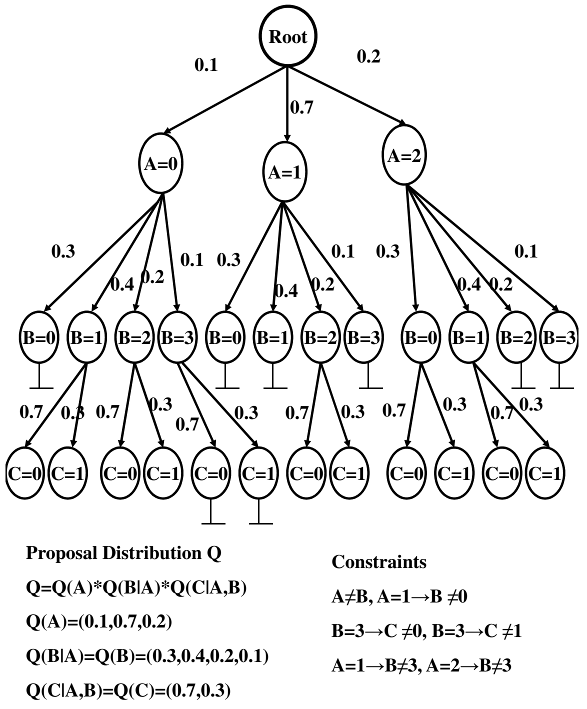


Figure 3.1: A full OR search tree given a set of constraints and a proposal distribution.

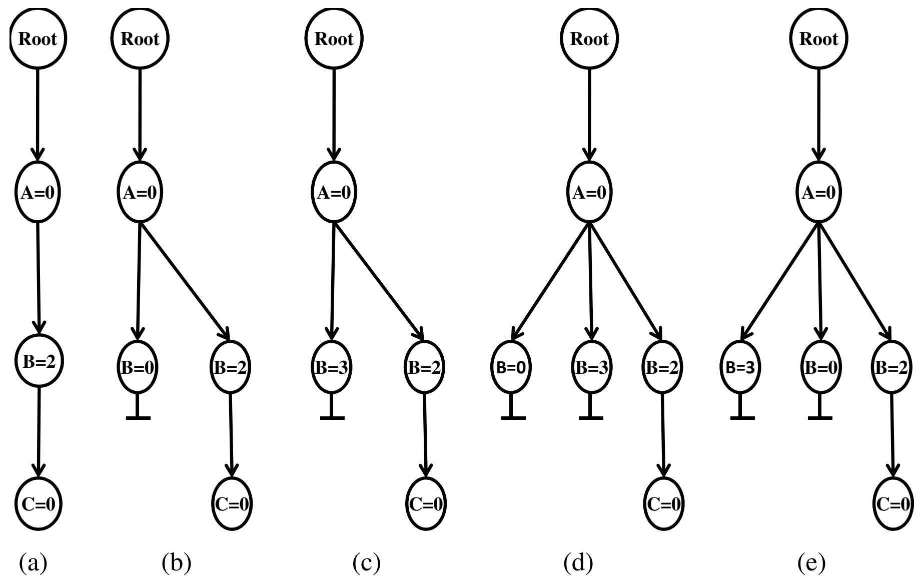


Figure 3.2: Five possible traces of SampleSearch which lead to the sample $(A = 0\ , B = 2\ , C = 0\ )$. The children of each node are specified from left to right in the order in which they are generated.

(d) $\langle B = 0, B = 3, B = 2\rangle | A = 0$ *and (e)* $\langle B = 3, B = 0, B = 2\rangle | A = 0$*. The notation* $\langle B = 3, B = 0, B = 2\rangle | A = 0$ *means that given* $A = 0$*, the states were sampled in the order from left to right* $(B = 3, B = 0, B = 2)$*. Clearly, the probability of* $I_B(B = 2|A = 0)$ *is the sum over the probability of these events. Let us now compute the probability of the event* $\langle B = 3, B = 0, B = 2\rangle | A = 0$*. The probability of sampling* $B = 3|A = 0$ *from* $Q(B|A = 0) = (0.3, 0.4, 0.2, 0.1)$ *is 0.1. The assignment* $(A = 0, B = 3)$ *is inconsistent and therefore the distribution* $Q(B|A = 0)$ *is changed by SampleSearch to* $Q'(B|A = 0) = (0.3/0.9, 0.4/0.9, 0.2/0.9, 0) = (3/9, 4/9, 2/9, 0)$*. Subsequently, the probability of sampling* $B = 0$ *from* Q' *is 3/9. However, the assignment* $(A = 0, B = 0)$ *is also globally inconsistent and therefore the distribution is changed to* $Q''(B|A = 0) \propto (0, 4/9, 2/9, 0) = (0, 2/3, 1/3, 0)$*. Next, the probability of sampling* $B = 2$ *from* Q'' *is* $1/3$*. Therefore, the probability of the event* $\langle B = 3, B = 0, B = 2\rangle | A = 0$ *is* $0.1 \times (3/9) \times (1/3) = 1/90$*. By calculating the probabilities of the remaining events using the approach described above and taking the sum, one can verify that the probability of sampling* $B = 2$ *given* $A = 0$ *i.e.* $I_B(B = 2|A = 0) = 1/3$*.*

We will now show that:

PROPOSITION **10.** *Given a mixed network* $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C}\rangle$*, an initial proposal distribution* $Q = \{Q_1, \ldots, Q_n\}$ *and a globally consistent partial assignment* $(x_1, \ldots, x_{i-1}, x_i)$ *w.r.t.* $\boldsymbol{C}$*, the probability* $I_i(x_i|x_1, \ldots, x_{i-1})$ *of sampling* x_i *given* $(x_1, \ldots, x_{i-1})$ *using SampleSearch is directly proportional to* $Q_i(x_i|x_1, \ldots, x_{i-1})$*. Namely,*

$$I_i(x_i|x_1, \ldots, x_{i-1}) \propto Q_i(x_i|x_1, \ldots, x_{i-1})$$

Proof. The proof is obtained by deriving a general expression for $I_i(x_i|x_1, \ldots, x_{i-1})$, summing the probabilities of all events that can lead to this desired partial sample. Consider a globally consistent partial assignment $\mathbf{x}_{i-1} = (x_1, \ldots, x_{i-1})$. Let us assume that the domain of the next variable X_i given $\mathbf{x}_{i-1}$, denoted by $\mathbf{D}_i^{\mathbf{x}_{i-1}}$ is partitioned into $\mathbf{D}_i^{\mathbf{x}_{i-1}} =$

$\mathbf{R}_i^{\mathbf{x}_{i-1}} \cup \mathbf{B}_i^{\mathbf{x}_{i-1}}$ where $\mathbf{R}_i^{\mathbf{x}_{i-1}} = \{x_i \in \mathbf{D}_i^{\mathbf{x}_{i-1}} | (x_1, \ldots, x_{i-1}, x_i) \textit{ is globally consistent}\}$ and $\mathrm{B}_i^{\mathbf{x}_{i-1}} = \mathbf{D}_i^{\mathbf{x}_{i-1}} \setminus \mathbf{R}_i^{\mathbf{x}_{i-1}}$.

We introduce some notation. Let $\mathbf{B}_i^{\mathbf{x}_{i-1}} = \{x_{i,1}, \ldots, x_{i,q}\}$. Let $j = 1, \ldots, 2^q$ index the sequence of all subsets of $\mathbf{B}_i^{\mathbf{x}_{i-1}}$ with $\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}}$ denoting the j-th element of this sequence. Let $\pi(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}})$ denote the sequence of all permutations of $\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}}$ with $\pi_k(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}})$ denoting the k-th element of this sequence. Finally, let $Pr(\pi_k(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}}), x_i|\mathbf{x}_{i-1})$ be the probability of generating x_i and $\pi_k(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}})$ given $\mathbf{x}_{i-1}$.

The probability of sampling $x_i \in \mathbf{R}_i^{\mathbf{x}_{i-1}}$ given $\mathbf{x}_{i-1}$ is:

$$I_i(x_i|\mathbf{x}_{i-1}) = \sum_{j=1}^{2^q} \sum_{k=1}^{|\pi(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}})|} Pr(\pi_k(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}}), x_i|\mathbf{x}_{i-1}) \tag{3.2}$$

where, $Pr(\pi_k(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}}), x_i|\mathbf{x}_{i-1})$ is given by:

$$Pr(\pi_k(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}}), x_i|\mathbf{x}_{i-1}) = Pr(\pi_k(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}})|\mathbf{x}_{i-1}) Pr(x_i|\pi_k(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}}), \mathbf{x}_{i-1}) \tag{3.3}$$

Substituting Equation 3.3 in Equation 3.2, we get:

$$I_i(x_i|\mathbf{x}_{i-1}) = \sum_{j=1}^{2^q} \sum_{k=1}^{|\pi(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}})|} Pr(\pi_k(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}})|\mathbf{x}_{i-1}) Pr(x_i|\pi_k(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}}), \mathbf{x}_{i-1}) \tag{3.4}$$

where $Pr(x_i|\pi_k(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}}), \mathbf{x}_{i-1})$ is given by an expression that takes the modified proposal distribution into account.

$$Pr(x_i|\pi_k(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}}), \mathbf{x}_{i-1}) = \frac{Q_i(x_i|\mathbf{x}_{i-1})}{1 - \sum_{x_i' \in \mathbf{B}_{i,j}^{\mathbf{x}_{i-1}}} Q_i(x_i'|\mathbf{x}_{i-1})} \tag{3.5}$$

From Equations 3.4 and 3.5, we get:

$$I_i(x_i|\mathbf{x}_{i-1}) = \sum_{j=1}^{2^q} \sum_{k=1}^{|\pi(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}})|} \frac{Q_i(x_i|\mathbf{x}_{i-1})}{1 - \sum_{x_i' \in \mathbf{B}_{i,j}^{\mathbf{x}_{i-1}}} Q_i(x_i'|\mathbf{x}_{i-1})} Pr(\pi_k(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}})|\mathbf{x}_{i-1}) \tag{3.6}$$

$Q_i(x_i|\mathbf{x}_{i-1})$ does not depend on the indices j and k in Equation 3.6 and therefore we can rewrite Equation 3.6 as:

$$I_i(x_i|\mathbf{x}_{i-1}) = Q_i(x_i|\mathbf{x}_{i-1}) \left(\sum_{j=1}^{2^q} \sum_{k=1}^{|\pi(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}})|} \frac{Pr(\pi_k(\mathbf{B}_{i,j}^{\mathbf{x}_{i-1}})|\mathbf{x}_{i-1})}{1 - \sum_{x_i' \in \mathbf{B}_{i,j}^{\mathbf{x}_{i-1}}} Q_i(x_i'|\mathbf{x}_{i-1})} \right) \tag{3.7}$$

The term enclosed in brackets in Equation 3.7 does not depend on x_i and therefore it follows that if $(x_1, \ldots, x_{i-1}, x_i)$ is globally consistent:

$$I_i(x_i|\mathbf{x}_{i-1}) \propto Q_i(x_i|\mathbf{x}_{i-1}) \tag{3.8}$$

which is what we wanted to prove. $\square$

We now have the necessary components to prove Theorem 9:

Proof of Theorem 9. From Proposition 9, $I_i(x_i|\mathbf{x}_{i-1})$ equals zero iff x_i is not globally consistent and from Proposition 10, for all other values, $I_i(x_i|\mathbf{x}_{i-1}) \propto Q_i(x_i|\mathbf{x}_{i-1})$. Therefore, the normalization constant equals $1 - \sum_{x_i' \in \mathbf{B}_i^{\mathbf{x}_{i-1}}} Q_i(x_i'|\mathbf{x}_{i-1})$. Consequently,

$$I_i(x_i|\mathbf{x}_{i-1}) = \frac{Q_i(x_i|\mathbf{x}_{i-1})}{1 - \sum_{x_i' \in \mathbf{B}_i^{\mathbf{x}_{i-1}}} Q_i(x_i'|\mathbf{x}_{i-1})} \tag{3.9}$$

The right hand side of Equation 3.9 is by definition equal to $Q_i^F(x_i|\mathbf{x}_{i-1})$ (see Equation 3.1). $\square$

Computing $Q^F(\mathbf{x})$

Given a set of i.i.d. samples $(\mathbf{x}^1 = (x_1^1, \ldots, x_n^1), \ldots, \mathbf{x}^N = (x_1^N, \ldots, x_n^N))$ generated by SampleSearch from the backtrack free distribution Q^F of Q w.r.t. $\mathbf{C}$, all we need in order to compute the estimates $\widehat{Z}_N$ and $\widetilde{P}_N(x_i)$ is to compute the appropriate weights so that we can substitute them in Equations 2.2 and 2.4. To derive these weights, we need to know the probability $Q^F(\mathbf{x})$ of the given sample $\mathbf{x}$.

From Equation 3.1, we see that to compute $Q_i^F(X_i|\mathbf{x}_{i-1})$, we have to determine the set $\mathbf{B}_i^{\mathbf{x}_{i-1}} = \{x_i' \in \mathbf{D}_i^{\mathbf{x}_{i-1}} | (x_1, \ldots, x_{i-1}, x_i')$ is not globally consistent $\}$. One way to accomplish that, as described in Algorithm 7 in Chapter 2 is to use a complete CSP oracle. The oracle should be invoked a maximum of $n \times (d-1)$ times where n is the number of variables and d is the maximum domain size. Methods such as adaptive consistency [29] (see Algorithm 8 in Chapter 2) or an exact CSP solver can be used as oracles. But then, we have gained nothing by SampleSearch if ultimately we need to use the oracle almost the same number of times as the sampling method presented in Algorithm 7. We will next show how to mitigate this computational problem by approximating the backtrack-free probabilities on the fly during the execution of SampleSearch. We will show that this process still maintains some desired statistical guarantees.

3.2.2 Approximating $Q^F(\mathbf{x})$

During the process of generating the sample $\mathbf{x}$, SampleSearch may have discovered one or more values in the set $\mathbf{B}_i^{\mathbf{x}_{i-1}}$ and thus we can build an approximation for $Q_i^F(x_i|\mathbf{x}_{i-1})$ as follows. Let $\mathbf{A}_i^{\mathbf{x}_{i-1}} \subseteq \mathbf{B}_i^{\mathbf{x}_{i-1}}$ be the set of values in the domain of X_i that were proved to be inconsistent given $\mathbf{x}_{i-1}$ while generating a sample $\mathbf{x}$. We use the set $\mathbf{A}_i^{\mathbf{x}_{i-1}}$ to compute an approximation $T_i^F(x_i|\mathbf{x}_{i-1})$ of $Q_i^F(x_i|\mathbf{x}_{i-1})$ as follows:

$$T_i^F(x_i|\mathbf{x}_{i-1}) = \frac{Q_i(x_i|\mathbf{x}_{i-1})}{1 - \sum_{x_i' \in \mathbf{A}_i^{\mathbf{x}_{i-1}}} Q_i(x_i'|\mathbf{x}_{i-1})} \tag{3.10}$$

Finally we compute $T^F(\mathbf{x}) = \prod_{i=1}^{n} T_i^F(x_i|\mathbf{x}_{i-1})$. However, $T^F(\mathbf{x})$ does not guarantee asymptotic unbiasedness when replacing $Q^F(\mathbf{x})$ for computing the weight $w^F(\mathbf{x})$ in Equation 2.3.

To remedy the situation, we can store each sample $(x_1, \ldots, x_n)$ and all its partial assignments $(x_1, \ldots, x_{i-1}, x_i')$ that were proved inconsistent during each trace of an independent execution of SampleSearch called DFS-traces (for example, Figure 3.2 shows the five DFS-traces that could generate the sample $(A = 0, B = 2, C = 0)$). After executing SampleSearch N times generating N samples, we can use all the stored DFS-traces to compute an approximation of $Q^F(\mathbf{x})$ as illustrated in the following example.

EXAMPLE **10.** *Consider the three traces given in Figure 3.3 (a). We can combine the information from the three traces as shown in Figure 3.3(b). Consider the assignment* $(A = 1, B = 2)$*. The backtrack-free probability of generating* $B = 2$ *given* $A = 1$ *requires the knowledge of all the values of* B *which are inconsistent. Based on the combined traces, we know that* $B = 0$ *and* $B = 1$ *are inconsistent (given* $A = 1$*) but we do not know whether* $B = 3$ *is consistent or not because it is not explored (indicated by "???" in Figure 3.3(b)). Setting the unexplored nodes to either inconsistent or consistent give us the two different approximations as shown in Figure 3.3(c).*

Generalizing Example 10, we consider two bounding approximations denoted by U_N^F and L_N^F respectively (the approximation is indexed to denote dependence on N) which are based on setting each unexplored node in the combined N traces to consistent or inconsistent respectively. As we will show, these approximations can be used to bound the sample

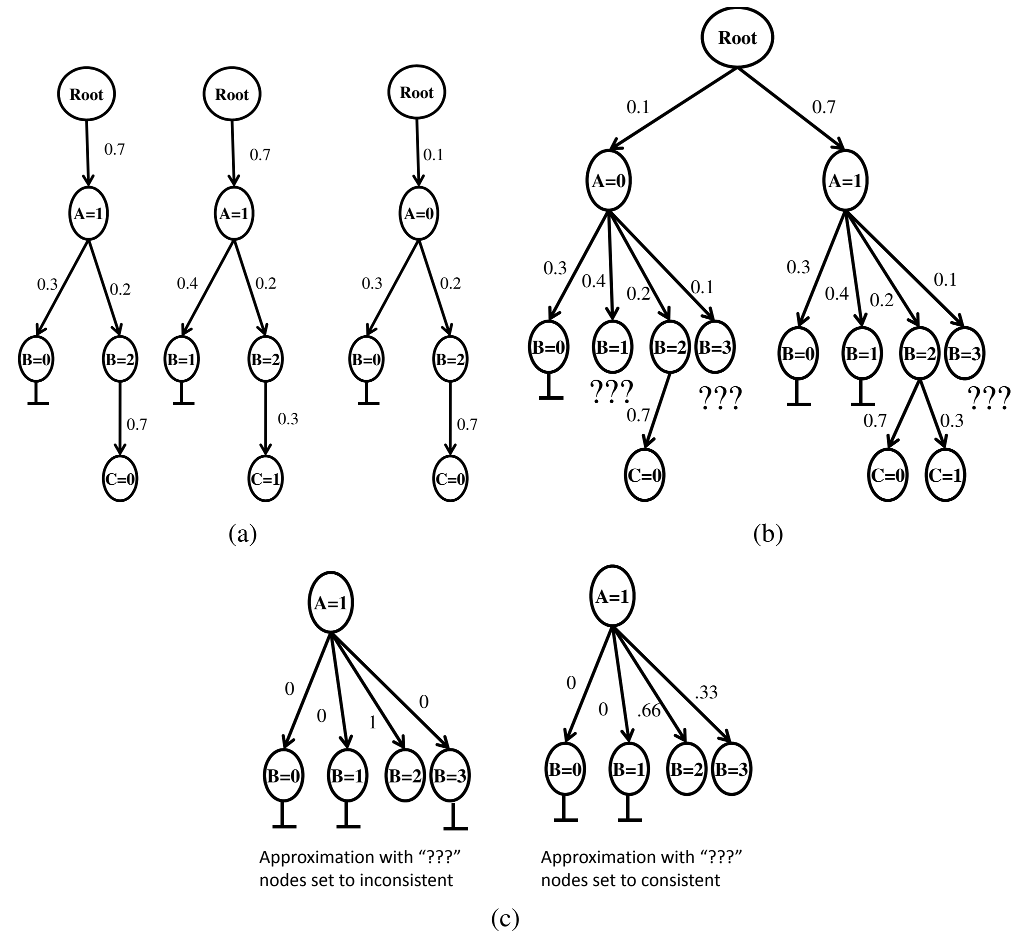


Figure 3.3: (a) Three DFS-traces (b) Combined information from the three DFS-traces given in (a) and (c) Two possible approximations of $I(B|A = 1)$

mean $\widehat{Z}_N$ from above and below respectively [1].

DEFINITION **34** (**Upper and Lower Approximations of Q^F by U_N^F and L_N^F**). *Given a mixed network $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$, an initial proposal distribution $Q = \{Q_1, \ldots, Q_n\}$, a combined sample tree generated from N independent runs of SampleSearch and a partial sample $\boldsymbol{x}_{i-1} = (x_1, \ldots, x_{i-1})$ generated in one of the N independent runs, we define two sets:*

- $\mathbf{A}_{N,i}^{\boldsymbol{x}_{i-1}} \subseteq \boldsymbol{B}_i^{\boldsymbol{x}_{i-1}} = \{x_i \in \boldsymbol{D}_i^{\boldsymbol{x}_{i-1}} |\ (x_1, \ldots, x_{i-1}, x_i)$ *was proved to be inconsistent during the N independent runs of SampleSearch* $\}$.
- $\mathbf{C}_{N,i}^{\boldsymbol{x}_{i-1}} \subseteq \mathbf{D}_i^{\boldsymbol{x}_{i-1}} = \{x_i \in \boldsymbol{D}_i^{\boldsymbol{x}_{i-1}} |\ (x_1, \ldots, x_{i-1}, x_i)$ *was not explored during the N independent runs of SampleSearch* $\}$.

We can set all the nodes in $\mathbf{C}_{N,i}^{\boldsymbol{x}_{i-1}}$ (i.e. the nodes which are not explored) either to consistent or inconsistent yielding:

$$U_N^F(\boldsymbol{x}) = \prod_{i=1}^{n} U_{N,i}^F(x_i|\mathbf{x}_{i-1}) \ \ where$$

$$U_{N,i}^F(x_i|\mathbf{x}_{i-1}) = \frac{Q_i(x_i|\mathbf{x}_{i-1})}{1 - \sum_{x'_i \in \mathbf{A}_{N,i}^{\boldsymbol{x}_{i-1}}} Q_i(x'_i|\mathbf{x}_{i-1})} \tag{3.11}$$

$$L_N^F(\boldsymbol{x}) = \prod_{i=1}^{n} L_{N,i}^F(x_i|\mathbf{x}_{i-1}) \ \ where$$

$$L_{N,i}^F(x_i|\mathbf{x}_{i-1}) = \frac{Q_i(x_i|\mathbf{x}_{i-1})}{1 - \sum_{x'_i \in \mathbf{A}_{N,i}^{\boldsymbol{x}_{i-1}} \cup \mathbf{C}_{N,i}^{\boldsymbol{x}_{i-1}}} Q_i(x'_i|\mathbf{x}_{i-1})} \tag{3.12}$$

[1]Note that it is easy to envision other approximations in which we designate some unexplored nodes as consistent while others as inconsistent based on the domain knowledge or via some other Monte Carlo estimate. We consider the two extreme options because they usually work well in practice and bound the sample mean from above and below.

It is clear that as N grows, the sample tree grows and therefore more inconsistencies will be discovered and as $N \to \infty$, all inconsistencies will be discovered making the respective sets approach $\mathbf{A}_{N,i}^{\mathbf{x}_{i-1}} = \mathbf{B}_{i}^{\mathbf{x}_{i-1}}$ and $\mathbf{C}_{N,i}^{\mathbf{x}_{i-1}} = \phi$. Clearly then,

PROPOSITION **11.** $\lim_{N\to\infty} U_N^F(\mathbf{x}) = \lim_{N\to\infty} L_N^F(\mathbf{x}) = Q^F(\mathbf{x})$

As before, given a set of i.i.d. samples $(\mathbf{x}^1 = (x_1^1, \ldots, x_n^1), \ldots, \mathbf{x}^N = (x_1^N, \ldots, x_n^N))$ generated by $SampleSearch$, we can estimate the weighted counts Z using the two statistics $U_N^F(\mathbf{x})$ and $L_N^F(\mathbf{x})$ by:

$$\widetilde{Z}_N^U = \frac{1}{N}\sum_{k=1}^{N} \frac{\prod_{i=1}^{m} F_i(\mathbf{x}^k)\prod_{j=1}^{p} C_j(\mathbf{x}^k)}{U_N^F(\mathbf{x}^k)} = \frac{1}{N}\sum_{k=1}^{N} w_N^U(\mathbf{x}^k) \tag{3.13}$$

where

$$w_N^U(\mathbf{x}^k) = \frac{\prod_{i=1}^{m} F_i(\mathbf{x}^k)\prod_{j=1}^{p} C_j(\mathbf{x}^k)}{U_N^F(\mathbf{x}^k)}$$

is the weight of the sample based on the combined sample tree using the upper approximation U_N^F.

$$\widetilde{Z}_N^L = \frac{1}{N}\sum_{k=1}^{N} \frac{\prod_{i=1}^{m} F_i(\mathbf{x}^k)\prod_{j=1}^{p} C_j(\mathbf{x}^k)}{L_N^F(\mathbf{x}^k)} = \frac{1}{N}\sum_{k=1}^{N} w_N^L(\mathbf{x}^k) \tag{3.14}$$

where

$$w_N^L(\mathbf{x}^k) = \frac{\prod_{i=1}^{m} F_i(\mathbf{x}^k)\prod_{j=1}^{p} C_j(\mathbf{x}^k)}{L_N^F(\mathbf{x}^k)}$$

is the weight of the sample based on combined sample tree using the lower approximation L_N^F.

Similarly, for marginals, we can develop the statistics.

$$\widetilde{P}_N^U(x_i) = \frac{\sum_{k=1}^{N} w_N^U(\mathbf{x}^k)\delta_{x_i}(\mathbf{x}^k)}{\sum_{k=1}^{N} w_N^U(\mathbf{x}^k)} \tag{3.15}$$

and

$$\widetilde{P}^L_N(x_i) = \frac{\sum_{k=1}^{N} w^L_N(\mathbf{x}^k)\delta_{x_i}(\mathbf{x}^k)}{\sum_{k=1}^{N} w^L_N(\mathbf{x}^k)} \tag{3.16}$$

Clearly,

THEOREM **10.** $\widetilde{Z}^L_N \leq \widehat{Z}_N \leq \widetilde{Z}^U_N$.

Proof. Because, $\mathbf{B}_i^{\mathbf{x}_{i-1}} \subseteq \mathbf{A}_{N,i}^{\mathbf{x}_{i-1}} \cup \mathbf{C}_{N,i}^{\mathbf{x}_{i-1}}$, we have:

$$\begin{aligned}
\sum_{x'_i \in \mathbf{B}_i^{\mathbf{x}_{i-1}}} Q_i(x'_i|\mathbf{x}_{i-1}) &\leq \sum_{x'_i \in \mathbf{A}_{N,i}^{\mathbf{x}_{i-1}} \cup \mathbf{C}_{N,i}^{\mathbf{x}_{i-1}}} Q_i(x'_i|\mathbf{x}_{i-1}) && (3.17)\\
\therefore 1 - \sum_{x'_i \in \mathbf{B}_i^{\mathbf{x}_{i-1}}} Q_i(x'_i|\mathbf{x}_{i-1}) &\geq 1 - \sum_{x'_i \in \mathbf{A}_{N,i}^{\mathbf{x}_{i-1}} \cup \mathbf{C}_{N,i}^{\mathbf{x}_{i-1}}} Q_i(x'_i|\mathbf{x}_{i-1}) && (3.18)\\
\therefore \frac{Q_i(x_i|\mathbf{x}_{i-1})}{1 - \sum_{x'_i \in \mathbf{B}_i^{\mathbf{x}_{i-1}}} Q_i(x'_i|\mathbf{x}_{i-1})} &\leq \frac{Q_i(x_i|\mathbf{x}_{i-1})}{1 - \sum_{x'_i \in \mathbf{A}_{N,i}^{\mathbf{x}_{i-1}} \cup \mathbf{C}_{N,i}^{\mathbf{x}_{i-1}}} Q_i(x'_i|\mathbf{x}_{i-1})} && (3.19)\\
\therefore Q^F_i(x_i|\mathbf{x}_{i-1}) &\leq L^F_{N,i}(x_i|\mathbf{x}_{i-1}) && (3.20)\\
\therefore \prod_{i=1}^{n} Q^F_i(x_i|\mathbf{x}_{i-1}) &\leq \prod_{i=1}^{n} L^F_{N,i}(x_i|\mathbf{x}_{i-1}) && (3.21)\\
\therefore Q^F(\mathbf{x}) &\leq L^F_N(\mathbf{x}) && (3.22)\\
\therefore \frac{\prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x})}{Q^F(\mathbf{x})} &\geq \frac{\prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x})}{L^F_N(\mathbf{x})} && (3.23)\\
\therefore w^F(\mathbf{x}) &\geq w^F_L(\mathbf{x}) && (3.24)\\
\therefore \frac{1}{N}\sum_{k=1}^{N} w^F(\mathbf{x}^k) &\geq \frac{1}{N}\sum_{k=1}^{N} w^F_L(\mathbf{x}^k) && (3.25)\\
\therefore \widehat{Z}_N &\geq \widetilde{Z}^L_N && (3.26)
\end{aligned}$$

Similarly, by using $\mathbf{A}_{N,i}^{\mathbf{x}_{i-1}} \subseteq \mathbf{B}_i^{\mathbf{x}_{i-1}}$, it is easy to prove that $\widehat{Z}^F_N \leq \widetilde{Z}^U_N$. □

We can prove that:

THEOREM **11.** *The estimates $\widetilde{Z}^U_N$ and $\widetilde{Z}^L_N$ of Z given in Equations 3.13 and 3.14 respectively are asymptotically unbiased. Similarly, the estimates $\widetilde{P}^U_N(x_i)$ and $\widetilde{P}^L_N(x_i)$ of $P(x_i)$ given in Equations 3.15 and 3.16 respectively are asymptotically unbiased.*

Proof. From Proposition 11, it follows that U_N^F and L_N^F in the limit of infinite samples coincide with the backtrack-free distribution Q^F. Therefore,

$$\lim_{N\to\infty} w_N^L(\mathbf{x}) = \lim_{N\to\infty} \frac{\prod_{i=1}^m F_i(\mathbf{x}) \prod_{j=1}^p C_j(\mathbf{x})}{L_N^F(\mathbf{x})} \tag{3.27}$$

$$= \frac{\prod_{i=1}^m F_i(\mathbf{x}) \prod_{j=1}^p C_j(\mathbf{x})}{Q^F(\mathbf{x})} \tag{3.28}$$

$$= w^F(\mathbf{x}) \tag{3.29}$$

Therefore,

$$\lim_{N\to\infty} \mathbb{E}_Q\left[\frac{1}{N}\sum_{k=1}^{N} w^L(\mathbf{x})\right] = \lim_{N\to\infty} \frac{1}{N}\sum_{\mathbf{x}\in\mathbf{X}} w_N^L(\mathbf{x})Q(\mathbf{x})\sum_{k=1}^{N}(1) \tag{3.30}$$

$$= \frac{1}{N} \times N \lim_{N\to\infty} \sum_{\mathbf{x}\in\mathbf{X}} w_N^L(\mathbf{x})Q(\mathbf{x}) \tag{3.31}$$

$$= \sum_{\mathbf{x}\in\mathbf{X}} w^F(\mathbf{x})Q(\mathbf{x}) \ldots \text{(From Equation 3.29)} \tag{3.32}$$

$$= Z \tag{3.33}$$

Similarly, we can prove that the estimator based on U_N^F in Equation 3.13 is asymptotically unbiased by replacing $w_N^L(\mathbf{x})$ with $w_N^U(\mathbf{x})$ in Equations 3.30-3.33.

Finally, because the estimates $\widetilde{P}_N^U(x_i)$ and $\widetilde{P}_N^L(x_i)$ of $P(x_i)$ given in Equations 3.15 and 3.16 respectively are ratios of two asymptotically unbiased estimators, by definition, they are asymptotically unbiased too. □

PROPOSITION **12.** *Given N samples output by SampleSearch for a mixed network $\mathcal{M} = \langle X, D, F, C\rangle$, the space and time complexity of computing $\widetilde{Z}_N^L$, $\widetilde{Z}_N^U$, $\widetilde{P}_N^L(x_i)$ and $\widetilde{P}_N^U(x_i)$ given in Equations 3.14, 3.13, 3.16 and 3.15 is $O(N \times d \times n)$.*

Proof. Because we store all full solutions $(x_1, \ldots, x_n)$ and all partial assignments $(x_1, \ldots, x_{i-1}, x_i')$ that were proved inconsistent during the N executions of SampleSearch, we require an additional $O(N \times n \times d)$ space to store the combined sample tree used to estimate Z and the marginals. Similarly, because we compute a sum or their ratios by visiting all nodes of this combined sample tree, the time complexity is also $O(N \times d \times n)$ $\square$

In summary, we presented two approximations to the backtrack-free probability Q^F which bound the sample mean $\widehat{Z}_N$. We proved that the two approximations yield an asymptotically unbiased estimate of the weighted counts and marginals. They will also enable trading bias with variance as we discuss next.

Bias-Variance Tradeoff

As pointed in Section 1.4.2, the mean squared error of an estimator can be reduced by either controling the bias or by increasing the number of samples. The estimators $\widetilde{Z}_N^U$ and $\widetilde{Z}_N^L$ have more bias than the unbiased estimator $\widehat{Z}_N^F$ (which has a bias of zero but requires an exact CSP solver). However, we expect the estimators $\widetilde{Z}_N^U$ and $\widetilde{Z}_N^L$ to allow larger sample size than $\widehat{Z}_N^F$. Moreover, $\widetilde{Z}_N^U$ and $\widetilde{Z}_N^L$ bound $\widehat{Z}_N^F$ from above and below and therefore the absolute distance $|\widetilde{Z}_N^U - \widetilde{Z}_N^L|$ can be used to estimate their bias. If $|\widetilde{Z}_N^U - \widetilde{Z}_N^L|$ is small enough, then we can expect $\widetilde{Z}_N^U$ and $\widetilde{Z}_N^L$ to perform better than $\widehat{Z}_N^F$ because they can be based on a larger sample size.

3.2.3 Incorporating Advanced Search Techniques in SampleSearch

Theorem 9 is applicable to any search procedure that is systematic i.e. once the search procedure encounters an assignment $(x_1, \ldots, x_i)$, it will either prove that the assignment is inconsistent or return with a full consistent sample extending $(x_1, \ldots, x_i)$. Therefore, we can use any advanced systematic search technique [29] instead of naive backtracking within SampleSearch and easily show that:

PROPOSITION **13.** *Given a mixed network $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$ and an initial proposal distribution $Q = \{Q_1, \ldots, Q_n\}$, SampleSearch augmented with any systematic advanced search technique generates independent and identically distributed samples from the backtrack-free probability distribution Q^F of Q w.r.t.* ***C****.*

While advanced search techniques would not change the sampling distribution of SampleSearch, they can have a significant impact on its time complexity. In particular, since SAT solvers are very efficient, we can represent the constraints in the mixed network using CNF expressions [2] and use minisat [125] as our SAT solver. However, we have to make minisat [125] (or any other state-of-the-art SAT solver like RSAT [108]) *systematic* via the following changes:

- **Turn off random restarts and far backtracks** The use of restarts and far backtracks makes a SAT solver non-systematic and therefore they cannot be used.
- **Change variable and value Ordering** We change the variable ordering to respect the structure of the input proposal distribution Q, namely given $Q(\mathbf{X}) = \prod_{i=1}^{n} Q_i$ $(X_i| \ X_1 \ , \ldots \ , X_{i-1})$, we order variables as $(X_1, \ldots, X_n)$. Also, at each decision point, variable X_i is assigned a value x_i by sampling it from $Q_i(X_i \ | x_1 \ , \ldots, x_{i-1})$.

3.3 Empirical Evaluation

In this section, we describe results of our extensive empirical study evaluating the performance of SampleSearch and comparing with other approaches. We conducted experiments on three tasks: (a) counting models of SAT formula, (b) computing probability of evidence and partition function in a Bayesian and Markov network respectively and (c) computing

[2] It is easy to convert any (relational) constraint network to a CNF formula (see [128] for details). In our implementation, we use the direct encoding described in [128].

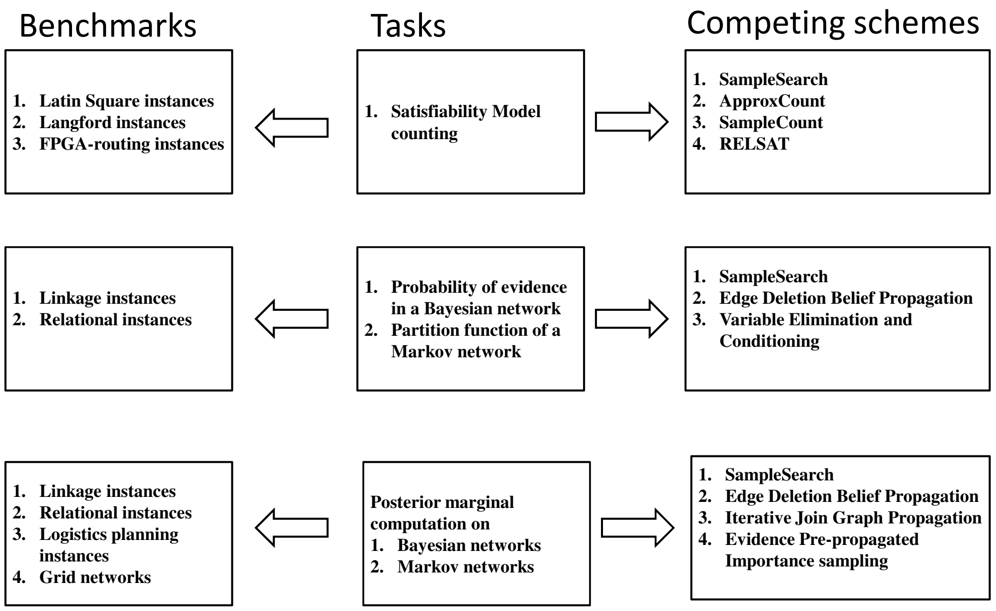

Benchmarks
Tasks
Competing schemes
1. Latin Square instances
2. Langford instances
3. FPGA-routing instances
1. Satisfiability Model counting
1. SampleSearch
2. ApproxCount
3. SampleCount
4. RELSAT
1. Linkage instances
2. Relational instances
1. Probability of evidence in a Bayesian network
2. Partition function of a Markov network
1. SampleSearch
2. Edge Deletion Belief Propagation
3. Variable Elimination and Conditioning
1. Linkage instances
2. Relational instances
3. Logistics planning instances
4. Grid networks
Posterior marginal computation on
1. Bayesian networks
2. Markov networks
1. SampleSearch
2. Edge Deletion Belief Propagation
3. Iterative Join Graph Propagation
4. Evidence Pre-propagated Importance sampling

Figure 3.4: Chart showing the scope of our experimental study

posterior marginals in a Bayesian and Markov network. The benchmarks and competing techniques for each task type are shown in Figure 3.4.

In the next subsection, we present implementation details of SampleSearch and other competing schemes. In Subsection 3.3.3, we focus on the three weighted counting tasks while in Subsection 3.3.4, we focus on the posterior marginals task.

3.3.1 SampleSearch with w-cutset and IJGP

In our experiments, we show how SampleSearch operates within the framework of w-cutset sampling [7] (see Section 1.4.4). Thus, we run SampleSearch only on a subset of the variables that form a w-cutset. We use IJGP to generate a proposal distribution and we used minisat [125] as a search scheme within SampleSearch. Below, we outline the details.

- *The Proposal distribution*: As in IJGP-sampling (see Algorithm 6 in Chapter 2), we obtain $Q = \{Q_1, \ldots, Q_n\}$ from the output of Iterative Join graph propagation

Algorithm 12: SampleSearch: Implementation Details

```
  Input: A mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$, integers $i$, $N$ and $w$.
  Output: A set of $N$ samples globally consistent with respect to $\mathbf{C}$
1 Create a min-fill ordering $o = (X_1, \ldots, X_n)$.;
2 Create a join-graph $JG$ with $i$-bound $i$ along $o$ using the join-graph structuring
  algorithm given in Algorithm 2 in Chapter 1 and run IJGP on it.;
3 Create a $w$-cutset $\mathbf{K} \subseteq \mathbf{X}$. Let $\mathbf{K} = \{K_1, \ldots, K_t\}$;
4 Create a proposal distribution $Q(\mathbf{K}) = \prod_{i=1}^{t} Q_i(K_i|K_1, \ldots, K_{i-1})$ from the
  messages and functions in $JG$ using the max heuristic described in Section 2.4 of
  Chapter 2. To recap, in this scheme, we find a cluster $A$ in $JG$ that mentions $K_i$ and
  has the largest number of variables common with the previous variables
  $\{K_1, \ldots, K_{i-1}\}$. Then, we construct $Q_i(K_i|K_1, \ldots, K_{i-1})$ by marginalizing out all
  variables not mentioned in $K_1, \ldots, K_i$ from the marginal over the variables of $A$;
5 for i=1 to N do do
6     Apply minisat based SampleSearch on the mixed network restricted to $\mathbf{K}$ with
      proposal distribution $Q(\mathbf{K})$ to get a sample $\mathbf{k}$.;
7     Output $\mathbf{k}$.;
```

(IJGP) [33]. Recall that IJGP is a generalized belief propagation [132] technique for approximating the posterior distribution in graphical models (for more details see Section 1.4.1). It runs the same message passing as join-tree propagation on a join graph rather than a tree. The time and space complexity of IJGP can be controlled by an i-bound (IJGP is exponential in its i-bound) and its accuracy generally increases with the i-bound. In our experiments, for every instance, we select the maximum i-bound that can be accommodated by 512 MB of space as follows.

Note that the space required by a message (or a function) is the product of the domain sizes of the variables in its scope. Given an i-bound, we can create a join graph using the join graph structuring scheme outlined in Algorithm 2 in Chapter 1 and compute, in advance, the space required by IJGP by summing over the space required by individual messages [3]. We iterate from $i = 1$ until the space bound is surpassed. This ensures that IJGP terminates in a reasonable amount of time and requires bounded space.

[3]Note that we can do this without constructing the messages explicitly

- *w-cutset sampling:* We combined SampleSearch with w-cutset sampling [7] (see Section 1.4.4) as follows. We partition the variables $\mathbf{X}$ into two sub-sets $\mathbf{K}$ and $\mathbf{R}$ such that the treewidth of the graphical model restricted to $\mathbf{R}$ after removing $\mathbf{K}$ is bounded by w. We only sample variables in $\mathbf{K}$ using a proposal distribution $Q(\mathbf{K})$ (see Equation 1.43 in Chapter 1). The weight of each sample $\mathbf{K} = \mathbf{k}$ is then given by a ratio between the weighted counts of the sub-problem over $\mathbf{R}$ given $\mathbf{k}$ and $Q(\mathbf{K})$. The exact counts can be computed using an exact inference technique such as bucket elimination. It was demonstrated that the higher the w-bound [7], the lower the sampling variance. Here also, we select the maximum w such that the resulting bucket elimination algorithm uses less than 512 MB of space. We can choose the appropriate w by using a similar iterative scheme to the one described above.

- *Variable Ordering:* We use the min-fill ordering for creating the join-graph for IJGP because it was shown to be a good heuristic for finding tree decompositions with low treewidth. Sampling is performed in reverse min-fill ordering.

The details of SampleSearch embedded in w-cutset sampling and IJGP-based proposal is given in Algorithm 12. The algorithm takes as input a mixed network and integer i, w and N which specify the i-bound for IJGP, w for creating a w-cutset and the number of samples N respectively [4]. In steps 1-2, the algorithm creates a join graph along the min fill ordering and runs IJGP. Then, in step 3, it computes a w-cutset $\mathbf{K}$ for the mixed network. In step 4, the algorithm creates a proposal distribution over the w-cutset $\mathbf{K}$, $Q(\mathbf{K}) = \prod_{i=1}^{t} Q_i(K_i|K_1, \ldots, K_{i-1})$ from the output of IJGP using the max-heuristic described in Section 2.4 in Chapter 2. Finally, the algorithm executes minisat based SampleSearch on the mixed network restricted to the w-cutset to generate the required N samples.

Note that we will refer to the estimates of SampleSearch generated using the upper and lower approximations of the backtrack-free probability given by Equations 3.13 and 3.14,

[4]This is done after we determine the i-bound and the w for the w-cutset

as SampleSearch/UB and SampleSearch/LB respectively (or SS/UB or SS/LB in short).

3.3.2 Other Competing Schemes

We experimented also with the following schemes.

1. Iterative Join Graph Propagation (IJGP)

In our experiments, we used an anytime version of IJGP (for more details see Section 1.4.1) in which we start with i-bound of 1, run IJGP until convergence or 10 iterations whichever is earlier. Then we increase the i-bound by one and reconstruct the join graph. We do this until one the following conditions is met: (a) i equals the treewidth in which case IJGP yields exact marginals or (b) the 2 GB space limit is reached or (c) the prescribed time-bound is reached.

3. ApproxCount and SampleCount

Wei and Selman [131] introduced an approximate solution counting scheme called ApproxCount. ApproxCount is based on the formal result of [126] that if one can sample uniformly (or close to it) from the set of solutions of a SAT formula F, then one can exactly count (or approximate with a good estimate) the number of solutions of F. Consider a SAT formula F with S solutions. If we are able to sample solutions uniformly, then we can exactly compute the fraction of the number of solutions, denoted by γ that have a variable X set to $True$ or 1 (and similarly to $False$ or 0). If γ is greater than zero, we can set X to that particular value and simplify F to F'. The estimate of the number of solutions is now equal to the product of $\frac{1}{\gamma}$ and the number of solutions of F'. Then, we recursively repeat the process, leading to a series of multipliers, until all variables are assigned a value or until the conditioned formula is easy for exact model counters like cachet [119]. To reduce the variance, [131] suggest to set the selected variable to a value that occurs more often in the sample. In this scheme, the fraction for each variable branching is selected via a solution sampling method called SampleSat [130], which is an extension of the well-known local search SAT solver Walksat [120]. We experimented with an anytime version of ApproxCount in which we report the cumulative average accumulated over several runs.

SampleCount [62] employs ApproxCount to yield a probabilistic lower bound on the number of solutions. SampleCount reduces the variance in ApproxCount by branching on variables which are more balanced i.e. variables having multipliers close to 2. Unlike ApproxCount, SampleCount assigns a value to a variable by sampling it with probability 0.5 yielding an unbiased estimate of the solution counts. We experimented with an anytime version of SampleCount in which we report the unbiased cumulative averages over several runs.

In our experiments, we used an implementation of ApproxCount and SampleCount available from the respective authors [130, 62]. Following the recommendations made in previous work by the authors [62], we use the following parameters for ApproxCount and SampleCount: (a) Number of samples for SampleSat $N_S = 20$, (b) Number of variables remaining to be assigned a value before running Cachet $N_R = 100$ and (c) local search cutoff $\alpha = 100K$.

4. Evidence Pre-propagation Importance sampling

The Evidence Pre-propagation Importance Sampling (EPIS) algorithm is an adaptive importance sampling algorithm for computing marginals in Bayesian networks [133]. The algorithm uses loopy belief propagation [106, 99] to construct the proposal distribution. We use the implementation of EPIS included in the Smile genie library available publicly.

5. Edge Deletion Belief Propagation

Edge deletion belief propagation (EDBP) [18] is an approximation algorithm for computing posterior marginals and for computing probability of evidence. EDBP solves exactly a simplified version of the original problem, obtained by deleting some of the edges of the problem graph. Deleted edges are selected based on two criteria : quality of approximation and complexity of computation (tree-width reduction) which is parameterized by an integer k, called the k-bound. Subsequently, information loss from the lost dependencies is compensated for by using several heuristic techniques (see [18] for details). The implementation of this scheme is available from the authors [18].

6. Variable Elimination + Conditioning (VEC)

When a problem having a high treewidth is encountered, variable or bucket elimination may be unsuitable, primarily because of its extensive memory demand. To alleviate the space complexity, we can use the w-cutset conditioning scheme (see Section 1.3.2). Namely, we condition or instantiate enough variables (or the w-cutset) so that the remaining problem after removing the instantiated variables can be solved exactly using bucket elimination [28]. In our experiments we select the w-cutset in such a way that bucket elimination would require less than 1.5GB of space. Exact weighted counts can be computed by summing over the exact solution output by bucket elimination for all possible instantiations of the w-cutset. When VEC is terminated before completion, it outputs a partial sum yielding a lower bound on the weighted counts. The implementation of this scheme is available publicly from our software website [30].

7. Relsat

Relsat [115] is an exact algorithm for counting solutions of a satisfiability problem. When Relsat is stopped before completion, it yields a lower bound on the number of solutions. The implementation of Relsat is available publicly from the authors web site [115].

Table 3.1 summarizes different query types that can be handled by the various solvers. A '$\surd$' indicates that the algorithm is able to approximately estimate the query while a lack of

Problem Type	SampleSearch	IJGP	EDBP	EPIS-BN	VEC	SampleCount ApproxCount Relsat
Bayesian networks $P(e)$	$\surd$		$\surd$		$\surd$	
Markov Networks Z	$\surd$		$\surd$		$\surd$	
Bayesian networks Mar	$\surd$	$\surd$	$\surd$	$\surd$		
Markov networks Mar	$\surd$	$\surd$	$\surd$			
Model counting	$\surd$					$\surd$

Z: partition function, P(e): probability of evidence and Mar: posterior marginals.

Table 3.1: Query types handled by various solvers.

$\surd$ indicates otherwise.

3.3.3 Results for Weighted Counts

Satisfiability instances

For the task of weighted counts, we evaluate the algorithms on formulas from three domains: (a) normalized Latin square problems, (b) Langford problems, (c) FPGA-Routing instances. We ran each algorithm for 10 hours on each instance. We will refer to the implemented algorithms as solvers.

Notation in Tables

The first column in each table (see Table 3.2 for example) gives the name of the instance. The second column provides various statistical information about the instance such as: (i) number of variables (n), (ii) average domain size (d), (iii) number of clauses (c) or number of evidence variables (e) and (iv) the treewidth of the instance (w) (computed using the min-fill heuristic). The third column provides the exact answer for the problem if available while the remaining columns display the actual output produced by the various solvers when terminated at the specified time-bound. The solver(s) giving the best results is highlighted in each row. A "*" next to the output of a solver indicates that it solved the

Problem	$\langle n, k, c, w\rangle$	**Exact**	**Sample Count**	**Approx Count**	**REL SAT**	**SS /LB**	**SS /UB**
ls8-norm.cnf	$\langle 512, 2, 5584, 255\rangle$	5.40E11	**5.15E+11**	3.52E+11	2.44E+08	5.91E+11	5.91E+11
ls9-norm.cnf	$\langle 729, 2, 9009, 363\rangle$	3.80E17	4.49E+17	1.26E+17	1.78E+08	**3.44E+17**	3.44E+17
ls10-norm.cnf	$\langle 1000, 2, 13820, 676\rangle$	7.60E24	**7.28E+24**	1.17E+24	1.36E+08	6.74E+24	6.74E+24
ls11-norm.cnf	$\langle 1331, 2, 20350, 956\rangle$	5.40E33	2.08E+34	4.91E+31	1.09E+08	**3.87E+33**	3.87E+33
ls12-norm.cnf	$\langle 1728, 2, 28968, 1044\rangle$		**2.23E+43**	6.54E+39	1.26E+08	1.25E+43	1.25E+43
ls13-norm.cnf	$\langle 2197, 2, 40079, 1558\rangle$		3.20E+54	1.41E+50	9.32E+07	**1.15E+55**	**1.15E+55**
ls14-norm.cnf	$\langle 2744, 2, 54124, 1971\rangle$		5.08E+65	5.96E+60	3.71E+07	**1.24E+70**	**1.24E+70**
ls15-norm.cnf	$\langle 3375, 2, 71580, 2523\rangle$		3.12E+79	2.82E+72	2.06E+07	**2.03E+83**	**2.03E+83**
ls16-norm.cnf	$\langle 4096, 2, 92960, 2758\rangle$		7.68E+95	2.04E+85	X	**2.08E+98**	**2.08E+98**

Table 3.2: Solution counts output by SampleSearch, ApproxCount, SampleCount and Relsat after 10 hours of CPU time for Latin Square instances. When the exact counts are not known, the entries for SampleCount and SS/LB contain the lower bounds computed by combining their respective sample weights with the Markov inequality based Average and Martingale schemes.

problem exactly while a "X" indicates that no solution was output.

When the exact weighted counts are not known, we compare the lower bounds on the weighted counts Z obtained by combining the estimates output by SampleSearch/LB and SampleCount with the Markov inequality based average and Martingale schemes; which we will present in Chapter 5. These lower bounding schemes take as input: (a) a set of unbiased sample weights and (b) a real number $0 < \alpha < 1$ and output a lower bound on Z that is correct with probability greater than α. In our experiments, we set $\alpha = 0.99$. SampleSearch/UB cannot be used to lower bound Z because it outputs upper bounds on the unbiased sample weights (except when its estimate equals the one output by SampleSearch/LB). Likewise, ApproxCount cannot be used to lower bound Z because it is not unbiased. Finally, note that Relsat and VEC always yield a lower bound on the weighted counts (with probability one). When we compare lower bounds, the higher the lower bound, the better the solver is.

Latin Square instances

Our first set of benchmark instances come from the normalized Latin squares domain. A Latin square of order s is an $s \times s$ table filled with s numbers from $\{1, \ldots, s\}$ in such a way that each number occurs exactly once in each row and exactly once in each column.

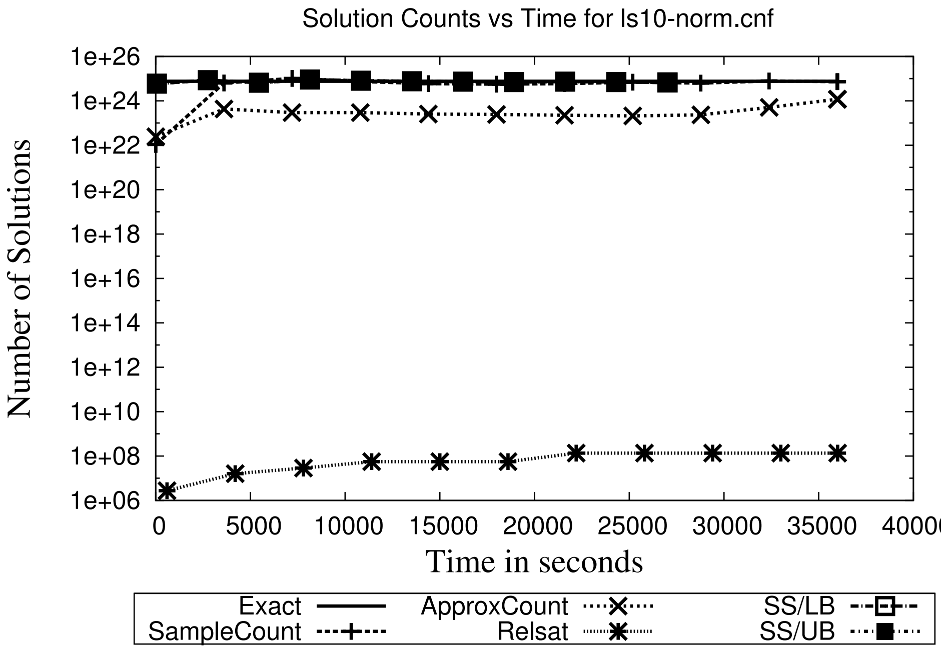
In a normalized Latin square the first row and column are fixed. The task here is to count the number of normalized Latin squares of a given order. The Latin squares were modeled as SAT formulas using the extended encoding given in [61]. The exact counts for these formulas are known up to order 11 (see [114] for details).

Table 3.2 shows the results. ApproxCount and Relsat underestimate the counts by several orders of magnitude. On the other hand, SampleSearch/UB, SampleSearch/LB and SampleCount yield very good estimates close to the true counts. The estimates of SampleCount are only slightly worse than SampleSearch/UB and SampleSearch/LB. The counts output by SampleSearch/UB and SampleSearch/LB are the same for all instances indicating that the sample mean is accurately estimated by the upper and lower approximations of the backtrack-free distribution (see the discussion on bias versus variance in Subsection 3.2.1). On Latin squares of size 12 through 15, the exact counts are not known and we compare lower bounding ability of SampleSearch/LB and SampleCount. From Table 3.2, we can see that SampleSearch/LB yields far better lower bounds than SampleCount as the problem size increases.

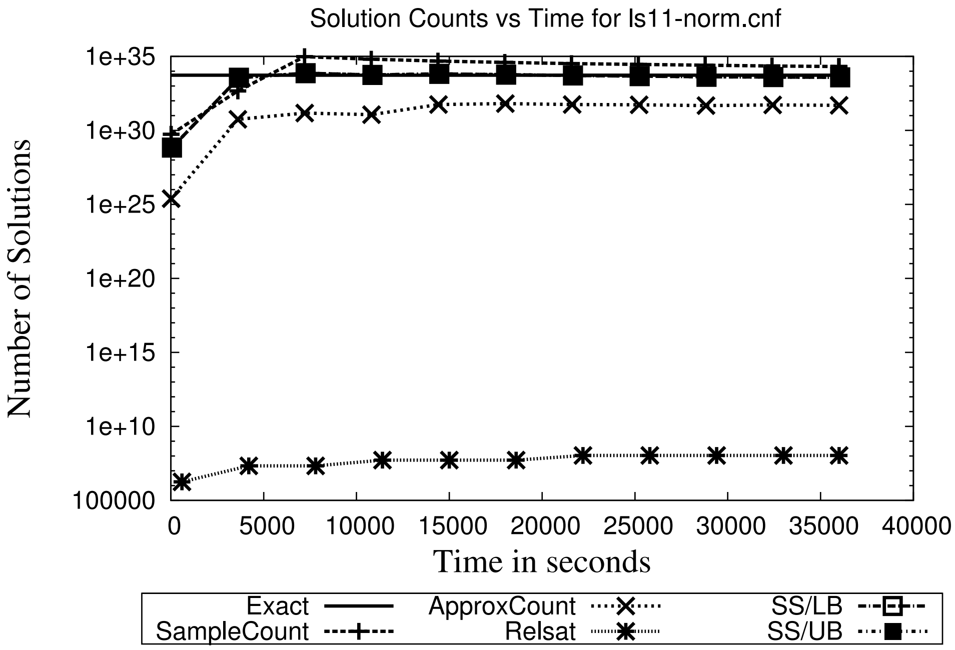
Finally, in Figure 3.5 (a) and (b), we show how the approximations change with time for the two largest instances for which the exact counts are known. Here, we can clearly see the superior convergence of SampleSearch/LB, SampleSearch/UB and SampleCount over other approaches.

Langford instances

Our second set of benchmark instances come from the Langford's problem domain. The problem is parameterized by its (integer) size denoted by s. Given a set of s numbers $\{1, 2, \ldots, s\}$, the problem is to produce a sequence of length $2s$ such that each $i \in \{1,$ $2, \ldots, s\}$ appears twice in the sequence and the two occurrences of i are exactly i apart from each other. This problem is satisfiable only if n is 0 or 3 modulo 4. We encoded



(a)



(b)

Figure 3.5: Time versus solution counts for two sample Latin square instances.

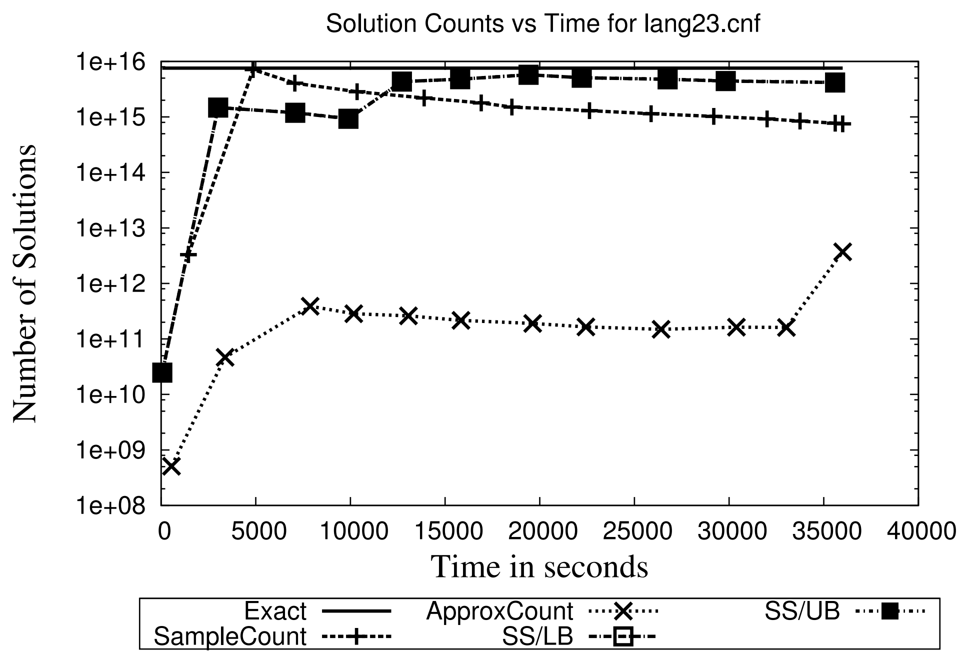
Problem	$\langle n,k,c,w\rangle$	**Exact**	**Sample Count**	**Approx Count**	**REL SAT**	**SS /LB**	**SS /UB**
lang12	$\langle 576, 2, 13584, 383\rangle$	2.16E+5	1.93E+05	2.95E+04	**2.16E+05**	2.16E+05	2.16E+05
lang16	$\langle 1024, 2, 32320, 639\rangle$	6.53E+08	5.97E+08	8.22E+06	6.28E+06	**6.51E+08**	6.99E+08
lang19	$\langle 1444, 2, 54226, 927\rangle$	5.13E+11	9.73E+10	6.87E+08	8.52E+05	**6.38E+11**	7.31E+11
lang20	$\langle 1600, 2, 63280, 1023\rangle$	5.27E+12	1.13E+11	3.99E+09	8.55E+04	2.83E+12	**3.45E+12**
lang23	$\langle 2116, 2, 96370, 1407\rangle$	7.60E+15	7.53E+14	3.70E+12	X	4.17E+15	**4.19E+15**
lang24	$\langle 2304, 2, 109536, 1535\rangle$	9.37E+16	1.17E+13	4.15E+11	X	8.74E+15	**1.40E+16**
lang27	$\langle 2916, 2, 156114, 1919\rangle$		4.38E+16	1.32E+14	X	**2.41E+19**	2.65E+19

Table 3.3: Solution counts output by SampleSearch, ApproxCount, SampleCount and Relsat after 10 hours of CPU time for Langford's problem instances. When the exact counts are not known, the entries for SampleCount and SS/LB contain the lower bounds computed by combining their respective sample weights with the Markov inequality based Average and Martingale schemes.

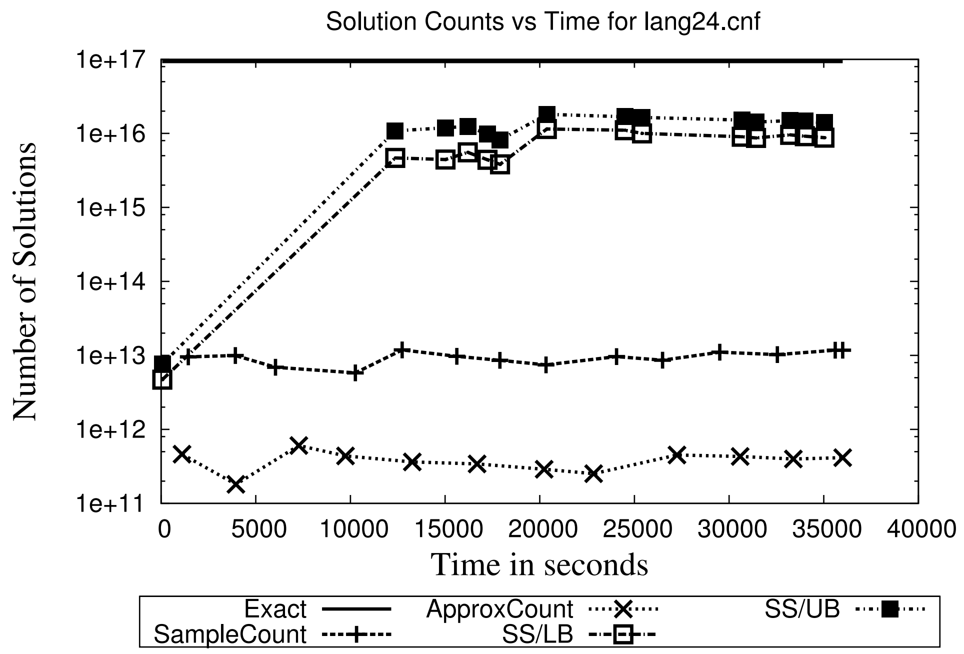
the Langford problem as a CNF instance using the channeling SAT encoding described in [129].

Table 3.3 presents the results. ApproxCount and Relsat severely under estimate the true counts except on the instance of size 12 (lang12 in Table 3.3) which Relsat solves exactly. SampleCount is inferior to SampleSearch/UB and SampleSearch/LB by several orders of magnitude. SampleSearch/UB is slightly better than SampleSearch/LB. Unlike the Latin square instances, the solution counts output by SampleSearch/LB and SampleSearch/UB are different for large problems but the difference is small. The exact counts for the Langford instance of size 27 is not known (see [96] for details). For this instance, we compare the lower bounds. We observe that the lower bound output by SampleSearch/LB are better by three orders of magnitude as compared to SampleCount.

Finally, in Figure 3.6 (a) and (b), we show how the approximations change with time for the two largest instances for which the exact counts are known. Here, we clearly see the superior anytime performance of SampleSearch/LB and SampleSearch/UB.



(a)



(b)

Figure 3.6: Time versus solution counts for two sample Langford instances.

Problem	$\langle n, k, c, w\rangle$	Exact	SampleCount	Relsat	SS/LB
9symml_gr_2pin_w6	$\langle 2604, 2, 36994, 413\rangle$		3.36E+51	3.41E+32	**3.06E+53**
9symml_gr_rcs_w6	$\langle 1554, 2, 29119, 613\rangle$		**8.49E+84**	3.36E+72	2.80E+82
alu2_gr_rcs_w8	$\langle 4080, 2, 83902, 1470\rangle$		1.21E+206	1.88E+56	**1.69E+235**
apex7_gr_2pin_w5	$\langle 1983, 2, 15358, 188\rangle$		5.83E+93	4.83E+49	**2.33E+94**
apex7_gr_rcs_w5	$\langle 1500, 2, 11695, 290\rangle$		**2.17E+139**	3.69E+46	9.64E+133
c499_gr_2pin_w6	$\langle 2070, 2, 22470, 263\rangle$		X	2.78E+47	**2.18E+55**
c499_gr_rcs_w6	$\langle 1872, 2, 18870, 462\rangle$		**2.41E+87**	7.61E+54	1.29E+84
c880_gr_rcs_w7	$\langle 4592, 2, 61745, 1024\rangle$		**1.50E+278**	1.42E+43	7.16E+255
example2_gr_2pin_w6	$\langle 3603, 2, 41023, 350\rangle$		3.93E+160	7.35E+38	**7.33E+160**
example2_gr_rcs_w6	$\langle 2664, 2, 27684, 476\rangle$		**4.17E+265**	1.13E+73	6.85E+250
term1_gr_2pin_w4	$\langle 746, 2, 3964, 31\rangle$		X	2.13E+35	**6.90E+39**
term1_gr_rcs_w4	$\langle 808, 2, 3290, 57\rangle$		X	1.17E+49	**7.44E+55**
too_large_gr_rcs_w7	$\langle 3633, 2, 50373, 1069\rangle$		X	1.46E+73	**1.05E+182**
too_large_gr_rcs_w8	$\langle 4152, 2, 57495, 1330\rangle$		X	1.02E+64	**5.66E+246**
vda_gr_rcs_w9	$\langle 6498, 2, 130997, 2402\rangle$		X	2.23E+92	**5.08E+300**

Table 3.4: Lower Bounds on Solution counts output by SampleSearch/LB, SampleCount and Relsat after 10 hours of CPU time for FPGA routing instances. The entries for SampleCount and SS/LB contain the lower bounds computed by combining their respective sample weights with the Markov inequality based Average and Martingale schemes.

FPGA Routing instances

The FPGA routing instances are constructed by reducing FPGA (Field Programmable Gate Array) detailed routing problems into a satisfiability problem. The instances were generated by Gi-Joon Nam and were used in the SAT 2002 competition [123]. Table 3.4 presents the results. The exact solution counts are not known and therefore we compare lower bounds. Relsat usually underestimates the solution counts by several orders of magnitude. SampleSearch/LB yields higher lower bounds than SampleCount and Relsat on 10 out of the 15 instances. On the remaining five instances SampleCount yields higher lower bounds than SampleSearch/LB.

Linkage networks

The Linkage networks are generated by converting biological linkage analysis data into a Bayesian or Markov network.

Problem	$\langle n,k,e,w\rangle$	**Exact**	**VEC**	**EDBP**	**SS/LB**	**SS/UB**
BN_69	$\langle 777,7,78,47\rangle$	5.28E-054	1.93E-61	2.39E-57	**3.00E-55**	**3.00E-55**
BN_70	$\langle 2315,5,159,87\rangle$	2.00E-71	7.99E-82	6.00E-79	**1.21E-73**	**1.21E-73**
BN_71	$\langle 1740,6,202,70\rangle$	5.12E-111	7.05E-115	1.01E-114	**1.28E-111**	**1.28E-111**
BN_72	$\langle 2155,6,252,86\rangle$	4.21E-150	1.32E-153	9.21E-155	**4.73E-150**	**4.73E-150**
BN_73	$\langle 2140,5,216,101\rangle$	2.26E-113	6.00E-127	2.24E-118	**2.00E-115**	**2.00E-115**
BN_74	$\langle 749,6,66,45\rangle$	3.75E-45	3.30E-48	5.84E-48	**2.13E-46**	**2.13E-46**
BN_75	$\langle 1820,5,155,92\rangle$	5.88E-91	5.83E-97	3.10E-96	**2.19E-91**	**2.19E-91**
BN_76	$\langle 2155,7,169,64\rangle$	4.93E-110	1.00E-126	3.86E-114	**1.95E-111**	**1.95E-111**

Table 3.5: Probability of evidence computed by VEC, EDBP and SampleSearch after 3 hours of CPU time for Linkage instances from the UAI 2006 evaluation.

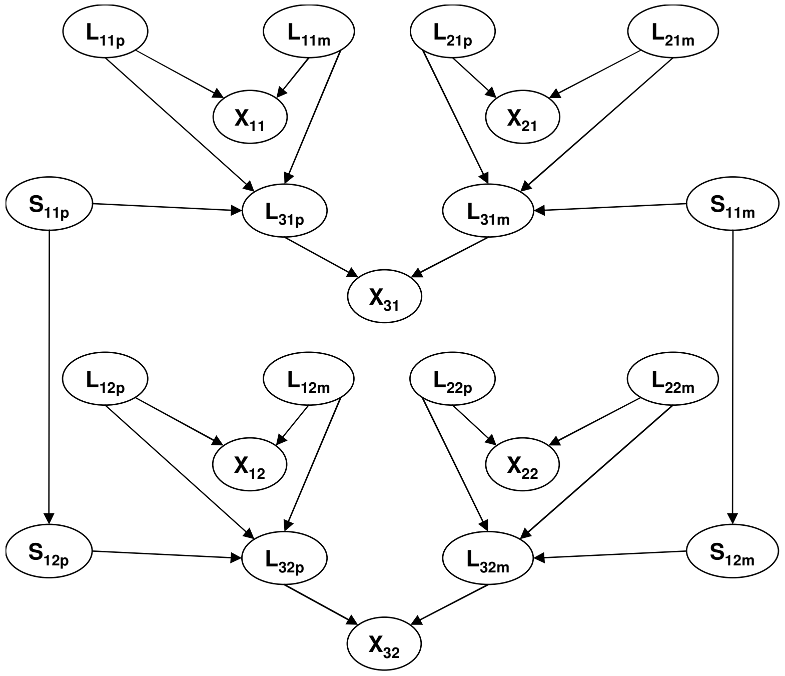


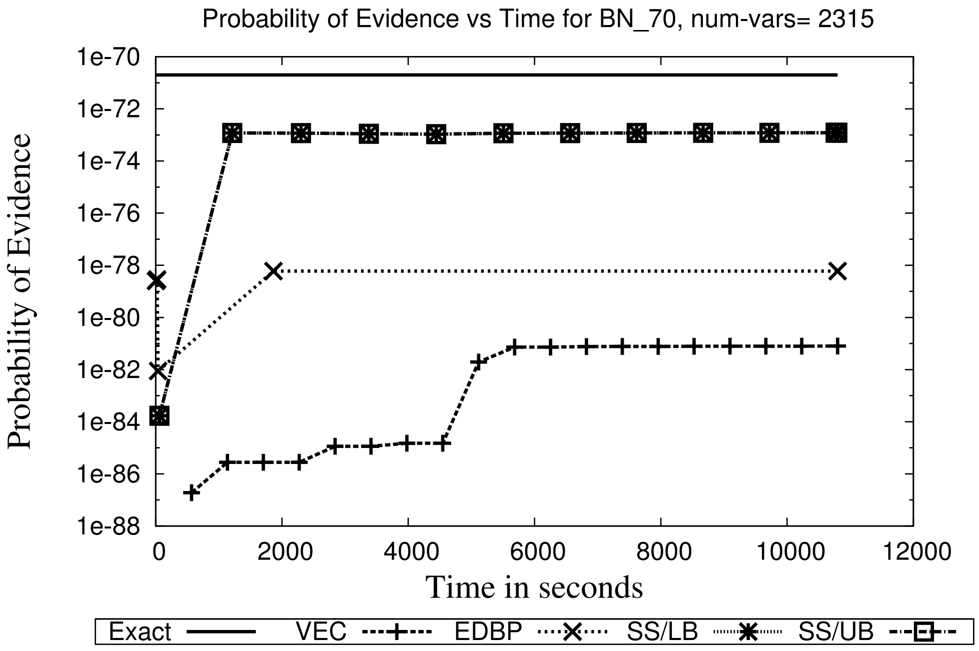
Figure 3.7: A fragment of a Bayesian network used in genetic linkage analysis.

Linkage analysis is a statistical method for mapping genes onto a chromosome [102]. This is very useful in practice for identifying disease genes. The input is an ordered list of loci $L_1, \ldots, L_{k+1}$ with allele frequencies at each locus and a pedigree with some individuals typed at some loci. The goal of linkage analysis is to evaluate the likelihood of a candidate vector $[\theta_1, \ldots, \theta_k]$ of recombination fractions for the input pedigree and locus order. The component θ_i is the candidate recombination fraction between the loci L_i and L_{i+1}.

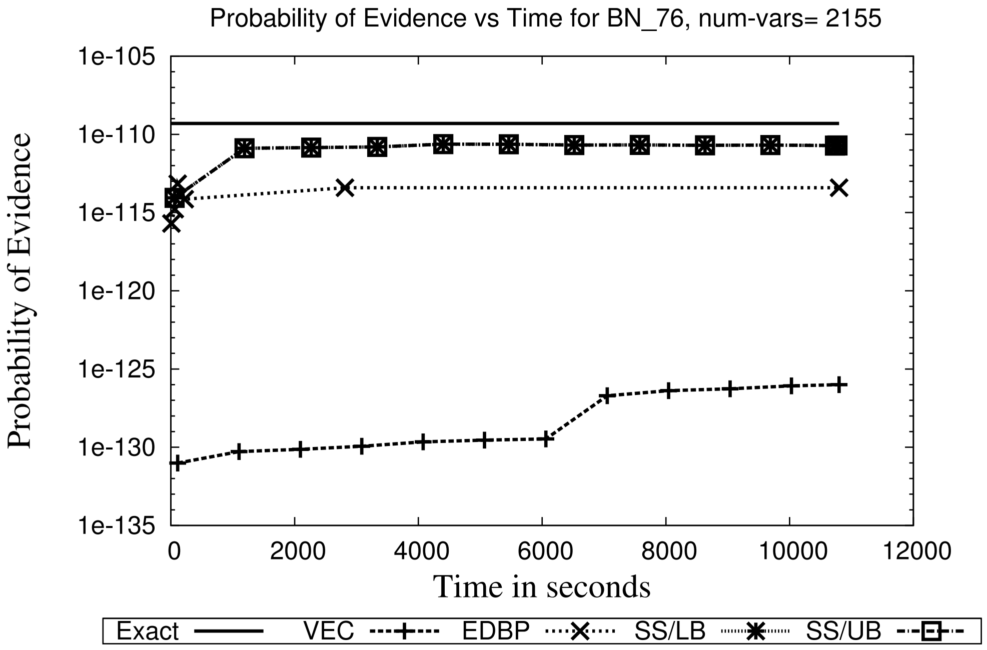
The pedigree data can be represented as a Bayesian network with three types of random variables: genetic loci variables which represent the genotypes of the individuals in the pedigree (two genetic loci variables per individual per locus, one for the paternal allele and one for the maternal allele), phenotype variables, and selector variables which are auxiliary variables used to represent the gene flow in the pedigree. Figure 3.7 represents a fragment of a network that describes parents-child interactions in a simple 2-loci analysis. The genetic loci variables of individual i at locus j are denoted by $L_{i,jp}$ and $L_{i,jm}$. Variables $X_{i,j}$, $S_{i,jp}$ and $S_{i,jm}$ denote the phenotype variable, the paternal selector variable and the maternal selector variable of individual i at locus j, respectively. The conditional probability tables that correspond to the selector variables are parameterized by the recombination ratio θ. The remaining tables contain only deterministic information. It can be shown that given the pedigree data, computing the likelihood of the recombination fractions is equivalent to computing the probability of evidence on the Bayesian network that model the problem (for more details consult [46]).

We first evaluate the solvers on Linkage (Bayesian) networks used in the UAI 2006 evaluation [8]. Table 3.5 contains the results. We see that SampleSearch/UB and SampleSearch/LB are very accurate usually yielding a few orders of magnitude improvement over VEC and EDBP. Because the estimates output by SampleSearch/UB and SampleSearch/LB are the same on all instances, they yield an exact approximation of the sample mean. Figures 3.8 contains time versus accuracy plots for two linkage instances. We see superior anytime performance of both SampleSearch schemes as compared with VEC and EDBP.

In Table 3.6, we present the results on Linkage instances encoded as Markov networks. These instances were used in the UAI 2008 evaluation [24]. We were able to compute exact solutions of 11 instances out of 20 using ACE [13] (which is not an anytime scheme). We see that VEC (as an anytime scheme) and EDBP exactly solve 10 and 6 instances respectively as indicated by a $*$ in Table 3.6. The instances for which exact solutions are known generally have smaller treewidth (< 30). On these instances, SampleSearch/LB and



(a)



(b)

Figure 3.8: Convergence of probability of evidence as a function of time for two sample Linkage instances

Problem	$\langle n,k,e,w\rangle$	**Exact**	**SS/LB**	**SS/UB**	**VEC**	**EDBP**
pedigree18	$\langle 1184,1,0,26\rangle$	7.18E-79	7.39E-79	7.39E-79	**7.18E-79***	**7.18E-79***
pedigree1	$\langle 334,2,0,20\rangle$	7.81E-15	7.81E-15	7.81E-15	**7.81E-15**	**7.81E-15***
pedigree20	$\langle 437,2,0,25\rangle$	2.34E-30	2.31E-30	2.31E-30	**2.34E-30***	6.19E-31
pedigree23	$\langle 402,1,0,26\rangle$	2.78E-39	2.76E-39	2.76E-39	**2.78E-39***	1.52E-39
pedigree25	$\langle 1289,1,0,38\rangle$	1.69E-116	**1.69E-116**	1.69E-116	**1.69E-116***	**1.69E-116***
pedigree30	$\langle 1289,1,0,27\rangle$	1.84E-84	1.90E-84	1.90E-84	**1.85E-84***	**1.85E-84***
pedigree37	$\langle 1032,1,0,25\rangle$	2.63E-117	1.18E-117	1.18E-117	**2.63E-117***	5.69E-124
pedigree38	$\langle 724,1,0,18\rangle$	5.64E-55	3.80E-55	3.80E-55	**5.65E-55***	8.41E-56
pedigree39	$\langle 1272,1,0,29\rangle$	6.32E-103	6.29E-103	6.29E-103	**6.32E-103***	**6.32E-103***
pedigree42	$\langle 448,2,0,23\rangle$	1.73E-31	1.73E-31	1.73E-31	**1.73E-31***	8.91E-32
pedigree19	$\langle 793,2,0,23\rangle$		**6.76E-60**	**6.76E-60**	1.597E-60	3.35E-60
pedigree31	$\langle 1183,2,0,45\rangle$		**2.08E-70**	**2.08E-70**	1.67E-76	1.34E-70
pedigree34	$\langle 1160,1,0,59\rangle$		**3.84E-65**	**3.84E-65**	2.58E-76	4.30E-65
pedigree13	$\langle 1077,1,0,51\rangle$		**7.03E-32**	**7.03E-32**	2.17E-37	6.53E-32
pedigree40	$\langle 1030,2,0,49\rangle$		**1.25E-88**	**1.25E-88**	2.45E-91	7.02E-17
pedigree41	$\langle 1062,2,0,52\rangle$		**4.36E-77**	**4.36E-77**	4.33E-81	1.09E-10
pedigree44	$\langle 811,1,0,29\rangle$		**3.39E-64**	**3.39E-64**	2.23E-64	7.69E-66
pedigree51	$\langle 1152,1,0,51\rangle$		**2.47E-74**	**2.47E-74**	5.56E-85	6.16E-76
pedigree7	$\langle 1068,1,0,56\rangle$		**1.33E-65**	**1.33E-65**	1.66E-72	2.93E-66
pedigree9	$\langle 1118,2,0,41\rangle$		**2.93E-79**	**2.93E-79**	8.00E-82	3.13E-89

Table 3.6: Partition function computed by VEC, EDBP and SampleSearch after 3 hours of CPU time for Linkage instances from the UAI 2008 evaluation.

SampleSearch/UB deviate only slightly from the true value of the partition function and on four instances for which EDBP does not output the exact answer, the estimates output by SampleSearch/LB and SampleSearch/UB are better than EDBP. On instances with large treewidth (> 30) and for which exact results are not known, we compare the lower bounding capability of VEC and SampleSearch. The lower bounds output by SampleSearch are superior to VEC on all the 9 instances. EDBP does not yield guaranteed lower bounds but we see that in 6 out of 9 instances, it yields an approximation which is smaller than the lower bound.

Relational Instances

The relational instances are generated by grounding the relational Bayesian networks using the primula tool (see Chavira et al. [14] for more information). We experimented with ten Friends and Smoker networks and six mastermind networks from this domain which have

Problem	$\langle n, k, e, w\rangle$	Exact	SS/LB	SS/UB	VEC	EDBP
fs-04	$\langle 262, 2, 226, 12\rangle$	1.52E-05	8.11E-06	8.11E-06	**1.53E-05***	X
fs-07	$\langle 1225, 2, 1120, 35\rangle$	9.80E-17	2.23E-16	2.23E-16	**1.78E-15***	X
fs-10	$\langle 3385, 2, 3175, 71\rangle$	7.88E-31	**2.49E-32**	2.49E-32	X	X
fs-13	$\langle 7228, 2, 6877, 119\rangle$	1.33E-51	**3.26E-55**	3.26E-55	X	X
fs-16	$\langle 13240, 2, 12712, 171\rangle$	8.63E-78	**6.04E-79**	6.04E-79	X	X
fs-19	$\langle 21907, 2, 21166, 243\rangle$	2.12E-109	**1.62E-114**	1.62E-114	X	X
fs-22	$\langle 33715, 2, 32725, 335\rangle$	2.00E-146	**4.88E-147**	4.88E-147	X	X
fs-25	$\langle 49150, 2, 47875, 431\rangle$	7.18E-189	**2.67E-189**	2.67E-189	X	X
fs-28	$\langle 68698, 2, 67102, 527\rangle$	9.82E-237	**4.53E-237**	4.53E-237	X	X
fs-29	$\langle 76212, 2, 74501, 559\rangle$	6.81E-254	**9.44E-255**	9.44E-255	X	X
mastermind_03_08_03	$\langle 1220, 2, 48, 20\rangle$	9.79E-8	9.87E-08	9.87E-08	**9.79E-08***	X
mastermind_03_08_04	$\langle 2288, 2, 64, 30\rangle$	8.77E-09	8.19E-09	8.19E-09	**8.77E-09***	X
mastermind_03_08_05	$\langle 3692, 2, 80, 42\rangle$	8.89E-11	7.27E-11	7.27E-11	**8.90E-11***	X
mastermind_04_08_03	$\langle 1418, 2, 48, 22\rangle$	8.39E-08	8.37E-08	8.37E-08	**8.39E-08***	X
mastermind_04_08_04	$\langle 2616, 2, 64, 33\rangle$	2.20E-08	**1.84E-08**	1.84E-08	1.21E-08	X
mastermind_05_08_03	$\langle 1616, 2, 48, 28\rangle$	5.29E-07	4.78E-07	4.78E-07	**5.30E-07***	X
mastermind_06_08_03	$\langle 1814, 2, 48, 31\rangle$	1.79E-08	1.12E-08	1.12E-08	**1.80E-08***	X
mastermind_10_08_03	$\langle 2606, 2, 48, 56\rangle$	1.92E-07	**5.01E-07**	5.01E-07	7.79E-08	X

Table 3.7: Probability of evidence computed by VEC, EDBP and SampleSearch after 3 hours of CPU time for relational instances.

between 262 to 76,212 variables. Table 3.7 summarizes the results. These networks are quite large and EDBP does not output any answer in 3 hours of CPU time. VEC solves 2 friends and smokers networks exactly while on the remaining instances, it fails to output any answer. The estimates computed by SampleSearch/LB and SampleSearch/UB on the other hand are very close to the exact value of probability of evidence. This shows that SampleSearch LB/UB are more scalable than VEC and EDBP. VEC solves exactly six out of the eight mastermind instances while on the remaining two instances VEC is worse than SampleSearch/UB and SampleSearch/LB.

3.3.4 Results for the Posterior Marginal Tasks

We experimented with the following four benchmark domains: (a) The linkage instances (b) The relational instances and (c) The grid instances and (d) The logistics planning instances. For this task, we measure the accuracy of the solvers using average Hellinger distance [77]. Given a mixed network with n variables, let $P(X_i)$ and $A(X_i)$ denote the exact and approximate marginals for a variable X_i, then the Hellinger distance is defined as:

$$Hellinger\ distance = \frac{\sum_{i=1}^{n} \frac{1}{2} \sum_{x_i \in \mathbf{D}_i} (\sqrt{P(x_i)} - \sqrt{A(x_i)})^2}{n} \tag{3.34}$$

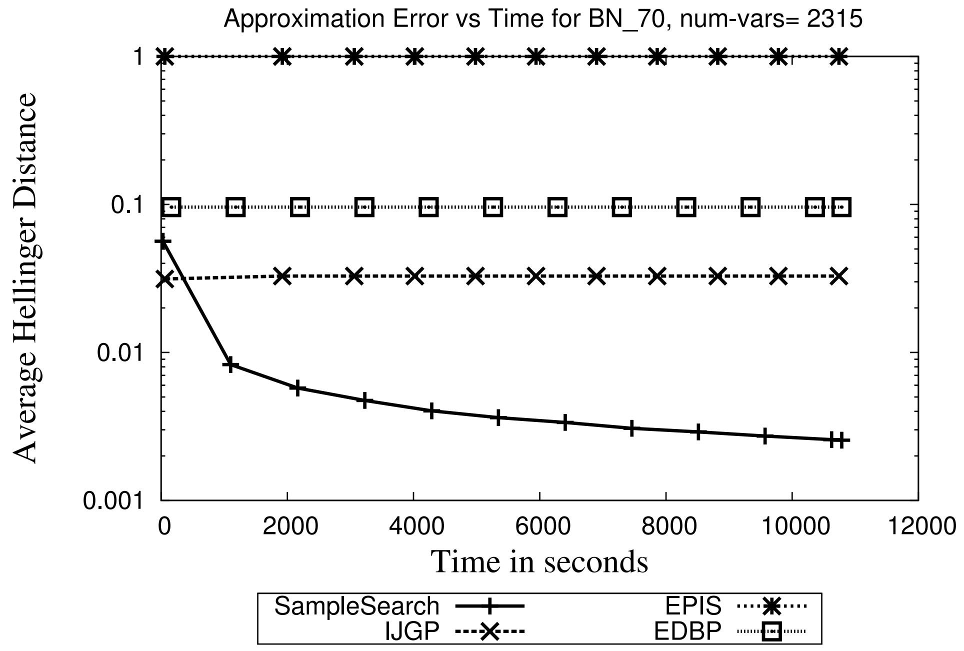
We chose Hellinger distance because as pointed out in [77], it is superior to other choices such as the Kullback-Leibler (KL) distance, the mean squared error and the relative error when zero probabilities are present in the graphical model. We do not use KL distance because it lies between 0 and ∞ and in practice when the exact marginals are close to 0, precision errors may cause the approximation to become zero, yielding infinite KL distance. Hellinger distance is between 0 and 1 and lower bounds the KL distance.

We did compute the error using other distance measures like the mean squared error and the relative error. All error measures show similar trends, with Hellinger distance being the most discriminative.

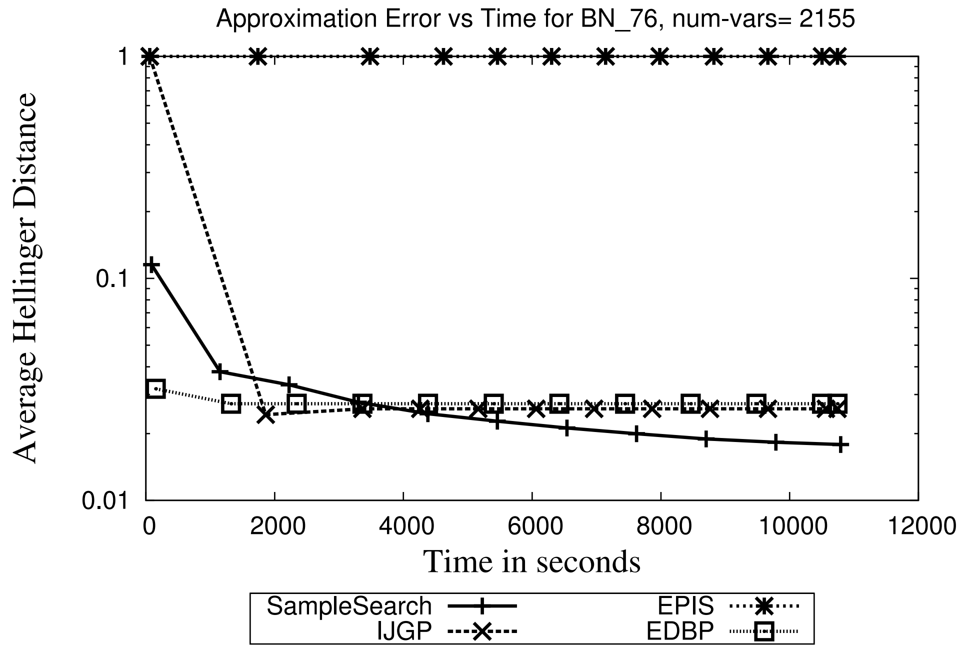
Finally, note that for the marginal task, SampleSearch/LB and SampleSearch/UB output the same marginals for all benchmarks that we experimented with and therefore we do not distinguish between them. This however, implies that our lower and upper approximations of the backtrack free probability are indeed strong. *Therefore, for the rest of this subsection, we will refer to SampleSearch/LB and SampleSearch/UB as SampleSearch.*

Linkage instances

We report the average Hellinger distance between exact and approximate marginals in Table 3.8. EPIS which is also an importance sampling technique like SampleSearch does not generate even a single consistent sample in 3 hours of CPU time and therefore its average Hellinger distance is 1. SampleSearch is more accurate than IJGP which in turn is more accurate than EDBP on 7 out of 8 instances. We can clearly see the relationship between treewidth and the performance of propagation based and sampling based techniques.



(a)



(b)

Figure 3.9: Time versus Hellinger distance for two sample Linkage instances.

Problem	$\langle n, K, e, w \rangle$	**SampleSearch**	**IJGP**	**EPIS**	**EDBP**
BN_69	$\langle 777, 7, 78, 47 \rangle$	**9.4E-04**	3.2E-02	1	8.0E-02
BN_70	$\langle 2315, 5, 159, 87 \rangle$	**2.6E-03**	3.3E-02	1	9.6E-02
BN_71	$\langle 1740, 6, 202, 70 \rangle$	**5.6E-03**	1.9E-02	1	2.5E-02
BN_72	$\langle 2155, 6, 252, 86 \rangle$	**3.6E-03**	7.2E-03	1	1.3E-02
BN_73	$\langle 2140, 5, 216, 101 \rangle$	**2.1E-02**	2.8E-02	1	6.1E-02
BN_74	$\langle 749, 6, 66, 45 \rangle$	6.9E-04	**4.3E-06**	1	4.3E-02
BN_75	$\langle 1820, 5, 155, 92 \rangle$	**8.0E-03**	6.2E-02	1	9.3E-02
BN_76	$\langle 2155, 7, 169, 64 \rangle$	**1.8E-02**	2.6E-02	1	2.7E-02

Table 3.8: Table showing Hellinger distance of SampleSearch, IJGP, EPIS and EDBP for Linkage instances from the UAI 2006 evaluation after 3 hours of CPU time.

Problem	$\langle n, K, e, w \rangle$	**SampleSearch**	**IJGP**	**EPIS**	**EDBP**
fs-04	$\langle 262, 2, 226, 12 \rangle$	5.4E-05	**4.6E-08**	1	6.4E-02
fs-07	$\langle 1225, 2, 1120, 35 \rangle$	**1.4E-02**	1.6E-02	1	3.0E-02
fs-10	$\langle 3385, 2, 3175, 71 \rangle$	1.2E-02	**6.3E-03**	1	2.7E-02
fs-13	$\langle 7228, 2, 6877, 119 \rangle$	2.0E-02	**6.5E-03**	1	2.3E-02
fs-16	$\langle 13240, 2, 12712, 171 \rangle$	**1.2E-03**	6.8E-03	1	1.7E-02
fs-19	$\langle 21907, 2, 21166, 243 \rangle$	**3.1E-03**	8.8E-03	1	1
fs-22	$\langle 33715, 2, 32725, 335 \rangle$	**2.5E-03**	8.6E-03	1	1
fs-25	$\langle 49150, 2, 47875, 431 \rangle$	**2.5E-03**	8.4E-03	1	1
fs-28	$\langle 68698, 2, 67102, 527 \rangle$	**1.3E-03**	7.4E-03	1	1
fs-29	$\langle 76212, 2, 74501, 559 \rangle$	**1.9E-03**	7.0E-03	1	1
mastermind_03_08_03	$\langle 1220, 2, 48, 20 \rangle$	**1.1E-03**	3.8E-02	1	3.8E-01
mastermind_03_08_04	$\langle 2288, 2, 64, 30 \rangle$	**1.1E-02**	4.4E-02	1	1
mastermind_03_08_05	$\langle 3692, 2, 80, 42 \rangle$	4.0E-02	**3.2E-02**	1	1
mastermind_04_08_04	$\langle 2616, 2, 64, 33 \rangle$	**3.1E-02**	3.5E-02	1	1
mastermind_05_08_03	$\langle 1616, 2, 48, 28 \rangle$	**1.0E-02**	3.6E-02	1	4.0E-02
mastermind_06_08_03	$\langle 1814, 2, 48, 31 \rangle$	**4.7E-03**	3.3E-02	5.6E-01	3.2E-01
mastermind_10_08_03	$\langle 2606, 2, 48, 56 \rangle$	**3.9E-02**	5.3E-02	1	8.3E-02

Table 3.9: Table showing Hellinger distance of SampleSearch, IJGP, EPIS and EDBP for Relational instances after 3 hours of CPU time.

When the treewidth is small (on BN_74.uai), a propagation based scheme like IJGP is more accurate than SampleSearch but as the treewidth increases, there is one to two orders of magnitude difference in the Hellinger distance. Finally in Figure 3.9, we demonstrate the superior anytime performance of SampleSearch compared with other solvers.

Relational Instances

As described before, the relational instances are generated by grounding the relational Bayesian networks using the primula tool [14]. We experimented again with the 10 Friends and Smoker networks and 6 mastermind networks from this domain. Table 3.9 shows the Hellinger distance after 3 hours of CPU time for each solver.

On the small friends and smoker networks, fs-04 to fs-13, IJGP performs better than SampleSearch. However, on large networks which have between 13240 and 76212 variables, and treewidth between 12712 to 74501, SampleSearch performs better than IJGP. EPIS is not able to generate a single consistent sample in 3 hours of CPU time as indicated by Hellinger distance of 1. EDBP is slightly worse than IJGP and runs out of memory on large instances, as indicated by a Hellinger distance of 1.

On the mastermind networks, SampleSearch is the best performing scheme followed by IJGP. EPIS fails to output even a single consistent sample in 3 hours on 6 out of the 7 instances. EDBP is slightly worse than IJGP on 5 out of the 6 instances. Figures 3.10 and 3.11 show the anytime performance of the solvers demonstrating clear superiority of SampleSearch.

Grid Networks

The Grid Bayesian networks benchmarks are available from the authors of Cachet [119]. A grid Bayesian network is a $s \times s$ grid, where there are two directed edges from a node to its neighbors right and down. The upper-left node is a source, and the bottom-right node is a sink. The sink node is the evidence node. The deterministic ratio p is a parameter specifying the fraction of nodes that are deterministic (functional in this case), that is, whose values are determined given the values of their parents. The grid instances are designated as $p - s$.

Problem	$\langle n, K, e, w \rangle$	**SampleSearch**	**IJGP**	**EPIS**	**EDBP**
50-12-5	$\langle 144, 2, 1, 16 \rangle$	4.3E-04	**3.2E-07**	2.6E-04	2.5E-02
50-14-5	$\langle 196, 2, 1, 20 \rangle$	4.9E-04	1.8E-02	**1.2E-04**	4.0E-02
50-15-5	$\langle 225, 2, 1, 23 \rangle$	4.9E-04	1.0E-02	**2.3E-04**	6.1E-02
50-17-5	$\langle 289, 2, 1, 25 \rangle$	8.0E-04	2.1E-02	**2.0E-04**	3.6E-03
50-18-5	$\langle 324, 2, 1, 27 \rangle$	9.3E-04	1.9E-02	**3.0E-04**	2.1E-03
50-19-5	$\langle 361, 2, 1, 28 \rangle$	1.1E-03	3.4E-02	4.0E-04	**3.4E-04**
75-16-5	$\langle 256, 2, 1, 24 \rangle$	6.5E-04	**2.5E-07**	1.7E-04	7.8E-02
75-17-5	$\langle 289, 2, 1, 25 \rangle$	1.4E-03	**2.6E-07**	2.7E-04	1.2E-03
75-18-5	$\langle 324, 2, 1, 27 \rangle$	1.2E-03	3.9E-02	**2.0E-04**	5.0E-03
75-19-5	$\langle 361, 2, 1, 28 \rangle$	9.0E-03	4.3E-02	2.5E-04	**6.7E-05**
75-20-5	$\langle 400, 2, 1, 30 \rangle$	6.2E-04	**3.1E-07**	1.9E-04	1.7E-02
75-21-5	$\langle 441, 2, 1, 32 \rangle$	1.9E-03	**2.9E-07**	2.8E-04	1.5E-02
75-22-5	$\langle 484, 2, 1, 35 \rangle$	3.2E-03	2.3E-02	**2.6E-04**	2.0E-02
75-23-5	$\langle 529, 2, 1, 35 \rangle$	2.0E-03	4.8E-02	**2.3E-04**	2.4E-02
75-24-5	$\langle 576, 2, 1, 38 \rangle$	8.4E-03	4.3E-02	**2.6E-04**	3.5E-02
75-26-5	$\langle 676, 2, 1, 44 \rangle$	2.4E-02	5.1E-02	**3.5E-04**	5.1E-02
90-20-5	$\langle 400, 2, 1, 30 \rangle$	1.6E-03	**2.7E-07**	2.5E-04	3.7E-02
90-22-5	$\langle 484, 2, 1, 35 \rangle$	4.6E-04	**2.8E-07**	1.5E-04	5.1E-02
90-23-5	$\langle 529, 2, 1, 35 \rangle$	2.8E-04	**3.2E-07**	3.9E-04	1.9E-02
90-24-5	$\langle 576, 2, 1, 38 \rangle$	5.0E-04	**3.9E-07**	3.5E-04	2.8E-02
90-25-5	$\langle 625, 2, 1, 39 \rangle$	**2.6E-07**	**2.7E-07**	3.4E-04	4.6E-02
90-26-5	$\langle 676, 2, 1, 44 \rangle$	1.0E-03	**1.9E-06**	2.3E-04	3.9E-02
90-34-5	$\langle 1156, 2, 1, 65 \rangle$	8.6E-04	**1.8E-07**	3.9E-04	4.1E-02
90-38-5	$\langle 1444, 2, 1, 69 \rangle$	1.6E-02	**4.3E-07**	1.7E-03	1.6E-01

Table 3.10: Table showing Hellinger distance of SampleSearch, IJGP, EPIS and EDBP for Grid networks after 3 hours of CPU time.

For example, the instance $50 - 18$ indicates a grid of size 18 in which 50% of the nodes are deterministic or functional. Table 3.10 shows the Hellinger distance after 3 hours of CPU time for each solver. Time versus approximation error plots are shown for six sample instances in Figures 3.12 through 3.14.

On grids with deterministic ratio of 50%, EPIS is the best performing scheme on all but one instance. SampleSearch is second best while EDBP is slightly better than IJGP. There is two orders of magnitude difference between SampleSearch and EDBP/IJGP while there is one order of magnitude difference between EPIS and SampleSearch.

On grids with deterministic ratio of 75%, IJGP is best on four out of the six smaller grids (up to size 21). EPIS dominates on the larger grids (size 22-26). Again, we see that there is an order of magnitude difference between EPIS and SampleSearch and it gets larger as the size of the grid increases.

On grids with deterministic ratio of 90%, IJGP is again the best performing scheme. EPIS is slightly better than SampleSearch while EDBP is the least accurate scheme.

Note that both EPIS and SampleSearch are importance sampling schemes. The poor performance of SampleSearch as compared to EPIS is due to the overhead of solving a satisfiability problem via search to generate a sample. Consequently, SampleSearch generates fewer samples than EPIS which uses relevancy-based reasoning [87] to determine the inconsistencies as a pre-processing step[5]. This highlights one of the advantages of using inference or reasoning to determine the inconsistencies before sampling rather than combining search and sampling. Inference schemes are however not scalable because of their extensive time and memory requirements when the treewidth of the constraint portion is large.

[5]Unfortunately, the EPIS program does not output the number of consistent samples that were used in computing the marginals and therefore we cannot experimentally compare the actual difference in sample size between SampleSearch and EPIS.

Problem	$\langle n, K, e, w \rangle$	SampleSearch	IJGP	EPIS	EDBP
log-1.uai	$\langle 4724, 2, 3785, 22 \rangle$	**2.2E-05**	0*	1	1
log-2.uai	$\langle 26114, 2, 24777, 51 \rangle$	**8.6E-04**	9.8E-03	1	1
log-3.uai	$\langle 30900, 2, 29487, 56 \rangle$	**1.2E-04**	7.5E-03	1	1
log-4.uai	$\langle 23266, 2, 20963, 52 \rangle$	**2.3E-02**	1.8E-01	1	1
log-5.uai	$\langle 32235, 2, 29534, 51 \rangle$	**8.6E-03**	1.2E-02	1	1

Table 3.11: Table showing Hellinger distance of SampleSearch, IJGP, EPIS and EDBP for Logistics planning instances after 3 hours of CPU time.

Logistics Planning instances

Our last domain is that of logistics planning. Given prior probabilities on actions and facts, the task is to compute marginal distribution of each variable. Goals and initial conditions are observed true. Bayesian networks are generated from the plan graphs, where additional nodes (all observed false) are added to represent mutex, action-effect and preconditions of actions. These benchmarks are available from the authors of Cachet [119].

Table 3.11 summarizes the results. Both EPIS and EDBP fail to output any answer after 3 hours of CPU time. IJGP solves the log-1 instance exactly as indicated by a * in Table 3.11 while on the remaining instances, SampleSearch is more accurate than IJGP. Finally, in Figure 3.15, we demonstrate the superior anytime performance of SampleSearch as compared with the other schemes.

3.3.5 Summary of Experimental Evaluation

To summarize, for satisfiability model counting, we compared SampleSearch with three other approximate solution counters available in literature: ApproxCount [130], SampleCount [62] and Relsat [115] on three benchmarks: (a) Latin Square instances (b) Langford instances and (c) FPGA-routing instances. We found that on most instances, SampleSearch yields solution counts which are closer to the true counts by a few orders of magnitude than those output by SampleCount and by several orders of magnitude than those output by ApproxCount and Relsat.

For the problem of computing the probability of evidence in a Bayesian network and the partition function in a Markov network, we compared SampleSearch with two other schemes available in literature: Variable Elimination and Conditioning (VEC) [28], and an advanced generalized belief propagation scheme called Edge Deletion Belief Propagation (EDBP) [18] on two benchmarks: (a) linkage analysis benchmarks and (b) relational Bayesian network benchmarks. We found that on most instances the estimates output by SampleSearch were closer to the exact answer than those output by EDBP. VEC solved some instances exactly, while on the remaining instances, VEC was substantially inferior.

For the posterior marginal tasks, we experimented with linkage analysis benchmarks, partially deterministic grid benchmarks, relational benchmarks and logistics planning benchmarks. Here, we compared the accuracy of SampleSearch in terms of a distance measure called the Hellinger distance with three other schemes: two generalized belief propagation schemes of Iterative Join Graph Propagation [33] and Edge Deletion Belief Propagation [18] and an adaptive importance sampling scheme called Evidence Pre-propagated Importance Sampling (EPIS) [133]. Again, we found that except on the grid instances, SampleSearch consistently yields estimates having smaller Hellinger distance than the competition. On the grid instances, IJGP and EPIS are the best schemes in terms of the Hellinger distance.

3.4 Conclusion

In this chapter we presented the SampleSearch scheme for performing effective importance sampling based inference in mixed probabilistic and deterministic graphical models. It is well known that on such graphical models, importance sampling performs quite poorly because of the rejection problem. SampleSearch remedies the rejection problem by interleaving random sampling with systematic backtracking. Specifically, when sampling variables one by one via logic sampling [106], instead of rejecting a sample when its inconsistency is detected, SampleSearch backtracks to the previous variable, modifies the proposal distribution to reflect the inconsistency and continues this process until a consistent sample is found.

We showed that SampleSearch can be viewed as a systematic search technique whose value selection is stochastically guided by sampling from a distribution. This view enables us to integrate any systematic SAT/CSP solver within SampleSearch (with minor modifications). Indeed, in our experiments, we used an advanced SAT solver called minisat [125]. Thus, advances in the systematic search community whose primary focus is solving "yes/no" type NP-complete problems can be leveraged through SampleSearch for approximating much harder #P-complete problems in Bayesian inference.

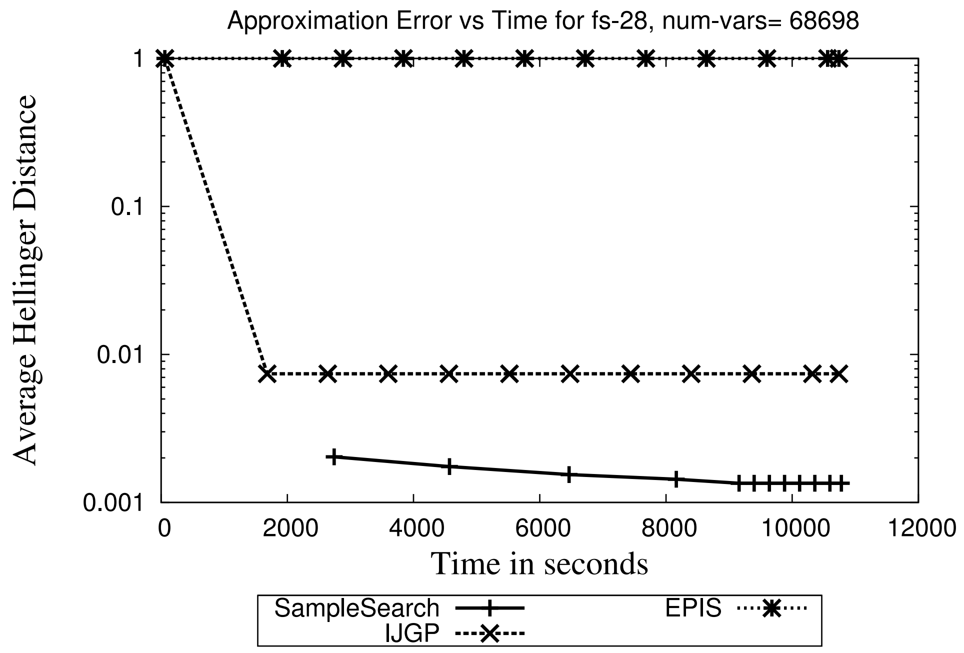
We showed that SampleSearch samples from the backtrack-free distribution, which is basically a modification of the proposal distribution from which all inconsistent partial assignments are removed. When the backtrack-free probability is too complex to compute, we proposed two approximations, which bound the backtrack-free probability from above and below and yield asymptotically unbiased estimates of the weighted counts and marginals. We demonstrated experimentally that these approximations were very accurate on most benchmarks.

We performed an extensive empirical evaluation on several benchmark graphical models and our results clearly demonstrate that SampleSearch is consistently superior to other state-of-the-art schemes on domains having a substantial amount of determinism.

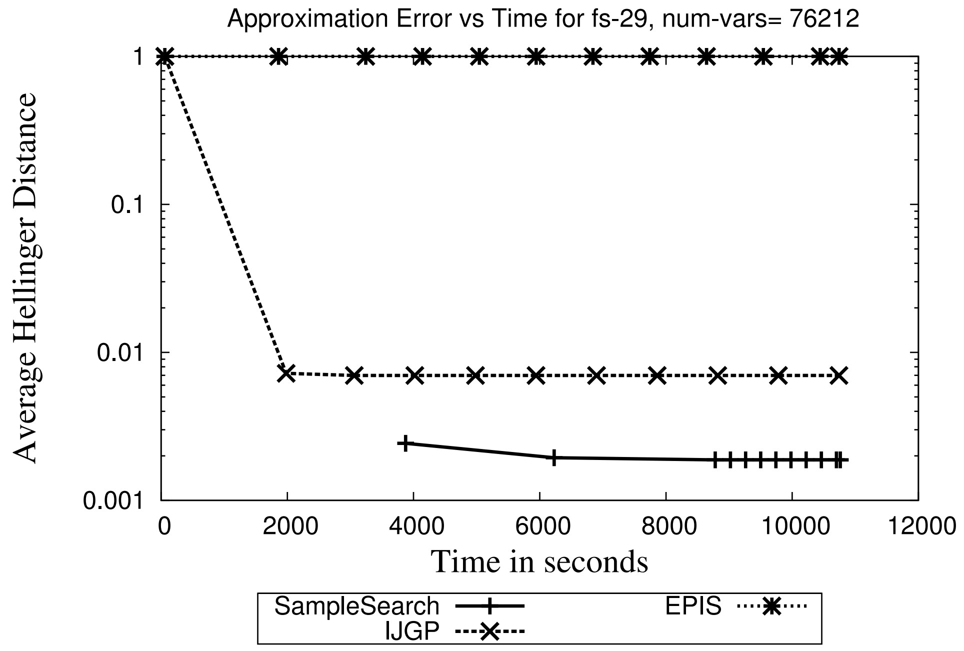
Specifically, on probabilistic graphical models, we showed that a state-of-the-art importance sampling technique of Evidence Pre-propagated Importance Sampling (EPIS) [133] which reasons about determinism in a limited way is unable to generate a single consistent sample on several hard linkage analysis and relational benchmarks. In such cases, SampleSearch is the only alternative importance sampling technique to date.

SampleSearch is also superior to generalized belief propagation schemes like Iterative Join Graph Propagation (IJGP) [33] and Edge Deletion Belief Propagation (EDBP) [18]. In theory, these propagation techniques are anytime, whose approximation quality can be improved by increasing their i-bound. However, their time and space complexity is exponential in i and in practice, beyond a certain i-bound (typically > 25), their memory requirement becomes a major bottleneck. Consequently, as we saw, on most benchmarks IJGP and EDBP quickly converge to an estimate which they are unable to improve with time. SampleSearch, being an importance sampling technique improves with time, and as we demonstrated yields superior anytime performance than IJGP and EDBP.

Finally, on the problem of counting solutions of a SAT/CSP, we showed that SampleSearch is slightly better than the recently proposed SampleCount [62] technique and substantially better than ApproxCount [130] and Relsat [115].

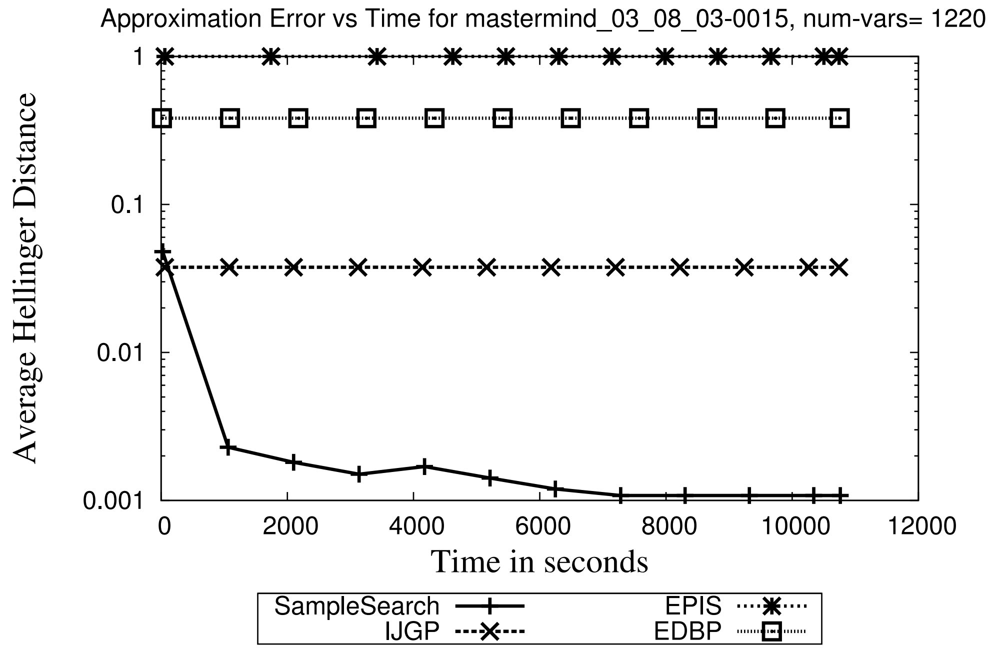


(a)

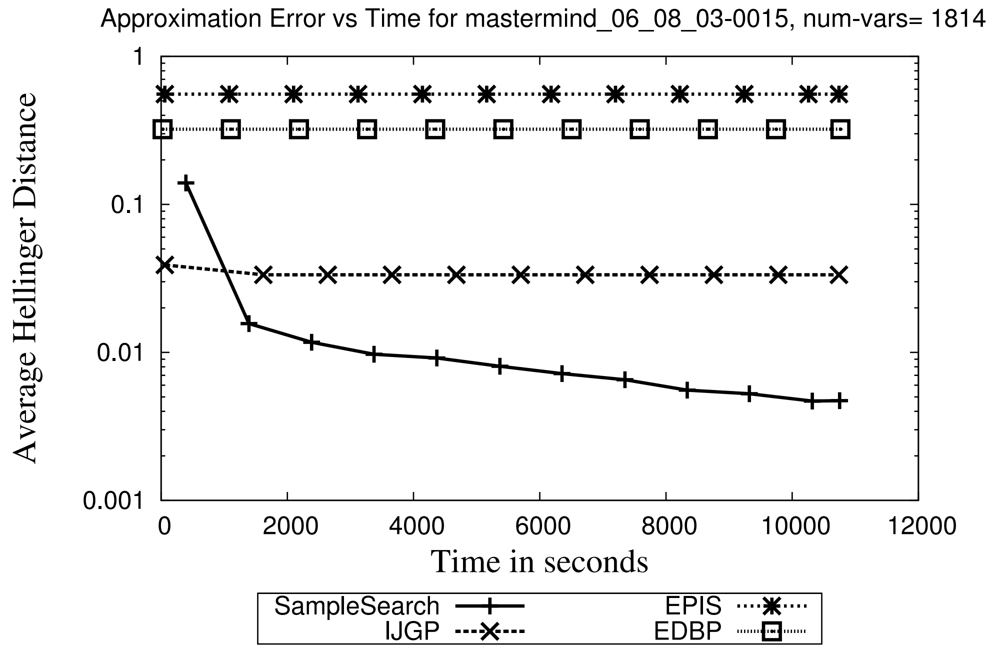


(b)

Figure 3.10: Time versus Hellinger distance for two sample Friends and Smokers networks.

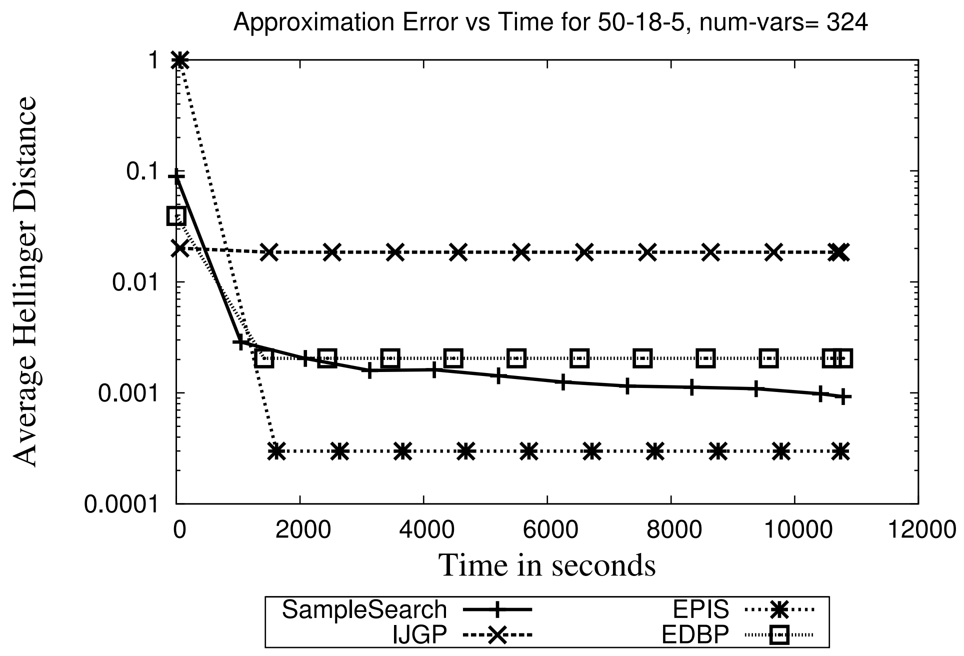


(a)

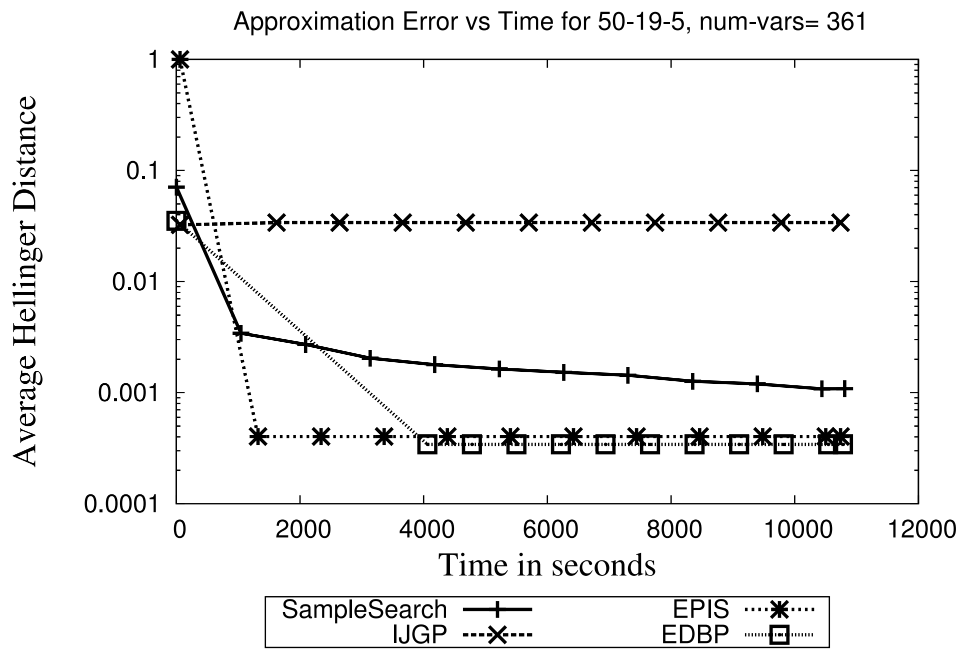


(b)

Figure 3.11: Time versus Hellinger distance for two sample Mastermind networks.

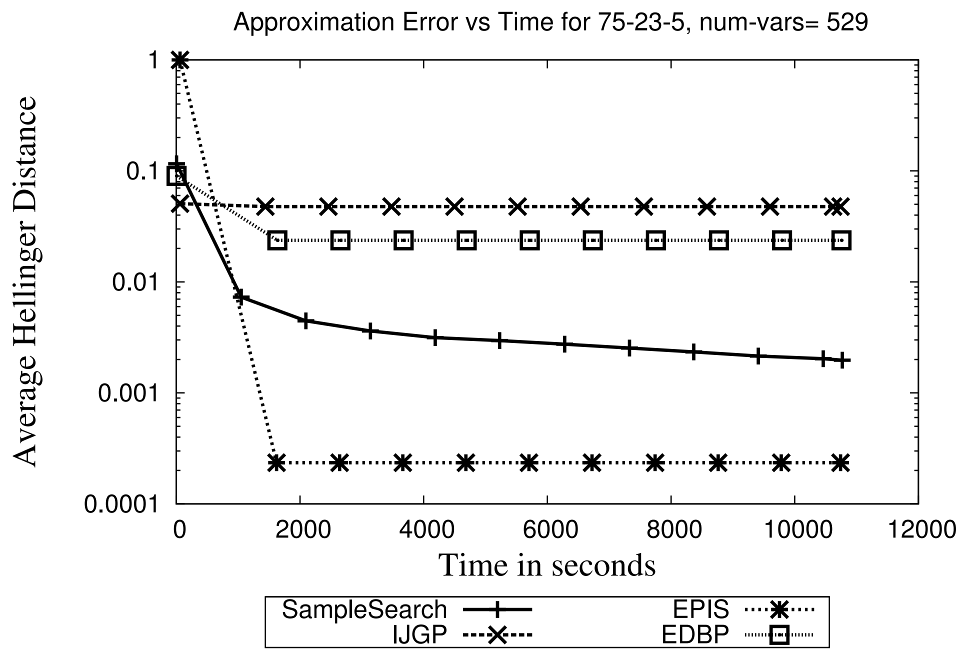


(a)

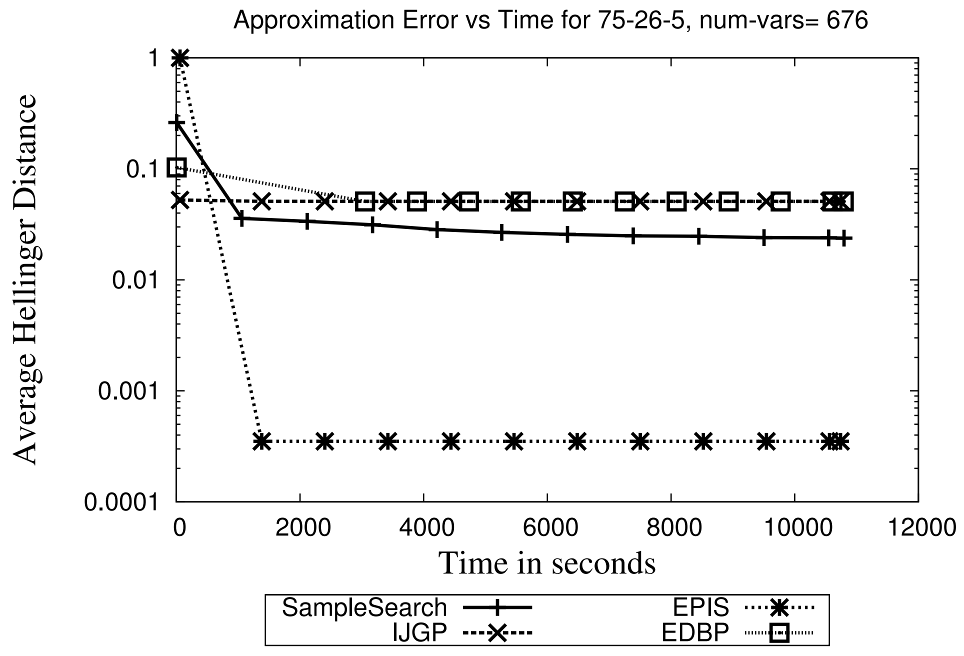


(b)

Figure 3.12: Time versus Hellinger distance for two sample Grid instances with deterministic ratio=50%.

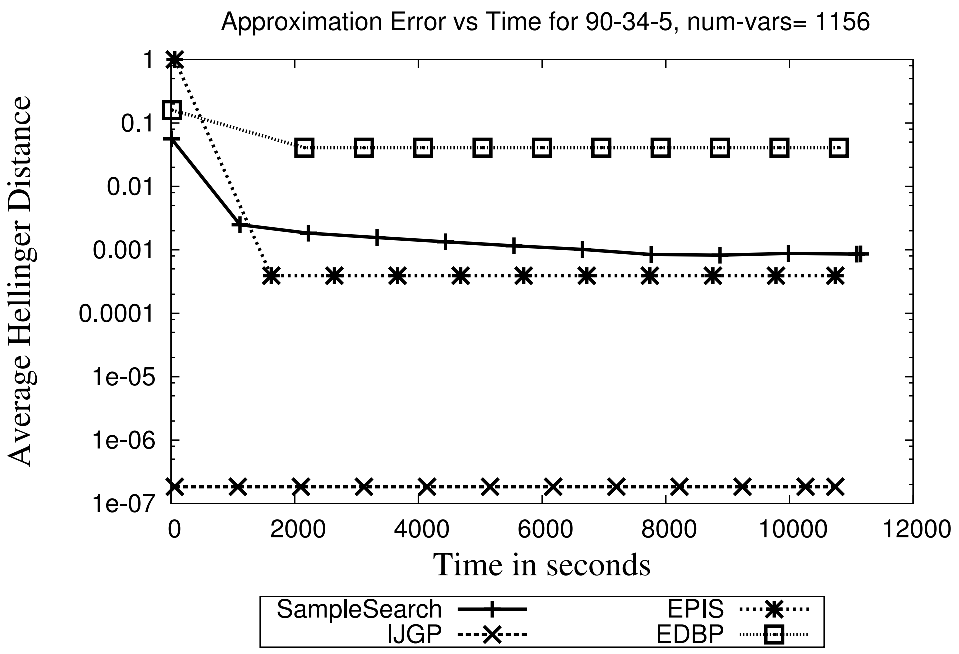


(a)

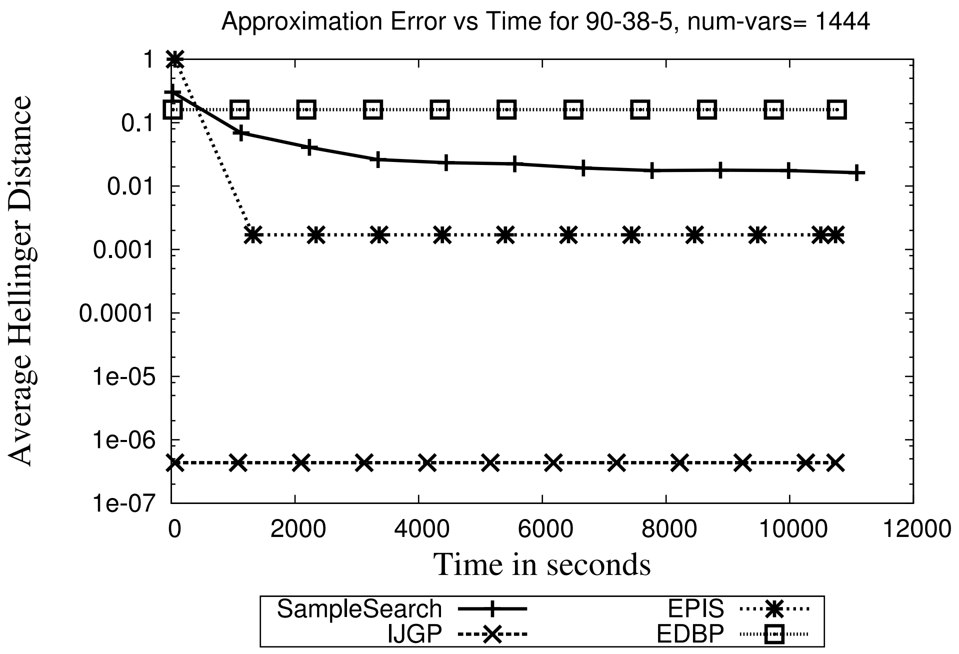


(b)

Figure 3.13: Time versus Hellinger distance for two sample Grid instances with deterministic ratio=75%.

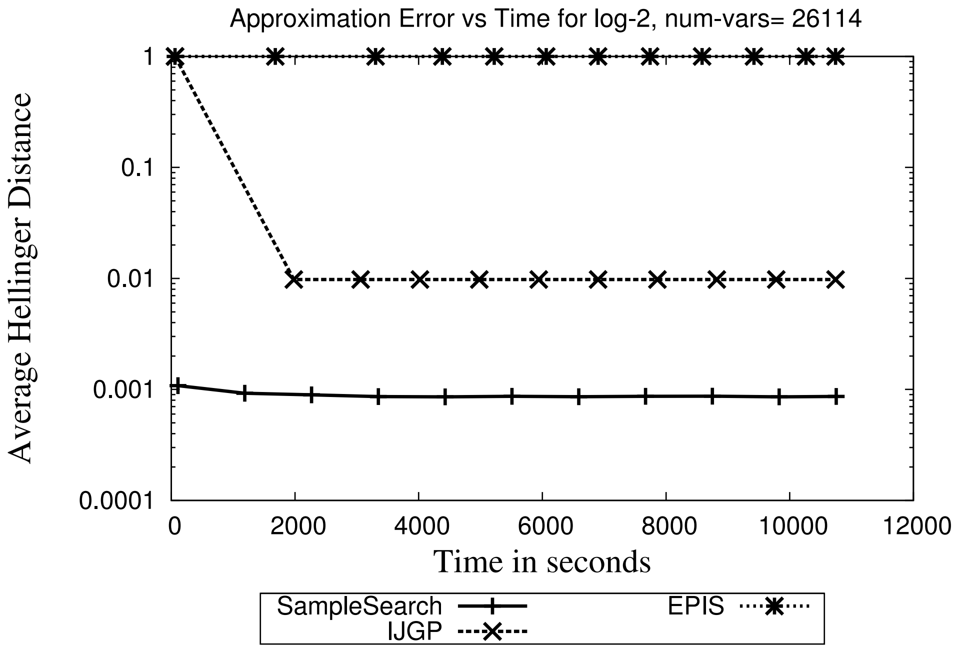


(a)

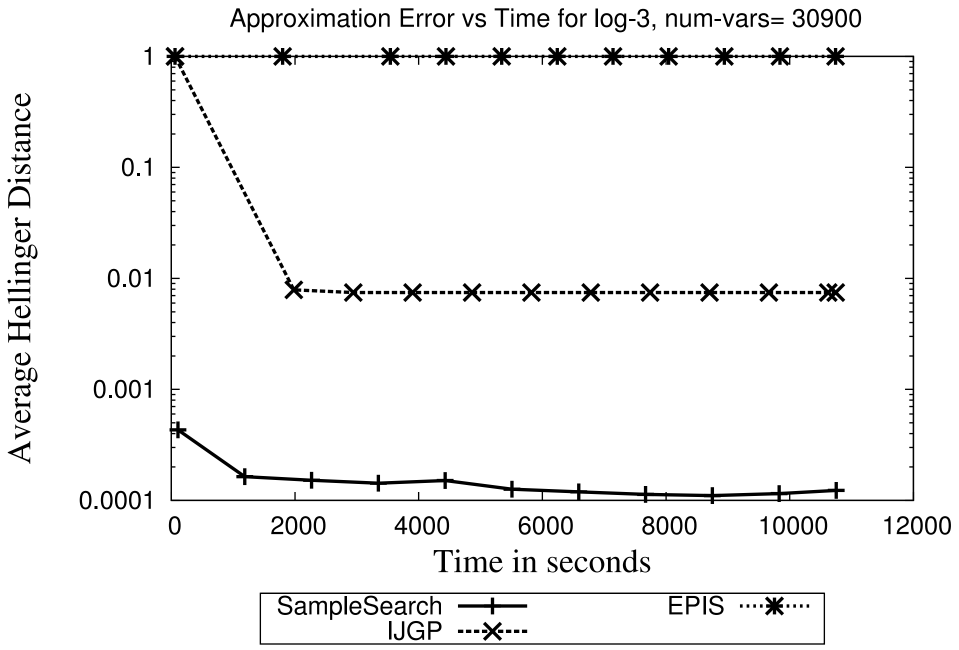


(b)

Figure 3.14: Time versus Hellinger distance for two sample Grid instances with deterministic ratio=90%.



(a)



(b)

Figure 3.15: Time versus Hellinger distance for two sample Logistics planning instances.

Chapter 4

Studies in Solution Sampling

4.1 Introduction

In this chapter, we extend the *SampleSearch with IJGP* scheme presented in Chapter 3 for sampling solutions uniformly from a Boolean satisfiability problem. The primary motivation for solution sampling comes from applications in the field of functional verification [134, 31] and first order probabilistic models [113, 95].

In the functional verification field for instance, the main vehicle for the verification of large and complex hardware designs is the simulation of a large number of random test programs for diagnosis of faults [4, 134]. Hardware designs can be modeled as Boolean satisfiability problems in which case the test programs are its solutions. Typically, the number of solutions of the modeled program could be as large as 10^{1000} and the number of selected test programs are in the range of 10^5 or 10^6. Naturally, the best test generator is the one that would uniformly sample the space of test programs so that every fault in the circuit has equal probability of being tested.

Another application for solution sampling is to perform inference in first-order probabilistic models such as Markov logic networks [113] and Bayesian logic [95]. A Markov logic network (or MLN) is a probabilistic language which applies the ideas of a Markov network to first-order logic. Markov logic networks (MLNs) generalize first-order logic, where in a certain limit, all unsatisfiable statements have a probability of zero, and all entailed formulas have probability one. Popular Markov Chain Monte Carlo (MCMC) inference techniques in MLNs such as MC-SAT [109] assume the ability to uniformly sample solutions of a SAT formula. Therefore, advances in random solution sampling would have impact on the accuracy of inference schemes for MLNs.

Theoretically, the solution sampling task is closely related to the well-known $\#$P-Complete problem of counting solutions of a satisfiability problem. In fact, it is known [31, 70] that if one can sample uniformly from the set of solutions of a satisfiability problem then one can design a highly accurate method for counting solutions of a satisfiability problem. Conversely, one can also design a highly efficient method to solve the solution sampling task from an exact counting algorithm.

This general principle was exploited in [31] in which the authors show how an exact counting method like Bucket Elimination can be used to solve the solution sampling task. The main idea here is to express the uniform distribution $\mathcal{P}$ in a product form $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ or a Bayesian network and then sample from the product form using the standard logic sampling algorithm [106]. The main drawback of Bucket Elimination is that it is time and space exponential in the treewidth and is infeasible when the treewidth is large.

Therefore, in order to make their solution sampling schemes practical, [31] propose to use approximate solution counters based on mini-bucket elimination (MBE) [39] instead of Bucket Elimination. In this chapter, we propose to use a more powerful scheme called Iterative Join Graph propagation (IJGP) instead of MBE. The main idea in these MBE and IJGP based schemes is to construct a product-form distribution $Q = \{Q_1, \ldots, Q_n\}$

that approximates the product form $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ of the uniform distribution over the solutions. However, both IJGP and MBE schemes have two issues. First, because Q is an approximation of $\mathcal{P}$, solutions generated from Q are biased. Second, just like the proposal distribution in importance sampling, Q suffers from the *rejection problem* in that it may generate non-solutions. Clearly, we can easily use the SampleSearch scheme presented in Chapter 3 to remedy rejection. However, even with SampleSearch, the problem of generating biased solution samples still remains in that SampleSearch samples from the backtrack-free distribution Q^F of Q which can be quite far from the required uniform distribution over the solutions.

The main contribution of this chapter is in correcting this deficiency by augmenting SampleSearch with statistical techniques of Sampling/Importance Resampling (SIR) and the Metropolis-Hastings (MH) method yielding SampleSearch-SIR and SampleSearch-MH respectively. Our new techniques operate in two phases. In the first phase, they use SampleSearch to generate a set of solution samples and in the second phase they thin down the samples by accepting only a subset of good samples. By carefully selecting this acceptance criteria, we can ensure that the generated samples converge in the limit to the required uniform distribution over the solutions. SampleSearch-MH and SampleSearch-SIR are the first to have such convergence guarantees, not available for state-of-the-art schemes such as SampleSat [130] and XorSample [63].

We present empirical evaluation, comparing with SampleSat [130] and XorSample [63]. We ran each scheme for the same amount of time and evaluated their performance using two criteria: (i) throughput, measured as the number of solutions generated per second and (ii) sampling accuracy measured using the Hellinger distance [77] between the uniform distribution and the sampling distribution of the generated random solutions. Our results demonstrate that SampleSearch is competitive with SampleSat and XorSample in terms of accuracy and throughput while the schemes we introduce here, SampleSearch-MH and

SampleSearch-SIR are usually more accurate.

The research presented in this chapter is based in part on [54, 59].

The rest of the chapter is organized as follows. In Section 4.2 we present preliminaries and background. In Section 4.3, we present the IJGP based solution sampling scheme while in Section 4.4, we review the SampleSearch scheme. In Sections 4.5 and 4.6, we present the SampleSearch-MH and SampleSearch-SIR schemes respectively. Experimental results are presented in section 4.7 and we conclude in section 4.8.

4.2 Background and Related work

In this section, we describe the notation and terminology, define the solution sampling problem, and describe earlier work.

4.2.1 Basic Notation and Definitions

We denote a SAT formula represented in conjunctive normal form (CNF) by $F(\mathbf{X}, \mathbf{C})$ where $\mathbf{X} = \{X_1, \ldots, X_n\}$ and $\mathbf{C} = \{C_1, \ldots, C_m\}$ denote the sets of variables and clauses respectively. Each variable X_i can take two values from the set $\{0, 1\}$ (or $\{False, True\}$). A literal is a variable X_i or its negation $\neg X_i$. A clause is a disjunction (denoted by $\vee$) of literals and the Formula F is a conjunction (denoted by $\wedge$) of clauses. For example, $F = (X_1 \vee \neg X_2) \wedge (\neg X_1 \vee \neg X_3 \vee X_4) \wedge (X_2 \vee X_4)$ is a CNF formula with four variables $\{X_1, X_2, X_4, X_4\}$ and three clauses.

A clause C_i is satisfied by an assignment **x** if at least one literal in C_i is set to 1 ($True$). (Note that if a variable X_i is assigned the value 0 ($False$) then it implies that the literal $\neg X_i$ is assigned a value 1 ($True$) and vice versa.) If **x** satisfies all clauses of F, then **x** is a solution or a model of F.

Given an assignment x_i, the assignment $\overline{x_i}$ denotes the negation of x_i. For example, if x_i is True, then $\overline{x_i}$ is False and vice versa.

With each clause C_i, we associate a $0/1$ function $\mathcal{C}_i$ defined on the variables corresponding to the literals of C_i called its scope and is denoted by $S(\mathcal{C}_i)$:

$$\mathcal{C}_i(\mathbf{x}) = \begin{cases} 1 & \text{If clause } C_i \text{ is satisfied by } \mathbf{x} \\ 0 & \text{Otherwise} \end{cases} \tag{4.1}$$

Similarly, with a formula F, we associate a $0/1$ function $\mathcal{F}$ whose scope is the set **X** of variables and is defined as:

$$\mathcal{F}(\mathbf{x}) = \begin{cases} 1 & \text{If } \mathbf{x} \text{ satisfies all clauses of } F \\ 0 & \text{Otherwise} \end{cases} \tag{4.2}$$

It is easy to see that:

$$\mathcal{F}(\mathbf{x}) = \prod_{i=1}^{m} \mathcal{C}_i(\mathbf{x}) \tag{4.3}$$

Given a SAT formula F and an integer M, the solution sampling task is to generate M solutions from F such that each solution is generated with equal probability. Formally,

DEFINITION **35** (**The Solution Sampling task**). *Given a SAT formula $F(\boldsymbol{X}, \boldsymbol{C})$ and an integer M, let $\#sol$ be the number of solutions of F given by:*

$$\#sol = \sum_{\mathbf{x} \in \mathbf{X}} \mathcal{F}(\mathbf{x}) = \sum_{\mathbf{x} \in \mathbf{X}} \prod_{i=1}^{m} \mathcal{C}_i(\mathbf{x}) \tag{4.4}$$

and let $\mathcal{P}$ be a uniform distribution over the solutions of F, where

$$\mathcal{P}(\mathbf{x}) = \begin{cases} \frac{1}{\#sol} & \textit{If } \boldsymbol{x} \textit{ is a solution of } F \\ 0 & \textit{Otherwise} \end{cases}$$

then the solution sampling task is to generate M solution samples from $\mathcal{P}$.

4.2.2 Earlier work

A brute-force way to sample solutions uniformly is to first generate and store all solutions and then generate M solutions from this stored set such that each solution is sampled with equal probability; but this is clearly impractical.

A more reasonable approach developed by Dechter et al. [31] is presented as Algorithm 13. We first express the uniform distribution $\mathcal{P}(\mathbf{x})$ over the solutions in a product form (or a Bayesian network): $\mathcal{P}(\mathbf{x}) = \prod_{i=1}^{n} \mathcal{P}_i(x_i|x_1,\ldots,x_{i-1})$. Then, we use a standard Monte Carlo (MC) sampler, also called logic sampling [106] to sample along the ordering $o = \langle X_1,\ldots,X_n\rangle$ implied by the product form. Namely, at each step, given a partial assignment $(x_1,\ldots,x_{i-1})$ to the previous $i-1$ variables, we assign a value to variable X_i by sampling it from the distribution $\mathcal{P}_i(X_i|x_1,\ldots,x_{i-1})$. Repeating this process for each variable in the sequence produces a solution sample $(x_1,\ldots,x_n)$. To generate the required M solutions from $\mathcal{P}$, all we have to do now is repeat the above process M times.

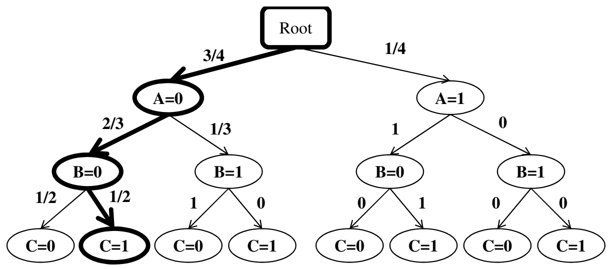
It was shown that:

PROPOSITION **14.** *[31] The probability $\mathcal{P}_i(X_i = x_i|x_1,\ldots,x_{i-1})$ is equal to the ratio between the number of solutions that the partial assignment $(x_1,\ldots,x_i)$ participates in and the number of solutions that $(x_1,\ldots,x_{i-1})$ participates in.*

Proposition 14 implies that a solution counting algorithm can be used to generate the sam-

Algorithm 13: Monte-Carlo Sampler $(F, O, \mathcal{P})$

```
Input: Formula $F$, An Ordering $o = (X_1, \ldots, X_n)$ and the uniform distribution over
       the solutions $\mathcal{P} = \prod_{i=1}^{n} \mathcal{P}_i(x_i|x_1, \ldots, x_{i-1})$
Output: $M$ Solutions drawn from the uniform distribution over the solutions of $F$
1 for $j = 1$ to $M$ do
2     $\mathbf{x} = \phi$;
3     for $i = 1$ to $n$ do
4         $r =$ a random real number in $(0, 1)$ ;
5         if $r < \mathcal{P}_i(X_i = 0|\mathbf{x})$ then
6             $\mathbf{x} = \mathbf{x} \cup (X_i = 0)$;
7         else
8             $\mathbf{x} = \mathbf{x} \cup (X_i = 1)$
9     Output $\mathbf{x}$
```



SAT formula $F = (A \vee \neg B \vee \neg C) \wedge (\neg A \vee B \vee C) \wedge (\neg A \vee \neg B \vee C) \wedge (\neg A \vee \neg B \vee \neg C)$

Figure 4.1: Search tree for the given SAT formula F annotated with probabilities for sampling solutions from a uniform distribution.

pling probabilities for sampling solutions.

EXAMPLE **11.** *Figure 4.1 shows a complete search tree for the given SAT formula F. Each arc from a parent $X_{i-1} = x_{i-1}$ to a child $X_i = x_i$ in the search tree is labeled with the ratio of the number of solutions below the child (i.e. the number of solutions that $(x_1, \ldots, x_i)$ participates in) and the number of solutions below the parent (i.e. the number of solutions that $(x_1, \ldots, x_{i-1})$ participates in) yielding the probability $\mathcal{P}_i(X_i = x_i|X_1 = x_1, \ldots, X_{i-1} = x_{i-1})$. Given a sequence of random numbers $\{0.2, 0.3, 0.6\}$, the reader can verify that the solution highlighted in bold in Figure 4.1 will be generated first by the ordered Monte Carlo sampler.*

Dechter et al. [31] used a bucket elimination based solution counting scheme to compute $\mathcal{P}_i(X_i|x_1,\ldots,x_{i-1})$. In this scheme, bucket elimination is run just once as a pre-processing step yielding a representation from which $\mathcal{P}_i(X_i|x_1,\ldots,x_{i-1})$ can be computed for any partial assignment $(x_1,\ldots,x_{i-1})$ to the previous $i-1$ variables by performing a constant time table look-up (For details see [31]). However, the time and space complexity of this pre-processing scheme is exponential in the induced width of the graph along the given ordering, and when the induced width is large, bucket elimination is impractical. Clearly, any state-of-the-art counting algorithms (e.g. Cachet [119]) can also be used to compute $\mathcal{P}_i(X_i = x_i|x_1,\ldots,x_{i-1})$ but the properties of utilizing these methods for sampling may be different.

To address the exponential blow up in large induced-width problems, Dechter et al. [31] proposed to approximate each component $\mathcal{P}_i(X_i|x_1,\ldots,x_{i-1})$ using the polynomial mini-bucket elimination scheme (MBE) [39]. In the next section, we propose to compute the approximation denoted by Q using Iterative Join Graph propagation (IJGP) rather than MBE. The quality of the sampling scheme depends upon the accuracy of Q (i.e. how close Q is to $\mathcal{P}$). Since IJGP was empirically shown to approximate $\mathcal{P}$ better than MBE [33], it is likely to be a better choice.

4.3 Using IJGP for solution sampling

To recap, Iterative Join-Graph Propagation (IJGP) [33] (see Section 1.4.1) is a class of generalized belief propagation algorithm [132] that applies the message passing algorithm of join-tree clustering to join-graphs iteratively. A join graph is a decomposition of functions into a graph of clusters (rather than a tree) that satisfies the running intersection property. IJGP uses a parameter i which controls the cluster size of the join-graph, yielding a class of algorithms (IJGP(i)) whose complexity is exponential in i, that allow a trade-off between

Algorithm 14: **ALGORITHM IJGP(i,p)-SAMPLING**

```
   Input: A SAT Formula $F(\mathbf{X}, \mathbf{C})$, an ordering $o = (X_1, \ldots, X_n)$, Integers $i$,
          $p \in \{0, 100\}$ and $M$
   Output: $M$ randomly generated solutions.
1  Run IJGP(i) on the formula $F$ and compute a proposal distribution
   $Q = \{Q_1, \ldots, Q_n\}$ from its output using Equation 4.5;
2  Randomly select $p/100 \times n$ variables without replacement from $\mathbf{X}$ and store them in
   a set $\mathbf{X}_p$;
3  repeat
4      Initialize s to the null assignment;
5      for $j = 1$ to $n$ do
6          $Q'_j = Q_j$;
7      for $j$=1 to $n$ do
8          if $X_j \in \boldsymbol{X}_p$ then
9              $F'$=Unit-propagate($F$,s);
10             Run IJGP on $F'$ and recompute $\{Q'_j, \ldots, Q'_n\}$ from IJGP's output using
               Equation 4.5 ;
11         r=random number between 0 and 1;
12         if $r < Q'_j(X_j = 0|\boldsymbol{s})$ then
13             $\mathbf{s} = \mathbf{s} \cup (X_j = 0)$ ;
14         else
15             $\mathbf{s} = \mathbf{s} \cup (X_j = 1)$;
           // Check if s is inconsistent
16         If s is not consistent according to $F$, then goto step 3.;
17     Output s;
18 until $M \leq 0$ ;
```

accuracy and complexity. As i increases, accuracy generally increases. When i is big enough to allow a tree-structure, IJGP(i) coincides with join-tree clustering and becomes exact.

Given a SAT Formula $F(\mathbf{X}, \mathbf{C})$, the output of IJGP (see Algorithm 1 in Chapter 1) is a join graph $JG = (G(\mathbf{V}, \mathbf{E}), \chi, \psi, \theta)$ in which each cluster $A \in \mathbf{V}$ contains the original clauses $\psi(A) \subseteq \mathbf{C}$ and messages $m_{B \rightarrow A}$ received from all its neighbors $B \in N_G(A)$. Each component $Q_i(X_i|X_1, \ldots, X_{i-1})$ can be constructed from JG as follows. Let A be the cluster of JG containing X_i, namely $X_i \in \chi(A)$. Let $\mathbf{Y} = \chi(A) \cap \{X_1, \ldots, X_{i-1}\}$. Then,

$$Q_i(x_i|\mathbf{y}) = \alpha \sum_{\mathbf{z} \in \chi(A) \backslash \{X_1, \ldots, X_i\}} \left(\prod_{C \in \psi(A)} C(\mathbf{z}, x_i, \mathbf{y}) \prod_{B \in N_G(A)} m_{B \rightarrow A}(\mathbf{z}, x_i, \mathbf{y}) \right) \tag{4.5}$$

where α is the normalization constant which ensures that $Q_i(X_i|\mathbf{y})$ is a proper probability distribution. Because there could be many clusters in the join graph that mention X_i, as mentioned in Chapter 2, we choose heuristically the cluster which mentions X_i and has the largest intersection with $\{X_1, \ldots, X_{i-1}\}$ (ties are broken randomly).

Algorithm 14 presents a IJGP-based technique for sampling solutions from a Satisfiability formula F. We introduce here a parameter p for controlling the recomputation of the sampling distribution periodically. Given a SAT formula $F(\mathbf{X}, \mathbf{C})$ and an ordering $o = (X_1, \ldots, X_n)$ of variables, we first run IJGP and create the components $Q = \{Q_1, \ldots, Q_n\}$ of the sampling distribution using Equation 4.5. Then, we randomly select $p\%$ of the variables and designate them as checkpoints at which IJGP will be rerun. We sample variables one by one using the latest proposal distribution component Q_j until a checkpoint variable X_j is encountered. At each check-point, we re-run IJGP on the Formula F' obtained from F by conditioning on the current assignment to the previous $j-1$ variables and recompute the components $\{Q_j, \ldots, Q_n\}$ for all the unassigned variables. The primary reason for introducing $p \in \{0, 100\}$ is to explore a time versus accuracy tradeoff. As we increase p, the accuracy of Q improves but so does the time required to generate a sample.

PROPOSITION **15.** *The time complexity and space complexity of IJGP(i,p)-sampling is* $O(h \times n \times \frac{p}{100} \times exp(i) \times N)$ *and* $O(h \times exp(i))$ *respectively where* n *is the number of variables,* h *is the number of clusters in the join graph of IJGP and* N *is the number of solution samples.*

Proof. To generate a sample, IJGP is executed $O(n \times \frac{p}{100})$ times. The time complexity of running IJGP is $O(h \times exp(i))$. Therefore, to generate N samples, we would require $O(h \times n \times \frac{p}{100} \times exp(i) \times N)$ time.

The space required by IJGP and Q is $O(h \times exp(i))$ and $O(n \times exp(i))$ respectively. Because $h \geq n$, the overall space complexity is $O(h \times exp(i))$. □

Another issue with IJGP(i,p) sampling is Step 16 of the algorithm in which a partial (or full) sample which is not consistent is rejected. Obviously, we can use SampleSearch presented in Chapter 3 in a straight forward manner to manage this problem. For completeness sake, we present this scheme next.

4.4 The SampleSearch scheme for solution sampling

Instead of returning with a sample that is inconsistent, $SampleSearch$ progressively revises the inconsistent sample via backtracking search until a solution is found. Since the focus of this chapter is on SAT formulas, we use the conventional backtracking procedure for SAT which is the DPLL algorithm [26].

$SampleSearch$ with DPLL is presented as Algorithm 15. It takes as input a formula F, an ordering $o = \langle X_1, \ldots, X_n \rangle$ of variables and a distribution $Q(\mathbf{X}) = \prod_{i=1}^{n} Q_i \ (X_i | \ X_1, \ldots, X_{i-1})$ along that ordering. Given a partial assignment $(x_1, ..., x_{i-1})$ already generated, the next variable in the ordering X_i is selected and its value $X_i = x_i$ is sampled from the

Algorithm 15: $SampleSearch(F = (\mathbf{X}, \mathbf{C}), Q, o)$

```
Input: A SAT formula $F$ in CNF form, a distribution $Q$ and an ordering $o$ of
       variables.
Output: $1$ if a solution sample is generated ( plus the solution stored in $F$) and $0$
        otherwise
1 if there is an empty clause in F then
2     Return 0;
3 if all clauses in F are unit then
4     Return 1;
5 Select the earliest variable $X_i$ in $o$ not yet assigned a value;
6 for every unit clause $C$ in $F$ do
7     Unit-propagate($F, C$);
8 Sample $X_i = x_i$ from $Q_i(X_i|x_1, \ldots, x_{i-1})$;
9 return $SampleSearch(F \wedge x_i, Q, o)$ OR $SampleSearch(F \wedge \overline{x_i}, Q, o)$;
```

conditional distribution $Q_i(X_i|x_1, \ldots, x_{i-1})$. Then the algorithm applies unit-propagation with the new unit clause $X_i = x_i$ created over the formula F. If no empty clause is generated, the algorithm proceeds to the next variable. Otherwise, the algorithm tries $X_i = \overline{x_i}$, performs unit propagation and either proceeds forward (if no empty clause generated) or it backtracks. On termination, the output of $SampleSearch$ is a solution to F (assuming one exists).

The distribution Q can be constructed using the output of IJGP described in the previous section or by any other scheme. The only restriction we have is that Q must satisfy $\mathcal{P}(\mathbf{x}) > 0 \rightarrow Q(\mathbf{x}) > 0$ to guarantee that every solution is sampled with non-zero probability (similar to importance sampling).

Next, we characterize the sampling distribution of SampleSearch which will be used to quantify how far the generated solutions are from the uniform distribution over the solutions.

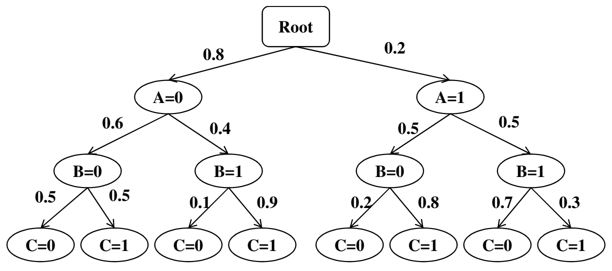
4.4.1 The Sampling Distribution of SampleSearch

As shown in Chapter 3, SampleSearch generates independent and identically distributed samples from the backtrack-free distribution denoted by Q^F. In this section, we recap the definition of the backtrack-free distribution relative to a SAT formula. We will also provide an alternative proof for the claim that SampleSearch generates samples from the backtrack-free distribution. Because of the restriction to binary domains as in a SAT formula, our proof is much simpler and more intuitive than the proof presented in Chapter 3.

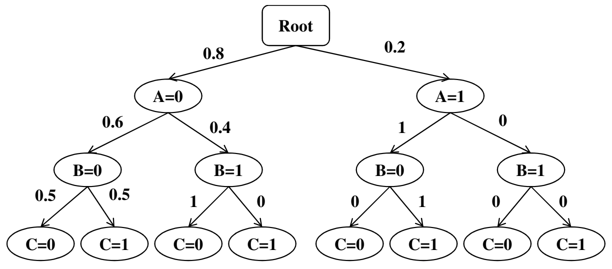
DEFINITION **36 (The Backtrack-free distribution of Q w.r.t. F).** *Given a distribution $Q(\mathbf{X})$ in the product form $Q(\mathbf{X}) = \prod_{i=1}^{n} Q_i(X_i|X_1,\ldots,X_{i-1})$, an ordering $o = \langle X_1,\ldots,X_n\rangle$ and a SAT formula $F(\boldsymbol{X},\boldsymbol{C})$, the backtrack-free distribution Q^F is factored into $Q^F(\mathbf{x}) = \prod_{i=1}^{n} Q_i^F(x_i|x_1,\ldots,x_{i-1})$ where $Q_i^F(x_i|x_1,\ldots,x_{i-1})$ is defined as follows:*

1. *$Q_i^F(x_i|x_1,\ldots,x_{i-1}) = 0$ if $(x_1,\ldots,x_{i-1},x_i)$ cannot be extended to a solution of F.*
2. *$Q_i^F(x_i|x_1,\ldots,x_{i-1}) = 1$ if $(x_1,\ldots,x_{i-1},x_i)$ can be extended to a solution of F but $(x_1,\ldots,x_{i-1},\overline{x_i})$ cannot.*
3. *$Q_i^F(x_i|x_1,\ldots,x_{i-1}) = Q_i(x_i|x_1,\ldots,x_{i-1})$ if both $(x_1,\ldots,x_{i-1},x_i)$ and $(x_1\ ,\ldots,$ $x_{i-1},\overline{x_i})$ can be extended to a solution of F.*

EXAMPLE **12.** *Figure 4.2(a) shows a distribution Q expressed as a probability tree. Each arc from a parent to a child in the probability tree is labeled with the conditional probability of the child given an assignment from the root to the parent. Figure 4.2(b) shows the backtrack-free distribution of Q w.r.t. the given SAT formula F. Q^F is constructed from Q as follows. Given a parent and its two children one which participates in a solution and the other which does not, the edge-label of the child participating in a solution is changed to 1 while the edge-label of the other child which does not participate in a solution is changed to 0. If both children participate in a solution, the edge labels are not changed.*



(a)



(b)

SAT formula $F = (A \vee \neg B \vee \neg C) \wedge (\neg A \vee B \vee C) \wedge (\neg A \vee \neg B \vee C) \wedge (\neg A \vee \neg B \vee \neg C)$

Figure 4.2: (a) A distribution Q expressed as a probability tree and (b) Backtrack-free distribution Q^F of Q w.r.t. F.

THEOREM **12.** *Given a distribution Q and a formula F, the sampling distribution of $SampleSearch$ is the backtrack-free distribution Q^F of Q w.r.t. F.*

Proof. Let $I(\mathbf{x}) = \prod_{i=1}^{n} I_i(x_i|x_1, \ldots, x_{i-1})$ be the factored sampling distribution of SampleSearch (F, Q, o). We will prove that for any arbitrary partial assignment $(x_1, \ldots, x_{i-1}, x_i)$, $I_i(x_i|x_1, \ldots, x_{i-1}) = Q_i^F(x_i|x_1, \ldots, x_{i-1})$. We consider three cases corresponding to the definition of the backtrack-free distribution (see Definition 36):

Case (1): $(x_1, \ldots, x_{i-1}, x_i)$ cannot be extended to a solution. Since, all samples generated by $SampleSearch$ are solutions of F, $I_i(x_i|x_1, \ldots, x_{i-1}) = 0 = Q_i^F(x_i|x_1, \ldots, x_{i-1})$.

Case (2): $(x_1, \ldots, x_{i-1}, x_i)$ can be extended to a solution but $(x_1, \ldots, x_{i-1}, \overline{x_i})$ cannot be extended to a solution. Since $SampleSearch$ is a systematic search procedure, if it explores a partial assignment $(x_1, \ldots, x_k)$ that can be extended to a solution of F, it is guaranteed to return a full sample (solution) extending this partial assignment. Otherwise, if $(x_1, \ldots, x_k)$ is not part of any solution of F then $SampleSearch$ will prove this inconsistency before it will finish generating a full sample. Consequently, if $(x_1, \ldots, x_{i-1}, x_i)$ is sampled first, a full assignment (a solution) extending $(x_1, \ldots, x_i)$ will be returned and if $(x_1, \ldots, \bar{x}_i)$ is sampled first, $SampleSearch$ will detect that $(x_1, \ldots, x_{i-1}, \overline{x_i})$ cannot be extended to a solution and eventually explore $(x_1, \ldots, x_{i-1}, x_i)$. Therefore, in both cases if $(x_1, \ldots, x_{i-1})$ is explored by $SampleSearch$, a solution extending $(x_1, \ldots, x_{i-1}, x_i)$ will definitely be generated i.e. $I_i(x_i|x_1, \ldots, x_{i-1}) = 1 = Q_i^F(x_i|x_1, \ldots, x_{i-1})$.

Case (3): Both $(x_1, \ldots, x_{i-1}, x_i)$ and $(x_1, \ldots, x_{i-1}, \overline{x_i})$ can be extended to a solution. Since $SampleSearch$ is a systematic search procedure, if it samples a partial assignment $(x_1, \ldots, x_{i-1}, \overline{x_i})$ it will return a solution extending $(x_1, \ldots, x_{i-1}, \overline{x_i})$ without sampling $(x_1, \ldots, x_{i-1}, x_i)$. Therefore, the probability of sampling x_i given $(x_1, \ldots, x_{i-1})$ equals $Q_i(x_i|x_1, \ldots, x_{i-1})$ which equals $Q_i^F(x_i|x_1, \ldots, x_{i-1})$. $\square$

Note that the only property of backtracking search that we have used in the proof is its *systematic nature*. Therefore, as pointed out in Chapter 3, if we replace naive backtracking search by any systematic SAT solver such as minisat [125] for example, Theorem 12 would still hold. The only modifications we have to make are: (a) use static variable ordering [1], (b) use value ordering based on the proposal distribution Q. Consequently:

COROLLARY **1.** *Given a Formula F, a distribution Q and an ordering o, any systematic SAT solver replacing DPLL in $SampleSearch$ will generate independent and identically distributed solution samples from the backtrack-free distribution Q^F.*

While SampleSearch samples from the backtrack-free distribution Q^F, it still does not solve the solution sampling problem because Q^F may still be quite far from the uniform distribution over the solutions. In the next two sections, we show how to augment SampleSearch

[1] We can also use some restricted dynamic variable orderings and still maintain correctness of Theorem 12

with ideas presented in the statistics literature – the Metropolis-Hastings method (MH) and the Sampling/Importance Resampling (SIR) theory so that in the limit the sampling distribution of the resulting techniques is the target uniform distribution over the solutions.

4.5 SampleSearch-MH

As explained in Chapter 1, the main idea in the Metropolis-Hastings (MH) simulation algorithm [65] is to generate a Markov Chain whose limiting or stationary distribution is equal to the target distribution $\mathcal{P}$. A Markov chain consists of a sequence of states and a proposal distribution $T(\mathbf{y}|\mathbf{x})$ for moving from state $\mathbf{x}$ to state $\mathbf{y}$. Given a proposal distribution $T(\mathbf{y}|\mathbf{x})$, an acceptance function $0 \leq r(\mathbf{y}|\mathbf{x}) \leq 1$ and a current state $\mathbf{x}^t$, the Metropolis-Hastings algorithm works as follows:

1. Draw $\mathbf{y}$ from $T(\mathbf{y}|\mathbf{x}^t)$.

2. Draw p uniformly from $[0, 1]$ and update

$$\mathbf{x}^{t+1} = \begin{cases} \mathbf{y} & \text{if } p \leq r(\mathbf{y}|\mathbf{x}^t) \\ \mathbf{x}^t & \text{otherwise} \end{cases}$$

Hastings [65] proved that the stationary distribution of the above Markov Chain converges to $\mathcal{P}$ when it satisfies the *detailed balance* condition. Formally,

THEOREM **13.** *[65] The stationary distribution of the Markov Chain generated using the two steps outlined above converges to $\mathcal{P}$, when the following detailed balanced condition is satisfied:*

$$\mathcal{P}(\mathbf{x})A(\mathbf{y}|\mathbf{x}) = \mathcal{P}(\mathbf{y})A(\mathbf{x}|\mathbf{y}) \tag{4.6}$$

where $A(\boldsymbol{x}|\boldsymbol{y}) = T(\mathbf{x}|\mathbf{y})r(\mathbf{x}|\mathbf{y})$ is called the transition function.

Algorithm 16: $SampleSearch - MH(F, Q, O, M)$

```
Input: A SAT formula $F$, A distribution $Q$, An ordering $O$, Integer $M$.
Output: $M$ solution samples
1 $\mathbf{x}^0$ = SampleSearch$(F, O, Q)$;
2 for $t = 0$ to $M - 1$ do
3     $\mathbf{y}$ = SampleSearch$(F, O, Q)$;
4     Generate a random number $p$ in the interval $[0, 1]$;
5     if $p \leq \frac{Q^F(\mathbf{x}^t)}{Q^F(\mathbf{x}^t)+Q^F(\mathbf{y})}$ then
6         $\mathbf{x}^{t+1} = \mathbf{y}$;
7     else
8         $\mathbf{x}^{t+1} = \mathbf{x}^t$;
```

Algorithm 16 uses the Metropolis Hastings algorithm (MH) [65] for solution sampling. We first generate a solution sample $\mathbf{y}$ using SampleSearch and then accept the solution with probability $\frac{Q^F(\mathbf{x})}{Q^F(\mathbf{x})+Q^F(\mathbf{y})}$. We can show that:

PROPOSITION **16.** *The Markov Chain of SampleSearch-MH (Algorithm 16) satisfies the detailed balance condition.*

Proof. Since our task is to generate samples from a uniform distribution over the solutions, for any two solution samples **x** and **y**, we have

$$\mathcal{P}(\mathbf{x}) = \mathcal{P}(\mathbf{y})$$

Therefore, the detailed balance condition given in Equation 4.6 reduces to:

$$A(\mathbf{y}|\mathbf{x}) = A(\mathbf{x}|\mathbf{y}) \tag{4.7}$$

Because each sample is generated independently from Q^F and we accept it with probability

$\frac{Q^F(\mathbf{x})}{Q^F(\mathbf{x})+Q^F(\mathbf{y})}$, the transition function $A(\mathbf{y}|\mathbf{x})$ is given by:

$$A(\mathbf{y}|\mathbf{x}) = Q^F(\mathbf{y})\frac{Q^F(\mathbf{x})}{Q^F(\mathbf{x})+Q^F(\mathbf{y})} \tag{4.8}$$

Similarly, the transition function $A(\mathbf{x}|\mathbf{y})$ is given by:

$$A(\mathbf{x}|\mathbf{y}) = Q^F(\mathbf{x})\frac{Q^F(\mathbf{y})}{Q^F(\mathbf{x})+Q^F(\mathbf{y})} \tag{4.9}$$

From Equations 4.8 and 4.9, we have $A(\mathbf{y}|\mathbf{x}) = A(\mathbf{x}|\mathbf{y})$ as required.

□

It follows from Proposition 16 and the Metropolis-Hastings theory that the stationary distribution of the Markov Chain generated by SampleSearch-MH is equal to the uniform distribution over the solutions. Therefore, if we run SampleSearch-MH long enough (also called as the burn-in time), the samples produced by it can be regarded as following the uniform distribution over the solutions. Integrating MH with SampleSearch opens the way for applying various improvements to MH proposed in the statistics literature. We describe next the integration of an improved MH algorithm with SampleSearch.

4.5.1 Improved SampleSearch-MH

In SampleSearch-MH, we generate a state (or a sample) from the backtrack-free distribution and subsequently a decision is made whether to accept the sample based on an acceptance function. In Improved SampleSearch-MH (Algorithm 17), we first *widen* the acceptance by generating multiple samples instead of just one and then accept a good sample from these multiple options. Such methods are referred to as *multiple trial MH* [89]. Given

Algorithm 17: Improved-SampleSearch-MH (F, Q, O, M, k)

```
Input: A formula $F$, A distribution $Q$, An ordering $O$, Integer $M$ and $k$
Output: $M$ solutions
1  Generate $\mathbf{x}^0$ using SampleSearch$(F, O, Q)$;
2  for $t = 0$ to $M - 1$ do
3      $\mathbf{Y} = \phi$;
4      for $j = 1$ to $k$ do
5          $\mathbf{y}_j$ = SampleSearch$(F, O, Q)$;
6          $\mathbf{Y} = \mathbf{Y} \cup \mathbf{y}_j$;
7      Compute $\mathcal{W} = \sum_{j=1}^{k} \frac{1}{Q^F(\mathbf{y}_j)}$;
8      Select $\mathbf{y}$ from the set $\mathbf{Y}$ by sampling each element $\mathbf{y}_j$ in $\mathbf{Y}$ with probability
       proportional to $\frac{1}{Q^F(\mathbf{y}_j)}$;
9      Generate a random number $p$ in the interval $[0, 1]$;
10     if $p \leq \frac{\mathcal{W}}{\mathcal{W} - \frac{1}{Q^F(\mathbf{y})} + \frac{1}{Q^F(\mathbf{x}^t)}}$ then
11         $\mathbf{x}^{t+1} = \mathbf{y}$;
12     else
13         $\mathbf{x}^{t+1} = \mathbf{x}^t$;
```

a current sample $\mathbf{x}$, the algorithm generates k candidates $\mathbf{y}_1, \ldots, \mathbf{y}_k$ from the backtrack-free distribution Q^F in the usual SampleSearch style. It then selects a (single) sample $\mathbf{y}$ from the k candidates with probability proportional to $1/Q^F(\mathbf{y}_j)$. Sample $\mathbf{y}$ is then accepted according to the following acceptance function:

$$r(\mathbf{y}|\mathbf{x}) = \frac{\mathcal{W}}{\mathcal{W} - \frac{1}{Q^F(\mathbf{y})} + \frac{1}{Q^F(\mathbf{x})}} \quad where \;\; \mathcal{W} = \sum_{j=1}^{k} \frac{1}{Q^F(\mathbf{y}_j)} \tag{4.10}$$

Using results from [89], it is easy to prove that:

PROPOSITION **17.** *[89] The Markov Chain generated by Improved-SampleSearch-MH satisfies the detailed balance condition.*

In summary,

THEOREM **14.** *The stationary distribution of SampleSearch-MH and Improved SampleSearch-MH is the uniform distribution over the solutions.*

Proof. Proof follows from Propositions 16 and 17 and the Metropolis-Hastings theory [65].

□

4.6 SampleSearch-SIR

```
Algorithm 18: $SampleSearch - SIR(F, Q, o, N, M)$
  Input: A formula $F$, A distribution $Q$, An ordering $O$, Integers $M$ and $N$, $N > M$
  Output: $M$ solution samples
1 Generate $N$ i.i.d. samples $\mathbf{A} = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ by executing SampleSearch(F,O,Q) $N$
  times;
2 Compute importance weights $\{w(\mathbf{x}^1) = \frac{1}{Q^F(\mathbf{x}^1)}, \ldots, w(\mathbf{x}^N) = \frac{1}{Q^F(\mathbf{x}^N)}\}$ for each
  sample where $Q^F$ is the backtrack-free distribution;
3 Normalize the importance weights using $\widetilde{w}(\mathbf{x}^i) = w(\mathbf{x}^i)/\sum_{j=1}^{N} w(\mathbf{x}^j)$;
  // Resampling Step
4 Generate $M$ i.i.d. samples $\{\mathbf{y}^1, \ldots, \mathbf{y}^M\}$ from $\mathbf{A}$ by sampling each sample $\mathbf{x}^i$ with
  probability $\widetilde{w}(\mathbf{x}^i)$.;
```

We now discuss our second scheme in which we augment SampleSearch with Sampling/Importance Resampling (SIR) [117] which will yield the SampleSearch-SIR technique. Standard SIR [117] aims at drawing random samples from a target distribution $\mathcal{P}(\mathbf{x})$ by using a given distribution $Q(\mathbf{x})$ which satisfies $\mathcal{P}(\mathbf{x}) > 0 \Rightarrow Q(\mathbf{x}) > 0$ (which is the same requirement as the importance sampling one). First, a set of independent and identically distributed random samples $\mathbf{A} = (\mathbf{x}^1, \ldots, \mathbf{x}^N)$ are drawn from a proposal distribution $Q(\mathbf{x})$. Second, a possibly smaller number of samples $\mathbf{B} = (\mathbf{y}^1, \ldots, \mathbf{y}^M)$ are drawn from $\mathbf{A}$ with sample probabilities which are proportional to the weights $w(\mathbf{x}^i) = \mathcal{P}(\mathbf{x}^i)/Q(\mathbf{x}^i)$ (this step is referred to as the *re-sampling step*). The samples from SIR will, consist of independent draws from $\mathcal{P}$ as $N \to \infty$ (see [117] for details).

SampleSearch augmented with SIR is described in Algorithm 18. Here, a set $\mathbf{A} = (\mathbf{x}^1, \ldots, \mathbf{x}^N)$ of solution samples ($N > M$) is generated from SampleSearch. Then, M samples are drawn from $\mathbf{A}$ with sample probabilities proportional to $w(\mathbf{x}^i) = 1/Q^F(\mathbf{x}^i)$.

THEOREM **15.** *As $N \rightarrow \infty$, the samples generated by SampleSearch-SIR consist of independent draws from the uniform distribution $\mathcal{P}$ over the solutions of F.*

Proof. From the SIR theory [117], we know that SampleSearch-SIR would i.i.d. samples from the uniform distribution $\mathcal{P}$ over the solutions of F as $N \rightarrow \infty$ if the following conditions hold:

- **C.1** The N samples are generated i.i.d from some distribution Q^F (satisfied trivially from Theorem 9).
- **C.2** The M samples are generated in the re-sampling step by sampling each sample $\mathbf{x}$ with probability proportional to $\mathcal{P}(\mathbf{x})/Q^F(\mathbf{x})$

Because, $\mathcal{P}$ is a uniform distribution over the solutions, we have

$$\frac{\mathcal{P}(\mathbf{x})}{Q^F(\mathbf{x})} \propto \frac{1}{Q^F(\mathbf{x})} \tag{4.11}$$

Since the M samples are generated in the re-sampling step of SampleSearch-SIR by resampling each generated solution with probability $\propto \frac{1}{Q^F(\mathbf{x})}$, condition $\mathbf{C.2}$ is also satisfied. $\square$

Note that Theorem 15 does not make any assumptions on the type or quality of Q^F. Therefore, even with a bad proposal distribution we are guaranteed to get independent samples in the limit. However, for a finite sample size, the quality of the proposal distribution is the most important factor.

4.6.1 Extensions of basic SampleSearch-SIR

The integration of SampleSearch within the SIR framework allows using various improvements to the SIR framework presented in the statistics literature over the past decade. In this subsection, we consider two such improvements: (a) sampling without replacement [49] and (b) Improved SIR [124].

Sampling without replacement

SampleSearch-SIR uses sampling with replacement in the re-sampling step i.e. the same solution sample may be drawn twice. Instead we could use sampling without replacement so that each solution will appear only once in the final set of samples. Sampling without replacement helps avoid replicates when Q^F is a poor approximation of $\mathcal{P}$. In this situation we may have a few very large weights and many small weights which causes the large weight samples to appear multiple times in the final set of samples. It was shown by [49] that sampling without replacement yields a more desirable intermediate approximation somewhere between the starting (proposal) Q^F and the target distribution $\mathcal{P}$. In the experimental section, we will revisit this issue and show how a simple statistical test based on co-efficient of variation [88] is able to predict which sampling method is likely to have better accuracy.

Improved SampleSearch-SIR

Under certain restrictions, [124] proved that the convergence of SIR is proportional to $O(1/N)$. To speed up this convergence to $O(1/N^2)$, they proposed the Improved SIR framework. For our purposes, Improved SIR only changes the weights during resampling step as follows.

In case of sampling with replacement the Improved SampleSearch-SIR weighs each sample as:

$$w(\mathbf{x}^i) \propto \frac{1}{S_{-i} \times Q^F(\mathbf{x}^i)} \; where \; S_{-i} = \sum_{j=1}^{M} \frac{1}{Q^F(\mathbf{x}^j)} - \frac{1}{Q^F(\mathbf{x}^i)} \tag{4.12}$$

The first draw of Improved SampleSearch-SIR without replacement is specified by the weights in Equation 4.12. For the k-th draw, $k > 1$, the distribution of w is modified to:

$$w(\mathbf{x}^k) \propto \frac{1}{Q^F(\mathbf{x}^k)(\sum_{j=1}^{N} \frac{1}{Q^F(\mathbf{x}^j)} - \sum_{j=1}^{k-1} \frac{1}{Q^F(\mathbf{x}^j)} - k\frac{1}{Q^F(\mathbf{x}^k)})} \tag{4.13}$$

4.6.2 Discussion on related work

Theorems 14 and 15 are important because they guarantee that as the sample size increases, the samples drawn by SampleSearch-SIR and SampleSearch-MH would converge to the uniform distribution over the solutions. Therefore, we can expect that the sampling error of both schemes will decrease as the number of samples is increased and will disappear in the limit. To the best of our knowledge, the state-of-the-art schemes in literature: (a) SampleSat [130] and (b) XorSample [63] do not have such guarantees. In particular, it is difficult to characterize the sampling distribution of SampleSat. It is possible to make SampleSat sample uniformly by setting the noise parameter appropriately but doing so was shown by [130] to compromise significantly the time required to generate a solution sample making SampleSat impractical. The $XorSample$ algorithm [63] can be adjusted to generate samples that can be within a constant factor of the uniform distribution by increasing its slack parameter α. However, if α is fixed, $XorSample$ may not generate solutions from a uniform distribution [63]. Finally, although the samples drawn from the scheme based on bucket elimination [31] do converge to the uniform distribution over the solutions, it is clearly not practical when the tree-width is large.

4.7 Experimental Results

We evaluated the performance of (1) SampleSearch, (2) SampleSearch-SIR with replacement (3) SampleSearch-SIR without replacement and (4) SampleSearch-MH on four domains: (a) circuit, (b) coloring, (c) logistics planning and (d) grid pebbling. The initial distribution Q of SampleSearch is generated using IJGP(i,p) (recall that we re-compute Q by executing IJGP periodically using the parameter p). For each scheme, we experimented with 5 values: $\{0, 10, 20, 50, 100\}$ of p. We also experimented with two other schemes available in literature: Wei et al.'s SampleSat scheme [130] and Gomes et al.'s XorSample scheme [63].

Next, we will describe the implementation details of each scheme.

1. SampleSearch

SampleSearch is implemented via the following components:

- *Search*: As already mentioned, the search component in SampleSearch was implemented using the minisat solver [125].
- *i-bound of IJGP(i,p)*: We use the iterative scheme outlined in Section 3.3.1 to select the maximum i-bound that can be accommodated by 512 MB of space. This was done to ensure that IJGP terminates in a reasonable amount of time.
- *w-cutset sampling*: Following the w-cutset idea [7], we partition the variables $\mathbf{X}$ into two sets $\mathbf{K}$ and $\mathbf{R}$ such that the treewidth of the graphical model induced by $\mathbf{R}$ after removing $\mathbf{K}$ is bounded by w. We only sample the set of variables $\mathbf{K}$ using SampleSearch and then exactly sample the variables in $\mathbf{R}$ given $\mathbf{K} = \mathbf{k}$ by using the Bucket Elimination based exact sampling scheme described in [31]. Again, we use

the iterative scheme outlined in Section 3.3.1 to select the maximum w such that the Bucket Elimination based exact sampling scheme uses only 512MB space.

- *Variable Ordering:*We use the min-fill ordering for creating the join-graph for IJGP because it was shown to be a good heuristic for finding tree decompositions with low treewidth. Sampling is performed in reverse min-fill ordering.

2. SampleSearch-MH

We experimented with the Improved SampleSearch-MH scheme given in Algorithm 17. We set k to 10 (set arbitrarily). *For brevity, henceforth we will refer to SampleSearch-MH as MH.*

3. SampleSearch-SIR with and without replacement

We experimented with the improved SampleSearch-SIR with and without replacement schemes outlined in section 4.6. We use a resampling ratio $M/N = 0.1 = 10\%$. The choice of the resampling ratio of 10% is arbitrary. *For brevity, henceforth we will refer to SampleSearch-SIR with replacement as SIRwr and SampleSearch-SIR without replacement as SIRwor.*

4. SampleSat

SampleSat [130] is based on the popular local search solver called Walksat [120]. Walksat starts with a random assignment to all variables and then flips a variable at each iteration by mixing local search style greedy moves with random walk moves. The inherent randomness due to random walk moves often leads the algorithm to different solutions in different runs. This however provides a biased sampling of solution space. To reduce this bias, Wei

Instance	#Vars	#Cl	$\#Sol$	#Xors(s)	Size of Xors (k)
Circuit					
ssa-7558-158	1363	3034	2.56e+31	85	20
ssa-7558-159	1363	3032	7.65e+33	X	X
2bitcomp_5	125	310	9.84e+15	96	20
2bitmax_6	252	766	2.06e+29	53	20
Coloring					
gcoloring-100	300	1117	1.8E+9	17	150
gcoloring-200	600	2237	1.28e+13	41	35
Logistics					
log-1	939	3785	5.64E+20	59	20
log-2	1337	24777	3.23E+10	26	50
log-3	1413	29487	2.80E+11	27	50
log-4	2303	20963	2.34E+28	65	25
log-5	2701	29534	7.24E+38	102	20
Pebbling					
grid-pbl-0010	110	191	5.93e+23	70	5
grid-pbl-0015	240	436	3.01e+54	X	X
grid-pbl-0020	420	781	5.06e+95	X	X
grid-pbl-0025	650	1226	1.81e+151	X	X
grid-pbl-0030	930	1771	1.54e+218	X	X

Table 4.1: Table showing the two parameters (Number of Xors and Size of Xors) used by XorSample for each problem instance. 'X' indicates that no parameter that satisfies our criteria was found.

and Selman [130] explore a hybrid strategy in which they select with probability p a random walk style move and with probability $1 - p$ a fixed temperature simulated annealing (SA) move (the authors suggest setting the annealing temperature to 1). More specifically, for each SA move, they consider a randomly selected neighbor of the current assignment. When the neighbor satisfies the same or more clauses as the current assignment, the algorithm moves to the neighbor. When the neighbor satisfies fewer clauses than the current assignment, the algorithm moves to the neighbor with probability e^{-Cost} where $Cost$ is the decrease in the number of satisfied clauses. Note that the simulated annealing moves, when used in excess tend to make the algorithm significantly slower. Therefore, Wei and Selman [130] suggest setting $p = 0.5$. In our experiments, we used an implementation of SampleSat available from the authors website [2].

[2]http://www.cs.cornell.edu/∼sabhar/

5. XorSample

XorSample [63] works by adding random Xor (parity) constraints to the original SAT formula. According to a formal result by Valiant and Vazirani [127], for a formula with n variables, each Xor constraint of length $n/2$ would cut the solution space of satisfying assignments approximately by half. Therefore, in expectation, if we add s Xor constraints of length $n/2$ for a formula with 2^s solutions, we will have a SAT instance with exactly one solution. Then one can use any SAT solver like minisat to generate the *surviving solution*. Repeating this process M times yields the required near uniform solution samples. In practice, however, adding Xor constraints of length $n/2$ i.e. large Xors is infeasible. Large Xor constraints cause poor constraint propagation making the underlying SAT solver quite inefficient. Therefore Gomes et al. [63] consider the use of small Xors. Small Xors, however, result in an algorithm of not as high a quality as with large Xors in that in most iterations, no solutions would survive after adding s constraints. Gomes et al. [63] suggest to overcome this issue by adding fewer than s small Xor constraints with the hope that more than one solution survives and then uniformly sampling the surviving solutions (if any) by using an exact sampling algorithm. In this scheme, assuming that the new formula obtained after adding $< s$ (short) Xor constraints has mc surviving solutions, the algorithm would output the i^{th} surviving solution by choosing i uniformly from $\{1, 2, \ldots, mc\}$.

The implementation available from the authors[3] is not a stand-alone implementation of XorSample. What the authors provide is a script that takes two arguments: (a) the number s of Xors used and (b) the size k of Xors as input and generates s random Xors of size k. We implemented XorSample on top of this implementation as follows. Note that s and k are problem dependent which were determined as follows. For each problem, we first fix $k = n/2$ and search for s by iteratively increasing it until the following two conditions are satisfied. Let F' be the formula obtained by adding s Xor constraints of size k to the

[3]http://www.cs.cornell.edu/∼sabhar/

Algorithm 19: XorSample

```
Input: A SAT formula $F$, Integers $k$ (Size of Xors) and $s$ (number of Xors)
Output: A solution sample $\mathbf{x}$
1 Add $s$ Xors of size $k$ to $F$ yielding a new Formula $F'$;
2 Use Cachet [119] to count the number of solutions of $F'$;
  // Let $mc$ be the number of solutions of $F'$
3 if $mc >$ 1 million then
4     Goto Step 1;
5 $i =$ random number between 1 and $mc$;
6 Output the $i^{th}$ solution of $F'$ using Relsat solution enumerator [115].;
```

original formula.

1. F' is easy enough so that solution counting techniques like Cachet [119] and Relsat [115] count its solutions in a reasonable amount of time (1hr).

2. F' has less than 1 million solutions so that the Relsat solution enumerator terminates in a reasonable amount of time. The process of enumerating solutions is more complex than counting and is quite expensive if too many solutions survive. Therefore, we restrict ourselves to one million solutions.

Given k, if no value of s satisfies the two conditions described above, we decrease k by 1 and continue. The parameters returned by our search for k and s are reported in Table 4.1. In some cases, however, no combination of s and k satisfies the two conditions mentioned above indicating that running XorSample on the particular formula is impractical (indicated by a 'X' in Table 4.1).

Once we fix k and s, we can generate a sample from a Formula F using Algorithm 19.

4.7.1 Evaluation Criteria

We evaluate the quality of various algorithms using two criteria:

1. *Accuracy:* We measure the accuracy of the solvers using the Hellinger distance [77]. Given a formula F with n variables, let $P(X_i)$ and $A(X_i)$ denote the exact and approximate marginal distribution respectively of variable X_i, then the Hellinger distance is defined as:

$$Hellinger\ distance = \frac{\sum_{i=1}^{n} \frac{1}{2} \sum_{x_i \in \mathbf{D}_i} (\sqrt{P(x_i)} - \sqrt{A(x_i)})^2}{n} \tag{4.14}$$

   The exact marginal for each variable X_i is defined as: $P(x_i) = |\mathbf{S}_{x_i}|/|\mathbf{S}|$ where $\mathbf{S}_{x_i}$ is the set of solutions of F having $X_i = x_i$ and $\mathbf{S}$ is the set of all solutions. The number of solutions of F were computed using Cachet [119]. After running various sampling algorithms, we get a set of solution samples ϕ from which we compute the approximate marginal distribution as: $A(x_i) = |\phi(x_i)|/|\phi|$ where $\phi(x_i)$ is the subset of solutions in ϕ having $X_i = x_i$. We chose Hellinger distance because as pointed out in [77], it is superior to other choices such as the Kullback-Leibler (KL) distance, mean squared error and relative error when the marginal probabilities are close to zero. We do not use KL distance because it lies between 0 and ∞ and in practice it may equal ∞. For example, if $P(x_i) > 0$ and x_i is not sampled, $A(x_i)$ would be zero and the KL distance would be infinite. Hellinger distance always lies between 0 and 1 and always yields a lower bound on the KL distance. We did compute the error using other distance measures like the mean squared error and the relative error. All error measures show similar trends, with Hellinger distance being the most discriminative.

2. *Throughput:* Throughput is defined as the number of random solutions generated per second.

Note that accuracy is the primary performance measure because our target is to sample solutions uniformly from the SAT formula and not specifically generating solutions quickly. Throughput only serves to show whether a particular scheme is practical and scalable as

Instance	n	k	SampleSat	XorSample	SampleSearch	SIRwr	SIRwor	MH
Circuit								
ssa7552-158.cnf	1363	3034	1.14E-01	2.61E-01	8.41E-02	1.25E-03	4.00E-03	**1.07E-03**
ssa7552-159.cnf	1363	3032	9.57E-02	1.00E+00	5.88E-02	1.40E-03	3.46E-03	**1.09E-03**
2bitcomp_5.cnf	125	310	2.65E-02	3.51E-01	1.02E-01	**2.09E-03**	3.42E-02	2.54E-03
2bitmax_6.cnf	252	766	**3.11E-02**	2.57E-01	8.10E-02	5.63E-02	5.99E-02	4.31E-02
Coloring								
gcoloring-100.cnf	300	1417	2.96E-03	1.50E-01	6.98E-02	9.21E-04	1.94E-03	**4.78E-04**
gcoloring-200.cnf	600	2237	7.37E-03	1.03E-01	6.59E-02	**7.21E-04**	7.32E-03	8.40E-04
Logistics								
log-1.cnf	939	3785	3.61E-02	5.93E-01	1.64E-02	**8.75E-04**	4.28E-03	1.02E-03
log-2.cnf	1337	24777	1.52E-01	8.60E-02	1.06E-01	1.65E-02	7.06E-02	**1.33E-02**
log-3.cnf	1413	29487	1.33E-01	7.76E-02	1.04E-01	**2.23E-03**	5.28E-02	2.59E-03
log-4.cnf	2303	20963	2.29E-01	1.39E-01	2.39E-01	8.07E-02	2.22E-01	**6.92E-02**
log-5.cnf	2701	29534	1.48E-01	**7.54E-02**	1.24E-01	2.20E-01	9.90E-02	1.09E-01
Pebbling								
sat-grid-pbl-0010.cnf	110	191	8.38E-02	1.00E+00	1.60E-02	1.16E-03	1.77E-03	**9.04E-04**
sat-grid-pbl-0015.cnf	240	436	9.22E-02	1.00E+00	4.23E-02	**3.75E-03**	9.09E-03	5.04E-03
sat-grid-pbl-0020.cnf	420	781	1.12E-01	1.00E+00	6.97E-02	**2.09E-02**	2.22E-02	2.56E-02
sat-grid-pbl-0025.cnf	650	1226	1.00E-01	1.00E+00	7.16E-02	7.73E-02	**4.33E-02**	8.20E-02
sat-grid-pbl-0030.cnf	930	1771	1.12E-01	1.00E+00	5.34E-02	1.68E-01	**3.13E-02**	7.72E-02

Table 4.2: Table showing Hellinger distance of SampleSat, XorSample, SampleSearch, SIRwR, SIRwoR and MH after 10 hrs of CPU time.

well as to quantify time vs accuracy tradeoffs.

In our evaluation, we had to use SAT instances whose solutions can be counted in a relatively small amount of time because to compute the Hellinger distance (see Equation 4.14) we have to count solutions to $n + 1$ SAT problems for each formula having n variables. Therefore, even though we are able to generate solution samples from large SAT instances, we are not able to evaluate the accuracy of our schemes on these instances. In general, the time required to generate a solution sample via our SampleSearch strategy is roughly the same as the time required by state-of-the-art solvers like minisat [125] and RSAT [108].

4.7.2 Results for $p = 0$

Table 4.2 summarizes the results of running each algorithm for exactly 10 hrs per problem instance on these benchmarks. Columns 1 contains the instance name, columns 2 and

Problem	#Var	#Cl	SampleSat	XorSample	SampleSearch
Circuit					
ssa-158	1363	3034	6.67E+03	1.51E-01	1.67E+04
ssa-159	1363	3032	8.89E+03	0.00E+00	1.67E+04
2bitmax_6	125	310	8.06E+04	4.10E+00	1.00E+05
2bitcomp_5	252	766	6.94E+04	8.50E+00	1.00E+05
Coloring					
gcoloring-100	300	1117	3.89E+03	1.98E+00	1.42E+04
gcoloring-200	600	2237	1.36E+03	5.68E-01	1.42E+03
Logistics					
log-1	939	3785	1.25E+04	9.44E-01	2.00E+04
log-2	1337	24777	1.39E+02	5.36E-01	3.06E+03
log-3	1413	29487	9.72E+00	3.15E-01	2.98E+03
log-4	2303	20963	9.56E+00	4.14E-02	2.95E+03
log-5	2701	29534	9.10E-01	2.50E-03	2.22E+03
Pebbling					
grid-10	110	191	5.00E+04	36	3.06E+03
grid-15	240	436	3.33E+04	0.00E+00	8.33E+03
grid-20	420	781	1.25E+04	0.00E+00	5.56E+03
grid-25	650	1226	3.33E+04	0.00E+00	3.33E+03
grid-30	930	1771	8.33E+03	0.00E+00	2.58E+02

Table 4.3: Table showing the number of samples generated per second by SampleSat, XorSample and SampleSearch.

3 report the number of variables and clauses for the instance, columns 4 to 9 report the Hellinger distance after 10 hrs per instance for the solvers. Table 4.3 shows the number of solution samples generated per second by SampleSearch, XorSample and SampleSat. Note that the number of samples of SIR and MH schemes equals 10% of that of SampleSearch and are therefore not shown in Table 4.3.

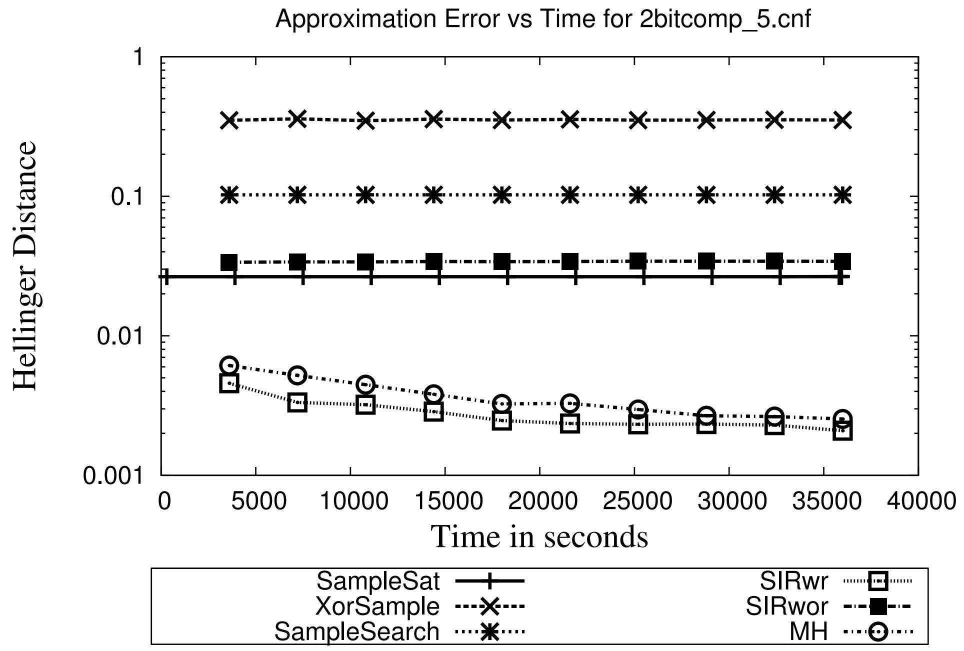
From Tables 4.2 and 4.3, we can make the following observations. First, on most instances the Hellinger distance of all SIR and MH schemes is less than pure SampleSearch, SampleSat and XorSample. Second, on most instances, SampleSearch has lower Hellinger distance and generates more samples than SampleSat.

Results on the Circuit instances

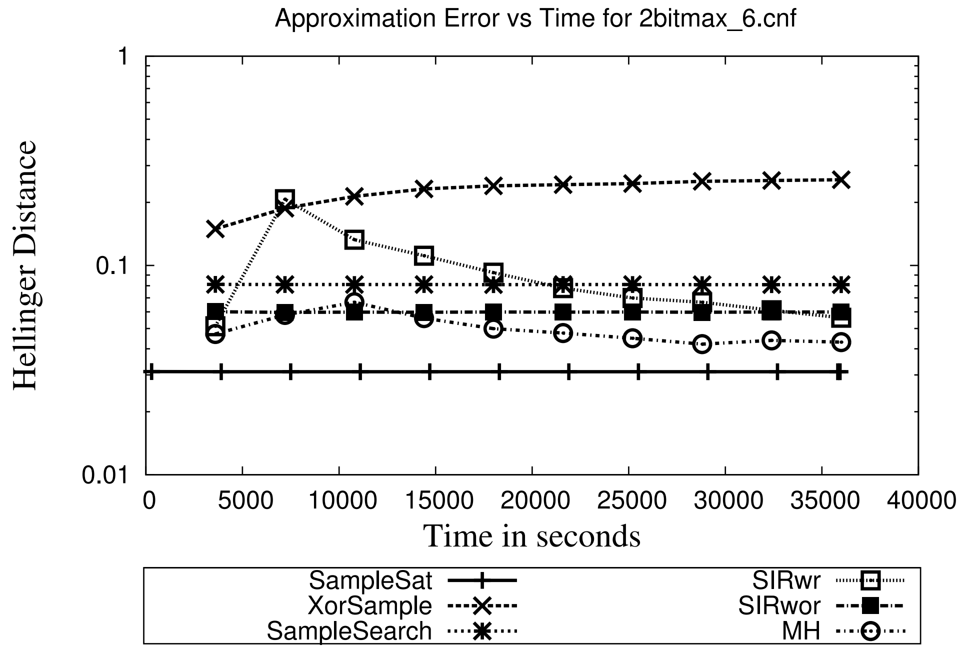
Time versus Hellinger distance plots for the circuit instances are shown in Figures 4.3(a) and (b) and Figures 4.4(a) and (b). We observe that MH and SIR with replacement are the best performing schemes on 3 instances while SampleSat performs better on the 2bitmax_6 instance. SampleSearch is more accurate than SampleSat and XorSample on the two ssa-7552-* instances while SampleSat is more accurate than SampleSearch on 2bitcomp_5 and 2bitmax_6 instances.

Results on Graph Coloring instances

Figure 4.5 shows the anytime performance of the solvers over two graph coloring instances. MH and SIR with replacement are the best performing schemes. On the instance having 100 vertices (see Figure 4.5(a)), MH and SIR with and without replacement have better sampling error than pure SampleSearch, XorSample and SampleSat. On the instance having 200 vertices (see Figure 4.5(b)), we see that SampleSat is slightly better than SIR without replacement but is worse than both SIR with replacement and MH. Pure Sample-

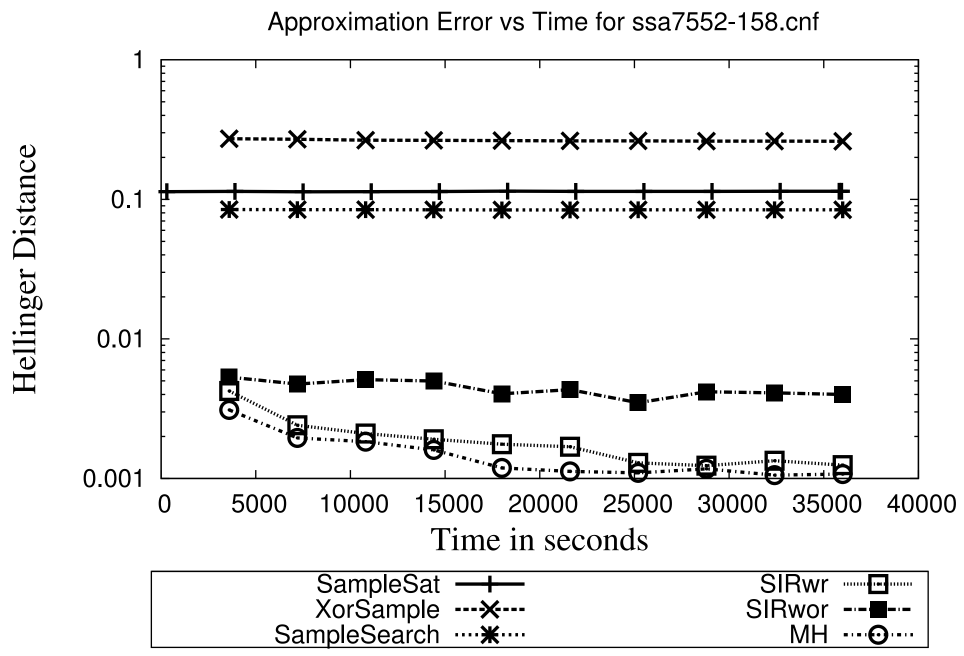


(a)

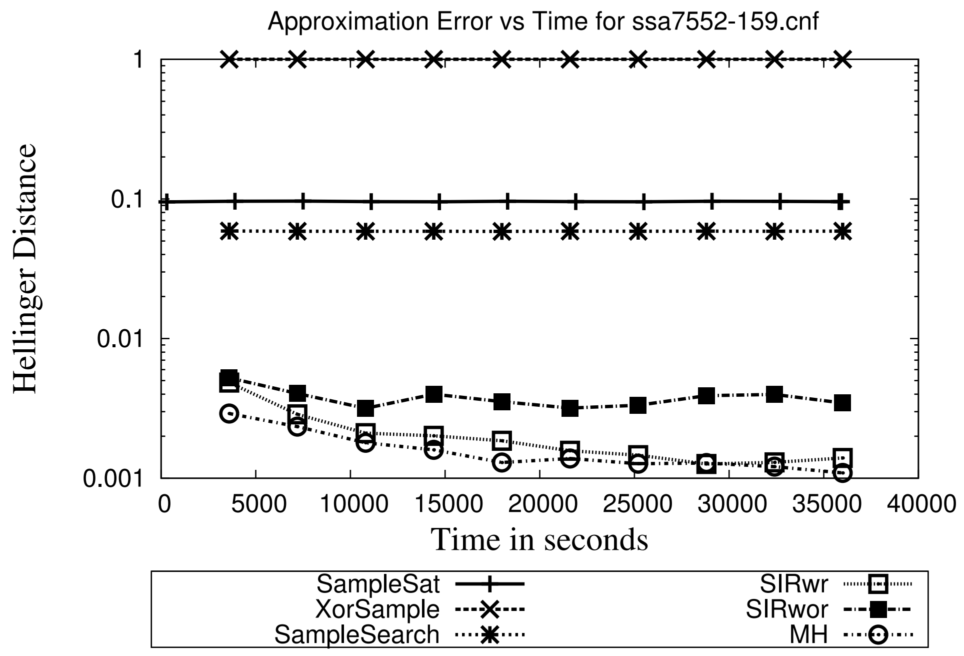


(b)

Figure 4.3: Time versus Hellinger distance plots for Circuit instances 2bitmax_6 and 2bit-comp_5.

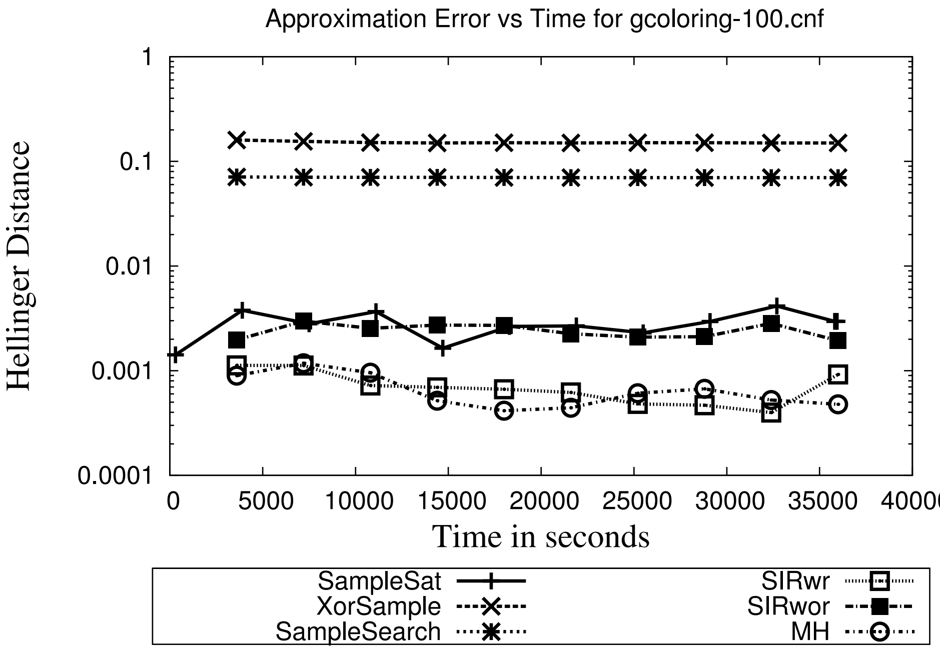


(a)

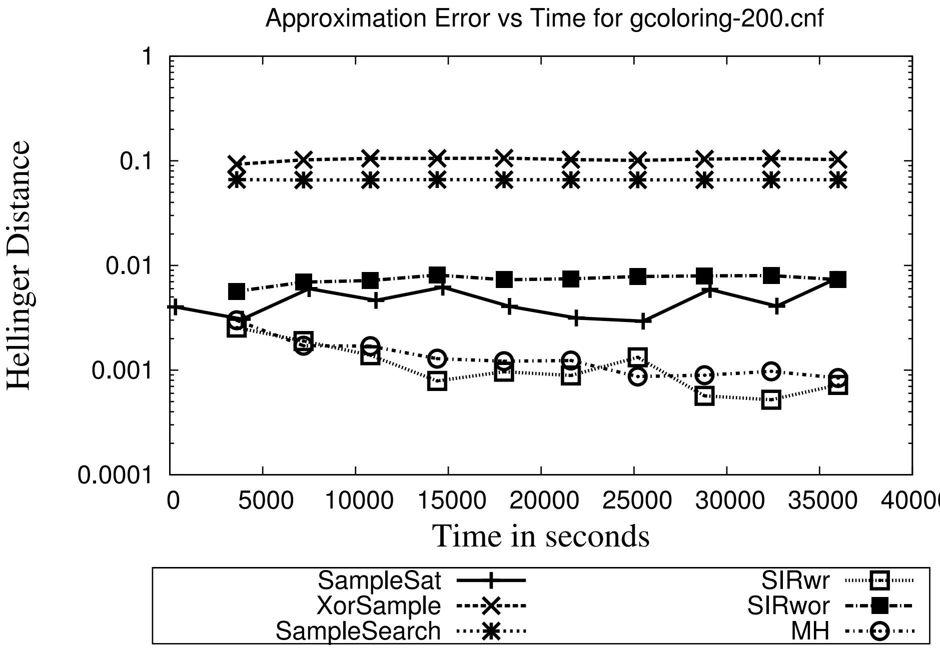


(b)

Figure 4.4: Time versus Hellinger distance plots for Circuit instances ssa7552-158 and ssa7552-159 instances.



(a)



(b)

Figure 4.5: Time versus Hellinger distance plots for two 3-coloring instances with 100 and 200 vertices respectively.

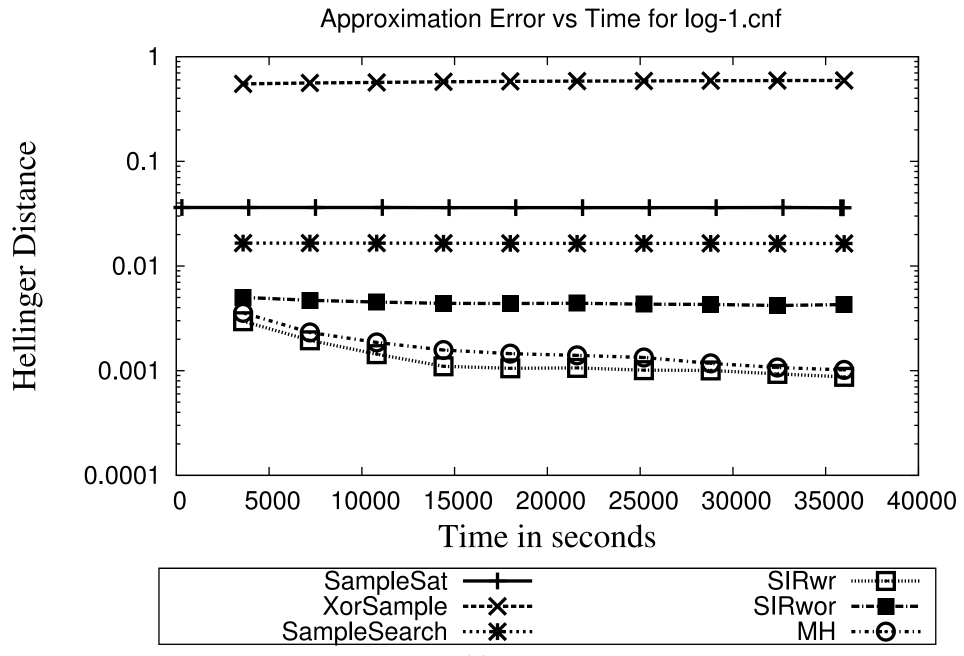
Search is the worst performing scheme. Note that because finding a solution is relatively easy on these instances, poor performance of SampleSearch does not translate down to SIR and MH schemes because it is offset by the relatively large sample size.

Results on Logistics planning instances

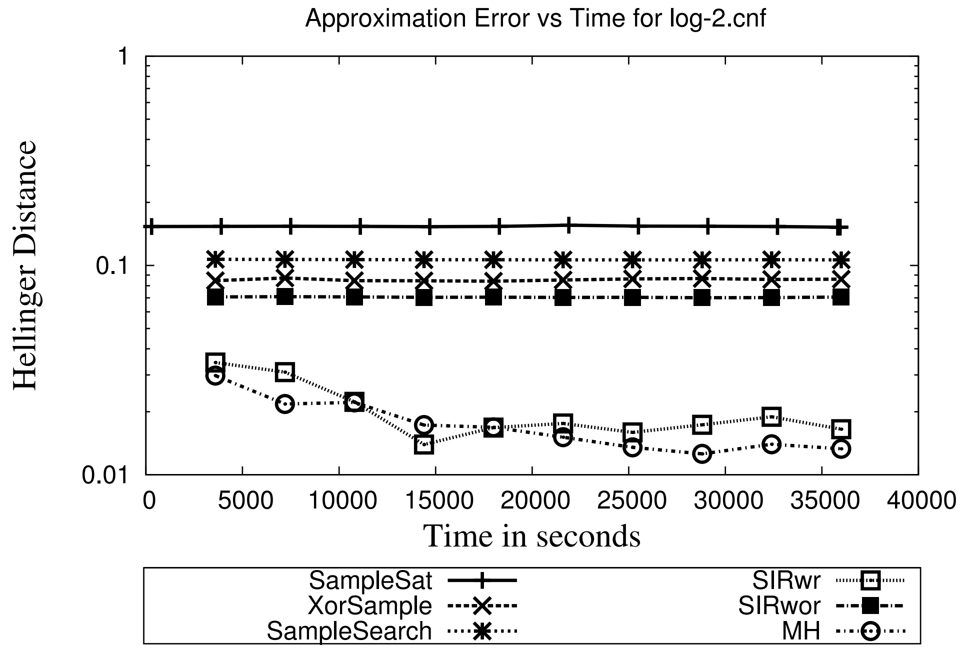
Time versus Hellinger distance plots for the logistics instances are shown in Figures 4.6, 4.7 and 4.8. SIR with replacement and MH are more accurate than other schemes on log-1, log-2 and log-3 and log-4 instances but on the log-5 instance, they are worse than pure SampleSearch. On the log-5 instance, SampleSearch-SIR without replacement is the better than SampleSearch. XorSample has the best accuracy for log-5 instance while on other instances it is worse than MH and SIR schemes. SampleSat is worse than SampleSearch on 3 instances.

Results on Grid Pebbling instances

Figures 4.9, 4.10 and 4.11 show a plot of time versus Hellinger distance on the grid pebbling instances. On the smaller Grid Pebbling benchmarks of size 10, 15 and 20, we observe that SIR with replacement and MH are the best performing schemes while on the larger instances of size 25 and 30, SIR without replacement performs better than other competing techniques. XorSample is able to generate solution samples on only the smallest pebbling instance of size 10. On this instance, it not only generates far less samples as compared with other competing schemes but also has lower accuracy. SampleSearch is slightly more accurate than SampleSat on 4 out of 5 instances.

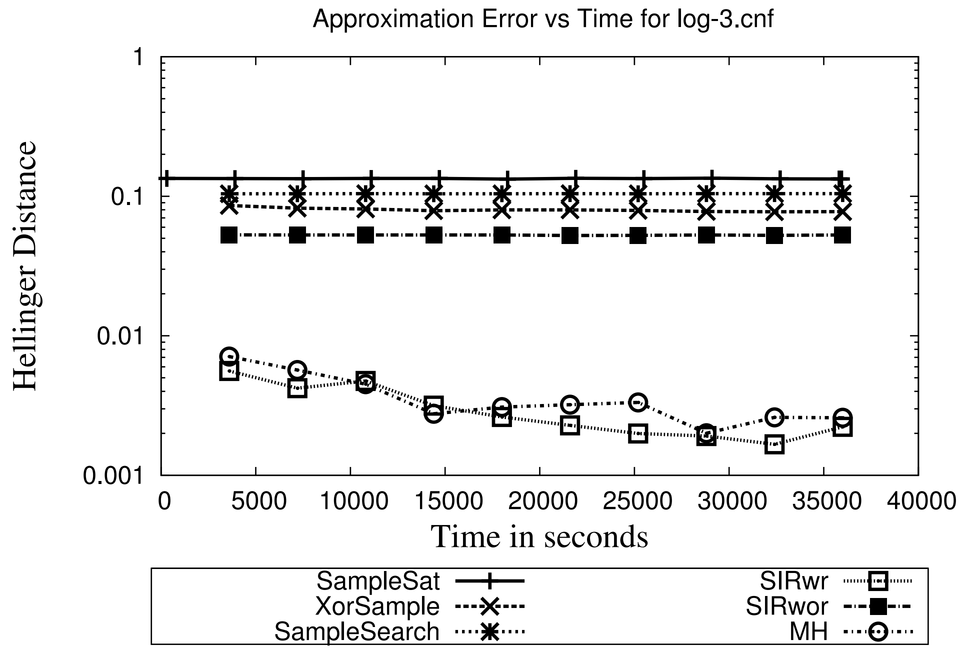


(a)

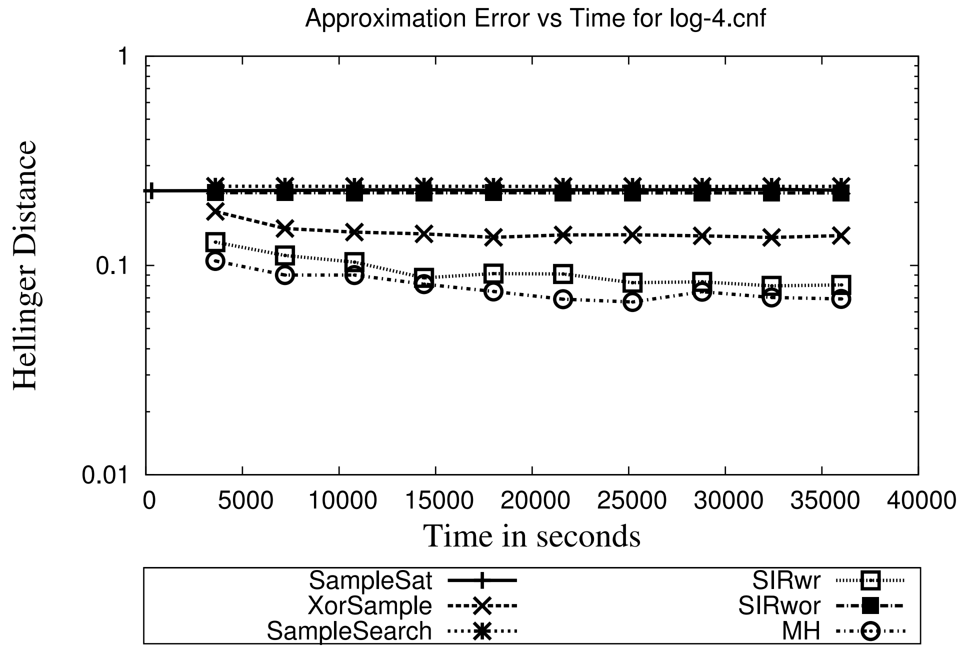


(b)

Figure 4.6: Time versus Hellinger distance plots for Logistics planning instances log-1 and log-2.

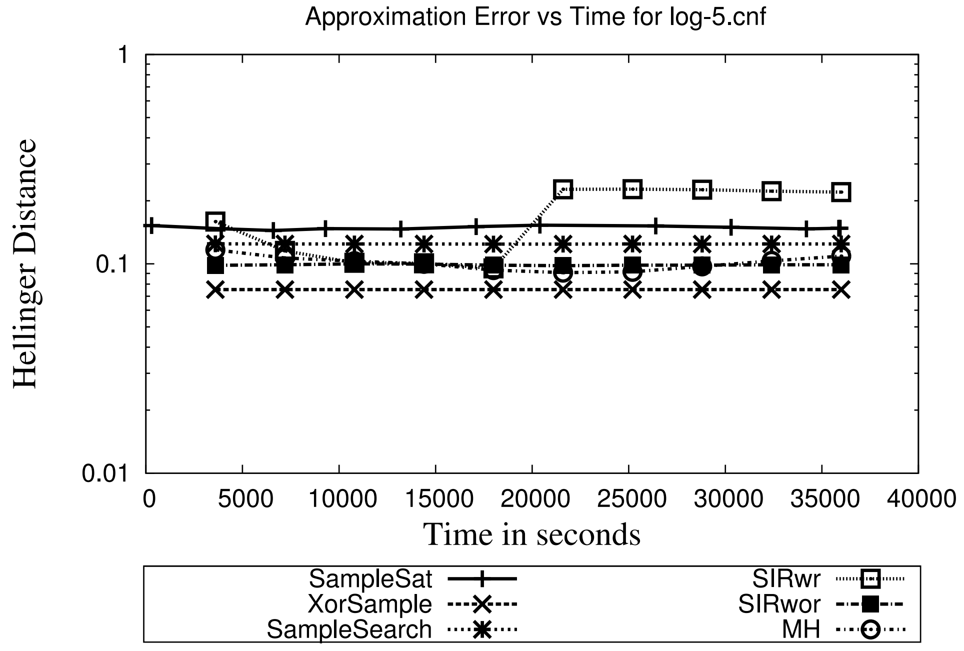


(a)



(b)

Figure 4.7: Time versus Hellinger distance plots for Logistics planning instances log-3 and log-4.



(a)

Figure 4.8: Time versus Hellinger distance plot for a Logistics planning instance log-5.

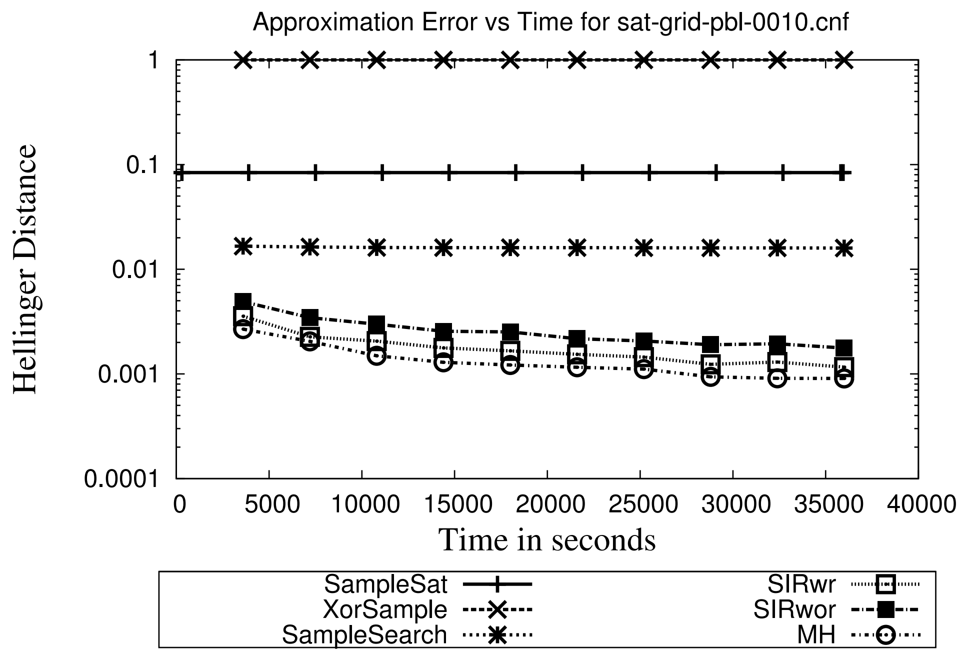
4.7.3 Predicting which sampling scheme to use

In this subsection, we discuss a simple statistical test based on co-efficient of variation [88] to predict which SampleSearch based sampling scheme would be better. Co-efficient of variation measures the dispersion of the probability distribution and can be used to estimate the variance of the normalized weights. Given a weight vector $(w_1, \ldots, w_N)$ for the samples generated by SampleSearch, the co-efficient of variation is defined as:

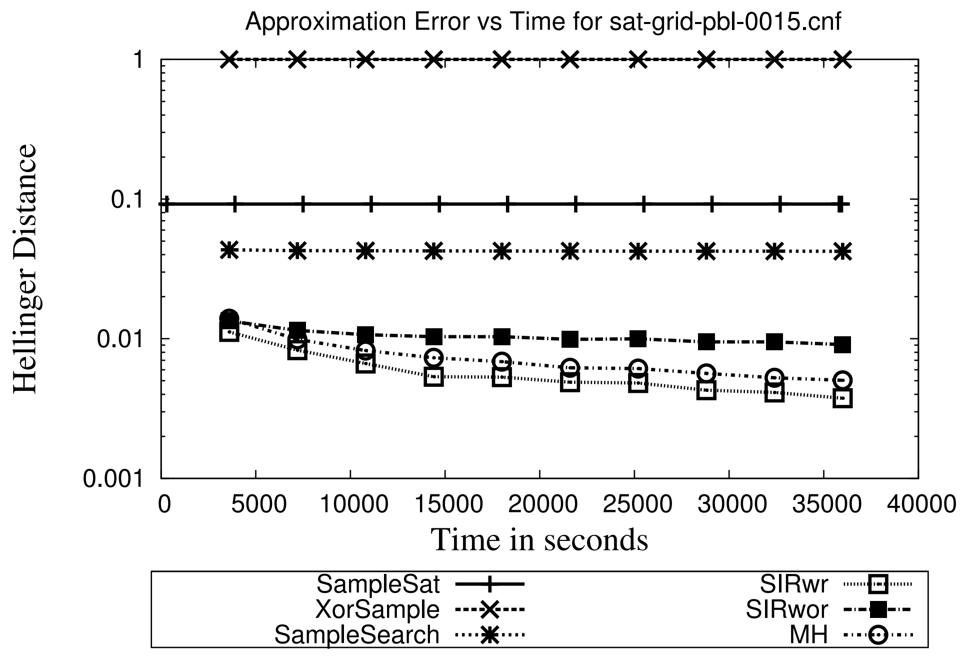
$$cv^2(w) = \frac{\sum_{i=1}^{N}(w^i - \overline{w})^2}{(N-1)\overline{w}^2} \tag{4.15}$$

where $\overline{w}$ is the average of the weight vector.

cv^2 measures how the samples output by SampleSearch compare to those drawn from the uniform distribution over the solutions. When it is large, it indicates that the samples are not as accurate yielding high variance while when small, it indicates good accuracy.

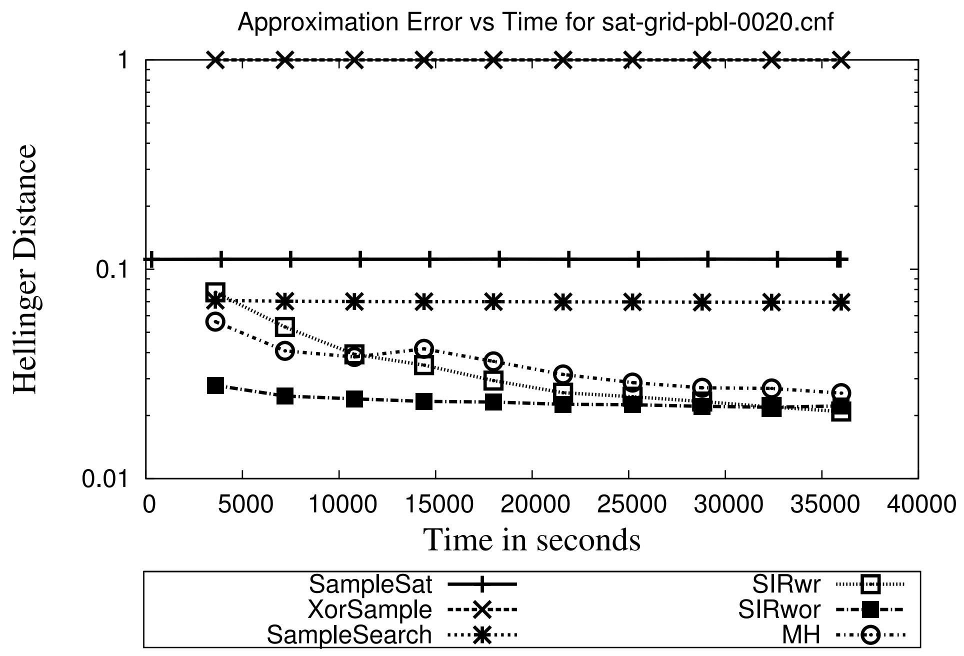


(a)

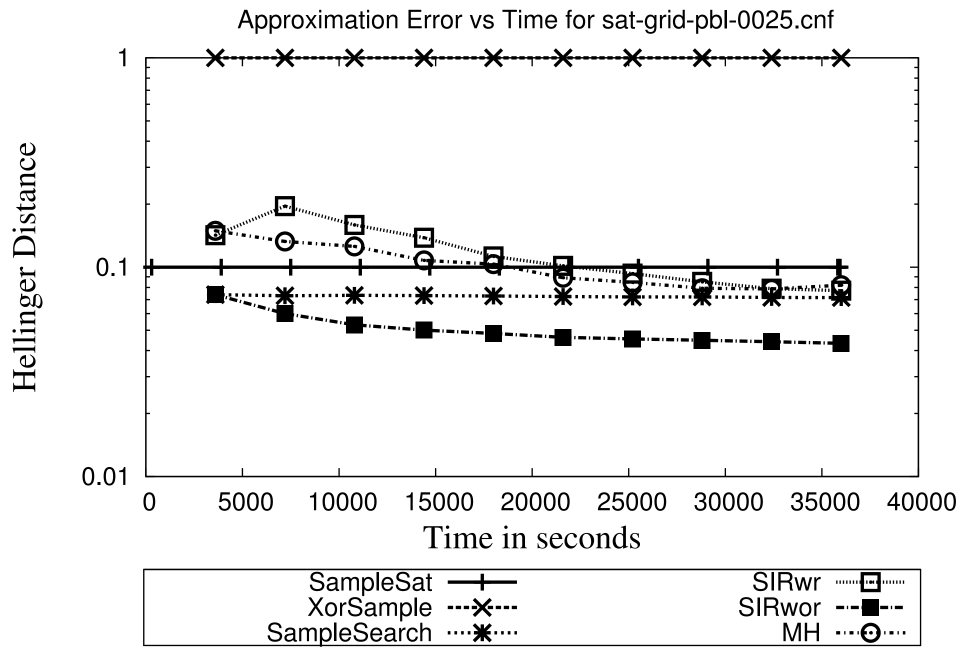


(b)

Figure 4.9: Time versus Hellinger distance plot for Grid pebbling instances of size 10 and 15.

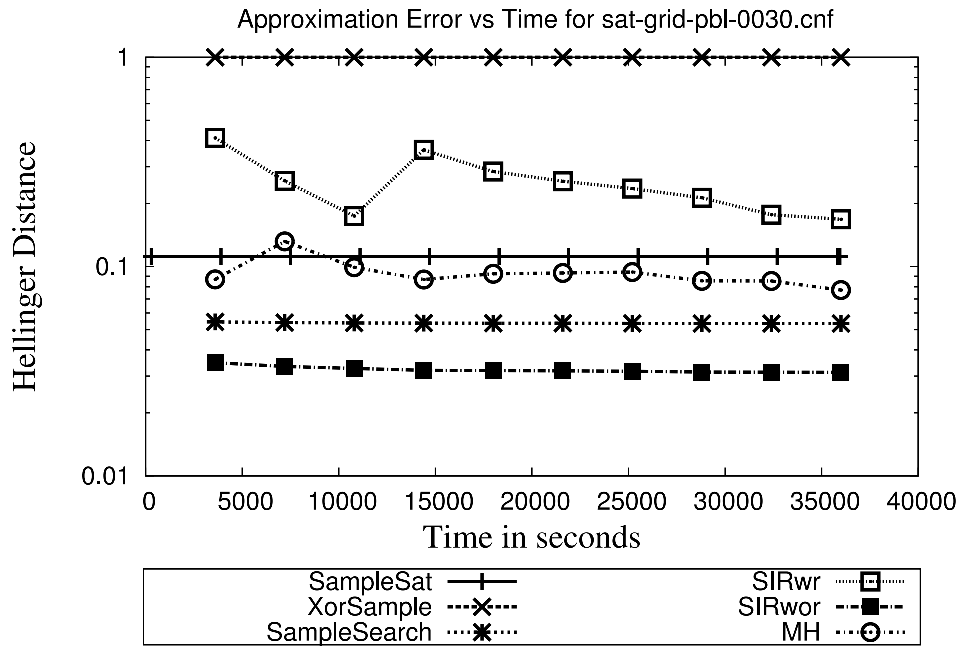


(a)



(b)

Figure 4.10: Time versus Hellinger distance plot for Grid pebbling instances of size 20 and 25



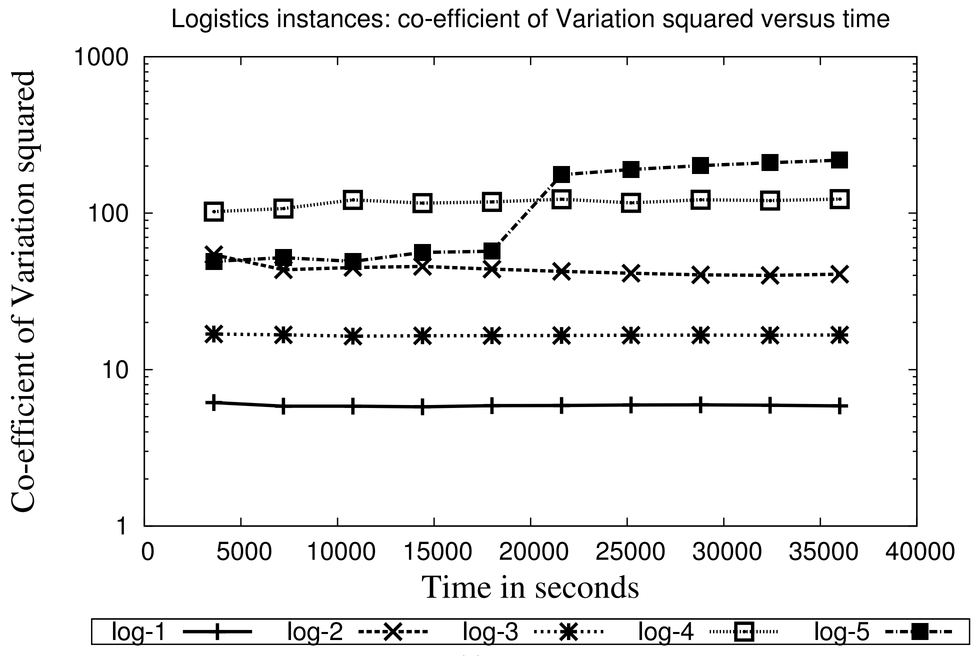
(a)

Figure 4.11: Time versus Hellinger distance plot for Grid pebbling instance of size 30

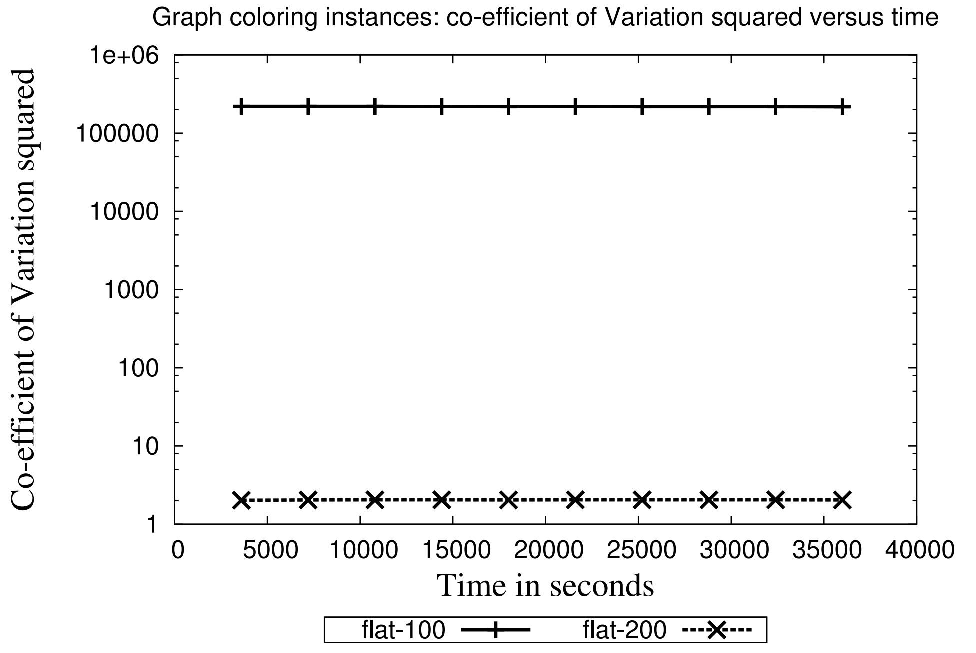
Figure 4.12 and 4.13 show the change in cv^2 of SampleSearch's weights with time for each benchmark category. We find that the performance of SIR and MH schemes is quite correlated with the co-efficient of variation. When the co-efficient of variation is greater than 10000, we find that SIR without replacement is more accurate than SIR with replacement and MH schemes while when it is below 10000, SIR with replacement and MH are more accurate than SIR without replacement. For example, in case of log-5 and grid-pbl-0030 instances, the co-efficient of variation is greater than 10000 and SIR without replacement is better than SIR with replacement (see Figures 4.12(a) and 4.8).

4.7.4 Results for $p > 0$

In this subsection, we study the impact of p on the accuracy of SampleSearch, MH and SIR methods.

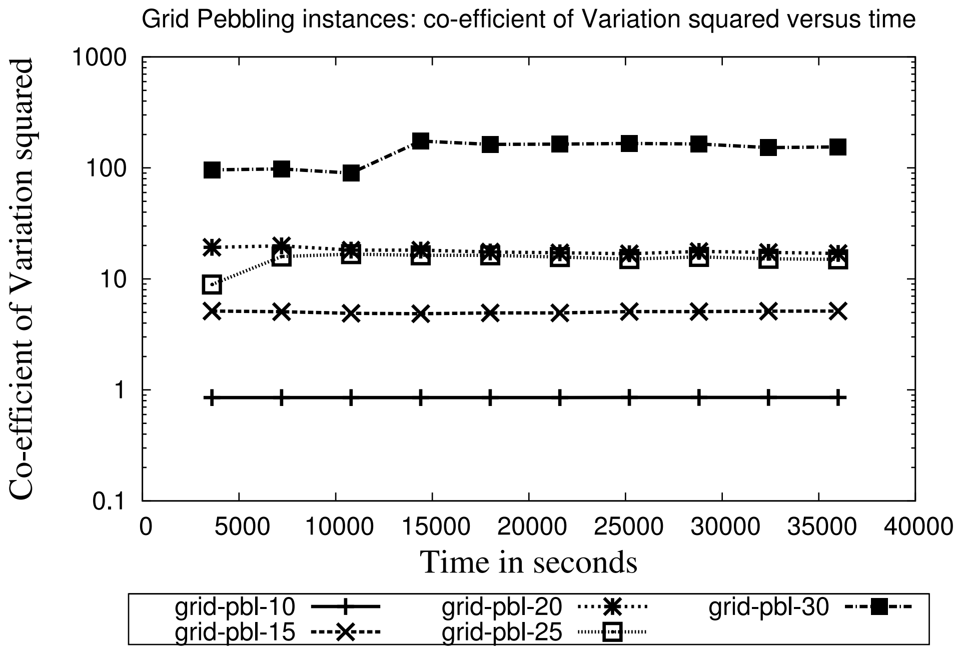


(a)

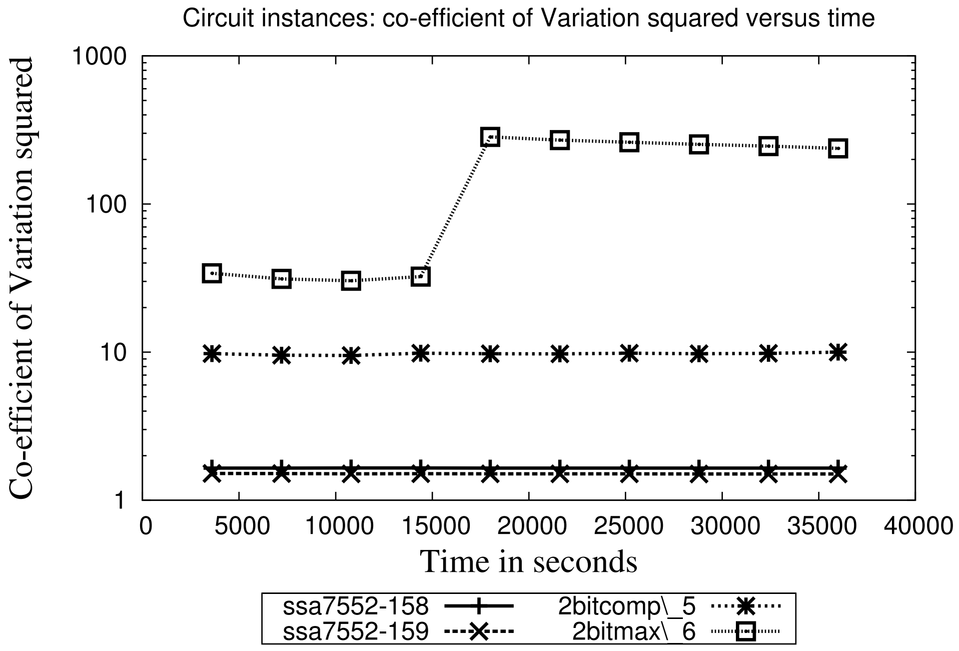


(b)

Figure 4.12: Time versus co-efficient of variation of SampleSearch for the Logistics and 3-coloring benchmarks.



(a)



(b)

Figure 4.13: Time versus co-efficient of variation of SampleSearch for the Grid pebbling and Circuit benchmarks.

			SampleSearch				
Problem	**#Var**	**#Cl**	**P=0**	**P=10**	**P=20**	**P=50**	**P=100**
Circuit							
ssa-158	1363	3034	16667	1.50	0.94	0.40	0.22
ssa-159	1363	3032	16667	1.39	0.86	0.33	0.17
2bitmax_6	125	310	100000	75.94	34.23	22.31	10.36
2bitcomp_5	252	766	100000	9.06	5.70	2.31	1.11
Coloring							
gcoloring-100	300	1117	14167	4.94	3.22	1.08	0.56
gcoloring-200	600	2237	1417	1.39	0.75	0.31	0.16
Logistics							
log-1	939	3785	20000	7.33	4.98	1.92	1.11
log-2	1337	24777	3060	0.22	0.14	0.06	0.00
log-3	1413	29487	2980	0.50	0.31	0.13	0.06
log-4	2303	20963	2950	0.86	0.47	0.14	0.06
log-5	2701	29534	2220	0.08	0.00	0.00	0.00
Pebbling							
grid-10	110	191	3060	81.14	81.00	81.08	52.83
grid-15	240	436	8330	11.75	7.22	3.33	2.04
grid-20	420	781	5560	6.33	1.44	0.75	0.36
grid-25	650	1226	3330	1.08	0.67	0.25	0.11
grid-30	930	1771	258	0.17	0.08	0.00	0.00

Table 4.4: Table showing the number of solutions output per second by SampleSearch as a function of p.

Instance	#Var	#Cl	SampleSearch				
			p=0	p=10	p=20	p=50	p=100
Circuit							
ssa7552-158.cnf	1363	3034	**8.41E-02**	1.01E-01	1.01E-01	1.02E-01	1.06E-01
ssa7552-159.cnf	1363	3032	**5.88E-02**	8.86E-02	8.92E-02	8.96E-02	8.61E-02
2bitcomp_5.cnf	125	310	1.02E-01	**9.34E-02**	9.34E-02	9.38E-02	9.39E-02
2bitmax_6.cnf	252	766	8.10E-02	7.54E-02	7.60E-02	7.51E-02	**7.43E-02**
Coloring							
gcoloring-100.cnf	300	1417	6.98E-02	5.35E-02	5.37E-02	5.31E-02	**5.11E-02**
gcoloring-200.cnf	600	2237	6.59E-02	4.56E-02	4.80E-02	5.39E-02	**4.37E-02**
Logistics							
log-1.cnf	939	3785	**1.64E-02**	2.39E-02	2.40E-02	2.36E-02	2.35E-02
log-2.cnf	1337	24777	1.06E-01	**9.02E-02**	1.00E-01	9.76E-02	1.08E-01
log-3.cnf	1413	29487	1.04E-01	9.88E-02	9.95E-02	**9.49E-02**	9.95E-02
log-4.cnf	2303	20963	2.39E-01	1.73E-01	**1.71E-01**	1.77E-01	1.71E-01
log-5.cnf	2701	29534	1.24E-01	**1.05E-01**	1.13E-01	X	X
Pebbling							
sat-grid-pbl-0010.cnf	110	191	1.60E-02	5.24E-03	5.24E-03	**5.24E-03**	5.41E-03
sat-grid-pbl-0015.cnf	240	436	4.23E-02	**2.08E-02**	2.11E-02	2.21E-02	2.26E-02
sat-grid-pbl-0020.cnf	420	781	6.97E-02	**4.00E-02**	4.22E-02	4.36E-02	4.50E-02
sat-grid-pbl-0025.cnf	650	1226	7.16E-02	**4.60E-02**	4.69E-02	5.06E-02	5.43E-02
sat-grid-pbl-0030.cnf	930	1771	**5.34E-02**	5.76E-02	5.84E-02	6.37E-02	6.83E-02

Table 4.5: Table showing the Hellinger distance of SampleSearch as a function of p after 10 hrs of CPU time.

Instance	#Var	#Cl	SIRwr				
			p=0	p=10	p=20	p=50	p=100
Circuit							
ssa7552-158.cnf	1363	3034	**1.25E-03**	2.72E-02	3.11E-02	4.11E-02	6.18E-02
ssa7552-159.cnf	1363	3032	**1.40E-03**	5.32E-02	5.85E-02	7.78E-02	1.01E-01
2bitcomp_5.cnf	125	310	**2.09E-03**	1.40E-02	1.45E-02	1.58E-02	1.90E-02
2bitmax_6.cnf	252	766	**5.63E-02**	1.37E-01	2.30E-01	3.58E-01	1.13E-01
Coloring							
gcoloring-100.cnf	300	1417	**9.21E-04**	7.81E-03	5.23E-03	1.49E-02	1.92E-02
gcoloring-200.cnf	600	2237	**7.21E-04**	1.73E-02	1.97E-02	1.78E-02	3.75E-02
Logistics							
log-1.cnf	939	3785	**8.75E-04**	6.05E-03	9.62E-03	8.57E-03	1.89E-02
log-2.cnf	1337	24777	**1.65E-02**	7.72E-02	1.21E-01	1.57E-01	1.66E-02
log-3.cnf	1413	29487	**2.23E-03**	4.61E-02	5.66E-02	8.10E-02	1.09E-01
log-4.cnf	2303	20963	**8.07E-02**	1.34E-01	1.52E-01	1.49E-01	1.72E-01
log-5.cnf	2701	29534	2.20E-01	2.25E-01	**2.80E-02**	X	X
Pebbling							
sat-grid-pbl-0010.cnf	110	191	**1.16E-03**	2.10E-03	2.07E-03	2.14E-03	2.50E-03
sat-grid-pbl-0015.cnf	240	436	**3.75E-03**	6.73E-03	8.81E-03	1.23E-02	1.52E-02
sat-grid-pbl-0020.cnf	420	781	**2.09E-02**	3.50E-02	3.95E-02	1.41E-01	7.44E-02
sat-grid-pbl-0025.cnf	650	1226	7.73E-02	2.43E-01	8.62E-02	**7.47E-02**	2.13E-01
sat-grid-pbl-0030.cnf	930	1771	**1.68E-01**	1.83E-01	2.24E-01	4.70E-01	2.86E-01

Table 4.6: Table showing the Hellinger distance of SampleSearch-SIR with replacement as a function of p after 10 hrs of CPU time.

Tables 4.4 and 4.5 show how increasing p affects the throughput and accuracy of SampleSearch respectively. Figure 4.14 shows how the accuracy varies with time for two sample instances. As expected, on an average the accuracy increases slightly with p but throughput decreases significantly. Low values of p (≤ 20) pay off more than high values of p in that accuracy increases at a higher rate when we move from $p = 0$ to $p = 20$ and remains mostly constant there after while throughput reduces substantially.

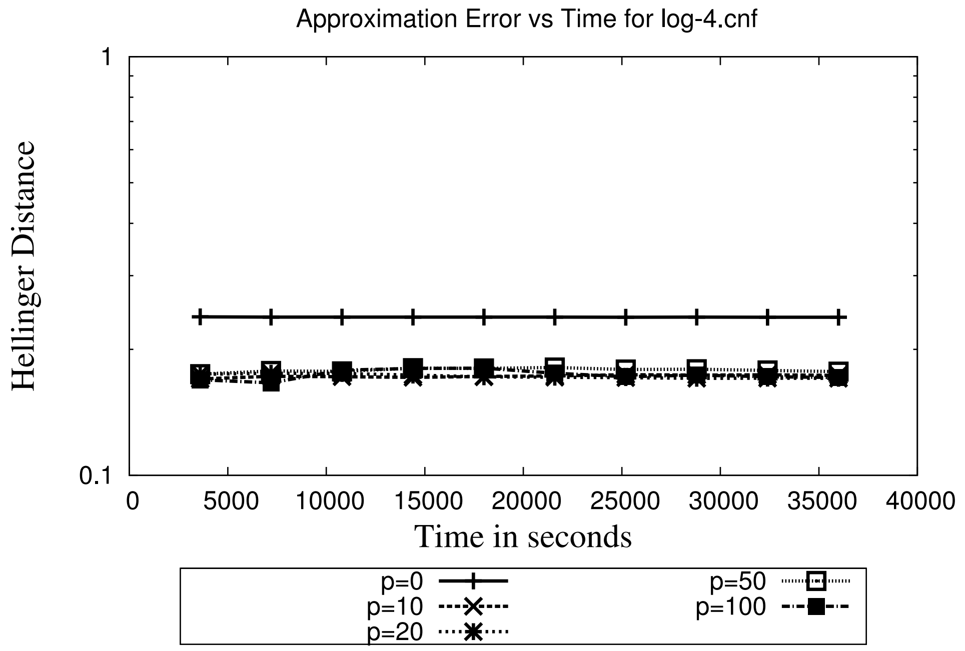
Tables 4.6, 4.7 and 4.8 show how increasing p affects the accuracy of SIR with replacement, SIR without replacement and MH respectively. Figures 4.15, 4.16 and 4.17 show how the accuracy varies with time for the SIR and MH schemes for two sample instances. We see that accuracy does not improve with p. This is because the accuracy of MH and SIR is dependent not only on the distance between the backtrack-free distribution and the uniform

			SIRwor				
Instance	**#Var**	**#Cl**	**p=0**	**p=10**	**p=20**	**p=50**	**p=100**
Circuit							
ssa7552-158.cnf	1363	3034	**4.00E-03**	3.75E-02	4.33E-02	5.26E-02	6.26E-02
ssa7552-159.cnf	1363	3032	**3.46E-03**	3.52E-02	4.39E-02	4.28E-02	6.20E-02
2bitcomp_5.cnf	125	310	3.42E-02	**1.82E-02**	1.82E-02	2.07E-02	2.05E-02
2bitmax_6.cnf	252	766	5.99E-02	4.53E-02	4.60E-02	**4.36E-02**	5.05E-02
Coloring							
gcoloring-100.cnf	300	1417	**1.94E-03**	6.19E-03	9.97E-03	2.06E-02	1.67E-02
gcoloring-200.cnf	600	2237	**7.32E-03**	1.96E-02	1.98E-02	3.77E-02	4.08E-02
Logistics							
log-1.cnf	939	3785	**4.28E-03**	7.13E-03	7.50E-03	8.88E-03	1.07E-02
log-2.cnf	1337	24777	7.06E-02	**6.79E-02**	7.72E-02	8.19E-02	9.23E-02
log-3.cnf	1413	29487	5.28E-02	**4.91E-02**	5.40E-02	4.96E-02	5.65E-02
log-4.cnf	2303	20963	2.22E-01	1.43E-01	**1.31E-01**	1.57E-01	1.58E-01
log-5.cnf	2701	29534	9.90E-02	**8.99E-02**	1.01E-01	X	X
Pebbling							
sat-grid-pbl-0010.cnf	110	191	**1.77E-03**	2.61E-03	2.82E-03	2.93E-03	3.53E-03
sat-grid-pbl-0015.cnf	240	436	9.09E-03	**9.04E-03**	1.02E-02	1.57E-02	1.94E-02
sat-grid-pbl-0020.cnf	420	781	2.22E-02	**1.84E-02**	2.68E-02	3.51E-02	5.28E-02
sat-grid-pbl-0025.cnf	650	1226	4.33E-02	**3.72E-02**	4.38E-02	6.37E-02	8.57E-02
sat-grid-pbl-0030.cnf	930	1771	**3.13E-02**	5.06E-02	6.02E-02	9.62E-02	1.39E-01

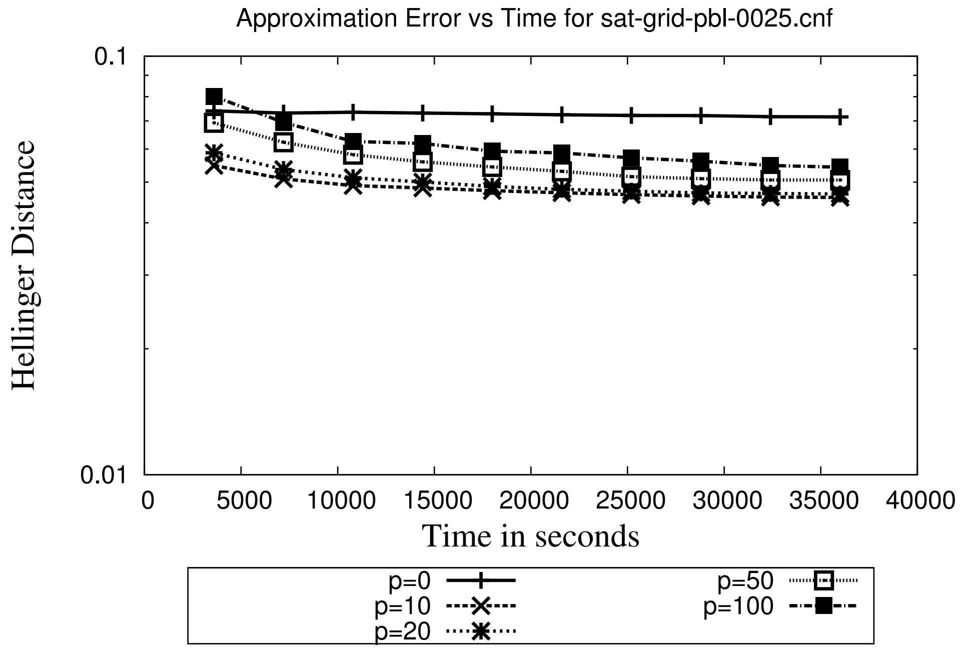
Table 4.7: Table showing the Hellinger distance of SampleSearch-SIR without replacement as a function of p after 10 hrs of CPU time.

Instance	#Var	#Cl	MH				
			p=0	**p=10**	**p=20**	**p=50**	**p=100**
Circuit							
ssa7552-158.cnf	1363	3034	**1.07E-03**	3.30E-02	3.73E-02	3.73E-02	7.78E-02
ssa7552-159.cnf	1363	3032	**1.09E-03**	3.85E-02	4.91E-02	5.41E-02	6.64E-02
2bitcomp_5.cnf	125	310	**2.54E-03**	1.38E-02	1.47E-02	1.49E-02	2.12E-02
2bitmax_6.cnf	252	766	**4.31E-02**	7.54E-02	7.29E-02	1.34E-01	8.42E-02
Coloring							
gcoloring-100.cnf	300	1417	**4.78E-04**	6.31E-03	6.68E-03	7.28E-03	1.23E-02
gcoloring-200.cnf	600	2237	**8.40E-04**	1.82E-02	1.27E-02	2.03E-02	3.37E-02
Logistics							
log-1.cnf	939	3785	**1.02E-03**	6.11E-03	1.28E-02	8.75E-03	1.57E-02
log-2.cnf	1337	24777	**1.33E-02**	7.83E-02	9.66E-02	1.28E-01	1.44E-02
log-3.cnf	1413	29487	**2.59E-03**	4.27E-02	3.65E-02	4.82E-02	7.58E-02
log-4.cnf	2303	20963	**6.92E-02**	9.17E-02	1.08E-01	1.28E-01	1.60E-01
log-5.cnf	2701	29534	1.09E-01	1.04E-01	**3.16E-02**	X	X
Pebbling							
sat-grid-pbl-0010.cnf	110	191	**9.04E-04**	1.43E-03	1.48E-03	1.51E-03	1.79E-03
sat-grid-pbl-0015.cnf	240	436	**5.04E-03**	6.41E-03	8.32E-03	1.06E-02	1.45E-02
sat-grid-pbl-0020.cnf	420	781	**2.56E-02**	3.52E-02	3.81E-02	1.01E-01	6.01E-02
sat-grid-pbl-0025.cnf	650	1226	**8.20E-02**	9.72E-02	8.63E-02	9.42E-02	1.89E-01
sat-grid-pbl-0030.cnf	930	1771	**7.72E-02**	1.24E-01	1.04E-01	1.39E-01	1.46E-01

Table 4.8: Table showing the Hellinger distance of SampleSearch-MH as a function of p after 10 hrs of CPU time.

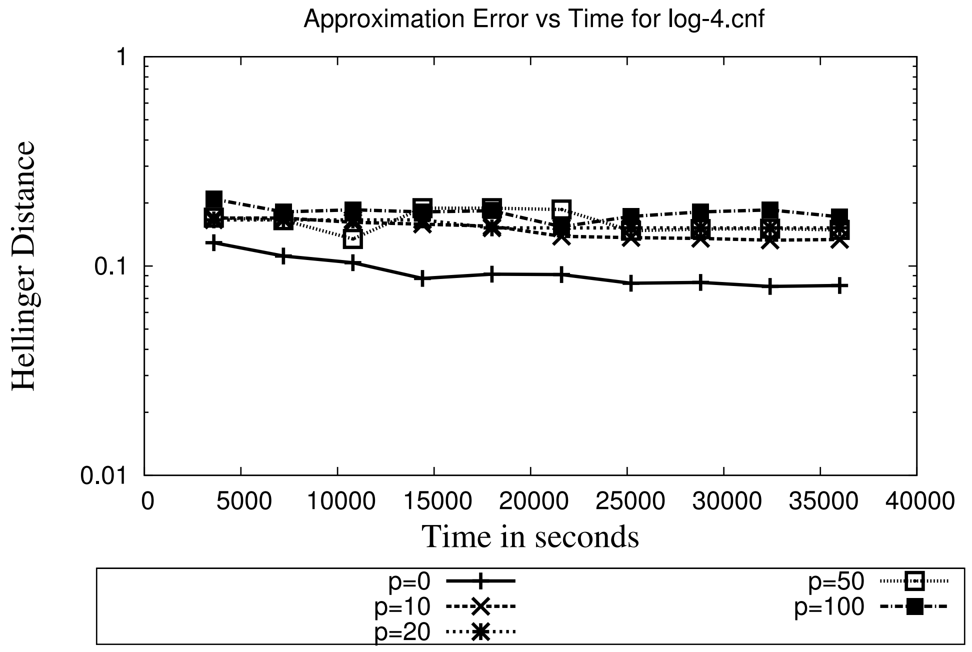


(a)

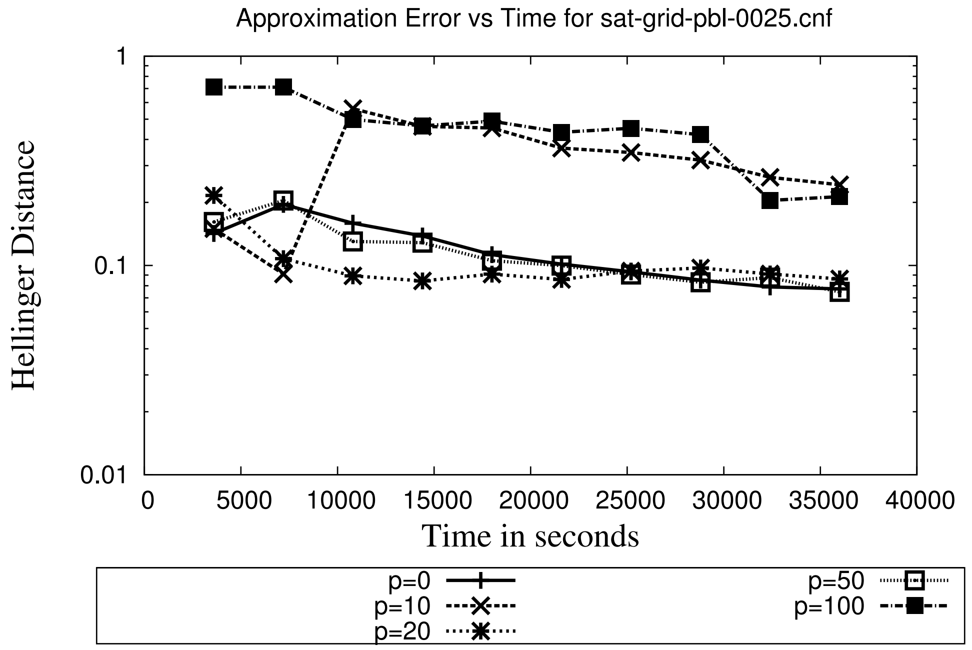


(b)

Figure 4.14: Time versus Hellinger distance plot for SampleSearch as a function of p for log-4 and Grid pebbling instance of size 25.



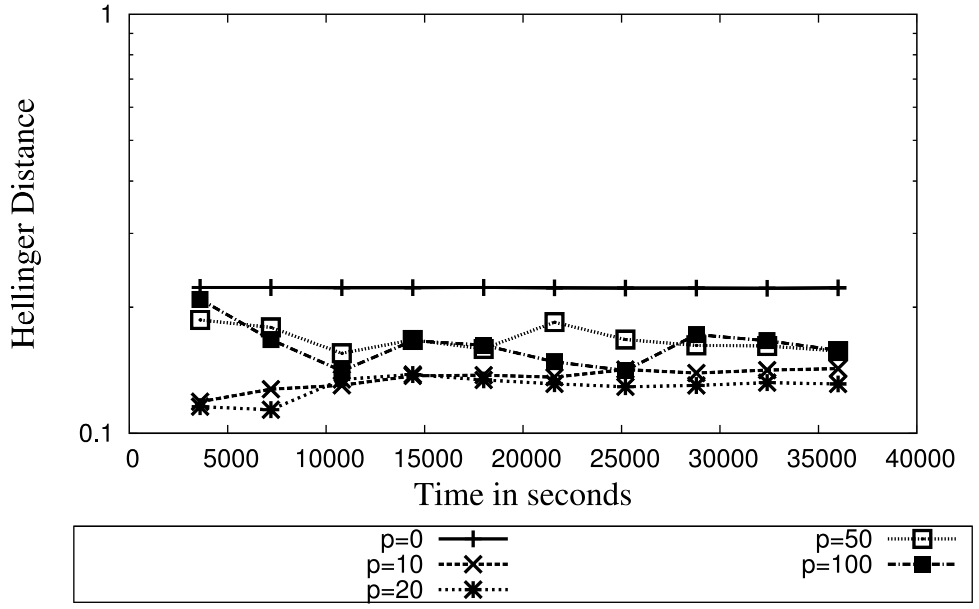
(a)



(b)

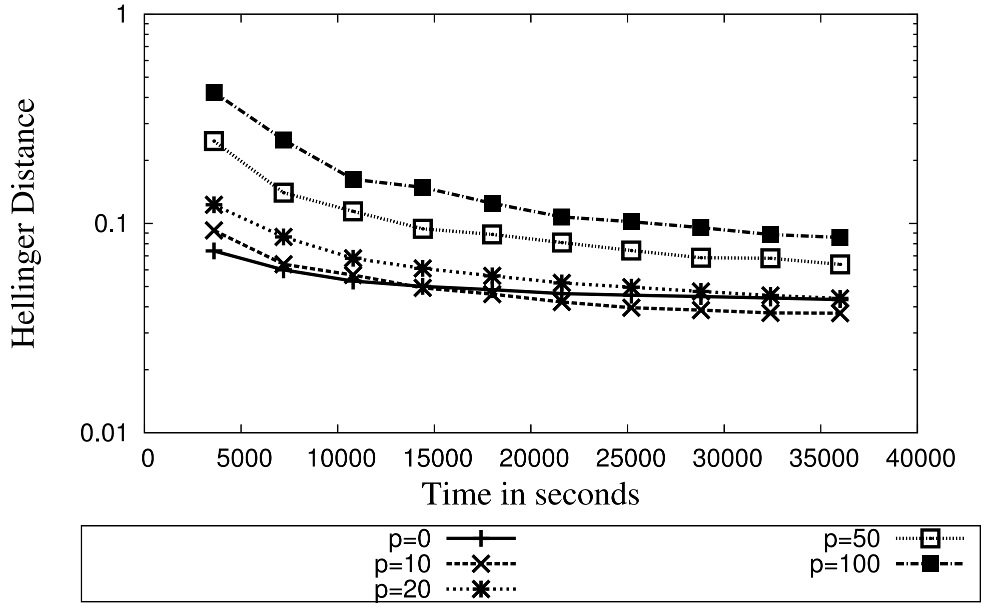
Figure 4.15: Time versus Hellinger distance plot for SampleSearch-SIR with replacement as a function of p for log-4 and Grid pebbling instance of size 25.

Approximation Error vs Time for log-4.cnf



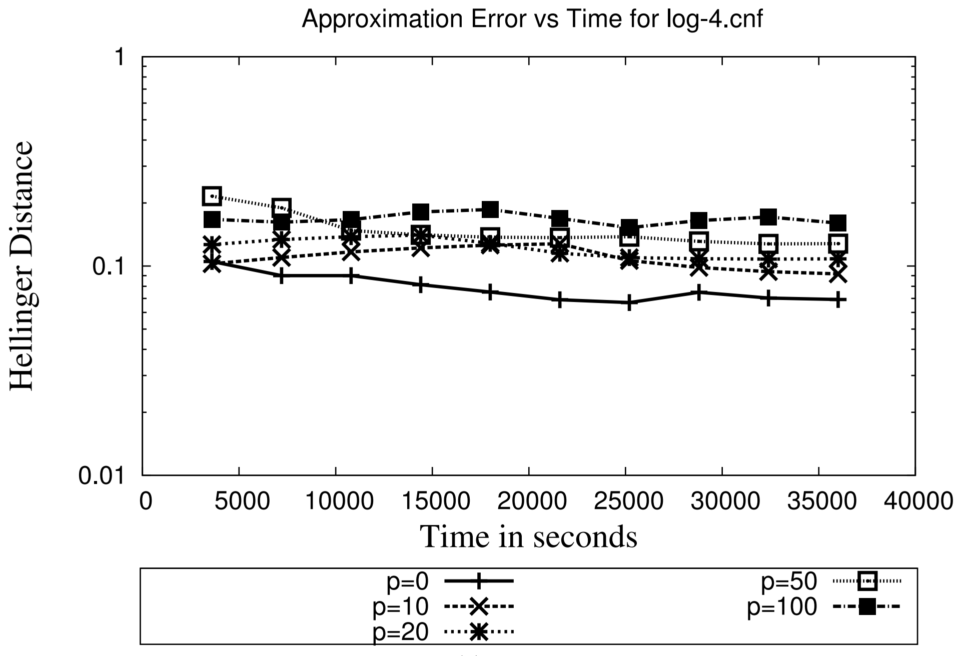
(a)

Approximation Error vs Time for sat-grid-pbl-0025.cnf

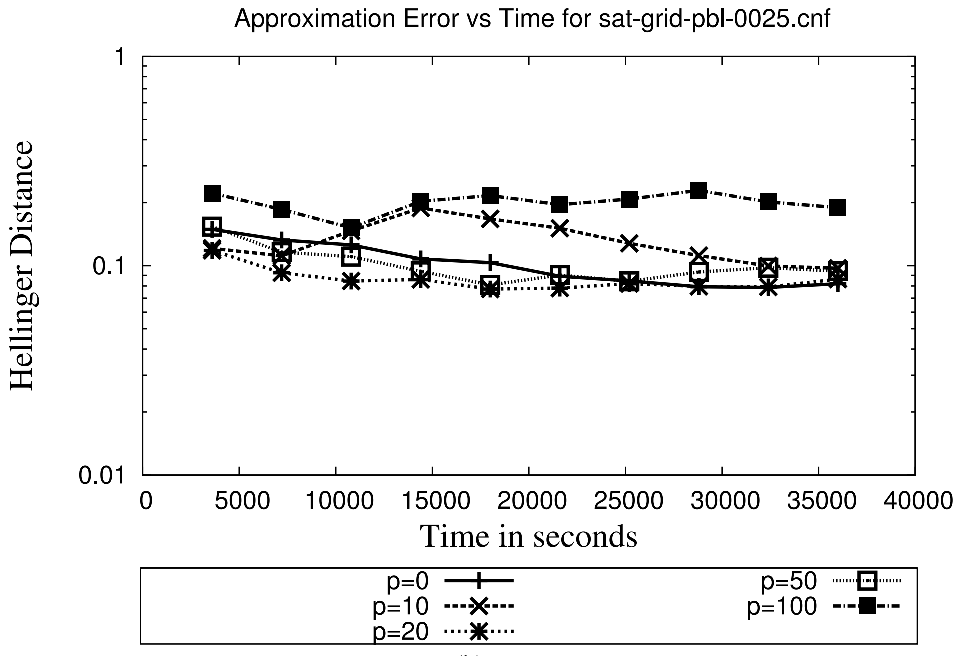


(b)

Figure 4.16: Time versus Hellinger distance plot for SampleSearch-SIR without replacement as a function of p for log-4 and Grid pebbling instance of size 25.



(a)



(b)

Figure 4.17: Time versus Hellinger distance plot for SampleSearch-MH as a function of p for log-4 and Grid pebbling instance of size 25.

distribution over the solutions but also on the number of samples. Both quantities decrease with p and we find that improving SampleSearch's accuracy at the expense of generating less samples does not pay off. In general, our empirical study suggests using low values of $p \leq 20$ in conjunction with MH and SIR.

4.8 Conclusion

The chapter introduces two schemes of SampleSearch-MH and SampleSearch-SIR for generating random, uniformly distributed solutions from a Boolean satisfiability formula. The origin for this task is the use of satisfiability based methods for random test program generation in the functional verification field and for performing MCMC inference in first order probabilistic models.

SampleSearch-MH and SampleSearch-SIR combine SampleSearch with statistical techniques of Metropolis Hastings (MH) and Sampling / Importance Resampling (SIR) respectively and guarantee that in the limit of infinite samples the distribution over the generated solution samples is the uniform distribution over the solutions. These guarantees are significant because state-of-the-art schemes of SampleSat [130] and XorSample [63] do not have them.

We provided a thorough empirical evaluation of SampleSearch, SampleSearch-MH and SampleSearch-SIR with SampleSat and XorSample. Our results showed conclusively that SampleSearch-MH and SampleSearch-SIR are superior to both SampleSat and XorSample in terms of accuracy.

We also studied experimentally the use of a parameter $p \in \{0, 100\}$ which guides recomputation of the sampling distribution by executing IJGP periodically. We found that values of $p < 20$ are more cost-effective.

Chapter 5

Lower Bounding weighted counts using the Markov Inequality

5.1 Introduction

Approximating weighted counting tasks such as computing the probability of evidence in a Bayesian network, the partition function of a Markov network and counting the number of solutions of a constraint satisfaction or a Boolean satisfiability problem, even with known error bounds is NP-hard [21]. In this chapter, we address this hard problem by proposing schemes that yield a high confidence lower bound on the weighted counts but do not have any guarantees of relative or absolute error.

Previous work on bounding the weighted counts comprises of *deterministic approximations* [39, 82, 5] and sampling based *randomized approximations* [15, 22]. An approximation algorithm for computing the lower bound is deterministic if it is always guaranteed to output a lower bound. On the other hand, an approximation algorithm is randomized if the approximation fails with a known probability $\delta > 0$. The work in this chapter falls under the class of randomized approximations.

Randomized approximations [15, 22] use known inequalities such as the Chebyshev and the Hoeffding inequalities [68] for lower (and upper) bounding the weighted counts. The Chebyshev and Hoeffding inequalities provide bounds on how the sample mean of N independently and identically distributed random variables deviates from the actual mean. The main idea in [15, 22], which is in some sense similar to importance sampling [118, 50] is to express the problem of computing the weighted counts as the problem of computing the mean (or expected value) of independent random variables and then use the mean over the sampled random variables to bound the deviation from the actual mean. The problem with these previous approaches is that the number of samples required to guarantee high confidence lower (or upper) bounds is inversely proportional to the weighted counts (or the actual mean). Therefore, if the counts are arbitrarily small (e.g. $< 10^{-20}$), a large number of samples (approximately 10^{19}) are required to guarantee the correctness of the bounds.

In this chapter, we address this problem by focusing on the Markov inequality which is independent of N. Recently, the Markov inequality was used to lower bound the number of solutions of a satisfiability formula [62] showing good empirical results. We adapt this scheme to compute lower bounds on weighted counts and extend it in the following ways. First, we address one of the well known concerns in statistics literature that the Markov inequality is quite weak and yields bad approximations. We argue that the Markov inequality is weak because it is based on a single sample and in fact good lower bounds can be obtained by extending it to multiple samples. Specifically, we propose three new schemes: one based on sample average and two based on the martingale theory [10] which utilize the maximum weight from the generated samples. Our new schemes guarantee that as more samples are drawn, the lower bound is likely to increase. Second, we show how we can estimate the weighted counts efficiently on mixed constraint and probabilistic graphical models by applying the Markov inequality on top of SampleSearch (see Chapter 3).

We provide thorough empirical results demonstrating the potential of our new scheme. On the probabilistic networks we compared against state-of-the-art deterministic approximations such as Variable elimination and Conditioning (VEC) [28] and bound propagation [82, 6]. For the task of lower bounding the number of models of a satisfiability formula, we compared against the Relsat [115] scheme which yields a deterministic approximation and a randomized approximation called SampleCount [62] which can also be combined with our extensions of the Markov inequality. Our results clearly show that our new randomized approximations based on the Markov inequality are more scalable than deterministic approximations like VEC, Relsat and bound propagation and in most cases yield highly accurate lower bounds which are closer to the exact answer than those output by these other schemes. Also on most instances, SampleSearch yields higher lower bounds as compared with SampleCount.

The research presented in this chapter is based in part on [51].

The rest of this chapter is organized as follows. In Section 5.2, we describe preliminaries and previous work. In Section 5.3, we present our basic lower bounding scheme and several enhancements. Experimental results are presented in Section 5.4 and we conclude in Section 5.5.

5.2 Background

In this section, we present previous work by Dagum and Luby [22] and Cheng [15] to approximate the weighted counts with a known relative error ϵ.

Given a mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$, consider the expression for weighted counts:

$$Z = \sum_{\mathbf{x} \in \mathbf{X}} \prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x}) \tag{5.1}$$

Recall that in importance sampling, given N independent and identically distributed (i.i.d.) samples $(\mathbf{x}^1, \ldots, \mathbf{x}^N)$ generated from a proposal distribution $Q(\mathbf{X})$ satisfying $\prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x}) > 0 \rightarrow Q(\mathbf{x}) > 0$, the weighted counts are estimated in an unbiased fashion by:

$$\widehat{Z}_N = \frac{1}{N}\sum_{k=1}^{N} \frac{\prod_{i=1}^{m} F_i(\mathbf{x}^k) \prod_{j=1}^{p} C_j(\mathbf{x}^k)}{Q(\mathbf{x}^k)} = \frac{1}{N}\sum_{k=1}^{N} w(\mathbf{x}^k) \tag{5.2}$$

where

$$w(\mathbf{x}) = \frac{\prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x})}{Q(\mathbf{x})} \tag{5.3}$$

is the weight of sample $\mathbf{x}$.

Dagum and Luby [22] provide an upper bound on the number of samples N required to guarantee that for any $\epsilon, \delta > 0$, the estimate $\widehat{Z}_N$ approximates Z with relative error ϵ with probability at least $1 - \delta$. Formally,

$$\mathbf{Pr}[Z(1-\epsilon) \leq \widehat{Z}_N \leq Z(1+\epsilon)] > 1 - \delta \tag{5.4}$$

when N satisfies:

$$N \geq \frac{4}{Z\epsilon^2} ln\frac{2}{\delta} \tag{5.5}$$

These bounds were later improved by [15] yielding:

$$N \geq \frac{1}{Z}\frac{1}{(1+\epsilon)ln(1+\epsilon) - \epsilon} ln\frac{2}{\delta} \tag{5.6}$$

In both these bounds (Equations 5.5 and 5.6) N is inversely proportional to Z and therefore when Z is small, a large number of samples are required to achieve an acceptable confidence level $(1 - \delta) > 0.99$.

A bound on N is required because [22, 15] insist on approximating Z with a known relative error ϵ. If we relax this relative error requirement, we can use just one sample and the Markov inequality to obtain a high confidence lower bound on Z. Furthermore, we can improve the lower bound with more samples, as we demonstrate in the next section.

5.3 Markov Inequality based Lower Bounds

PROPOSITION **18** (**Markov Inequality**). *For any random variable X and a real number $r > 1$, $\mathbf{Pr}(X > r\mathbb{E}[X]) < \frac{1}{r}$.*

The Markov inequality states that the probability that a random variable is r times its expected value is less than or equal to $1/r$. The weight of each sample generated by importance sampling is a random variable. Because the expected value of the weight equals the weighted counts Z (see Equation 5.1), it is straightforward to see that given a real number $r > 1$, the probability that the weight of a sample is greater than r times Z is less than $1/r$. Alternately, the weight of the sample divided by r is a lower bound on Z with probability greater than $1 - 1/r$. Formally, given a sample $\mathbf{x}$ drawn independently from a proposal distribution Q, we have:

$$\mathbf{Pr}(w(\mathbf{x}) > r \times Z) < \frac{1}{r} \tag{5.7}$$

Rearranging Equation 5.7, we get:

$$\mathbf{Pr}\left(\frac{w(\mathbf{x})}{r} < Z\right) > 1 - \frac{1}{r} \tag{5.8}$$

Equation 5.8 can be used to probabilistically lower bound Z as shown in the following example.

EXAMPLE **13.** *Let $r = 100$ and let $w(\mathbf{x})$ be the weight of a sample drawn independently from a proposal distribution Q. Then from Equation 5.8, $\frac{w(\mathbf{x})}{100}$ is a lower bound on Z with* *probability greater than* $1 - (1/100) = 0.99$.

The lower bound based on the Markov inequality uses just one sample. In the following, we consider ways in which multiple samples could be utilized to improve the lower bound. We formalize the probabilistic lower bounding problem as follows:

DEFINITION **37** (**Probabilistic Lower Bounding problem**). *Given N samples $(\mathbf{x}^1, \ldots, \mathbf{x}^N)$ drawn independently from a proposal distribution Q, such that $\mathbb{E}[w(\mathbf{x}^i)] = Z$ for $i \in \{1, \ldots, N\}$ and a constant $0 < \alpha < 1$, the probabilistic lower bounding problem is to output a statistic $\bar{Z}_N$ (based on the N samples) which is a lower bound on Z with probability greater than α.*

The minimum scheme was proposed to address this probabilistic lower bounding problem.

5.3.1 The Minimum scheme

This scheme due to Gomes et al. [62] uses the minimum over the sample weights to compute a lower bound on Z. Although this scheme was originally introduced in the context of lower bounding the number of solutions to a satisfiability (SAT) problem, we can easily modify it to compute a lower bound on the weighted counts as we show next.

THEOREM **16** (**minimum scheme**). *Given N samples $(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^N)$ drawn independently from a proposal distribution Q such that $\mathbb{E}[w(\boldsymbol{x}^i)] = Z$ for $i = 1, \ldots, N$ and a constant $0 < \alpha < 1$,*

$$\boldsymbol{Pr}\left[min_{i=1}^{N}\left[\frac{w(\mathbf{x}^i)}{\beta}\right] < Z\right] > \alpha, \;\; where\, \beta = \left(\frac{1}{1-\alpha}\right)^{\frac{1}{N}}$$

Proof. Consider an arbitrary sample $\mathbf{x}^i$. From the Markov inequality, we get:

$$\mathbf{Pr}\left[\frac{w(\mathbf{x}^i)}{\beta} > Z\right] < 1/\beta \tag{5.9}$$

Since, the generated N samples are independent, the probability that the minimum over them is also a lower bound is given by:

$$\mathbf{Pr}\left[min_{i=1}^{N}\left[\frac{w(\mathbf{x}^i)}{\beta}\right] > Z\right] < 1/\beta^N \tag{5.10}$$

Rearranging Equation 5.10, we get:

$$\mathbf{Pr}\left[min_{i=1}^{N}\left[\frac{w(\mathbf{x}^i)}{\beta}\right] < Z\right] > 1 - \frac{1}{\beta^N} \tag{5.11}$$

Substituting $\beta = \left(\frac{1}{1-\alpha}\right)^{\frac{1}{N}}$ in $1 - \frac{1}{\beta^N}$, we get:

$$\begin{aligned} 1 - \frac{1}{\beta^N} &= 1 - \frac{1}{\left(\left(\frac{1}{1-\alpha}\right)^{\frac{1}{N}}\right)^N} \\ &= 1 - \frac{1}{\frac{1}{1-\alpha}} \\ &= 1 - (1-\alpha) \\ &= \alpha \end{aligned} \tag{5.12}$$

Therefore, from Equations 5.11 and 5.12, we have

$$\mathbf{Pr}\left[min_{i=1}^{k}\left[\frac{w(\mathbf{x}^i)}{\beta}\right] < Z\right] > \alpha \tag{5.13}$$

□

Algorithm 20 describes the minimum scheme based on Theorem 16. The algorithm first calculates β based on the value of α and N. It then returns the minimum of $\frac{w(\mathbf{x}^i)}{\beta}$ (minCount

Algorithm 20: Minimum-scheme

```
Input: A mixed network $\mathcal{M} = \langle$X,D,F,C$\rangle$, proposal distribution $Q$, integer $N$ and a
       real number $0 < \alpha < 1$
Output: Lower Bound on Z that is correct with probability greater than $\alpha$
1 $minCount \leftarrow \infty$;
2 $\beta = \left(\frac{1}{1-\alpha}\right)^{\frac{1}{N}}$;
3 for $i = 1$ to $N$ do
4     Generate a sample $\mathbf{x}^i$ from $Q$ ;
5     IF $minCount > \frac{w(\mathbf{x}^i)}{\beta}$ THEN $minCount = \frac{w(\mathbf{x}^i)}{\beta}$;
6 Return $minCount$;
```

in Algorithm 20) over the N samples.

The problem with the minimum scheme is that because it computes a minimum over the sample weights, unless the variance of the weights is very small, we expect the lower bound to decrease with increase in the number of samples N. On the other hand, a good property of the minimum scheme is that with more samples, the divisor $\beta = \frac{1}{(1-\alpha)^{\frac{1}{N}}}$ decreases, thereby possibly increasing the lower bound. Next, we present our average scheme for computing a probabilistic lower bound which avoids this problem.

5.3.2 The Average Scheme

An obvious scheme is to use the unbiased importance sampling estimator $\widehat{Z}_N$ given in Equation 5.2. Because $\mathbb{E}_Q[\widehat{Z}_N] = Z$, from the Markov inequality $\frac{\widehat{Z}_N}{\beta}$ where $\beta = \frac{1}{1-\alpha}$ is a lower bound of Z with probability greater than α. Formally,

$$\mathbf{Pr}\left[\frac{\widehat{Z}_N}{\beta} < Z\right] > \alpha, \;\; where \;\; \beta = \frac{1}{1-\alpha} \tag{5.14}$$

As more samples are drawn the average is likely to stabilize and will usually be larger than the minimum value. However, unlike the minimum scheme in which the divisor β decreases with increase in the sample size thereby increasing the lower bound, the divisor β in the average scheme remains constant. As a consequence, for example, if all the generated samples have the same weight (or almost the same weight), the lower bound due to the minimum scheme would be greater than the lower bound output by the average scheme. However, in practice the variance is typically never close to zero and therefore the average scheme is likely to better than the minimum scheme.

5.3.3 The Maximum scheme

We can even use the maximum instead of the average over the N i.i.d samples as shown in the following Lemma.

LEMMA **1** (**maximum scheme**). *Given N samples $(\mathbf{x}^1, \ldots, \mathbf{x}^N)$ drawn independently from a proposal distribution Q such that $\mathbb{E}[w(\mathbf{x}^i)] = Z$ for $i = 1, \ldots, N$ and a constant $0 < \alpha < 1$,*

$$\mathbf{Pr}\left[\frac{max_{i=1}^{N} w(\mathbf{x}^i)}{\beta} < Z\right] > \alpha, \ where\ \beta = \frac{1}{1-\alpha^{\frac{1}{N}}}$$

Proof. From Markov inequality, we have:

$$\mathbf{Pr}\left[\frac{w(\mathbf{x}^i)}{\beta} < Z\right] > 1 - \frac{1}{\beta} \tag{5.15}$$

Given a set of N independent events such that each event occurs with probability $> (1 - 1/\beta)$, the probability that all events occur is $> (1 - 1/\beta)^N$. In other words, given N independent samples such that the weight of each sample is a lower bound on Z with probability $> (1 - 1/\beta)$, the probability that all samples are a lower bound on Z is $> (1 - 1/\beta)^N$. Consequently,

$$\mathbf{Pr}\left[\frac{max_{i=1}^{N} w(\mathbf{x}^i)}{\beta} < Z\right] > \left(\frac{1}{1-\beta}\right)^N \tag{5.16}$$

Substituting the value of β in $\left(\frac{1}{1-\beta}\right)^N$, we have:

$$\begin{aligned}\left(\frac{1}{1-\beta}\right)^N &= \left(\frac{1}{1-\frac{1}{1-\alpha^{\frac{1}{N}}}}\right)^N \\ &= (1-(1-\alpha^{\frac{1}{N}}))^N \\ &= \alpha \end{aligned} \tag{5.17}$$

From Equations 5.16 and 5.17, we get:

$$\mathbf{Pr}\left[\frac{max_{i=1}^{N} w(\mathbf{x}^i)}{\beta} < Z\right] > \alpha \tag{5.18}$$

□

The problem with the maximum scheme is that increasing the number of samples increases β and consequently the lower bound decreases. However, when only a few samples are available and the variance of the weights $w(\mathbf{x}^i)$ is large, the maximum value is likely to be larger than the sample average and obviously the minimum.

5.3.4 Using the Martingale Inequalities

Another approach to utilize the maximum over the N samples is to use the martingale inequalities.

DEFINITION **38** (**Martingale**). *A sequence of random variables $X_1, \ldots, X_N$ is a martingale with respect to another sequence $Y_1, \ldots, Y_N$ defined on a common probability space Ω iff $\mathbb{E}[X_i|Y_1, \ldots, Y_{i-1}] = X_{i-1}$ for all i.*

It is easy to see that given i.i.d. samples $(\mathbf{x}^1, \ldots, \mathbf{x}^N)$ generated from Q, the sequence

$\Lambda_1, \ldots, \Lambda_N, \quad where \;\; \Lambda_p = \prod_{i=1}^{p} \frac{w(\mathbf{x}^i)}{Z}$ forms a martingale as shown below:

$$\begin{aligned}\mathbb{E}[\Lambda_p|\mathbf{x}^1, \ldots, \mathbf{x}^{p-1}] &= \mathbb{E}\left[\Lambda_{p-1} * \frac{w(\mathbf{x}^p)}{Z}|\mathbf{x}^1, \ldots, \mathbf{x}^{p-1}\right] \\ &= \Lambda_{p-1} * \mathbb{E}\left[\frac{w(\mathbf{x}^p)}{Z}|\mathbf{x}^1, \ldots, \mathbf{x}^{p-1}\right]\end{aligned}$$

Because $\mathbb{E}[\frac{w(\mathbf{x}^p)}{Z}|\mathbf{x}^1, \ldots, \mathbf{x}^{p-1}] = 1$, we have $\mathbb{E}[\Lambda_p|\mathbf{x}^1, \ldots, \mathbf{x}^{p-1}] = \Lambda_{p-1}$ as required. The expected value $\mathbb{E}[\Lambda_1] = 1$ and for such martingales which have a mean of 1, Breiman [10] provides the following extension of the Markov inequality:

$$\mathbf{Pr}(max_{i=1}^{N}\Lambda_i > \beta) < \frac{1}{\beta} \tag{5.19}$$

and therefore,

$$\mathbf{Pr}\left(\left[max_{i=1}^{N}\prod_{j=1}^{i}\frac{w(\mathbf{x}^j)}{Z}\right] > \beta\right) < \frac{1}{\beta} \tag{5.20}$$

From Inequality 5.20, we can prove that:

THEOREM **17 (Random permutation scheme).** *Given N samples $(\mathbf{x}^1, \ldots, \mathbf{x}^N)$ drawn independently from a proposal distribution Q such that $\mathbb{E}[w(\mathbf{x}^i)] = Z$ for $i = 1, \ldots, N$ and a constant $0 < \alpha < 1$,*

$$\mathbf{Pr}\left[max_{i=1}^{N}\left(\frac{1}{\beta}\prod_{j=1}^{i}w(\mathbf{x}^j)\right)^{1/i} < Z\right] > \alpha, \; where \; \beta = \frac{1}{1-\alpha}$$

Proof. From Inequality 5.20, we have:

$$\mathbf{Pr}\left(\left[max_{i=1}^{N}\prod_{j=1}^{i}\frac{w(\mathbf{x}^j)}{Z}\right] > \beta\right) < \frac{1}{\beta} \tag{5.21}$$

Rearranging Inequality 5.21, we have:

$$\mathbf{Pr}\left[max_{i=1}^{N}\left(\frac{1}{\beta}\prod_{j=1}^{i}w(\mathbf{x}^{j})\right)^{1/i} < Z\right] > 1-\frac{1}{\beta}=\alpha \tag{5.22}$$

□

Therefore, given N samples, the following quantity

$$max_{i=1}^{N}\left(\frac{1}{\beta}\prod_{j=1}^{i}w(\mathbf{x}^{j})\right)^{1/i} \quad \text{where } \beta=\frac{1}{1-\alpha}$$

is a lower bound on Z with a confidence greater than α. In general one could use any randomly selected permutation of the samples $(\mathbf{x}^1,\ldots,\mathbf{x}^N)$ and apply inequality 5.20. We therefore call this scheme as the *random permutation scheme*.

Another related extension of Markov inequality for martingales deals with the order statistics of the samples. Let $\frac{w(\mathbf{x}^{(1)})}{Z} \leq \frac{w(\mathbf{x}^{(2)})}{Z} \leq \ldots \leq \frac{w(\mathbf{x}^{(N)})}{Z}$ be the order statistics of the sample. Using martingale theory, Kaplan [71] proved that the random variable

$$\Theta^* = max_{i=1}^{N}\prod_{j=1}^{i}\frac{w(\mathbf{x}^{(N-j+1)})}{Z\times\binom{N}{i}}$$

satisfies the inequality $\mathbf{Pr}(\Theta^* > k) < 1/k$. Therefore,

$$\mathbf{Pr}\left(\left[max_{i=1}^{N}\prod_{j=1}^{i}\frac{w(\mathbf{x}^{(N-j+1)})}{Z\times\binom{N}{i}}\right] > \beta\right) < \frac{1}{\beta} \tag{5.23}$$

From Inequality 5.23, we can prove that:

THEOREM **18** (**Order Statistics scheme**). *Given an order statistics of the weights* $\frac{w(\mathbf{x}^{(1)})}{Z} \leq \frac{w(\mathbf{x}^{(2)})}{Z} \leq \ldots \leq \frac{w(\mathbf{x}^{(N)})}{Z}$ *of N samples $(\mathbf{x}^1,\ldots,\mathbf{x}^N)$ drawn independently from a proposal* *distribution Q, such that* $\mathbb{E}[w(\mathbf{x}^i)] = Z$ *for* $i = 1, \ldots, N$ *and a constant* $0 < \alpha < 1$*,*

$$\mathbf{Pr}\left[max_{i=1}^{N}\left(\frac{1}{\beta}\prod_{j=1}^{i}\frac{w(\mathbf{x}^{(N-j+1)})}{\binom{N}{i}}\right)^{1/i} < Z\right] > \alpha,\ where\ \beta = \frac{1}{1-\alpha}$$

Proof. From Inequality 5.23, we have:

$$\mathbf{Pr}\left(\left[max_{i=1}^{N}\prod_{j=1}^{i}\frac{w(\mathbf{x}^{(N-j+1)})}{Z\times\binom{N}{i}}\right] > \beta\right) < \frac{1}{\beta} \tag{5.24}$$

Rearranging Inequality 5.24, we have:

$$\mathbf{Pr}\left[max_{i=1}^{N}\left(\frac{1}{\beta}\prod_{j=1}^{i}\frac{w(\mathbf{x}^{(N-j+1)})}{\binom{N}{i}}\right)^{1/i} < Z\right] > 1 - \frac{1}{\beta} = \alpha \tag{5.25}$$

□

Thus, given N samples, the following quantity

$$max_{i=1}^{N}\left(\frac{1}{\beta}\prod_{j=1}^{i}\frac{w(\mathbf{x}^{(N-j+1)})}{\binom{N}{i}}\right)^{1/i},\ \text{where } \beta = \frac{1}{1-\alpha}$$

is a lower bound on Z with probability greater than α. Because the lower bound is based on the order statistics, we call this scheme as the *order statistics* scheme.

To summarize, we have proposed five schemes that generalize the Markov inequality to multiple samples: (1) The minimum scheme, (2) The average scheme, (3) The maximum scheme, (4) The martingale random permutation scheme and (5) The martingale order statistics scheme.

All these schemes can be used with any sampling scheme that outputs unbiased sample weights to yield a probabilistic lower bound on the weighted counts.

5.4 Empirical Evaluation

Because SampleSearch samples from the backtrack-free distribution, it is easy to see that to use the lower bounding schemes on the samples output by SampleSearch, all we have to do is replace the weight $w(\mathbf{x})$ of sample $\mathbf{x}$ by its backtrack-free weight $w^F(\mathbf{x})$ (see Chapter 3). Interestingly, we can even use the lower backtrack free approximation $w^L_N(\mathbf{x})$ instead of $w^F(\mathbf{x})$ because a lower bound of a lower bound is also a lower bound, and as we already proved in Chapter 3, $w^L_N(\mathbf{x})$ is a lower bound of $w^F(\mathbf{x})$.

We evaluate the performance of the probabilistic lower bounding schemes presented in this chapter using the IJGP-sampling and SampleSearch schemes presented in this thesis (see Chapters 2 and 3 respectively). We also compare against deterministic approximations, which always yield a lower bound (with probability one) such as Bound propagation and its improvements [5], Variable elimination and conditioning (VEC) [28] and Relsat [115].

We conducted experiments on three weighted counting tasks: (a) Satisfiability Model counting, (b) Computing probability of evidence in a Bayesian network and (c) computing the partition function of a Markov network. Our experimental data clearly demonstrates that our new lower bounding schemes are more accurate, robust and scalable than deterministic approximations such as VEC, Relsat and Bound propagation yielding far better (higher) lower bounds on large instances.

5.4.1 The Algorithms Evaluated

We experimented with the following schemes. Hence forth, we call our new probabilistic lower bounding schemes as Markov-LB.

Markov-LB with SampleSearch and IJGP-sampling: We used the same implementation of IJGP-sampling and SampleSearch described in Chapters 2 and 3 respectively. We used IJGP-sampling on networks which have no determinism (or a small amount) while on networks with substantial amount of determinism, we used the SampleSearch scheme.

Bound Propagation with Cut-set Conditioning We also experimented with the state of the art any-time bounding scheme that combines sampling-based cut-set conditioning and bound propagation [82] and which is a part of Any-Time Bounds framework for bounding posterior marginals [5]. Given a subset of variables $\mathbf{C} \subset \mathbf{X} \backslash \mathbf{E}$, we can compute $P(\mathbf{e})$ exactly as follows:

$$P(\mathbf{e}) = \sum_{i=1}^{k} P(\mathbf{c^i}, \mathbf{e}) \tag{5.26}$$

The lower bound on $P(\mathbf{e})$ is obtained by computing $P(\mathbf{c^i}, \mathbf{e})$ for h high probability tuples of $\mathbf{C}$ (selected through sampling) and bounding the remaining probability mass by computing a lower bound $P^L(\mathbf{c_1}, ..., \mathbf{c_q}, \mathbf{e})$ on $P(\mathbf{c_1}, ..., \mathbf{c_q}, \mathbf{e})$, $q < |\mathbf{C}|$, for a polynomial number of partially instantiated tuples of subset C, resulting in:

$$P(\mathbf{e}) \geq \sum_{i=1}^{h} P(\mathbf{c^i}, \mathbf{e}) + \sum_{i=1}^{k'} P^L_{BP}(\mathbf{c^i_1}, ..., \mathbf{c^i_q}, \mathbf{e}) \tag{5.27}$$

where lower bound $P^L_{BP}(\mathbf{c_1}, ..., \mathbf{c_q}, \mathbf{e})$ is obtained using bound propagation. Although bound propagation bounds marginal probabilities, it can be used to bound any joint probability $P(\mathbf{z})$ as follows:

$$P^L_{BP}(\mathbf{z}) = \prod_{i} P^L_{BP}(z_i | z_1, ..., z_{i-1}) \tag{5.28}$$

where lower bound $P^L_{BP}(z_i | z_1, ..., z_{i-1})$ is computed directly by bound propagation. We use here the same variant of bound propagation described in [6] that is used by the Any-Time Bounds framework. The lower bound obtained by Equation 5.27 can be improved by exploring a larger number of tuples h. After generating h tuples by sampling, we can stop the computation at any time after bounding $p < k'$ out of k' partially instantiated tuples and produce the result.

In our experiments we run the bound propagation with cut-set conditioning scheme until convergence or until a stipulated time bound has expired. Finally, we should note that the bound propagation with cut-set conditioning scheme provides deterministic lower and upper bounds on $P(\mathrm{e})$ while our Markov-LB scheme provides only a lower bound and it may fail with a probability $\delta \leq 0.01$.

Variable Elimination and Conditioning (VEC) When a problem having a high treewidth is encountered, variable or bucket elimination may be unsuitable, primarily because of its extensive memory demand. To alleviate the space complexity, we can use the w-cutset conditioning scheme (see Section 1.3.2). Namely, we condition or instantiate enough variables (or the w-cutset) so that the remaining problem after removing the instantiated variables can be solved exactly using bucket elimination [28]. In our experiments we select the w-cutset in such a way that bucket elimination would require less than 1.5GB of space. Exact weighted counts can be computed by summing over the exact solution output by bucket elimination for all possible instantiations of the w-cutset. When VEC is terminated before completion, it outputs a partial sum yielding a lower bound on the weighted counts. The implementation of VEC is available publicly from our software website [30].

Markov-LB with SampleCount

We use the same implementation of SampleCount [62] as described in Chapter 3. To recap, SampleCount outputs unbiased solution counts of a Boolean Satisfiability formula. Therefore, it can be easily combined with all Markov-LB schemes yielding the Markov-LB with SampleCount scheme.

Relsat

As mentioned in Chapter 3, Relsat [115] is an exact algorithm for counting solutions of a satisfiability problem. If stopped before termination, Relsat yields a lower bound on the number of solutions. The implementation of Relsat is available publicly from the first authors website [115].

We experimented with four versions of Markov-LB (run with SampleSearch, SampleCount and IJGP-Sampling): (a) Markov-LB as given in Algorithm 20, (b) Markov-LB with the average scheme, (c) Markov-LB with the martingale random permutation scheme and (d) Markov-LB with the martingale order statistics scheme. Note that the maximum scheme is subsumed by the Markov-LB with the martingale order statistics scheme. In all our experiments, we set $\alpha = 0.99$.

Evaluation Criteria

On each instance, we compared the log relative error between the exact value of probability of evidence (or the solution counts for satisfiability problems) and the lower bound generated by the respective techniques. Formally, if Z is the actual probability of evidence (or solution counts) and $\overline{Z}$ is the approximate probability of evidence (or solution counts), the log-relative error denoted by Δ is given by:

$$\Delta = \frac{log(Z) - log(\overline{Z})}{log(Z)} \tag{5.29}$$

When the exact results are not known, we use the highest lower bound reported by the schemes as a substitute for Z in Equation 5.29. We compute the log relative error instead of the usual relative error because when the probability of evidence is extremely small ($< 10^{-10}$) or when the solution counts are large (e.g. $> 10^{10}$) the relative error between the exact and the approximate probability of evidence will be arbitrarily close to 1 and we would need a large number of digits to determine the best performing scheme.

Notation in Tables

The first column in each table (see for example Table 5.1) gives the name of the instance. The second column provides raw statistical information about the instance such as: (i) number of variables (n), (ii) average domain size (d), (iii) number of clauses (c) or number of evidence variables (e) and (iv) the treewidth of the instance (w). The third column provides the exact answer for the problem if available while the remaining columns display the output produced by the various schemes after the specified time-bound. The columns Min, Avg, Per and Ord give the log-relative-error Δ for the minimum, the average, the martingale random permutation and the martingale order statistics schemes respectively. The final column Best LB reports the best lower bound reported by all the schemes whose log-relative error is highlighted by bold in each row.

We organize our results in two parts. We first consider networks which do not have determinism and compare Bound propagation and its improvements with IJGP-sampling based Markov-LB schemes. Then, we consider networks which have determinism and compare SampleSearch based Markov-LB with Variable elimination and Conditioning for probabilistic networks and with SampleCount for Boolean satisfiability problems.

5.4.2 Results on networks having no determinism

Table 5.1 summarizes the results. We ran each algorithm for 2 minutes. We see that our new strategy of Markov-LB scales well with problem size and provides good quality high-confidence lower bounds on most problems. It clearly outperforms the bound propagation with cut-set conditioning scheme. We discuss the results in detail below.

Non-deterministic Alarm networks The Alarm networks are one of the earliest belief networks designed by medical experts for monitoring patients in intensive care. The evi-

Problem	$\langle n,d,e,w\rangle$	Exact	Markov-LB with IJGP-sampling Min	Avg	Per	Ord	Bound propa-gation	Best
		P(e)	Δ	Δ	Δ	Δ	Δ	LB
Alarm								
BN_3	$\langle 100,2,36\rangle$	2.8E-13	0.157	**0.031**	0.040	0.059	0.090	1.1E-13
BN_4	$\langle 100,2,51\rangle$	3.6E-18	0.119	**0.023**	0.040	0.045	0.025	1.4E-18
BN_5	$\langle 125,2,55\rangle$	1.8E-19	0.095	**0.020**	0.021	0.030	0.069	7.7E-20
BN_6	$\langle 125,2,71\rangle$	4.3E-26	0.124	**0.016**	0.024	0.030	0.047	1.6E-26
BN_11	$\langle 125,2,46\rangle$	8.0E-18	0.185	**0.023**	0.061	0.064	0.102	3.3E-18
CPCS								
CPCS-360-1	$\langle 360,2,20\rangle$	1.3E-25	0.012	0.012	**0.000**	0.001	0.002	1.3E-25
CPCS-360-2	$\langle 360,2,30\rangle$	7.6E-22	0.045	0.015	0.010	0.010	**0.000**	7.6E-22
CPCS-360-3	$\langle 360,2,40\rangle$	1.2E-33	0.010	0.009	0.000	**0.000**	**0.000**	1.2E-33
CPCS-360-4	$\langle 360,2,50\rangle$	3.4E-38	0.022	0.009	0.002	**0.000**	**0.000**	3.4E-38
CPCS-422-1	$\langle 422,2,20\rangle$	7.2E-21	0.028	0.016	0.001	**0.001**	0.002	6.8E-21
CPCS-422-2	$\langle 422,2,30\rangle$	2.7E-57	0.005	0.005	0.000	**0.000**	**0.000**	2.7E-57
CPCS-422-3	$\langle 422,2,40\rangle$	6.9E-87	0.003	0.003	0.000	**0.000**	0.001	6.9E-87
CPCS-422-4	$\langle 422,2,50\rangle$	1.4E-73	0.007	0.004	0.000	**0.000**	0.001	1.3E-73
Random								
BN_94	$\langle 53,50,6\rangle$	4.0E-11	0.235	0.029	0.063	**0.025**	0.028	2.2E-11
BN_96	$\langle 54,50,5\rangle$	2.1E-09	0.408	0.036	0.095	**0.013**	0.131	1.6E-09
BN_98	$\langle 57,50,6\rangle$	1.9E-11	0.131	0.024	**0.013**	0.024	0.147	1.4E-11
BN_100	$\langle 58,50,8\rangle$	1.6E-14	0.521	**0.022**	0.079	0.041	0.134	8.1E-15
BN_102	$\langle 76,50,15\rangle$	1.5E-26	0.039	0.007	**0.007**	0.012	0.056	9.4E-27

Table 5.1: Table showing the log-relative error Δ of bound propagation and four versions of Markov-LB combined with IJGP-sampling for Bayesian networks having no determinism after 2 minutes of CPU time.

dence in these networks was set at random. These networks have between 100-125 binary nodes. We can see that Markov-LB with IJGP-sampling is slightly superior to the bound propagation based scheme accuracy-wise. Among the different versions of Markov-LB with IJGP-sampling, the average scheme performs better than the martingale schemes. The minimum scheme is the worst performing scheme.

The CPCS networks The CPCS networks are derived from the Computer-based Patient Case Simulation system [110]. The nodes of CPCS networks correspond to diseases and findings and conditional probabilities describe their correlations. The CPCS360b and CPCS422b networks have 360 and 422 variables respectively. We report results on the two networks with 20,30,40 and 50 randomly selected evidence nodes. We see that the lower bounds reported by the bound propagation based scheme are slightly better than Markov-LB with IJGP-sampling on the CPCS360b networks. However, on the CPCS422b networks, Markov-LB with IJGP-sampling gives higher lower bounds. The martingale schemes (the random permutation and the order statistics) give higher lower bounds than the average scheme. Again, the minimum scheme is the weakest.

Random networks The random networks are randomly generated graphs available from the UAI 2006 evaluation web site. The evidence nodes are generated at random. The networks have between 53 and 76 nodes and the maximum domain size is 50. We see that Markov-LB is better than the bound propagation based scheme on all random networks. The random permutation and the order statistics martingale schemes are slightly better than the average scheme on most instances.

5.4.3 Results on networks having determinism

In this subsection, we report on experiments for networks which have determinism. We experimented with five benchmark domains: (a) Latin square instances, (b) Langford in-

			Markov-LB with SampleSearch				Markov-LB with SampleCount				REL SAT	
Problem	$\langle n, k, c, w \rangle$	**Exact**	**Min**	**Avg**	**Per**	**Ord**	**Min**	**Avg**	**Per**	**Ord**		**Best**
			Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	**LB**
ls8-norm	$\langle 512, 2, 5584, 255 \rangle$	5.40E+11	0.387	**0.012**	0.068	0.095	0.310	0.027	0.090	0.090	0.344	3.88E+11
ls9-norm	$\langle 729, 2, 9009, 363 \rangle$	3.80E+17	0.347	**0.021**	0.055	0.070	0.294	0.030	0.097	0.074	0.579	1.59E+17
ls10-norm	$\langle 1000, 2, 13820, 676 \rangle$	7.60E+24	0.304	**0.002**	0.077	0.044	0.237	0.016	0.054	0.050	0.710	6.93E+24
ls11-norm	$\langle 1331, 2, 20350, 956 \rangle$	5.40E+33	0.287	0.023	0.102	0.026	0.227	0.036	0.094	**0.034**	0.783	7.37E+34
ls12-norm	$\langle 1728, 2, 28968, 1044 \rangle$		0.251	0.007	0.045	0.011	0.232	**0.000**	0.079	0.002	0.833	3.23E+43
ls13-norm	$\langle 2197, 2, 40079, 1558 \rangle$		0.250	0.005	0.080	**0.000**	0.194	0.015	0.087	0.044	0.870	1.26E+55
ls14-norm	$\langle 2744, 2, 54124, 1971 \rangle$		0.174	0.010	0.057	**0.000**	0.140	0.043	0.065	0.026	0.899	2.72E+67
ls15-norm	$\langle 3375, 2, 71580, 2523 \rangle$		0.189	0.015	0.080	**0.000**	0.130	0.053	0.077	0.062	0.923	4.84E+82
ls16-norm	$\langle 4096, 2, 92960, 2758 \rangle$		0.158	**0.000**	0.055	0.001	0.108	0.030	0.053	0.007	X	1.16E+97

Table 5.2: Table showing the log-relative error Δ of Relsat and four versions of Markov-LB combined with SampleSearch and SampleCount respectively for Latin Square instances after 10 hours of CPU time.

stances, (c) FPGA routing instances, (d) Linkage instances and (e) Relational instances. The task of interest on the first three domains is counting solutions while the task of interest on the final two domains is computing probability of evidence. All these domains were also used to evaluate SampleSearch in Chapter 3.

Results on Satisfiability model counting

For model counting, we evaluate the lower bounding power of Markov-LB with SampleSearch and Markov-LB with SampleCount [62]. We ran both algorithms for 10 hours on each instance.

Results on Latin Square instances

Table 5.2 shows the detailed results. The exact counts for Latin square instances are known only up to order 11. As pointed out earlier, when the exact results are not known, we use the highest lower bound reported by the schemes as a substitute for Z in Equation 5.29.

Among the different versions of Markov-LB with SampleSearch, we see that the average scheme performs better than the martingale order statistics scheme on 5 out of 8 instances while the martingale order statistics scheme is superior on the other 3 instances. The min-

			Markov-LB with SampleSearch				Markov-LB with SampleCount				REL SAT	
Problem	$\langle n,k,c,w\rangle$	**Ex act**	**Min**	**Avg**	**Per**	**Ord**	**Min**	**Avg**	**Per**	**Ord**		**Best**
			Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	**LB**
lang12	$\langle 576, 2, 13584, 383\rangle$	2.16E+05	0.464	0.051	0.128	0.171	0.455	0.067	0.103	0.175	**0.000**	2.16E+05
lang16	$\langle 1024, 2, 32320, 639\rangle$	6.53E+08	0.475	**0.008**	0.106	0.131	0.378	0.019	0.097	0.023	0.365	7.68E+08
lang19	$\langle 1444, 2, 54226, 927\rangle$	5.13E+11	0.405	**0.041**	0.109	0.095	0.420	0.156	0.219	0.200	0.636	1.70E+11
lang20	$\langle 1600, 2, 63280, 1023\rangle$	5.27E+12	0.411	**0.031**	0.150	0.102	0.424	0.217	0.188	0.123	0.685	2.13E+12
lang23	$\langle 2116, 2, 96370, 1407\rangle$	7.60E+15	0.389	**0.058**	0.119	0.100	0.418	0.215	0.284	0.211	X	9.15E+14
lang24	$\langle 2304, 2, 109536, 1535\rangle$	9.37E+16	0.258	0.076	**0.043**	0.054	0.283	0.220	0.203	0.220	X	1.74E+16
lang27	$\langle 2916, 2, 156114, 1919\rangle$		0.261	**0.000**	0.093	0.107	0.364	0.264	0.291	0.267	X	7.67E+19

Table 5.3: Table showing the log-relative error Δ of Relsat and four versions of Markov-LB combined with SampleSearch and SampleCount respectively for Langford instances after 10 hours of CPU time.

imum scheme is the weakest scheme while the martingale random permutation scheme is between the minimum scheme and the average and martingale order statistics scheme.

Among the different versions of Markov-LB with SampleCount, we see very similar performance.

SampleSearch with Markov-LB generates better lower bounds than SampleCount with Markov-LB on 6 out of the 8 instances. The lower bounds output by Relsat are several orders of magnitude lower than those output by Markov-LB with SampleSearch and Markov-LB with SampleCount.

Results on Langford instances

Table 5.3 shows the results. Among the different versions of Markov-LB with SampleSearch, we see again the superiority of the average scheme. The martingale order statistics and random permutation schemes are the second and the third best respectively. Among the different versions of SampleCount based Markov-LB, we see a similar trend where the average scheme performs better than other schemes on 6 out of the 7 instances.

Markov-LB with SampleSearch outperforms Markov-LB with SampleCount on 6 out of the 7 instances. The lower bounds output by Relsat are inferior by several orders of magnitude to the Markov-LB based lower bounds except on the lang12 instance which RESLAT solves

Problem	$\langle n, k, c, w\rangle$	Ex-act	Markov-LB with SampleSearch Min	Avg	Per	Ord	Markov-LB with SampleCount Min	Avg	Per	Ord	REL SAT	Best
			Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	**LB**
9symml_gr_2pin_w6	$\langle 2604, 2, 36994, 413\rangle$		0.192	**0.000**	0.075	0.006	0.087	0.073	0.076	0.075	0.491	2.76E+53
9symml_gr_rcs_w6	$\langle 1554, 2, 29119, 613\rangle$		0.237	0.016	0.117	0.023	0.117	0.060	0.041	0.009	**0.000**	9.95E+84
alu2_gr_rcs_w8	$\langle 4080, 2, 83902, 1470\rangle$		0.224	0.097	0.152	0.102	**0.000**	0.906	0.023	0.345	0.762	1.47E+235
apex7_gr_2pin_w5	$\langle 1983, 2, 15358, 188\rangle$		0.158	0.003	0.073	**0.000**	0.064	0.023	0.047	0.036	0.547	2.71E+93
apex7_gr_rcs_w5	$\langle 1500, 2, 11695, 290\rangle$		0.228	0.037	0.118	0.038	0.099	**0.000**	0.028	0.008	0.670	3.04E+139
c499_gr_2pin_w6	$\langle 2070, 2, 22470, 263\rangle$		0.262	0.012	0.092	**0.000**	X	X	X	X	0.376	6.84E+54
c499_gr_rcs_w6	$\langle 1872, 2, 18870, 462\rangle$		0.310	0.046	0.164	0.043	0.083	0.042	0.062	**0.000**	0.391	1.07E+88
c880_gr_rcs_w7	$\langle 4592, 2, 61745, 1024\rangle$		0.223	0.110	0.142	0.110	**0.000**	0.000	0.000	0.003	0.845	1.37E+278
example2_gr_2pin_w6	$\langle 3603, 2, 41023, 350\rangle$		0.112	**0.000**	0.026	0.000	0.005	0.005	0.005	0.005	0.756	2.78E+159
example2_gr_rcs_w6	$\langle 2664, 2, 27684, 476\rangle$		0.176	0.050	0.079	0.054	0.056	0.005	**0.000**	0.005	0.722	1.47E+263
term1_gr_2pin_w4	$\langle 746, 2, 3964, 31\rangle$		0.199	**0.000**	0.077	0.002	X	X	X	X	0.141	7.68E+39
term1_gr_rcs_w4	$\langle 808, 2, 3290, 57\rangle$		0.252	**0.000**	0.090	0.017	X	X	X	X	0.175	4.97E+55
too_large_gr_rcs_w7	$\langle 3633, 2, 50373, 1069\rangle$		0.156	0.026	0.073	**0.000**	X	X	X	X	0.608	7.73E+182
too_large_gr_rcs_w8	$\langle 4152, 2, 57495, 1330\rangle$		0.147	**0.000**	0.038	0.020	X	X	X	X	0.750	8.36E+246
vda_gr_rcs_w9	$\langle 6498, 2, 130997, 2402\rangle$		0.088	0.009	0.030	**0.000**	X	X	X	X	0.749	5.04E+300

Table 5.4: Table showing the log-relative error Δ of Relsat and four versions of Markov-LB combined with SampleSearch and SampleCount respectively for FPGA routing instances after 10 hours of CPU time.

exactly.

Results on FPGA routing instances

Table 5.4 shows the results for FPGA routing instances. We see a similar behavior to the Langford and Latin square instances in that the average and the martingale order statistics schemes are better than other schemes with the average scheme performing the best. SampleSearch based Markov-LB yields better lower bounds than SampleCount based Markov-LB on 11 out of the 17 instances. As in the other benchmarks, the lower bounds output by Relsat are inferior by several orders of magnitude.

Results on Linkage instances

The Linkage instances that we experimented with in Chapter 3 are generated by converting biological linkage analysis data into a Bayesian or Markov network.

Table 5.5 show the results for linkage instances used in the UAI 2006 evaluation [8]. Here, we compare Markov-LB with SampleSearch with VEC. The bound propagation scheme [5]

Problem	$\langle n,k,c,w\rangle$	Exact	Markov-LB with SampleSearch Min	Avg	Per	Ord	VEC	Best LB
			Δ	Δ	Δ	Δ	Δ	
BN_69.uai	$\langle 777, 7, 78, 47\rangle$	5.28E-54	0.082	**0.029**	0.031	0.034	0.140	1.56E-55
BN_70.uai	$\langle 2315, 5, 159, 87\rangle$	2.00E-71	0.275	**0.035**	0.101	0.046	0.147	6.24E-74
BN_71.uai	$\langle 1740, 6, 202, 70\rangle$	5.12E-111	0.052	**0.009**	0.019	0.017	0.035	5.76E-112
BN_72.uai	$\langle 2155, 6, 252, 86\rangle$	4.21E-150	0.021	**0.002**	0.004	0.007	0.023	2.38E-150
BN_73.uai	$\langle 2140, 5, 216, 101\rangle$	2.26E-113	0.172	**0.020**	0.059	0.026	0.121	1.19E-115
BN_74.uai	$\langle 749, 6, 66, 45\rangle$	3.75E-45	0.233	**0.035**	0.035	0.049	0.069	1.09E-46
BN_75.uai	$\langle 1820, 5, 155, 92\rangle$	5.88E-91	0.077	**0.005**	0.024	0.019	0.067	1.98E-91
BN_76.uai	$\langle 2155, 7, 169, 64\rangle$	4.93E-110	0.109	**0.015**	0.043	0.018	0.153	1.03E-111

Table 5.5: Table showing the log-relative error Δ of VEC and four versions of Markov-LB combined with SampleSearch for Linkage instances from the UAI 2006 evaluation after 3 hours of CPU time.

does not work on instances having determinism and therefore we do not report on it here. We clearly see that SampleSearch based Markov-LB yields higher lower bounds than VEC. Remember, however that the lower bounds output by VEC are correct (with probability 1) while the lower bounds output by Markov-LB are correct with probability > 0.99. We see that the average scheme is the best performing scheme.

Table 5.6 reports the results on Linkage instances encoded as Markov networks, used in the UAI 2008 evaluation [24]. VEC solves 10 instances exactly. On these instances, the lower bound output by SampleSearch based Markov-LB are quite accurate as evidenced by the small log relative error. On instances which VEC does not solve exactly, we clearly see that Markov-LB with SampleSearch yields higher lower bounds than VEC.

Comparing between different versions of Markov-LB, we see that the average scheme is overall the best performing scheme. The Martingale order statistics scheme is the second best scheme while the Martingale random permutation scheme is the third best.

Problem	$\langle n,k,c,w \rangle$	**Exact**	**Markov-LB with SampleSearch**				**VEC**	**Best LB**
			Min	**Avg**	**Per**	**Ord**		
			Δ	Δ	Δ	Δ	Δ	
pedigree18.uai	$\langle 1184,1,0,26 \rangle$	7.18E-79	0.062	0.004	0.011	0.016	**0.000**	7.18E-79
pedigree1.uai	$\langle 334,2,0,20 \rangle$	7.81E-15	0.034	0.020	0.020	0.020	**0.000**	7.81E-15
pedigree20.uai	$\langle 437,2,0,25 \rangle$	2.34E-30	0.208	0.010	0.011	0.029	**0.000**	2.34E-30
pedigree23.uai	$\langle 402,1,0,26 \rangle$	2.78E-39	0.093	0.007	0.016	0.019	**0.000**	2.78E-39
pedigree25.uai	$\langle 1289,1,0,38 \rangle$	2.12E-119	0.006	0.022	0.019	0.019	**0.024**	1.69E-116
pedigree30.uai	$\langle 1289,1,0,27 \rangle$	4.03E-88	0.014	0.039	0.039	0.035	**0.042**	1.85E-84
pedigree37.uai	$\langle 1032,1,0,25 \rangle$	2.63E-117	0.031	0.005	0.005	0.006	**0.000**	2.63E-117
pedigree38.uai	$\langle 724,1,0,18 \rangle$	5.64E-55	0.197	0.010	0.024	0.023	**0.000**	5.65E-55
pedigree39.uai	$\langle 1272,1,0,29 \rangle$	6.32E-103	0.039	0.003	**0.001**	0.007	0.000	7.96E-103
pedigree42.uai	$\langle 448,2,0,23 \rangle$	1.73E-31	0.024	0.009	0.007	0.010	**0.000**	1.73E-31
pedigree19.uai	$\langle 793,2,0,23 \rangle$		0.158	0.018	**0.000**	0.031	0.011	3.67E-59
pedigree31.uai	$\langle 1183,2,0,45 \rangle$		0.059	**0.000**	0.003	0.011	0.083	1.03E-70
pedigree34.uai	$\langle 1160,1,0,59 \rangle$		0.211	0.006	**0.000**	0.012	0.174	4.34E-65
pedigree13.uai	$\langle 1077,1,0,51 \rangle$		0.175	**0.000**	0.038	0.023	0.163	2.94E-32
pedigree40.uai	$\langle 1030,2,0,49 \rangle$		0.126	**0.000**	0.036	0.008	0.025	4.26E-89
pedigree41.uai	$\langle 1062,2,0,52 \rangle$		0.079	**0.000**	0.012	0.010	0.049	2.29E-77
pedigree44.uai	$\langle 811,1,0,29 \rangle$		0.045	0.002	0.007	0.009	**0.000**	2.23E-64
pedigree51.uai	$\langle 1152,1,0,51 \rangle$		0.150	0.003	0.027	**0.000**	0.139	1.01E-74
pedigree7.uai	$\langle 1068,1,0,56 \rangle$		0.127	**0.000**	0.019	0.009	0.101	6.42E-66
pedigree9.uai	$\langle 1118,2,0,41 \rangle$		0.072	**0.000**	0.009	0.009	0.028	1.41E-79

Table 5.6: Table showing the log-relative error Δ of VEC and four versions of Markov-LB combined with SampleSearch for Linkage instances from the UAI 2008 evaluation after 3 hours of CPU time.

Problem	$\langle n,k,c,w\rangle$	**Exact**	**Markov-LB with SampleSearch**				**VEC**	**Best LB**
			Min	**Avg**	**Per**	**Ord**		
			Δ	Δ	Δ	Δ	Δ	
fs-01.uai	$\langle 10,2,7,2\rangle$	5.00E-01	0.000	0.000	0.000	0.000	**0.000**	5.00E-01
fs-04.uai	$\langle 262,2,226,12\rangle$	1.53E-05	0.116	0.116	0.116	0.116	**0.000**	1.53E-05
fs-07.uai	$\langle 1225,2,1120,35\rangle$	9.80E-17	0.028	0.004	0.014	0.016	**0.079**	1.78E-15
fs-10.uai	$\langle 3385,2,3175,71\rangle$	7.89E-31	0.071	**0.064**	0.064	0.065	X	9.57E-33
fs-13.uai	$\langle 7228,2,6877,119\rangle$	1.34E-51	0.077	0.077	**0.077**	0.077	X	1.69E-55
fs-16.uai	$\langle 13240,2,12712,171\rangle$	8.64E-78	0.085	**0.019**	0.048	0.025	X	3.04E-79
fs-19.uai	$\langle 21907,2,21166,243\rangle$	2.13E-109	0.051	**0.050**	0.050	0.050	X	8.40E-115
fs-22.uai	$\langle 33715,2,32725,335\rangle$	2.00E-146	0.053	**0.006**	0.022	0.009	X	2.51E-147
fs-25.uai	$\langle 49150,2,47875,431\rangle$	7.18E-189	0.050	0.005	0.026	**0.004**	X	1.57E-189
fs-28.uai	$\langle 68698,2,67102,527\rangle$	9.83E-237	0.231	0.017	0.023	**0.011**	X	4.53E-237
fs-29.uai	$\langle 76212,2,74501,559\rangle$	6.82E-254	0.259	0.101	0.201	**0.027**	X	9.44E-255
mastermind_03_08_03	$\langle 1220,2,48,20\rangle$	9.79E-08	0.283	0.039	0.034	0.096	**0.000**	9.79E-08
mastermind_03_08_04	$\langle 2288,2,64,30\rangle$	8.77E-09	0.562	0.045	0.145	0.131	**0.000**	8.77E-09
mastermind_03_08_05	$\langle 3692,2,80,42\rangle$	8.90E-11	0.432	0.041	0.021	0.095	**0.000**	1.44E-10
mastermind_04_08_03	$\langle 1418,2,48,22\rangle$	8.39E-08	0.297	0.041	0.072	0.082	**0.000**	8.39E-08
mastermind_04_08_04	$\langle 2616,2,64,33\rangle$	2.20E-08	0.640	**0.026**	0.155	0.103	0.034	1.38E-08
mastermind_05_08_03	$\langle 1616,2,48,28\rangle$	5.30E-07	0.625	0.062	0.188	0.185	**0.000**	5.30E-07
mastermind_06_08_03	$\langle 1814,2,48,31\rangle$	1.80E-08	0.510	0.058	0.193	0.175	**0.000**	1.80E-08
mastermind_10_08_03	$\langle 2606,2,48,56\rangle$	1.92E-07	0.839	**0.058**	0.297	0.162	0.058	7.90E-08

Table 5.7: Table showing the log-relative error Δ of VEC and four versions of Markov-LB combined with SampleSearch for relational instances after 3 hours of CPU time.

Results on Relational instances

To recap, the relational instances are generated by grounding the relational Bayesian networks using the primula tool (see Chavira et al. [14] for more information). In Table 5.7, we report the results on instances with 10 Friends and Smoker networks and 6 mastermind networks from this domain which have between 262 to 76,212 variables. On the 11 friends and smokers network, we can see that as the problems get larger the lower bounds output by Markov-LB with SampleSearch are higher than VEC. This clearly indicates that Markov-LB with SampleSearch is more scalable than VEC. VEC solves exactly six out of the eight mastermind instances while on the remaining two instances Markov-LB with SampleSearch yields higher lower bounds than VEC.

5.4.4 Summary of Experimental Results

Based on our large scale experimental evaluation, we see that applying Markov inequality and its generalization to multiple samples generated by SampleSearch and IJGP-Sampling is more scalable than deterministic approaches for lower bounding the weighted counts like Variable elimination and conditioning (VEC), Relsat and improved bound propagation. Among the different versions of Markov-LB, we find that the average and martingale order statistics schemes consistently yield higher lower bounds and should be used.

5.5 Conclusion and Summary

In this chapter, we proposed a randomized approximation algorithm, *Markov-LB* for computing high confidence lower bounds on weighted counting tasks such as computing the probability of evidence in a Bayesian network, counting the number of solutions of a constraint network and computing the partition function of a Markov network. Markov-LB is based on importance sampling and the Markov inequality. Since a straight-forward application of the Markov inequality may lead to poor lower bounds, we proposed several improved measures such as the average scheme which utilizes the sample average and martingale schemes which utilize the maximum values from the sample weights. A highlight of Markov-LB is that it can be used with any scheme that outputs unbiased (or a lower bound on the unbiased) sample weights. We showed that Markov-LB applied to the IJGP-sampling and SampleSearch schemes presented in this thesis provide high confidence good quality lower bounds on most instances. Furthermore, the lower bounds are often better than those ouput by state-of-the-art schemes such as bound propagation [6], SampleCount [62], Relsat [115] and variable elimination and conditioning.

Chapter 6

AND/OR Importance Sampling

6.1 Introduction

Importance sampling [118] is a general scheme which can be used to approximate various weighted counting tasks defined over graphical models such as computing the probability of evidence in a Bayesian network, computing the partition function of a Markov network and counting the number of solutions of a constraint network. The main idea in importance sampling is to first transform the counting or summation problem to an expectation problem using a special distribution called the proposal or importance distribution and then to estimate the counts by a weighted average over the generated samples, also called the sample mean. As pointed out in Chapter 1, the quality of estimation of importance sampling is highly dependent on the variance of the weights or the sample mean and therefore over the past few years significant research effort has been devoted to reducing variance [118, 88]. In this chapter, we propose a family of variance reduction schemes in the context of graphical models called AND/OR importance sampling.

The central idea in AND/OR importance sampling is to exploit problem decomposition introduced by the conditional independencies in the graphical model. Recently, graph based problem decomposition was introduced in the context of systematic search in graphical models [23, 3, 37], using the notion of AND/OR search spaces, and was shown to substantially reduce search time; sometimes exponentially. The usual way of performing search is to systematically go over all possible instantiations of the variables, which can be organized in an OR search tree. In AND/OR search, additional AND nodes are interleaved with OR nodes to capture decomposition into conditionally independent subproblems.

We can organize the generated samples as covering a part of a full AND/OR search tree yielding an AND/OR sample tree. Likewise, the OR sample tree is the portion of the full OR search tree that is covered by the samples. The main intuition in moving from OR space to AND/OR space is that at AND nodes, we can combine samples in independent components to yield a virtual increase in the sample size. For example, if X is conditionally independent of Y given Z, we can consider N samples of X independently from those of Y given $Z = z$, thereby yielding an effective sample size of N^2 instead of the input N. Since the variance reduces as the number of samples increases (see for example [118]), the sample mean computed over the AND/OR sample tree has lower variance than the sample mean computed over the OR sample tree.

We can take this idea a step further and look at the AND/OR search graph [37] as the target for compiling the given set of samples. Since the AND/OR search graph is smaller than the AND/OR search tree, generating a partial cover of the AND/OR graph by the current set of samples yields the AND/OR sample graph mean with an even reduced variance. However, due to caching, computing the AND/OR sample graph mean is $O(w^*)$ (where w^* is the treewidth) times more expensive time-wise and $O(N)$ times more expensive space wise.

Finally, we combine AND/OR sampling with the w-cutset sampling scheme [7]. Since (based on the Rao-Blackwell theorem [12]) w-cutset sampling reduces the variance by combining sampling with exact inference; it leads to additional variance reduction.

We provide a thorough empirical evaluation of all the new estimators. We compare the impact of exploiting varying levels of graph decompositions via (a) AND/OR tree, (b) AND/OR graph and (c) their w-cutset generalizations on a variety of benchmark probabilistic networks. We demonstrate that exploiting more decomposition improves the accuracy of the estimates as a function of time in most cases. In particular, AND/OR tree sampling usually yields better estimates than (conventional) OR tree sampling, AND/OR graph sampling is superior to AND/OR tree sampling and w-cutset sampling (OR and AND/OR) yields additional improvement. Our results clearly show that the scheme that is the most aggressive in exploiting decomposition - AND/OR w-cutset graph importance sampling, is consistently superior.

The research presented in this chapter is based in part on [57, 58].

The rest of the chapter is organized as follows. In the next section, we present preliminaries on AND/OR search spaces for graphical models. In Section 6.3, we present AND/OR tree sampling and in Section 6.4, we prove that it yields lower variance than pure importance sampling. AND/OR graph sampling is presented in Section 6.5 and AND/OR w-cutset sampling is presented in Section 6.6. Section 6.7 presents empirical results and related work is presented in Section 6.8. We conclude in Section 6.9.

6.2 AND/OR search spaces

Given a mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$, we can compute the weighted counts (see Definition 11 in Chapter 1) by search, by accumulating probabilities over the search space of instantiated variables. In the simplest case, the algorithm traverses an OR search tree, whose nodes represent states in the space of partial assignments. This traditional search space does not capture any of the structural properties of the underlying graphical model, however. Introducing AND nodes into the search space can capture the conditional independencies in the graphical model.

The AND/OR search space is a well known problem solving approach developed in the area of heuristic search [100], that exploits the problem structure to decompose the search space. The AND/OR search space for graphical models was introduced in [37]. It is guided by a pseudo tree that spans the original graphical model.

DEFINITION **39** (**Pseudo Tree**). *Given an undirected graph $G = (\boldsymbol{V}, \boldsymbol{E}')$, a directed rooted tree $T = (\boldsymbol{V}, \boldsymbol{E})$ defined on all its nodes is called pseudo tree if any edge in $\boldsymbol{E}'$ which is not included in $\boldsymbol{E}$ is a back-arc, namely it connects a node to an ancestor in T.*

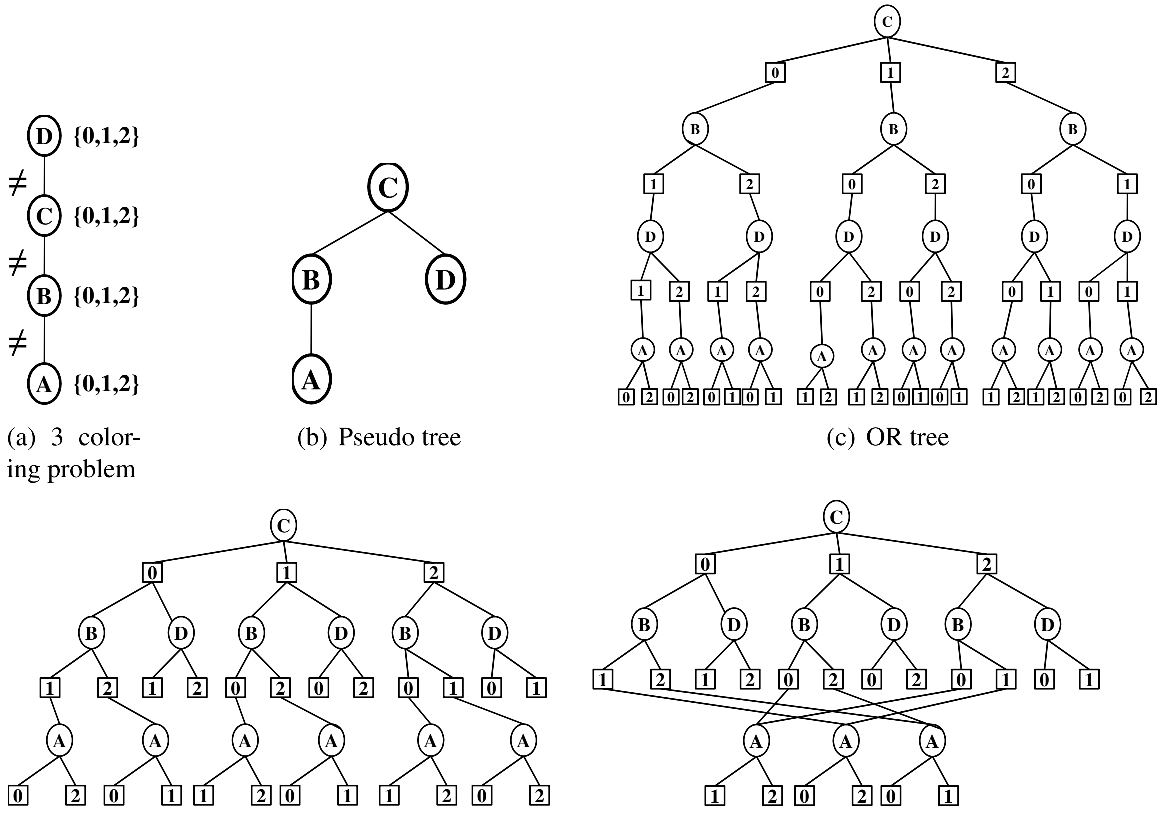
DEFINITION **40** (**AND/OR search tree**). *Given a mixed network $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$, its primal graph G and a pseudo tree T of G, the associated AND/OR search tree, has alternating levels of AND and OR nodes. The OR nodes are labeled X_i and correspond to the variables. The AND nodes are labeled x_i and correspond to the value assignments. The structure of the AND/OR search tree is based on T. The root of the AND/OR search tree is an OR node labeled by the root of T.*

*The children of an OR node X_i are AND nodes labeled with assignments x_i that are consistent with the assignments $(x_1, \ldots, x_{i-1})$ relative to **C** along the path from the root. Inconsistent assignments are not extended.*

The children of an AND node x_i are OR nodes labeled with the children of X_i in T.

DEFINITION **41** (**Solution Subtree**). *A solution subtree of a labeled AND/OR search tree (or graph) contains the root node. For every OR node it contains one of its child nodes and for each of its AND nodes it contains all its child nodes.*

Semantically, the OR states represent alternative assignments, whereas the AND states represent problem decomposition into independent subproblems, all of which need to be solved. When the pseudo tree is a chain, the AND/OR search tree coincides with the regular OR search tree.



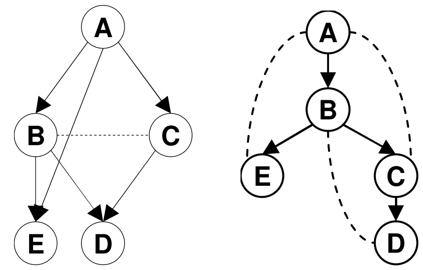
(a) 3 coloring problem (b) Pseudo tree (c) OR tree (d) AND/OR tree (e) AND/OR graph

Figure 6.1: AND/OR search spaces for graphical models

EXAMPLE **14.** *Figure 6.1(a) shows a constraint network for a 3-coloring problem over* 4 *variables. A possible pseudo tree is given in Figure 6.1(b). Figure 6.1(c) shows an OR tree and 6.1(d) shows an AND/OR tree.*

In [92, 36] it was shown how the posterior probability distribution associated with a mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$ can be expressed using AND/OR search spaces, and how queries of interest, such as computing the weighted counts, can be computed by a depth-first search traversal. All we need is to annotate the OR-to-AND arcs with weights derived from the relevant constraints $\mathbf{C}$ and functions $\mathbf{F}$, such that the product of weights on the arc of any solution subtree i.e. a full assignment **x** is equal to the weight $\prod_{i=1}^{m} F_i(\mathbf{x})$ of that solution.

DEFINITION **42** (**Weighted AND/OR tree).** *Given a mixed network $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$ and its AND/OR tree along some pseudo tree T, the weight $w(n,m)$ of an arc from an OR node n to an AND node m such that n is labeled with X_i and m is labeled with x_i, is the product of all functions $F \in \boldsymbol{F}$ which become fully instantiated by the last assignment from the root to X_i. A weighted AND/OR tree is the AND/OR tree annotated with weights.*



(a) Belief Network (b) Pseudo tree

P(A)

A	P(A)
0	.6
1	.4

P(B \| A)

A	B=0	B=1
0	1	0
1	.1	.9

P(C \| A)

A	C=0	C=1
0	.2	.8
1	.7	.3

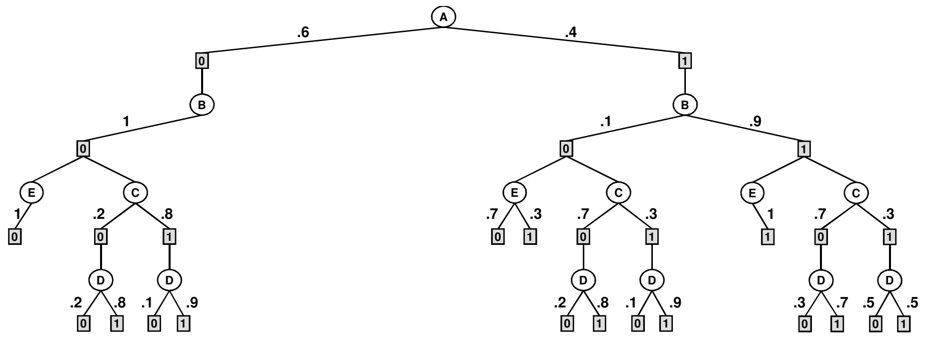
P(D \| B,C)

B	C	D=0	D=1
0	0	.2	.8
0	1	.1	.9
1	0	.3	.7
1	1	.5	.5

P(E \| A,B)

A	B	E=0	E=1
0	0	1	0
0	1	.5	.5
1	0	.7	.3
1	1	0	1

(c) CPTs



(d) Weighted AND/OR Search tree

Figure 6.2: Assigning weights to OR-to-AND arcs of an AND/OR search tree

EXAMPLE **15.** *Figure 6.2 shows how to assign weights to an AND/OR tree. Figure 6.2 (a) shows a Bayesian network, 6.2 (b) is a pseudo tree, and 6.2(c) shows the conditional probability tables. Figure 6.2(d) shows the weighted AND/OR search tree.*

The probability of evidence or weighted counts can be computed by traversing a weighted AND/OR tree in a DFS manner and computing the value of all nodes from leaves to the root [37, 92], as defined next. The value of a node is the weighted counts of the sub-tree that it roots.

DEFINITION **43** (**Value of a node for computing the weighted counts**). *The value of a node is defined recursively as follows. The value of leaf AND nodes are "1" and the value of leaf OR nodes are "0". Let $chi(n)$ denote the children and $v(n)$ denote the value of an OR or an AND node n respectively. If n is an OR node then:*

$$v(n) = \sum_{n' \in chi(n)} v(n')w(n, n')$$

If n is an AND node then:

$$v(n) = \prod_{n' \in chi(n)} v(n')$$

The value of the root node is equal to the weighted counts Z.

The AND/OR search tree may contain nodes that root identical subproblems. These nodes are unifiable and can be merged yielding a search graph whose size is smaller than the AND/OR search tree. Traversing the AND/OR search graph requires additional memory, however. A depth first search algorithm can cache previously computed results, and retrieve them when the same sub-problem is encountered. Some unifiable nodes can be identified based on their context which express the set of ancestor variables in the pseudo tree that completely determine a conditioned subproblem [37].

DEFINITION **44** (**context**). *Given a pseudo tree $T(\boldsymbol{X}, \boldsymbol{E})$ and a primal graph $G(\boldsymbol{X}, \boldsymbol{E}')$, the context of a node X_i in T denoted by $context_T(X_i)$ is the set of ancestors of X_i in T , ordered descendingly, that are connected in G to X_i or to descendants of X_i.*

DEFINITION **45** (**context minimal AND/OR graph**). *Given an AND/OR search tree, two OR nodes n_1 and n_2 are context unifiable if they have the same variable label X_i and the assignments of their contexts are identical. In other words, if **y** and **z** are the partial* *assignment of variables along the path to n_1 and n_2 respectively, and if their restriction to the context of X_i satisfy $\mathbf{y}_{context_T(X_i)} = \mathbf{z}_{context_T(X_i)}$, then n_1 and n_2 are unifiable. The context minimal AND/OR graph is obtained from the AND/OR search tree by merging all the context unifiable OR nodes.*

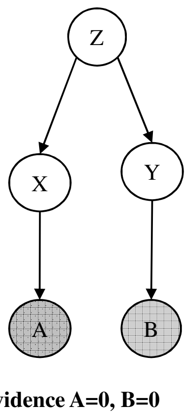
EXAMPLE **16.** *Figure 6.1(e) shows a context minimal AND/OR graph created from the AND/OR tree of Figure 6.1(d) by merging all context unifiable nodes.*

In the next five sections, we describe AND/OR importance sampling and its w-cutset generalizations; which are the main contributions of this chapter.

6.3 AND/OR Tree importance sampling

P(X|Z)	X=0	X=1	X=2
Z=0	0.3	0.4	0.3
Z=1	0.2	0.7	0.1

P(A|X)	A=0	A=1
X=0	0.1	0.9
X=1	0.2	0.8
X=2	0.6	0.4



P(Z)	Z=0	Z=1
	0.8	0.2

P(Y|Z)	Y=0	Y=1	Y=2
Z=0	0.5	0.1	0.4
Z=1	0.2	0.6	0.2

P(B|Y)	B=0	B=1
Y=0	0.2	0.9
Y=1	0.7	0.8
Y=2	0.1	0.4

Figure 6.3: A Bayesian network and its CPTs

We start by discussing computing expectation by parts; which forms the backbone of AND/OR importance sampling. We then present our first version based on AND/OR tree search.

6.3.1 Estimating Expectation by Parts

In Equation 1.14 (in Chapter 1), the expectation of a function defined over a set of variables is computed by summing over the Cartesian product of the domains of all variables. This method is clearly inefficient because it does not take into account the inherent decomposition in the graphical model as we illustrate below.

Consider the tree graphical model given in Figure 6.3. Let $A = a$ and $B = b$ be the evidence. By definition, the probability of evidence $P(a, b)$ is given by:

$$P(a,b) = \sum_{xyz \in XYZ} P(z)P(x|z)P(a|x)P(y|z)P(b|y) \tag{6.1}$$

Let $Q(Z, X, Y) = Q(Z)Q(X|Z)Q(Y|Z)$ be a proposal distribution. We can express $P(a, b)$ in terms of Q as:

$$P(a,b) = \sum_{xyz \in XYZ} \frac{P(z)P(x|z)P(a|x)P(y|z)P(b|y)}{Q(z)Q(x|z)Q(y|z)} Q(z)Q(x|z)Q(y|z) \tag{6.2}$$

We can now apply some simple symbolic manipulations, and rewrite Equation 6.2 as:

$$P(a,b) = \sum_{z \in Z} \frac{P(z)Q(z)}{Q(z)} \sum_{x \in X} \frac{P(x|z)P(a|x)Q(x|z)}{Q(x|z)} \sum_{y \in Y} \frac{P(y|z)P(b|y)Q(y|z)}{Q(y|z)} \tag{6.3}$$

By definition of conditional expectation[1]:

$$\mathbb{E}\left[\frac{P(x|z)P(a|x)}{Q(x|z)} \mid z\right] = \sum_{x \in X} \frac{P(x|z)P(a|x)Q(x|z)}{Q(x|z)} \tag{6.4}$$

[1]Because, the expectation is always taken with respect to the component of the proposal distribution in the denominator, we write $\mathbb{E}_Q[X]$ as $\mathbb{E}[X]$ for clarity.

and

$$\mathbb{E}\left[\frac{P(y|z)P(b|y)}{Q(y|z)} \mid z\right] = \sum_{y\in Y}\frac{P(y|z)P(b|y)Q(y|z)}{Q(y|z)} \tag{6.5}$$

Substituting Equations 6.4 and 6.5 in Equation 6.3, we get:

$$P(a,b) = \sum_{z\in Z}\frac{P(z)}{Q(z)}\mathbb{E}\left[\frac{P(x|z)P(a|x)}{Q(x|z)} \mid z\right]\mathbb{E}\left[\frac{P(y|z)P(b|y)}{Q(y|z)} \mid z\right]Q(z) \tag{6.6}$$

By definition (of expectation), we can rewrite Equation 6.6 as:

$$P(a,b) = \mathbb{E}\left[\frac{P(z)}{Q(z)}\mathbb{E}\left[\frac{P(x|z)P(a|x)}{Q(x|z)} \mid z\right]\mathbb{E}\left[\frac{P(y|z)P(b|y)}{Q(y|z)} \mid z\right]\right] \tag{6.7}$$

We will refer to Equations of the form 6.7 as *expectation by parts*. If the domain size of all variables is $d = 3$, for example, computing $P(a,b)$ using Equation 6.2 would require summing over $d^3 = 3^3 = 27$ terms while computing $P(a,b)$ using Equation 6.7 would require summing over $d + d^2 + d^2 = 3 + 3^2 + 3^2 = 21$ terms.

We will now describe how to estimate $P(a,b)$ using Equation 6.7. Assume that we are given N samples $(z^1, x^1, y^1), \ldots, (z^N, x^N, y^N)$ generated from Q. Let $\{0,1\}$ be the domain of Z and let $Z = 0$ and $Z = 1$ be sampled N_0 and N_1 times respectively. We define two sets $S(j) = \{k|k \in \{1,\ldots,N\}$ and $z^k = j\}$ for $j \in \{0,1\}$ which store the indices of the samples in which the value j is assigned to Z.

We can unbiasedly estimate $\mathbb{E}\left[\frac{P(x|z)P(a|x)}{Q(x|z)} \mid z\right]$ and $\mathbb{E}\left[\frac{P(y|z)P(b|y)}{Q(y|z)} \mid z\right]$ by replacing the expectation by the sample average. The unbiased estimates denoted by $\widehat{g}_X(Z = j)$ and

$\widehat{g}_Y(Z=j); j \in \{0,1\}$ are given by:

$$\widehat{g}_X(Z=j) = \frac{1}{N_j} \sum_{i \in S(j)} \frac{P(x^i|Z=j)P(a|x^i)}{Q(x^i|Z=j)}, \tag{6.8}$$

$$\widehat{g}_Y(Z=j) = \frac{1}{N_j} \sum_{i \in S(j)} \frac{P(y^i|Z=j)P(b|y^i)}{Q(y^i|Z=j)} \tag{6.9}$$

Substituting the unbiased estimates for $\mathbb{E}\left[\frac{P(x|z)P(a|x)}{Q(x|z)} \mid z\right]$ and $\mathbb{E}\left[\frac{P(y|z)P(b|y)}{Q(y|z)} \mid z\right]$ in Equation 6.7, we get the following unbiased estimate of $P(a,b)$:

$$\widehat{P}(a,b) = \mathbb{E}\left[\frac{P(z)}{Q(z)}\widehat{g}_X(Z=j)\widehat{g}_Y(Z=j)\right] \tag{6.10}$$

Given samples $(z^1,\ldots,z^N)$ generated from $Q(Z)$, we can unbiasedly estimate $\widehat{P}(a,b)$ by replacing the expectation in Equation 6.10 by the sample average as given below:

$$\widehat{P}_{ao-is}(a,b) = \frac{1}{N}\sum_{i=1}^{N} \frac{P(z^i)}{Q(z^i)}\widehat{g}_X(z^i)\widehat{g}_Y(z^i) \tag{6.11}$$

where $\widehat{P}_{ao-is}(a.b)$ stands for an AND/OR estimate of $P(a,b)$. Based on our assumption that $Z=0$ and $Z=1$ are sampled N_0 and N_1 times respectively, we can collect together all samples in which the value $Z=j, j \in \{0,1\}$ is generated and rewrite Equation 6.10 as:

$$\widehat{P}_{ao-is}(a,b) = \frac{1}{N}\sum_{j=0}^{1} \frac{N_j P(Z=j)}{Q(Z=j)}\widehat{g}_X(Z=j)\widehat{g}_Y(Z=j) \tag{6.12}$$

It is easy to show that $\mathbb{E}[\widehat{P}_{ao-is}(a,b)] = P(a,b)$, namely $\widehat{P}_{ao-is}$ is unbiased.

Conventional importance sampling, on the other hand would estimate $P(a, b)$ as follows:

$$\widehat{P}_{is}(a, b) = \frac{1}{N} \sum_{i=1}^{N} \frac{P(z^i)P(x^i|z^i)P(y^i|z^i)P(a|x^i)P(b|y^i)}{Q(z^i)Q(x^i|z^i)Q(y^i|z^i)} \tag{6.13}$$

As before assuming that $Z = 0$ and $Z = 1$ are sampled N_0 and N_1 times respectively, sets $S(j)$ for $j \in \{0, 1\}$ and by collecting together all samples in which the value $Z = j$, $j \in \{0, 1\}$ is generated, we can rewrite Equation 6.13 as:

$$\widehat{P}_{is}(a, b) = \frac{1}{N} \sum_{j=0}^{1} \frac{N_j P(Z = j)}{Q(Z = j)} \left(\frac{1}{N_j} \sum_{i \in S(j)} \frac{P(x^i|Z = j)P(y^i|Z = j)P(a|x^i)P(b|y^i)}{Q(x^i|Z = j)Q(y^i|Z = j)} \right) \tag{6.14}$$

For simplicity denote:

$$\widehat{g}_{X,Y}(Z = j) = \frac{1}{N_j} \sum_{i \in S(j)} \frac{P(x^i|Z = j)P(y^i|Z = j)P(a|x^i)P(b|y^i)}{Q(x^i|Z = j)Q(y^i|Z = j)}$$

and rewrite Equation 6.14 as:

$$\widehat{P}_{is}(a, b) = \frac{1}{N} \sum_{j=0}^{1} \frac{N_j P(Z = j)}{Q(Z = j)} \widehat{g}_{X,Y}(Z = j) \tag{6.15}$$

It is easy to show that $\widehat{g}_{X,Y}(Z = j)$ is an unbiased estimate of $\mathbb{E}\left[\frac{P(x|z)P(y|z)P(a|x)P(b|y)}{Q(x|z)Q(y|z)}|z\right]$, namely,

$$\mathbb{E}\left[\widehat{g}_{X,Y}(Z = j)\right] = \mathbb{E}\left[\frac{P(x|z)P(y|z)P(a|x)P(b|y)}{Q(x|z)Q(y|z)}|z\right] \tag{6.16}$$

Let us now compare $\widehat{P}_{ao-is}$ given by Equation 6.12 with $\widehat{P}_{is}$ given by Equation 6.15. The only difference is that in $\widehat{P}_{ao-is}$, we compute a product of $\widehat{g}_X(Z = j)$ and $\widehat{g}_Y(Z = j)$ instead of $\widehat{g}_{XY}(Z = j)$. The product of $\widehat{g}_X(Z = j)$ and $\widehat{g}_Y(Z = j)$ combines an estimate

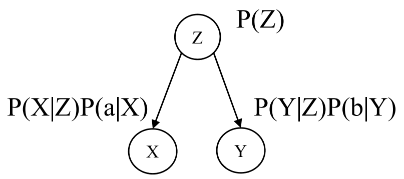


Figure 6.4: A pseudo tree of the Bayesian network given in Figure 6.3(a) in which each variable is annotated with its bucket function.

over two separate quantities defined over the random variables $X|Z = z$ and $Y|Z = z$ respectively from the generated samples. While in conventional importance sampling, we estimate only one quantity defined over the joint random variable $XY|Z = z$ using the generated samples. Because the samples for $X|Z = z$ and $Y|Z = z$ are considered independently in Equation 6.12, N_j samples drawn over the joint random variable $XY|Z = z$ in Equation 6.15 correspond to $N_j \times N_j = N_j^2$ virtual samples in Equation 6.12. Consequently, because of a larger sample size our new estimation technique will likely have lower error than the conventional approach.

6.3.2 Estimating weighted counts using an AND/OR sample tree

In this subsection, we generalize the idea of estimating expectation by parts using an AND/OR search tree [37]. We will define an AND/OR sample tree which is a restriction of the full AND/OR search space to the generated samples. On this AND/OR sample tree, we define a new sample mean and show that it yields an unbiased estimate of the weighted counts. We start with some required definitions.

DEFINITION **46** (**Bucket function**). *Given a mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$ and a rooted pseudo tree $T(\boldsymbol{X}, \boldsymbol{E})$, the bucket function of X_i relative to T, denoted by B_{T,X_i} is the product of all functions and constraints in $\mathcal{M}$ which mention X_i but do not mention any variables that are descendants of X_i in T.*

EXAMPLE **17.** *Figure 6.4 shows a possible pseudo tree over the non evidence variables of* *the Bayesian network given in Figure 6.3. Each variable in the pseudo tree is annotated with its bucket function. Note that after instantiating the evidence variable A to a, the CPT $P(A|X)$ yields a function $P(a|X)$ defined over X. Similarly, after instantiating B to the value b, the CPT $P(B|Y)$ yields a function $P(b|Y)$ defined over Y. Therefore, by definition, the bucket function at X is the product of $P(a|X)$ and $P(X|Z)$ while the bucket function at Y is the product of $P(b|Y)$ and $P(Y|Z)$. The bucket function of Z is $P(Z)$.*

DEFINITION **47 (AND/OR Sample Tree).** *Given a mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$, a pseudo tree $T(\boldsymbol{X}, \boldsymbol{E})$, a proposal distribution along the pseudo tree $Q(\boldsymbol{X}) = \prod_{i=1}^{n} Q_i(X_i| \mathbf{Y}_i)$ such that $\mathbf{Y}_i \subseteq context_T(X_i)$ [2], a sequence of samples $\mathbf{S}$ and a complete AND/OR search tree ϕ_{AOT}, an AND/OR sample tree S_{AOT} is obtained from ϕ_{AOT} by removing all nodes and corresponding edges which do not appear in* ***S****.*

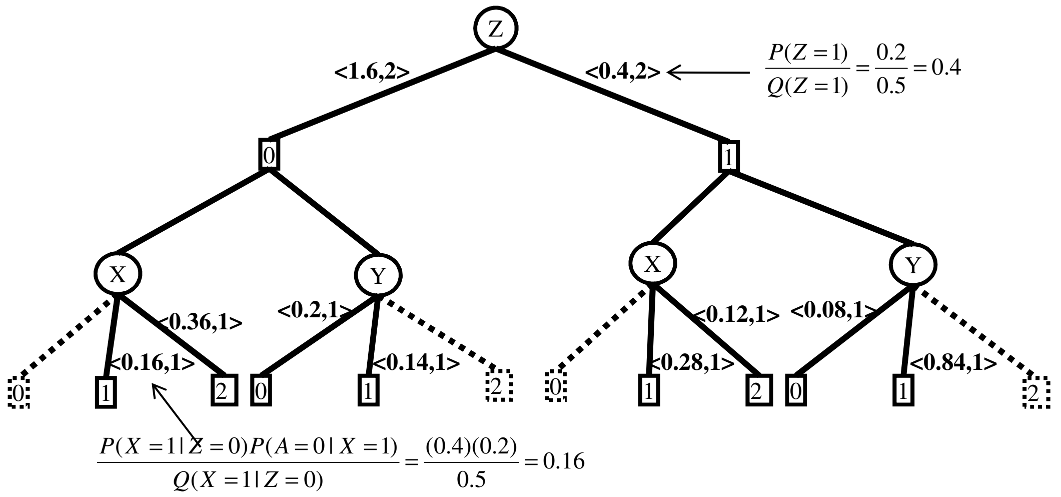
A path from the root of S_{AOT} to a node n is denoted by π_n. If n is an OR node labeled with X_i or an AND node labeled with x_i, the path will be denoted by $\pi_n(X_i)$ or $\pi_n(x_i)$ respectively. The assignment sequence along the path π_n, denoted by $A(\pi_n)$ is the set of assignments associated with the sequence of AND nodes along π_n:

$$A(\pi_n(X_i)) = \{x_1, \ldots, x_{i-1}\}$$

$$A(\pi_n(x_i)) = \{x_1, \ldots, x_i\}$$

The set of variables associated with OR nodes along the path π_n is denoted by $V(\pi_n)$: $V(\pi_n(X_i)) = \{X_1, \ldots, X_{i-1}\}$, $V(\pi_n(x_i)) = \{X_1, \ldots, X_i\}$. Clearly, if an OR node n is labeled X_i then $V(\pi_n)$ is the set of variables mentioned on the path from the root to X_i in the pseudo tree T, denoted by $path_T(X_i)$.

[2] For simplicity, we assume that $\mathbf{Y}_i$ is a subset of context of X_i. When the proposal distribution is specified externally, it may not obey this constraint. In that case, we construct a pseudo tree from a graph G obtained by combining the primal graphs of the proposal distribution and the mixed network.


Z
<1.6,2>
<0.4,2>
$\frac{P(Z=1)}{Q(Z=1)} = \frac{0.2}{0.5} = 0.4$
0
1
X
Y
X
Y
<0.36,1>
<0.2,1>
<0.12,1>
<0.08,1>
<0.16,1>
<0.14,1>
<0.28,1>
<0.84,1>
0
1
2
0
1
2
0
1
2
0
1
2
$\frac{P(X=1 \mid Z=0)P(A=0 \mid X=1)}{Q(X=1 \mid Z=0)} = \frac{(0.4)(0.2)}{0.5} = 0.16$

Samples: (a) $Z = 0, X = 1, Y = 0$, (b) $Z = 0, X = 2, Y = 1$,
(c) $Z = 1, X = 1, Y = 1$ and (d) $Z = 1, X = 2, Y = 0$.

Figure 6.5: Figure showing four samples arranged on an AND/OR sample tree.

The arc-label for an OR node n to an AND node m in S_{AOT}, where X_i labels n and x_i labels m, is a pair $\langle w(n,m), \#(n,m)\rangle$ where:

- $w(n,m) = \frac{B_{T,X_i}(x_i, A(\pi_n))}{Q_i(x_i|A(\pi_n))}$ *is called the weight of the arc.*
- $\#(n,m)$ *is the frequency of the arc. Namely, it is equal to the number of times the partial assignment $A(\pi_m)$ occurs in* ***S****.*

where B_{T,X_i} be the bucket function of X_i (see Definition 46).

We define an OR sample tree as an AND/OR sample tree whose pseudo tree is a chain.

EXAMPLE **18.** *Consider again the Bayesian network given in Figure 6.3. Assume that the proposal distribution is given by $Q(X,Y,Z) = Q(X)Q(Y)Q(Z)$ where $Q(X)$, $Q(Y)$ and $Q(Z)$ are uniform distributions. Figure 6.5 shows a full AND/OR search tree over the Bayesian network and four hypothetical random samples drawn from Q. The AND/OR sample tree is obtained by removing the dotted edges and nodes which are not sampled from the full AND/OR search tree. Each arc from an OR node to an AND node in the AND/OR* *sample tree is labeled with appropriate frequencies and weights according to Definition 47. Figure 6.5 shows the derivation of weights for two arcs.*

We can now compute an approximation of node values by mimicking the value computation on the AND/OR sample tree.

DEFINITION **48** (**Value of a node).** *Given an AND/OR sample tree (or graph) S_{AOT} and an initialization function $\psi_l(V(\pi_l))$ which defines the value of each leaf AND node l, the value of a node n, denoted by $v(n)$ is defined recursively as follows. The value of leaf AND node l is given by the initialization function ψ_l and the value of leaf OR node is 0. If n is an AND node then:*

$$v(n) = \prod_{n' \in chi(n)} v(n')$$

and if n is an OR node then

$$v(n) = \frac{\sum_{n' \in chi(n)} \#(n,n') \times w(n,n') \times v(n')}{\sum_{n' \in chi(n)} \#(n,n')}$$

We will show that the value of an OR node n is equal to an unbiased estimate of the conditional expectation of the subproblem conditioned on the assignment from the root to n (see Theorem 19). Note that unlike previous work on AND/OR search spaces [37] in which the leaf AND nodes are always initialized to one, we require an initialization function to accommodate w-cutset extensions to AND/OR sample trees presented in Section 6.6.

DEFINITION **49** (**AND/OR Sample Tree Mean).** *Given an AND/OR sample tree S_{AOT} with arcs labeled according to Definition 47, the* **AND/OR Sample Tree mean** *denoted by $\widehat{Z}_{ao-tree}$ is the value of the root node of S_{AOT}, where the value of each leaf AND node of S_{AOT} is initialized to "1".*

EXAMPLE **19.** *The calculations involved in computing the AND/OR sample tree mean are shown in Figure 6.6. Each node in Figure 6.6 is marked with a value that is computed*

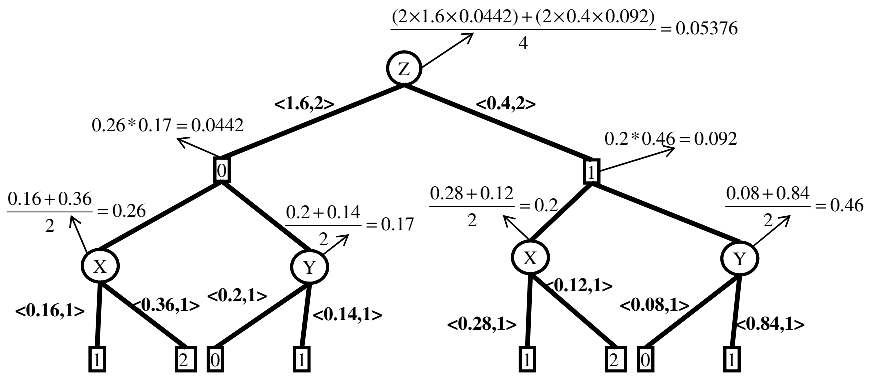


Figure 6.6: Value computation on an AND/OR sample tree

recursively using Definition 49. The value of OR nodes X and Y given $Z = j \in \{0, 1\}$ is equal to $\widehat{g}_X(Z = j)$ and $\widehat{g}_Y(Z = j)$ respectively defined in Equations 6.8 and 6.9. The value of the root node labeled by Z is equal to the AND/OR sample tree mean which is equal to the sample mean computed by parts in Equation 6.12.

THEOREM **19.** *The AND/OR sample tree mean $\widehat{Z}_{ao-tree}$ is an unbiased estimate of the weighted counts Z.*

Proof. We prove by induction that the value of any OR node n is an unbiased estimate of the conditional expectation of the subproblem rooted at n (conditioned on the assignment from the root to n).

Consider the expression for weighted counts:

$$Z = \sum_{\mathbf{x} \in \mathbf{X}} \prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x}) \tag{6.17}$$

Let $T(\mathbf{X}, \mathbf{E})$ be a pseudo tree, $path_T(X_i)$ be the set of variables along the path from root up to node X_i (note that $path_T(X_i)$ does not include X_i) in T and B_{T,X_i} be the bucket function (see Definition 46) of X_i w.r.t. T. For any full assignment **x**, we have:

$$\prod_{i=1}^{n} B_{T,X_i}(x_i, \mathbf{x}_{path_T(X_i)}) = \prod_{i=1}^{m} F_i(\mathbf{x}) \prod_{j=1}^{p} C_j(\mathbf{x}) \tag{6.18}$$

To recap our notation, $\mathbf{x}_{path_T(X_i)}$ is the projection of the assignment **x** onto the subset $path_T(X_i)$ of **X**.

Substituting Equation 6.18 into Equation 6.17, we get:

$$Z = \sum_{\mathbf{x}\in\mathbf{X}} \prod_{i=1}^{n} B_{T,X_i}(x_i, \mathbf{x}_{path_T(X_i)}) \tag{6.19}$$

Let $Q(\mathbf{X}) = \prod_{i=1}^{n} Q_i(X_i|\mathbf{Y}_i)$ be the proposal distribution where $\mathbf{Y}_i \subseteq context_T(X_i)$. Because $context_T(X_i) \subseteq path_T(X_i)$, we can express Q as:

$$Q(\mathbf{X}) = \prod_{i=1}^{n} Q_i(X_i|\mathbf{Y}_i) = \prod_{i=1}^{n} Q_i(X_i|path_T(X_i)) \tag{6.20}$$

We can express Z in Equation 6.19 in terms of Q as:

$$Z = \sum_{\mathbf{x}\in\mathbf{X}} \prod_{i=1}^{n} \frac{B_{T,X_i}(\mathbf{x}_{path_T(X_i)})}{Q_i(x_i|\mathbf{x}_{path_T(X_i)})} Q_i(x_i|\mathbf{x}_{path_T(X_i)}) \tag{6.21}$$

Using the notation $\mathbf{x}_i = (x_1, \ldots, x_i)$ and $\mathbf{x}_{i,path_T(X_j)}$ as the projection of $\mathbf{x}_i$ on $path_T(X_j)$ and migrating the functions to the left of summation variables which it does not reference, we can rewrite Equation 6.21 as:

$$\begin{aligned} Z \;=\; & \sum_{x_1\in X_1} \frac{B_{T,X_1}(x_1)}{Q_1(x_1)} Q_1(x_1) \times \ldots \times \\ & \sum_{x_i\in X_i} \frac{B_{T,X_i}(x_i, \mathbf{x}_{i-1,path_T(X_i)})}{Q_i(x_i|\mathbf{x}_{i-1,path_T(X_i)})} Q_i(x_i|\mathbf{x}_{i-1,path_T(X_i)}) \times \ldots \times \\ & \sum_{x_n\in X_n} \frac{B_{T,X_i}(x_i, \mathbf{x}_{n-1,path_T(X_n)})}{Q_n(x_n|\mathbf{x}_{n-1,path_T(X_n)})} Q_n(x_n|\mathbf{x}_{n-1,path_T(X_n)}) \end{aligned} \tag{6.22}$$

Using the definition of expectation and conditional expectation, we can rewrite Equation 6.22 as:

$$\begin{aligned} Z &= \mathbb{E}\left[\frac{B_{T,X_1}(x_1)}{Q_1(x_1)} \times \ldots \times \mathbb{E}\left[\frac{B_{T,X_i}(x_i, \mathbf{x}_{i-1,path_T(X_i)})}{Q_i(x_i|\mathbf{x}_{i-1,path_T(X_i)})} \times \ldots \times \right.\right. \\ & \left.\left. \mathbb{E}\left[\left. \frac{B_{T,X_n}(x_n, \mathbf{x}_{n-1,path_T(X_n)})}{Q_n(x_n|\mathbf{x}_{n-1,path_T(X_n)})} \right| \mathbf{x}_{n-1,path_T(X_n)}\right] \ldots \right| \mathbf{x}_{i-1,path_T(X_i)}\right] \ldots \Bigg] \qquad (6.23) \end{aligned}$$

Let $chi(X_i)$ be the set of children of X_i in the pseudo tree T and let us denote the component of conditional expectation at a node X_i, along an assignment $\mathbf{x}_{i-1,path_T(X_i)}$ by $V_{X_i}(X_i|\mathbf{x}_{i-1,path_T(X_i)})$. $V_{X_i}(X_i|\mathbf{x}_{i-1,path_T(X_i)})$ can be recursively defined as follows:

$$\begin{aligned} V_{X_i}(X_i|\mathbf{x}_{i-1,path_T(X_i)}) &= \mathbb{E}\left[\frac{B_{T,X_i}(x_i, \mathbf{x}_{i-1,path_T(X_i)})}{Q_i(x_i|\mathbf{x}_{i-1,path_T(X_i)})} \times \right. \\ & \left. \prod_{X_j \in chi(X_i)} V_{X_j}(X_j|x_i, \mathbf{x}_{i-1,path_T(X_i)}) \right| \mathbf{x}_{i-1,path_T(X_i)}\Bigg] \qquad (6.24) \end{aligned}$$

It is easy to see that Z equals $V_{X_1}(X_1)$, namely,

$$Z = \mathbb{E}\left[\frac{B_{T,X_1}(X_1)}{Q_1(X_1)} \prod_{X_j \in chi(X_1)} V_{X_j}(X_j|x_1)\right] = V_{X_1}(X_1) \qquad (6.25)$$

We will now derive an unbiased estimate of $V_{X_i}\ (X_i|\ \mathbf{x}_{i-1,path_T(X_i)})$. Assume that for all $X_j \in chi(X_i)$ in T, we have an unbiased estimate of $V_{X_j}(X_j|x_i, \mathbf{x}_{i-1,path_T(X_i)})$ denoted by $\widehat{v}_{X_j}\ (X_j|\ x_i, \mathbf{x}_{i-1,path_T(X_i)})$. Assume that given $\mathbf{x}_{i-1,path_T(X_i)}$, we have generated N samples $(x_i^1, \ldots, x_i^N)$ from $Q_i(X_i|\mathbf{x}_{i-1,path_T(X_i)})$. By replacing the (conditional) expectation by its sample average, we get the following unbiased estimate of $V_{X_i}\ (X_i|\ \mathbf{x}_{i-1,path_T(X_i)})$.

$$\widehat{v}_{X_i}(X_i|\mathbf{x}_{i-1,path_T(X_i)}) = \frac{1}{N}\sum_{a=1}^{N}\frac{B_{T,X_i}(x_i^a,\mathbf{x}_{i-1,path_T(X_i)})}{Q_i(x_i^a|\mathbf{x}_{i-1,path_T(X_i)})}\prod_{X_j\in chi(X_i)}\widehat{v}_{X_j}(X_j|x_i^a,\mathbf{x}_{i-1,path_T(X_i)}) \tag{6.26}$$

Assume that the domain of X_i is $\{x_{i,1},\ldots,x_{i,k}\}$. Also, assume that each value $x_{i,j}$ is sampled $N_{i,j}$ times. By collecting together all the samples in which the value $x_{i,j}$ is generated and substituting $N=\sum_{a=1}^{k}N_{i,a}$, we can rewrite Equation 6.26 as:

$$\widehat{v}_{X_i}(X_i|\mathbf{x}_{i-1,path_T(X_i)}) = \frac{\sum_{a=1}^{k}N_{i,a}\frac{B_{T,X_i}(x_{i,a},\mathbf{x}_{i-1,path_T(X_i)})}{Q_i(x_{i,a}|\mathbf{x}_{i-1,path_T(X_i)})}\prod_{X_j\in chi(X_i)}\widehat{v}_{X_j}(X_j|x_{i,a},\mathbf{x}_{i-1,path_T(X_i)})}{\sum_{a=1}^{k}N_{i,a}} \tag{6.27}$$

Next, we show that given an AND/OR sample tree S_{AOT} and the same samples from which $\widehat{v}_{X_i}(X_i|\mathbf{x}_{i-1,path_T(X_i)})$ is derived, the value of an OR node n in S_{AOT} labeled by X_i such that $A(\pi_n)=\mathbf{x}_{i-1,path_T(X_i)}$ is equal to $\widehat{v}_{X_i}(X_i|\mathbf{x}_{i-1,path_T(X_i)})$. Let us denote the k^{th} child AND node by m_k. By definition the frequencies and weights of the arcs from n to m_a are given by:

$$\#(n,m_a) = N_{i,a}$$

$$w(n,m_a) = \frac{B_{T,X_i}(x_{i,a},A(\pi_n))}{Q_i(x_{i,a}|A(\pi_n))} = \frac{B_{T,X_i}(x_{i,a},\mathbf{x}_{i-1,path_T(X_i)})}{Q_i(x_{i,a}|\mathbf{x}_{i-1,path_T(X_i)})}$$

By definition, the value of each AND node m_a is given by:

$$v(m_a) = \prod_{n'\in chi(m_a)} v(n')$$

Similarly by definition, the value of the OR node n is given by:

$$\begin{aligned} v(n) &= \frac{\sum_{a=1}^{k} \#(n,m_a) \times w(n,m_a) \times v(m_a)}{\sum_{a=1}^{k} \#(n,m_a)} \\ &= \frac{\sum_{a=1}^{k} \#(n,m_a) \times w(n,m_a) \times \prod_{n' \in chi(m_a)} v(n')}{\sum_{a=1}^{k} \#(n,m_a)} \end{aligned} \tag{6.28}$$

Substituting the expressions for $\#(n,m_a)$ and $w(n,m_a)$ in Equation 6.28, we get:

$$v(n) = \frac{\sum_{a=1}^{k} N_{i,a} \frac{B_{T,X_i}(x_{i,a},\mathbf{x}_{i-1,path_T(X_i)})}{Q_i(x_{i,a},\mathbf{x}_{i-1,path_T(X_i)})} \prod_{n' \in chi(m_a)} v(n')}{\sum_{a=1}^{k} N_{i,a}} \tag{6.29}$$

Assuming $v(n') = \widehat{v}_{X_j}(X_j|x_{i,a},\mathbf{x}_{i-1,path_T(X_i)})$, we can see that the right hand sides of Equations 6.27 and 6.29 are equal yielding $v(n) = \widehat{v}_{X_i}(X_i|\mathbf{x}_{i-1,path_T(X_i)})$. Namely, we have proved that if the value of a child OR node n' of a child AND node of an OR node n is equal to an unbiased estimate of the conditional expectation of the sub-problem rooted at n', then the value of the OR node n is also an unbiased estimate of the conditional expectation of the sub-problem rooted at n.

Since this result is true for any OR node, the value of the root OR node is equal to an unbiased estimate of $Z = V_{X_1}(X_1)$, which is what we wanted to prove.

$\square$

We look now at the special case of an OR tree.

THEOREM **20.** *Given a mixed network $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$, a pseudo tree T, a proposal distribution $Q(\boldsymbol{X}) = \prod_{i=1}^{n} Q_i(X_i|context_T(X_i))$, N i.i.d. samples drawn from Q and a chain pseudo tree T' obtained by any topological linearization order o of T, the AND/OR sample tree mean computed on an OR sample tree based on T' is same as the conventional importance sampling estimate $\widehat{Z}$ defined in Equation 1.15.*

Proof. We prove this theorem by induction over the nodes of the pseudo tree $T(\mathbf{X}, \mathbf{E})$.

Base Case: Here we prove that the statement of the theorem is true for $n = 1$. Assume that T has only one variable X_1. In this case, the chain pseudo tree T' obtained by any topological linearization order o of T coincides with T. Given samples $\mathbf{S} = (x_1^1, \ldots, x_1^N)$ generated from a proposal distribution $Q_1(X_1)$ and bucket function $B_{T',X_1}(X_1)$ (see Definition 46), the conventional importance sampling estimate is:

$$\widehat{Z} = \frac{1}{N} \sum_{i=1}^{N} \frac{B_{T',X_1}(x_1^i)}{Q_1(x_1^i)} \tag{6.30}$$

Let $\{x_{1,1}, \ldots, x_{1,k}\}$ be the domain of X_1 and $N_{1,j}$ be the number of times the value $x_{1,j}$ appears in $\mathbf{S}$, $j \in \{1, \ldots, k\}$. Then, by collecting together all the samples in which the value $x_{1,j}$ is generated and substituting $N = \sum_{a=1}^{k} N_{1,a}$, we can rewrite Equation 6.30 as:

$$\widehat{Z} = \frac{\sum_{a=1}^{k} N_{1,a} \frac{B_{T',X_1}(x_{1,a})}{Q_1(x_{1,a})}}{\sum_{a=1}^{k} N_{1,a}} \tag{6.31}$$

Since T' has just one node, the OR sample tree based on T has just one OR node denoted by n. Let $m_1, \ldots, m_k$ be the child AND nodes of n. By definition, the value of all leaf AND nodes is 1, while the weight and frequency of the arcs (n, m_a) are given by:

$$\#(n, m_a) = N_{1,a}$$

$$w(n, m_a) = \frac{B_{T',X_1}(x_{1,a})}{Q_1(x_{1,a})}$$

Also, by definition, the value of the OR node n is given by:

$$\begin{aligned} v(n) &= \frac{\sum_{a=1}^{k} \#(n,m_a) w(n,m_a)}{\sum_{a=1}^{k} \#(n,m_a)} \\ &= \frac{\sum_{a=1}^{k} N_{1,a} \frac{B_{T',X_1}(x_{1,a})}{Q_1(x_{1,a})}}{\sum_{a=1}^{k} N_{1,a}} \end{aligned} \tag{6.32}$$

From, Equations 6.31 and 6.32, we have $\widehat{Z} = v(n)$ which proves the base case. Next, we prove the induction case.

Induction case: In this case, we assume that the statement of the theorem is true for n variables $\{X_1, \ldots, X_n\}$ and then prove that it is also true for $n+1$ variables $\{X_1, \ldots, X_{n+1}\}$.

Consider a pseudo tree T over $n+1$ variables with X_{n+1} as the root. Let T' be the chain pseudo tree corresponding to the topological linearization order o of T. By definition, both T and T' have the same root node X_{n+1}.

Given samples $\mathbf{S} = ((\mathbf{x}_n^1, x_{n+1}^1), \ldots, (\mathbf{x}_n^N, x_{n+1}^N))$ generated from the proposal distribution $Q(\mathbf{X}_n, X_{n+1})$, the conventional importance sampling estimate is given by:

$$\widehat{Z} = \frac{1}{N} \sum_{i=1}^{N} \frac{\prod_{j=1}^{n+1} B_{T',X_j}(\mathbf{x}_n^i, x_{n+1}^i)}{\prod_{j=1}^{n+1} Q_j(\mathbf{x}_n^i, x_{n+1}^i)} \tag{6.33}$$

Let X_{n+1} have k values in its domain given by $\{x_{n+1,1}, \ldots, x_{n+1,k}\}$ and $N_{n+1,j}$ be the number of times the value $x_{n+1,j}$ appears in $\mathbf{S}$. Let $\mathbf{S}(x_{n+1,j}) \subseteq \mathbf{S}$ be the subset of all samples which mention the value $x_{n+1,j}$. Then, by collecting together all the samples in which the value $x_{n+1,j}$ is generated and substituting $N = \sum_{a=1}^{k} N_{n+1,a}$, we can rewrite Equation 6.33 as:

$$\widehat{Z} = \frac{\sum_{a=1}^{k} N_{n+1,a} \frac{B_{T',X_{n+1}}(x_{n+1,a})}{Q_{n+1}(x_{n+1,a})} \left[\frac{1}{N_{n+1,a}} \sum_{\mathbf{x}^k \in \mathbf{S}(x_{n+1,j})} \frac{\prod_{j=1}^{n} B_{T',X_j}(\mathbf{x}_n^k, x_{n+1,a})}{\prod_{j=1}^{n} Q_j(\mathbf{x}_n^k | x_{n+1,a})} \right]}{\sum_{a=1}^{k} N_{n+1,a}} \tag{6.34}$$

Without loss of generality, let X_1 be the child of X_{n+1} in T'. From the induction case assumption, the quantity in the brackets in Equation 6.34 is equal to the value of the OR node labeled by X_1 given $x_{n+1,a}$. Let the OR node be denoted by r_a. We can rewrite Equation 6.34 as:

$$\widehat{Z} = \frac{\sum_{a=1}^{k} N_{n+1,a} \frac{B_{T',X_{n+1}}(x_{n+1,a})}{Q_{n+1}(x_{n+1,a})} v(r_a)}{\sum_{a=1}^{k} N_{n+1,a}} \tag{6.35}$$

Consider the root OR node denoted by r of the OR sample tree which is labeled by X_{n+1}. r has k child AND nodes $m_1, \ldots, m_k$ which in turn have one child OR node each. By definition, the value of an AND node is the product of the values of all its child nodes. Since each AND node m_a $a = 1, \ldots, k$ has only one child OR node denoted by r_a in an OR sample tree, the value of m_a is equal to the value of r_a. Namely,

$$v(m_a) = v(r_a)$$

By definition, the weights of the arcs between r and m_a for $a = 1, \ldots, k$ are given by:

$$\#(r, m_a) = N_{n+1,a}$$

$$w(r, m_a) = \frac{B_{T',X_{n+1}}(x_{n+1,a})}{Q_{n+1}(x_{n+1,a})}$$

By definition, the value of the root OR node r denoted by $v(r)$ is given by:

$$v(r) = \frac{\sum_{a=1}^{k} \#(r, m_a) \times w(r, m_a) \times v(m_a)}{\sum_{a=1}^{k} \#(r, m_a)} = \frac{\sum_{a=1}^{k} N_{n+1,a} \frac{B_{T',X_{n+1}}(x_{n+1,a})}{Q_{n+1}(x_{n+1,a})} v(r_a)}{\sum_{a=1}^{k} N_{n+1,a}} \tag{6.36}$$

From Equations 6.35 and 6.36, we have $\widehat{Z} = v(r)$, which proves the induction case. Therefore, from the principle of induction, the proof follows. □

Algorithm 21: AND/OR Tree Importance Sampling

```
   Input: A Mixed Network ⟨$\mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C}$⟩, a pseudo tree $T(\mathbf{X}, \mathbf{E})$ and a proposal distribution
          $Q(\mathbf{X}) = \prod_{i=1}^{n} Q(X_i|context(X_i))$
   Output: AND/OR sample tree mean
 1 Generate samples $\mathbf{x}^1, \ldots, \mathbf{x}^N$ from $Q$;
 2 Build an AND/OR sample tree $S_{AOT}$ for the samples $\mathbf{x}^1, \ldots, \mathbf{x}^N$;
 3 Initialize all labeling functions $\langle w(n,m), \#(n,m) \rangle$ on each arc from an OR node $n$ to an
   AND node $m$ using Definition 47;
   // Start: Value computation phase
 4 Initialize the value of all leaf OR nodes $n$ to 0 and leaf AND nodes to 1;
 5 for every node $n$ from leaves to the root of $S_{AOT}$ do
 6     Let $chi(n)$ denote the child nodes of node $n$ ;
       // denote value of a node by $v(n)$
 7     if $n$ is an AND node then
 8         $v(n) = \prod_{n' \in chi(n)} v(n')$ ;
 9     else
10         $v(n) = \frac{\sum_{n' \in chi(n)} \#(n,n') \times w(n,n') \times v(n')}{\sum_{n' \in chi(n)} \#(n,n')}$.
   // End: Value computation phase
11 return $v$(root node of $S_{AOT}$)
```

Algorithm AND/OR tree importance sampling is presented as Algorithm 21. In steps 1-3, the algorithm generates samples from Q and stores them on an AND/OR sample tree. The algorithm then computes the AND/OR sample tree mean over the AND/OR sample tree recursively from leaves to the root in steps $4 - 10$ (value computation phase).

We summarize the complexity of Algorithm 21 in the following theorem.

THEOREM **21 (Complexity of AND/OR tree importance sampling).** *Given N samples, a mixed network with n variables and a pseudo tree of depth h, the time and space complexity of computing the AND/OR sample tree mean is $O(nN)$ and $O(h)$ respectively.*

Proof. Because only N full samples are generated, the number of nodes of the AND/OR sample tree S_{AOT} is bounded by $O(nN)$. Because each node of S_{AOT} is processed only once during the value computation phase of Algorithm 21 (with each node processed in constant time) the overall time complexity is $O(nN)$. We could perform a depth first search traversal of the AND/OR sample tree (i.e. build it on the fly). In this case, we only have to store the current search path, whose maximum size is bounded by the depth h of the pseudo tree. Therefore, the space complexity is $O(h)$. $\square$

In summary, we defined AND/OR and OR sample tree mean and showed that they yield an unbiased estimate of the weighted counts Z. We also proved that conventional importance sampling mean can be derived over an OR sample tree. We provided an algorithm for computing the AND/OR sample tree mean and proved that it has the same time complexity as conventional importance sampling. In the next section, we will prove that AND/OR importance sampling is indeed a more powerful approach than OR importance sampling in that it has lower variance.

6.4 Variance Reduction

In this section, we prove that the AND/OR sample tree mean has lower (or equal) variance than that of the OR sample tree mean (which in turn is equal to conventional sample mean given by Equation 1.15).

THEOREM **22** (Variance Reduction)**.** *Given a pseudo tree T_{AO} of a mixed network $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$, a chain pseudo tree T_{OR} obtained from any topological linearization order o*

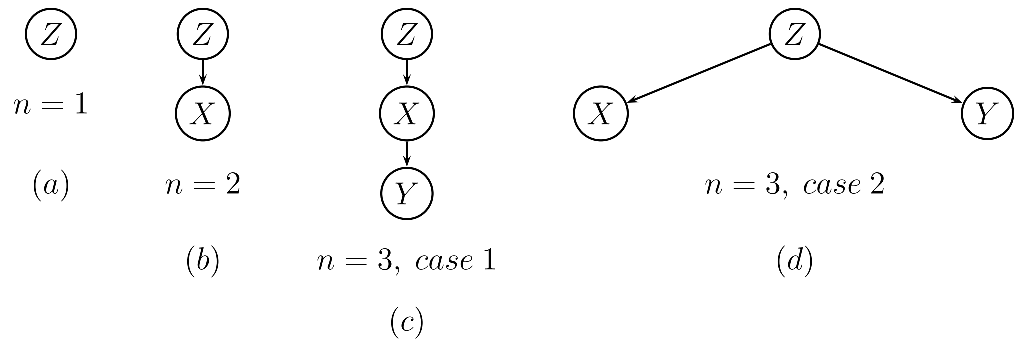


Figure 6.7: The possible pseudo trees for $n = 1, 2, 3$ excluding a permutation of the variables, where n is the number of variables.

of T_{AO}, and a set of samples $\mathbf{S} = (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^N)$, the variance of the AND/OR sample tree mean on the AND/OR sample tree of **S** *along T_{AO} is always smaller than or equal to the variance of the OR sample tree mean on the OR sample tree of* **S** *along T_{OR}.*

Proof. We prove this theorem by an induction over the number of variables n of T_{AO}.

Base case, n=1,2,3: Trivial to prove for $n = 1, 2$ and case 1 for $n = 3$ (see Figures 6.7(a), (b) and (c)). In these cases, the pseudo trees form a chain and therefore the OR and AND/OR sample tree means are equal.

Next, we consider the second case for $n = 3$ corresponding to the pseudo tree T_{AO} given in Figure 6.7(d). The OR pseudo tree T_{OR} corresponding to a possible topological linearization order o of T_{AO} is the one in Figure 6.7(c).

Without loss of generality, let $\{z_1, z_2\}$ be the domain of Z and let $Q(Z) \times Q(X|Z) \times Q(Y|Z)$ be the proposal distribution. Let us assume that we have generated N samples from Q and each value z_j of Z appears N_j times in the generated N samples. Let $\mathbf{S}_j = \{(x^{j,1}, y^{j,1}), \ldots, (x^{j,N_j}, y^{j,N_j})\}$ be the set of samples on X, Y given $Z = z_j$, $j \in \{1, 2\}$.

Let $B_{T_{OR},X}$ and $B_{T_{OR},Y}$ be the bucket functions (see Definition 46) associated with T_{OR} and let $B_{T_{AO},X}$ and $B_{T_{AO},Y}$ be the bucket functions associated with T_{AO}.

Consider the OR sample tree S_{ORT} relative to the samples $\mathbf{S} = \{\mathbf{S}_j |\ j \in \{1,2\}\}$ and T_{OR}. By definition, in the OR sample tree S_{ORT}, the value of the AND node labeled by z_j, denoted by a_j is given by:

$$v_{S_{ORT}}(a_j) = \frac{1}{N_j} \sum_{i=1}^{N_j} \frac{B_{T_{OR},Y}(y^{j,i}, z_j)}{Q(y^{j,i}|z_j)} \frac{B_{T_{OR},X}(x^{j,i}, z_j)}{Q(x^{j,i}|z_j)} \tag{6.37}$$

Consider the AND/OR sample tree S_{AOT} relative to the samples $\mathbf{S} = \{\mathbf{S}_j |\ j \in \{1,2\}\}$ and T_{AO}. By definition, in the AND/OR sample tree S_{AOT}, the value of the AND node b_j labeled by z_j is given by:

$$v_{S_{AOT}}(b_j) = \left(\frac{1}{N_j} \sum_{i=1}^{N_j} \frac{B_{T_{AO},Y}(y^{j,i}, z_j)}{Q(y^{j,i}|z_j)} \right) \left(\frac{1}{N_j} \sum_{i=1}^{N_j} \frac{B_{T_{OR},X}(x^{j,i}, z_j)}{Q(x^{j,i}|z_j)} \right) \tag{6.38}$$

Based on [64], we will show that variance of $v_{S_{ORT}}(a_j)$ is greater than or equal to the variance of $v_{S_{AOT}}(b_j)$. Goodman [64] proved that the variance of product of two independent random variables X and Y, denoted by $V[XY]$ can be expressed in terms of their variances $V[X]$, $V[Y]$ and expected values $\mathbb{E}[X]$ and $\mathbb{E}[Y]$ respectively as follows:

$$V(XY) = V[X]\mathbb{E}[Y]^2 + V[Y]\mathbb{E}[X]^2 + V[Y]V[X] \tag{6.39}$$

Given random samples $(x^1, \ldots, x^N)$ and $(y^1, \ldots, y^N)$, Goodman [64] considered the following two unbiased estimators of XY:

$$\widehat{xy} = \left(\frac{1}{N} \sum_{i=1}^{N} x^i \right) \left(\frac{1}{N} \sum_{i=1}^{N} y^i \right) \tag{6.40}$$

$$\widehat{xy} = \frac{1}{N} \sum_{i=1}^{N} x^i y^i \tag{6.41}$$

Let $V[\widehat{xy}]$ and $V[\widehat{xy}]$ denote the variances of the two unbiased estimators respectively.

Goodman [64] showed that:

$$V[\widehat{x}\widehat{y}] = \frac{V[X]\mathbb{E}[Y]^2}{N} + \frac{V[Y]\mathbb{E}[X]^2}{N} + \frac{V[Y]V[X]}{N^2} \tag{6.42}$$

$$V[\widehat{xy}] = \frac{V[X]\mathbb{E}[Y]^2}{N} + \frac{V[Y]\mathbb{E}[X]^2}{N} + \frac{V[Y]V[X]}{N} \tag{6.43}$$

Notice that Equations 6.42 and 6.43 differ only in the last term. In Equation 6.42, the last term is divided by N^2 while in Equation 6.43, the last term is divided by N. Therefore, if $N > 1$, then $V[\widehat{x}\widehat{y}] < V[\widehat{xy}]$.

Since $v_{S_{ORT}}(a_j)$ has the same form as the estimator $\widehat{xy}$ while $v_{S_{AOT}}(b_j)$ has the same form as the estimator $\widehat{x}\widehat{y}$, the variances of $v_{S_{ORT}}(a_j)$ and $v_{S_{AOT}}(b_j)$ obey:

$$\begin{aligned} V[v_{S_{ORT}}(a_j)] = \frac{1}{N_j}\Bigg[&\mathbb{E}\left[\left.\frac{B_{T_{OR},Y}(y,z_j)}{Q(y|z_j)}\right| z_j\right]^2 V\left[\left.\frac{B_{T_{OR},X}(x,z_j)}{Q(x|z_j)}\right| z_j\right] + \\ &\mathbb{E}\left[\left.\frac{B_{T_{OR},X}(x,z_j)}{Q(x|z_j)}\right| z_j\right]^2 V\left[\left.\frac{B_{T_{OR},Y}(y,z_j)}{Q(y|z_j)}\right| z_j\right] + \\ &V\left[\left.\frac{B_{T_{OR},X}(x,z_j)}{Q(x|z_j)}\right| z_j\right] V\left[\left.\frac{B_{T_{OR},Y}(y,z_j)}{Q(y|z_j)}\right| z_j\right]\Bigg] \end{aligned} \tag{6.44}$$

$$\begin{aligned} V[v_{S_{AOT}}(b_j)] = \frac{1}{N_j}\Bigg[&\mathbb{E}\left[\left.\frac{B_{T_{AO},Y}(y,z_j)}{Q(y|z_j)}\right| z_j\right]^2 V\left[\left.\frac{B_{T_{AO},X}(x,z_j)}{Q(x|z_j)}\right| z_j\right] + \\ &\mathbb{E}\left[\left.\frac{B_{T_{AO},X}(x,z_j)}{Q(x|z_j)}\right| z_j\right]^2 V\left[\left.\frac{B_{T_{AO},Y}(y,z_j)}{Q(y|z_j)}\right| z_j\right] + \\ &V\left[\left.\frac{B_{T_{AO},X}(x,z_j)}{Q(x|z_j)}\right| z_j\right] V\left[\left.\frac{B_{T_{AO},Y}(y,z_j)}{Q(y|z_j)}\right| z_j\right] / N_j\Bigg] \end{aligned} \tag{6.45}$$

By definition (see Definition 46), $B_{T_{OR},X}(x,z_j) = B_{T_{AO},X}(x,z_j)$ and $B_{T_{OR},Y}(y,z_j) =$

$B_{T_{AO},Y}(y, z_j)$. Therefore, from Equations 6.44 and 6.45, we can see that if $N_j > 1$, $V[v_{S_{AOT}}(b_j)] < V[v_{S_{ORT}}(a_j)]$. This proves the base case for $n = 3$.

Induction case: Here, we assume that the theorem holds for any pseudo tree T_{AO} having n variables or fewer and then we prove that the theorem also holds for any pseudo tree having $n+1$ variables.

Consider a pseudo tree T_{AO} having $n+1$ variables. Let T_{OR} be the pseudo tree obtained from any topological linearization order o of T_{AO}. Assume that the root node of T_{AO} and therefore that of T_{OR} is labeled by Z. Without loss of generality, let Z contain two child nodes labeled by X and Y in T_{AO} and let X be the child of Z and Y be the child of X in T_{OR}. Let $\{z_1, z_2\}$ be the domain of Z. Let us assume that each value z_j $j \in \{1, 2\}$ is sampled N_j times. Let $\mathbf{S}_j = \{(x^{j,1}, y^{j,1}), \ldots, (x^{j,N_j}, y^{j,N_j})\}$ be the samples on X, Y given $Z = z_j$, $j \in \{1, 2\}$. Consider the OR sample tree S_{ORT} relative to $\mathbf{S} = \{\mathbf{S}_j |\ j \in \{1, 2\}\}$ and T_{OR}. By definition, in the OR sample tree S_{ORT}, the value of the AND node labeled by z_j, denoted by c_j is given by:

$$v_{S_{ORT}}(c_j) = \frac{1}{N_j} \sum_{i=1}^{N_j} \frac{B_{T_{OR},Y}(y^{j,i}, z_j)}{Q(y^{j,i}|z_j)} \frac{B_{T_{OR},X}(x^{j,i}, z_j)}{Q(x^{j,i}|z_j)} \times \widehat{v}_{ORT}(\mathbf{X}|x^{j,i}, y^{j,i}, z_j) \qquad (6.46)$$

where $\widehat{v}_{ORT}(\mathbf{X}|x^{j,i}, y^{j,i}, z_j)$ is the estimate of conditional expectation of the sub-problem given $x^{j,i}, y^{j,i}, z_j$ in S_{ORT}.

By definition, in the AND/OR sample tree S_{AOT}, the value of the AND node d_j labeled by z_j is given by:

$$v_{S_{AOT}}(d_j) = \left(\frac{1}{N_j} \sum_{i=1}^{N_j} \frac{B_{T_{AO},Y}(y^{j,i}, z_j)}{Q(y^{j,i}|z_j)} \right) \left(\frac{1}{N_j} \sum_{i=1}^{N_j} \frac{B_{T_{OR},X}(x^{j,i}|z_j)}{Q(x^{j,i}|z_j)} \right) \times \widehat{v}_{AOT}(\mathbf{X}|x^{j,i}, y^{j,i}, z_j) \qquad (6.47)$$

where $\widehat{v}_{AOT}(\mathbf{X}|x^{j,i}, y^{j,i}, z_j)$ is the estimate of conditional expectation of the sub-problem given $x^{j,i}, y^{j,i}, z_j$ in S_{AOT}.

From the inductive assumption, the variance of $\widehat{v}_{AOT}(\mathbf{X}|x^{j,i}, y^{j,i}, z_j)$ is less than or equal to $\widehat{v}_{ORT}(\mathbf{X}|x^{j,i}, y^{j,i}, z_j)$. Assuming a weak case of equality, namely $V[\widehat{v}_{AOT}(\mathbf{X}|x^{j,i}, y^{j,i}, z_j)] = V[\widehat{v}_{ORT}(\mathbf{X}|x^{j,i}, y^{j,i}, z_j)]$, we can ignore these two quantities from Equations 6.46 and 6.47. Doing so, the expression for $v_{S_{ORT}}(c_j)$ given in Equation 6.46 has the same form as the expression for $v_{S_{ORT}}(a_j)$ given in Equation 6.37. Similarly, the expression for $v_{S_{AOT}}(d_j)$ given in Equation 6.47 has the same form as $v_{S_{AOT}}(b_j)$ given in Equation 6.38. As already proved in the base case, if $N_j > 1$, then $V[v_{S_{AOT}}(b_j)] < V[v_{S_{ORT}}(a_j)]$ and therefore it follows that $V[v_{S_{AOT}}(d_j)] < V[v_{S_{ORT}}(c_j)]$. This proves the induction case and the proof follows.

□

6.4.1 Remarks on Variance Reduction

From the proof of Theorem 22, it is easy to see that if each value $Z = z_j$ is sampled only once i.e. $N_j = 1$ then there is no variance reduction. We can tie variance reduction to the difference between the (virtual) number of AND/OR tree and OR tree samples (defined below).

DEFINITION **50 (Virtual AND/OR Tree samples).** *Given an AND/OR sample tree S_{AOT}, its virtual samples are all the solution subtrees of the AND/OR sample tree (see Definition 41).*

EXAMPLE **20.** *Figure 6.8(a) and Figure 6.8(b) show four samples arranged on an OR sample tree and an AND/OR sample tree respectively. The four samples correspond to 8 virtual samples (solution subtrees) on an AND/OR sample tree. The AND/OR sample tree includes for example the assignment $(C = 0, B = 2, D = 1, A = 1)$ which is not represented in the OR sample tree.*

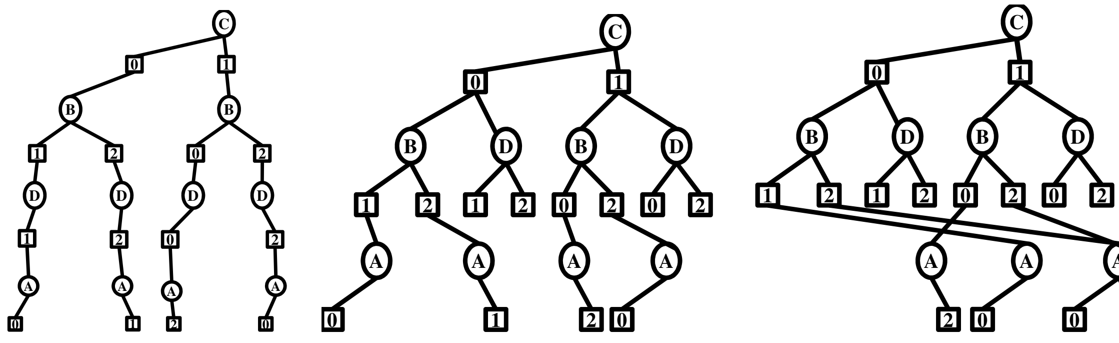

(a) OR sample tree (b) AND/OR sample tree (c) AND/OR sample graph

Figure 6.8: Figure showing 4 samples arranged on an OR sample tree, AND/OR sample tree and AND/OR sample graph. The AND/OR sample tree and graph are based on the pseudo tree given in Figure 6.1(b).

It is trivial to show that:

PROPOSITION **19.** *The number of virtual AND/OR tree samples is greater than or equal to the number of OR (tree) samples.*

So clearly, if the number of virtual samples rooted at an AND node labeled by x_i is equal to the number of OR samples, it does not matter whether x_i is a part of an OR tree or an AND/OR tree. Variance reduction therefore only occurs at AND nodes which are sampled more than once and which have at least two child nodes.

To summarize, in this section, we proved that the variance of AND/OR sample tree mean is less than or equal to the variance of the OR sample tree mean. We showed how variance reduction can be tied to the difference between the number of virtual AND/OR tree samples and the original number of samples.

6.5 Estimating Sample mean in AND/OR graphs

Next, we describe a more powerful strategy for estimating sample mean in the AND/OR space by moving from AND/OR trees to AND/OR graphs [37]. The idea is similar to

AND/OR graph search in that we merge nodes in an AND/OR sample tree, which are unifiable based on *context* (see Definition 44), to form an AND/OR sample graph. This can result in an even larger increase in the number of virtual samples.

DEFINITION **51** (**AND/OR Sample Graph**). *Given an AND/OR sample tree S_{AOT}, the AND/OR sample graph S_{AOG} is obtained from S_{AOT} by merging all OR nodes that have the same context.*

EXAMPLE **21.** *Figure 6.8(c) shows an AND/OR sample graph obtained from the AND/OR sample tree in Figure 6.8(b) by merging all context unifiable nodes.*

DEFINITION **52** (**AND/OR Sample Graph mean** $\widehat{Z}_{ao-graph}$). *The AND/OR sample graph mean, denoted by $\widehat{Z}_{ao-graph}$ is the value of the root node of S_{AOG} (see Definition 48) with initialization functions $\psi_l(V(\pi_l)) = 1$ for all leaf AND nodes.*

DEFINITION **53** (**Virtual AND/OR Graph samples**). *The set of virtual AND/OR graph samples is the set of solution subtrees of the AND/OR sample graph.*

EXAMPLE **22.** *Figure 6.8(b) and Figure 6.8(c) show four OR samples arranged over an AND/OR sample tree and an AND/OR sample graph respectively. The 8 virtual AND/OR tree samples (solution subtrees) correspond to 12 virtual AND/OR graph samples. The AND/OR sample graph includes for example the sample $(C = 0, B = 2, D = 1, A = 0)$ which is not part of the virtual samples of the AND/OR sample tree.*

It is trivial to show that:

PROPOSITION **20.** *The number of virtual AND/OR graph samples is greater than or equal to the number of virtual AND/OR tree samples if both are based on the same underlying pseudo tree.*

Since the AND/OR graph captures more virtual samples, the variance of AND/OR sample graph mean may be smaller than the variance of AND/OR sample tree mean. Formally, THEOREM **23.** *The variance of AND/OR sample graph mean $\widehat{Z}_{ao-graph}$ is less than or equal to that of AND/OR sample tree mean $\widehat{Z}_{ao-tree}$.*

Proof. Given a pseudo tree T, in an AND/OR sample graph S_{AOG}, the value of an OR node, denoted by n_{AOG} and labeled by X_i is computed using a subset of the samples that have the same assignment to the $context_T(X_i)$ of X_i while in an AND/OR sample tree S_{AOT}, the value of the corresponding OR node n_{AOT} is computed using a subset of the samples that have the same assignment to all variables along the path from root to X_i, denoted by $path_T(X_i)$. Because, $context_T(X_i) \subseteq path_T(X_i)$, the value of n_{AOG} is based on a larger (or equal) number of samples compared with the value of n_{AOT}. Because of its larger virtual sample size, from Equation 6.45, the variance of the value of n_{AOG} is less than (or equal to) the variance of the value of n_{AOT}.

□

The algorithm for computing the AND/OR sample graph mean is identical to that of AND/OR sample tree mean (Steps 4-10 of Algorithm 21). Obviously, the only difference is that we store the samples and perform value computations over an AND/OR sample graph instead of an AND/OR sample tree.

THEOREM **24** (Complexity of computing AND/OR graph sample mean)**.** *Given a graphical model with n variables, a pseudo tree T with maximum context size (treewidth) w^* and N samples, the time complexity of AND/OR graph sampling is $O(nNw^*)$ while its space complexity is $O(nN)$.*

Proof. Let X_j be the child node of X_i in T. Given N samples and maximum context size w^*, the number of edges emanating from AND nodes corresponding to X_i to OR nodes labeled by X_j in the AND/OR sample graph is bounded by $O(Nw^*)$. Since each such edge is visited just once in the value computation phase, the overall time complexity is

$O(nNw^*)$. To store N samples it takes $O(nN)$ space and therefore the space complexity is $O(nN)$. $\square$

6.6 AND/OR w-cutset sampling

We now show how the w-cutset sampling framework [7] can be combined with AND/OR importance sampling to reduce variance even further. Recall from Section 1.4.4 that the w-cutset sampling framework facilitates the use of the Rao-Blackwell theorem (see Theorem 4 in Chapter 1). To recap, given a w-cutset $\mathbf{K}$ (see Definition 16) of a mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$, a proposal distribution $Q(\mathbf{K})$ defined over the w-cutset, and a set of samples $(\mathbf{k}^1, \ldots, \mathbf{k}^N)$ generated from Q, the w-cutset estimate of weighted counts Z is given by:

$$\widehat{Z}_{or-wc-tree} = \frac{1}{N} \sum_{i=1}^{N} \frac{\sum_{\mathbf{r} \in \mathbf{R}} \prod_{j=1}^{m} F_j(\mathbf{r}, \mathbf{K} = \mathbf{k}^i) \prod_{k=1}^{p} C_k(\mathbf{r}, \mathbf{K} = \mathbf{k}^i)}{Q(\mathbf{c}^i)} \tag{6.48}$$

From the Rao-Blackwell theorem, it follows that the variance of $\widehat{Z}_{or-wc-tree}$ is less than or equal to the variance of $\widehat{Z}_{or-tree}$ when the two are based on the same samples on the w-cutset variables.

We now illustrate how w-cutset sampling can be easily combined with AND/OR sampling in the following example.

EXAMPLE **23.** *Consider the primal graph in Figure 6.9 (a). The minimal 1-cutset or cycle cutset contains three nodes $\{A, B, C\}$. Given a proposal distribution $Q(A, B, C) = Q(A)$ $Q(B|A)$ $Q(C|A)$, in 1-cutset sampling the variables A, B and C are sampled as if they form a chain pseudo tree as in Figure 6.9(c). Bucket elimination is applied on the remaining network to compute the sample weight. However once A is sampled, the remaining sub-problem is split into two components and therefore as presented in [91] we can organize*

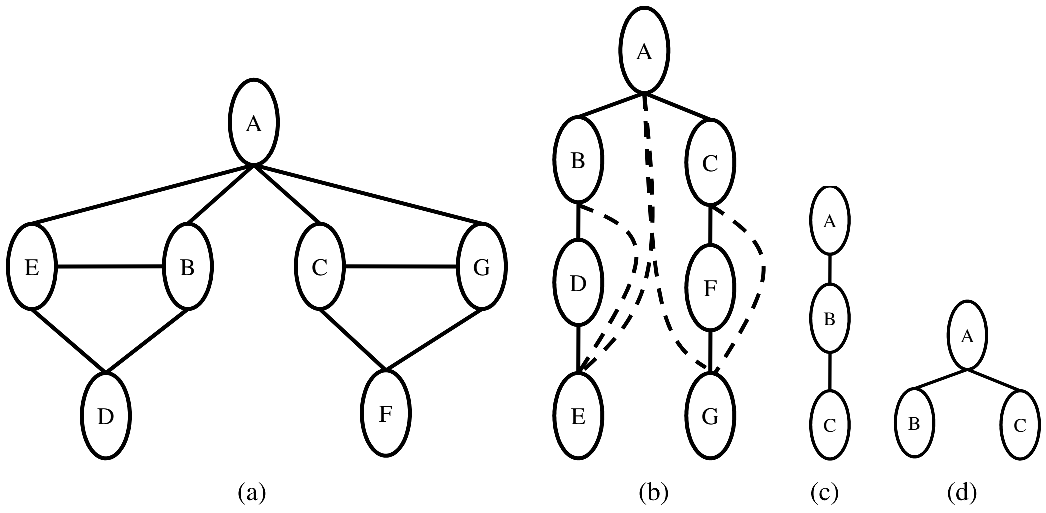


Figure 6.9: (a) Example primal graph of a mixed network (b) Pseudo tree (c) Start pseudo tree of OR w-cutset tree sampling and (d) Start pseudo tree of AND/OR w-cutset tree sampling.

the 1-cutset into two portions as in the start pseudo tree given in Figure 6.9(d). Thus, the main idea is to arrange the samples on an AND/OR sample tree restricted over the 1-*cutset variables* $\{A,B,C\}$, *perform exact bucket elimination separately on the sub-problems rooted at* B *and* C *respectively in the pseudo tree of Figure 6.9(b) and then compute a new AND/OR* w*-cutset sample tree mean over the AND/OR* w*-cutset sample tree.*

DEFINITION **54** (**start and full pseudo trees**). *[91] Given a pseudo tree* $T(\boldsymbol{V},\boldsymbol{E})$, *a directed rooted tree* $T'(\boldsymbol{V'},\boldsymbol{E'})$ *where* $\boldsymbol{V'} \subseteq \boldsymbol{V}$ *and* $\boldsymbol{E}' \subseteq \boldsymbol{E}$ *is a start pseudo tree of* T *if it has the same root as* T *and is a connected subgraph of* T. T *is called the* full pseudo tree *of its start pseudo tree* T'.

EXAMPLE **24.** *The pseudo tree given in Figure 6.9(d) is a start pseudo tree of the (full) pseudo tree given in Figure 6.9(b).*

DEFINITION **55** (**AND/OR w-cutset**). *[91] Given a mixed network* $\mathcal{M} = \langle\, \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C}\,\rangle$, *an AND/OR w-cutset is a pair* $\langle T', \boldsymbol{K}\rangle$ *where* $\boldsymbol{K} \subseteq \boldsymbol{X}$ *is a w-cutset of* $\mathcal{M}$ *and* $T'(\boldsymbol{K},\boldsymbol{E})$ *is a start pseudo tree defined over* $\boldsymbol{K}$.

DEFINITION **56** (**AND/OR w-cutset search tree**). *Given a mixed network* $\mathcal{M} = \langle\, \boldsymbol{X}, \boldsymbol{D},$

F, ***C*** $\rangle$ *, an AND/OR w-cutset* $\langle T'$*,* ***K***$\rangle$*, a full pseudo tree* T *of* T' *and an AND/OR search tree* ϕ_{AOT} *based on* T *and* $\mathcal{M}$*, the AND/OR w-cutset search tree* ϕ_{AOWT} *is obtained from* ϕ_{AOT} *by removing all nodes and edges that are not included in* ***K****.*

DEFINITION **57** (**AND/OR w-cutset sample tree**). *Given a mixed network* $\mathcal{M} = \langle$ ***X***, ***D***, ***F***, ***C*** $\rangle$*, an AND/OR w-cutset* $\langle T'$*,* ***K***$\rangle$*, a full pseudo tree* T *of* T'*, a proposal distribution along the pseudo tree* $Q(\boldsymbol{X}) = \prod_{i=1}^{n} Q_i(X_i|\boldsymbol{Y}_i)$ *such that* $\boldsymbol{Y}_i \subseteq context_T(X_i)$*, a sequence of samples* $\mathrm{S}_{T'}$ *and a complete AND/OR w-cutset search tree* ϕ_{AOWT} *defined relative to* $\mathcal{M}$*,* $\langle T'$*,* ***K***$\rangle$ *and* T'*, an AND/OR w-cutset sample tree* S_{AOWT} *is obtained from* ϕ_{AOWT} *by removing all edges and corresponding nodes which do not appear in* $\boldsymbol{S}_{T'}$*. The arc-labels from an OR node* n *to an AND node* m *in* S_{AOWT} *are defined in a similar way to those of an AND/OR sample tree (see Definition 47). Let* X_i *be the label of* n *and* x_i *be the label of* m*, then the arc-label from* n *to* m *is a pair* $\langle w(n,m), \#(n,m)\rangle$ *where:*

- $w(n,m) = \frac{B_{T,X_i}(x_i, A(\pi_n))}{Q_i(x_i|A(\pi_n))}$ *is called the weight of the arc.*
- $\#(n,m)$ *is the frequency of the arc. Namely, it is equal to the number of times the partial assignment* $A(\pi_m)$ *occurs in* $\boldsymbol{S}_{T'}$*.*

where B_{T,X_i} *be the bucket function of* X_i *w.r.t.* T *(see Definition 46). Note that the Bucket function is defined relative to the full pseudo tree* T *and not the start pseudo tree* T'*.*

DEFINITION **58** (**AND/OR w-cutset sample tree mean** $\widehat{Z}_{ao-wc-tree}$). *Given a mixed network* $\mathcal{M} = \langle$***X,D,F,C***$\rangle$*, an AND/OR w-cutset* $\langle T'$*,* ***K***$\rangle$*, a full pseudo tree* T *of* T'*, a proposal distribution along the pseudo tree* $Q(\boldsymbol{X}) = \prod_{i=1}^{n} Q_i(X_i|\boldsymbol{Y}_i)$ *such that* $\boldsymbol{Y}_i \subseteq context_T(X_i)$*, a sequence of samples* $\mathrm{S}_{T'}$*, an AND/OR* w*-cutset sample tree* S_{AOWT} *defined relative to* $\mathcal{M}$*,* $\langle T'$*,* ***K***$\rangle$ *and* T *and an initialization function* $\psi(V(\pi_l))$ *given by Equation 6.49 for each leaf AND node* l*, labeled with* x_i *, the AND/OR w-cutset sample tree mean is the value (see Definition 48) of the root node of* S_{AOWT}*.*

Algorithm 22: AND/OR w-cutset Importance Sampling

```
Input: A mixed network $\mathcal{M}$, and an integer $w$
Output: AND/OR $w$-cutset sample mean
1  Construct an AND/OR w-cutset (see Definition 55) $\langle T', \mathbf{K} \rangle$ of $\mathcal{M}$. Let $T$ be the full
   pseudo tree of $T'$;
2  Construct a proposal distribution $Q(\mathbf{K}) = \prod_{i=1}^{|\mathbf{K}|} Q_i(K_i|\mathbf{Y}_i)$ where
   $\mathbf{Y}_i \subseteq context_T(X_i)$;
3  Generate samples $\mathbf{k}^1, \ldots, \mathbf{k}^N$ from $Q$;
4  Build a AND/OR $w$-cutset sample tree $S_{AOWT}$ for the samples $\mathbf{k}^1, \ldots, \mathbf{k}^N$ (see
   Definition 57);
5  Initialize all labeling functions $\langle w, \# \rangle$ on each arc from an OR-node $n$ to an AND
   node $n'$ using Definition 57;
6  for all leaf AND nodes $l$ of $S_{AOWT}$ do
       // Let $l$ be labeled by $x_i$
7      Compute $v(l) = \sum_{\mathbf{u} \in lpath_T(X_i)} \prod_{X_j \in lpath_T(X_i)} B_{T,X_j}(\mathbf{u}, A(\pi_l))$ using Bucket
       elimination [28];
8  Initialize all leaf OR nodes to 0;
9  for every node $n$ from leaves to the root do
       // Let $chi(n)$ denote the child nodes of node $n$
10     if $n$ is an AND node then
11         $v(n) = \prod_{n' \in chi(n)} v(n')$;
12     else
13         $$v(n) = \frac{\sum_{n' \in chi(n)} (\#(n,n') w(n,n') v(n'))}{\sum_{n' \in chi(n)} \#(n,n')}$$
14 return v(root node of $S_{AOWT}$)
```

$$\psi(A(\pi_l)) = \sum_{\boldsymbol{u} \in lpath_T(X_i)} \prod_{X_j \in lpath_T(X_i)} B_{T,X_j}(\boldsymbol{u}, A(\pi_l)) \tag{6.49}$$

where $lpath_T(X_i)$ is the set of variables along the path from X_i to the leaf in the full pseudo tree T.

6.6.1 The algorithm and its properties

The AND/OR w-cutset importance sampling scheme is presented in Algorithm 22. Given a mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C} \rangle$ and an integer w, we generate an AND/OR w-cutset $\langle T', \mathbf{K} \rangle$ of $\mathcal{M}$. Then, we create a proposal distribution $Q(\mathbf{K})$ over the w-cutset along the start pseudo tree structure and generate N samples. We store these samples on the AND/OR w-cutset sample tree over $\mathbf{K}$ and perform value computations as follows. For each leaf AND node l, its value is computed by exact bucket elimination on the sub-tree of T rooted at l. Then, the algorithm computes the AND/OR sample tree mean recursively by performing product at each AND node and weighted average at each OR node (steps 9-13). Finally, the algorithm returns the value of the root-node.

From the proof of Theorem 19, it follows trivially that:

THEOREM **25.** *The AND/OR w-cutset sample mean is an unbiased estimate of the weighted counts.*

From the Rao-Blackwell theorem, it follows that:

THEOREM **26.** *The variance of AND/OR w-cutset sample tree mean is less than or equal to the variance of AND/OR sample tree mean.*

Next, we prove that:

THEOREM **27.** *Given a mixed network $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \boldsymbol{C} \rangle$, an AND/OR w-cutset $\langle T', \boldsymbol{K} \rangle$ of $\mathcal{M}$, a full pseudo tree T of T', a chain pseudo tree T'_{OR} obtained by any topological linearization order of T' and a full pseudo tree T_{OR} of T'_{OR}, the variance of $\widehat{Z}_{ao-wc-tree}$ based on T, $\langle T', \boldsymbol{K} \rangle$ and $\mathcal{M}$ is less than or equal to the variance of $\widehat{Z}_{or-wc-tree}$ based on T_{OR}, $\langle T'_{OR}, \boldsymbol{K} \rangle$ and $\mathcal{M}$.*

Proof. To prove this theorem, we show that AND/OR w-cutset sample tree is a special case of an AND/OR sample tree and OR w-cutset sample tree is a special case of an OR sample tree. Therefore, from Theorem 22, it follows that the variance of AND/OR w-cutset sample tree mean is smaller than or equal to OR w-cutset sample tree mean.

Consider an AND/OR w-cutset sample tree S_{AOWT} defined relative to $\mathcal{M}$, $\langle T', \mathbf{K} \rangle$ and T.

Let l be a leaf AND node of S_{AOWT}, labeled by x_i. For the purpose of discussion, let $\mathbf{U} = lpath_T(X_i)$ and $\mathbf{U} = \mathbf{u}$ be an assignment. We define the *perfect proposal distribution*, denoted by $\mathcal{P}(lpath_T(X_i)|V(\pi_l))$ as follows.

$$\mathcal{P}(\mathbf{u}|A(\pi_l)) = \alpha \prod_{X_j \in lpath_T(X_i)} B_{T,X_j}(\mathbf{u}, A(\pi_l))$$

where B_{T,Z_j} is a bucket function of X_j relative to T (see Definition 46) and α is a normalization constant given by:

$$\alpha = \frac{1}{\sum_{\mathbf{u} \in lpath_T(X_i)} \prod_{X_j \in lpath_T(X_i)} B_{T,X_j}(\mathbf{u}, A(\pi_l))}$$

If we draw a sample $\mathbf{u}^i$ from $\mathcal{P}(\mathbf{U}|A(\pi_l))$, then, by definition, the weight of $\mathbf{u}^i$ is given by:

$$w_{\mathcal{P}}(\mathbf{u}^i) = \frac{\prod_{X_j \in lpath_T(X_i)} B_{T,X_j}(\mathbf{u}^i, A(\pi_l))}{\mathcal{P}(\mathbf{u}^i|\mathbf{k})} \quad (6.50)$$

$$= \frac{\prod_{X_j \in lpath_T(X_i)} B_{T,X_j}(\mathbf{u}^i, A(\pi_l))}{\alpha \prod_{X_j \in lpath_T(X_i)} B_{T,X_j}(\mathbf{u}^i, A(\pi_l))} \quad (6.51)$$

$$= \frac{1}{\alpha} \quad (6.52)$$

$$= \sum_{\mathbf{u} \in lpath_T(X_i)} \prod_{X_j \in lpath_T(X_i)} B_{T,X_j}(\mathbf{u}, A(\pi_l)) \quad (6.53)$$

From Equations 6.53 and 6.49, it follows that the value of a leaf AND node l of S_{AOWT} is equal to the weight of any sample $\mathbf{u}^i$ drawn from $\mathcal{P}$. Namely,

$$v(l) = w_{\mathcal{P}}(\mathbf{u}^i) \tag{6.54}$$

Assume that S_{AOT} is a full AND/OR sample tree relative to T which is obtained from S_{AOWT} by adding a single sample from $\mathcal{P}$ to each leaf AND node of S_{AOWT}. We will now prove that S_{AOT} and S_{AOWT} are equivalent in the sense that the AND/OR sample tree mean computed on S_{AOT} is equal to the AND/OR w-cutset sample tree mean computed on S_{AOWT}. Consider the AND node g in S_{AOT} corresponding to the leaf AND node l in S_{AOWT}. By construction, to prove that S_{AOT} is equivalent to S_{AOWT}, all we need to prove is that the value of l is equal to the value of g. Because, we only added one sample at l in S_{AOWT} to form S_{AOT}, the value of g in S_{AOT} equals the weight of the added sample, namely

$$v(g) = w_{\mathcal{P}}(\mathbf{u}^i) \tag{6.55}$$

Therefore, from Equations 6.54 and 6.55, we get $v(l) = v(g)$, as required.

Thus, we can create an AND/OR sample tree S_{AOT} which is equivalent to a given AND/OR w-cutset sample tree S_{AOWT} by adding a sample from $\mathcal{P}$ to each leaf node of S_{AOWT}. Similarly, we can create an OR sample tree S_{ORT} which is equivalent to the OR w-cutset sample tree S_{ORWT} by adding a sample from $\mathcal{P}$ to each leaf node of S_{ORWT}.

From Theorem 22, the variance of AND/OR sample tree mean computed on S_{AOT} is less than or equal to the variance of the AND/OR sample tree mean computed on S_{ORT}. Since these full trees are equivalent to their w-cutset counterparts, it follows that the variance of AND/OR w-cutset sample tree mean computed on S_{AOWT} is less than or equal to the variance of the AND/OR w-cutset sample tree mean computed on S_{ORWT}.

$\square$

As in full AND/OR sampling, we can restrict the AND/OR sample graph mean to the start pseudo tree. Clearly, we will get additional reduction in variance.

DEFINITION **59** (**AND/OR w-cutset sample graph and mean**). *Given an AND/OR w-cutset sample tree S_{AOWT}, the AND/OR w-cutset sample graph S_{AOWG} is obtained from S_{AOWT} by context-based merging. The AND/OR w-cutset sample graph mean, denoted by $\widehat{Z}_{ao-wc-graph}$ is AND/OR sample mean computed over S_{AOWG}.*

The algorithm for computing AND/OR w-cutset graph mean is identical to Algorithm 22. The only difference is that we store all the generated samples on an AND/OR w-cutset sample graph rather than the AND/OR w-cutset sample tree. We can prove that:

THEOREM **28.** *The variance of AND/OR w-cutset sample graph mean $\widehat{Z}_{ao-wc-graph}$ is less than or equal to that of AND/OR w-cutset sample tree mean $\widehat{Z}_{ao-wc-tree}$.*

Proof. The proof of this theorem is along the similar lines as Theorem 27 in that we construct an equivalent AND/OR sample graph S_{AOG} from the AND/OR w-cutset sample graph S_{AOWG} and an equivalent AND/OR sample tree S_{AOT} from the AND/OR w-cutset sample tree S_{AOWT} by adding a sample from a perfect proposal distribution. Then, the proof follows from Theorem 23. □

6.6.2 Variance Hierarchy and Complexity

Theorems 22, 23, 27 and 28 along with the Rao-Blackwell theorem help establish the variance hierarchy shown in Figure 6.10. The main assumption is that all sample means are based on the same set of samples and the same (full) pseudo tree. Semantically, given a w-cutset where $0 < w < w^*$ (w^* is the treewidth), the directed arcs in Figure 6.10 indicate that the variance of the child node is smaller than (or equal to) the variance of the parent. We see that the variance of AND/OR sample tree mean is incomparable with w-cutset sample

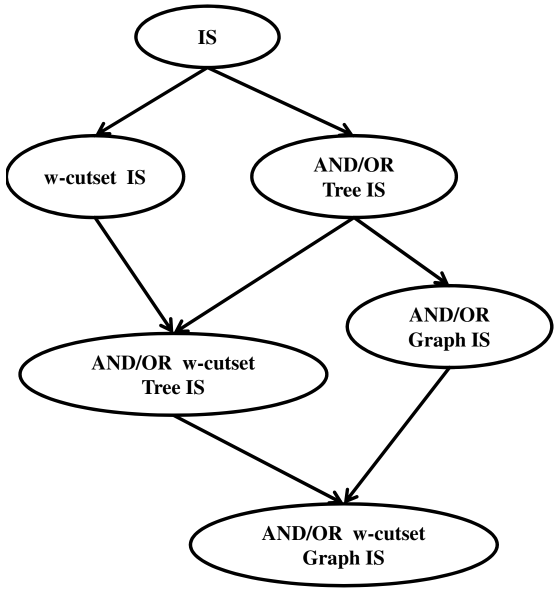


Figure 6.10: Variance Hierarchy

Sampling Scheme	**Time Complexity**	**Space Complexity**
OR tree	$O(nN)$	$O(1)$
AND/OR tree	$O(nN)$	$O(h)$
AND/OR graph	$O(nNw^*)$	$O(nN)$
OR w-cutset tree	$O(cN + (n-c)Nexp(w))$	$O((n-c)exp(w))$
AND/OR w-cutset tree	$O(cN + (n-c)Nexp(w))$	$O(h + (n-c)exp(w))$
AND/OR w-cutset graph	$O(cNw^* + (n-c)Nexp(w))$	$O(cN + (n-c)exp(w))$

(N: number of samples, n: number of variables, w: w-cutset bound, c: number of variables in the w-cutset, h: height of pseudo tree and w^*: treewidth)

Figure 6.11: Complexity of various AND/OR importance sampling schemes and their w-cutset generalizations.

mean and that AND/OR w-cutset sample graph mean has the lowest variance. Finally, variance reduction obviously comes at extra computational cost. As we move down the variance hierarchy, the time and space complexity of compiling the various means typically increases except when moving from OR tree schemes to AND/OR tree schemes and their respective w-cutset generalizations. The complexities are depicted in Figure 6.11.

6.7 Empirical Evaluation

We provide empirical evaluation which demonstrates the properties of the various schemes. In particular, we show how moving from OR space sampling to AND/OR space sampling improves performance. We evaluate the performance as an anytime scheme. Namely, we measure accuracy as a function of time.

6.7.1 The Algorithms evaluated

We use the SampleSearch with IJGP(i) scheme presented in Chapter 3 (see Algorithm 12) for evaluating AND/OR sampling.

We experimented with six versions of SampleSearch based importance sampling:

1. OR tree importance sampling (or-tree-IS) introduced in [53] which computes the OR sample tree mean.
2. AND/OR tree importance sampling (ao-tree-IS), which computes the AND/OR sample tree mean.
3. AND/OR graph importance sampling (ao-graph-IS) which computes the AND/OR graph sample tree mean.
4. OR w-cutset tree importance sampling (or-wc-tree-IS) introduced in [7, 53] which computes the OR w-cutset sample tree mean.
5. AND/OR w-cutset tree importance sampling (ao-wc-tree-IS) which computes the AND/OR w-cutset tree sample mean and
6. AND/OR w-cutset graph importance sampling (ao-wc-graph-IS) which computes AND/OR w-cutset graph sample mean.

We set the w of w-cutset to 5 to ensure that the bucket elimination component of w-cutset sampling does not run out of memory and terminates in a reasonable amount of time. The underlying scheme for generating the samples is identical in all schemes and is done via SampleSearch. What changes is the method of accumulating the samples and deriving the estimates.

6.7.2 Evaluation Criteria

We experimented with three sets of benchmarks: (a) Grid networks, (b) Linkage networks, and (c) 4-coloring problems. On each instance, we compare the log relative error (see Chapter 5) between the exact probability of evidence (or the solution counts for 4-coloring problems) and the approximate one. If Z is the exact value and $\overline{Z}$ is the approximate value, the log-relative error denoted by Δ is defined as:

$$\Delta = \frac{log(Z) - log(\overline{Z})}{log(Z)} \tag{6.56}$$

When the exact results are not known, we compute a lower bound on the probability of evidence (or the solution counts) by the martingale and average heuristics described in Chapter 5. Recall that the lower bounding scheme takes a set of unbiased sample weights and a confidence $0 < \alpha < 1$ as input, and outputs a probabilistic lower bound on the weighted counts, which are correct with probability greater than α. We use $\alpha = 0.99$ in all our experiments which means our lower bounds are correct with probability greater than 0.99. Again, we use the highest lower bound available as a substitute for Z in Equation 6.56.

As mentioned in Chapter 5, we compute the log relative error instead of the usual relative error because when the probability of evidence is extremely small ($< 10^{-10}$) or when the solution counts are large (e.g. $> 10^{10}$) the relative error between the exact and the approximate answer will be arbitrarily close to 1 and we would need a large number of digits to distinguish between the results.

Notation in Tables

Tables 6.1, 6.2, 6.3 and 6.4 presents the results. For each instance, in column 2, we report the number of variables (N), average domain size (k), number of evidence nodes or constraints (E), treewidth w^* and w-cutset size (c). The third column reports the exact value of probability of evidence or solution counts for that instance; if known. The remaining columns (from Column 4 onwards) contain the log-relative error of the approximate solution output by the competing schemes in 1 hour of CPU time. Next, we discuss the results for each benchmark in more detail. The best results are indicated by bold in each row.

6.7.3 Results on the Grid Networks

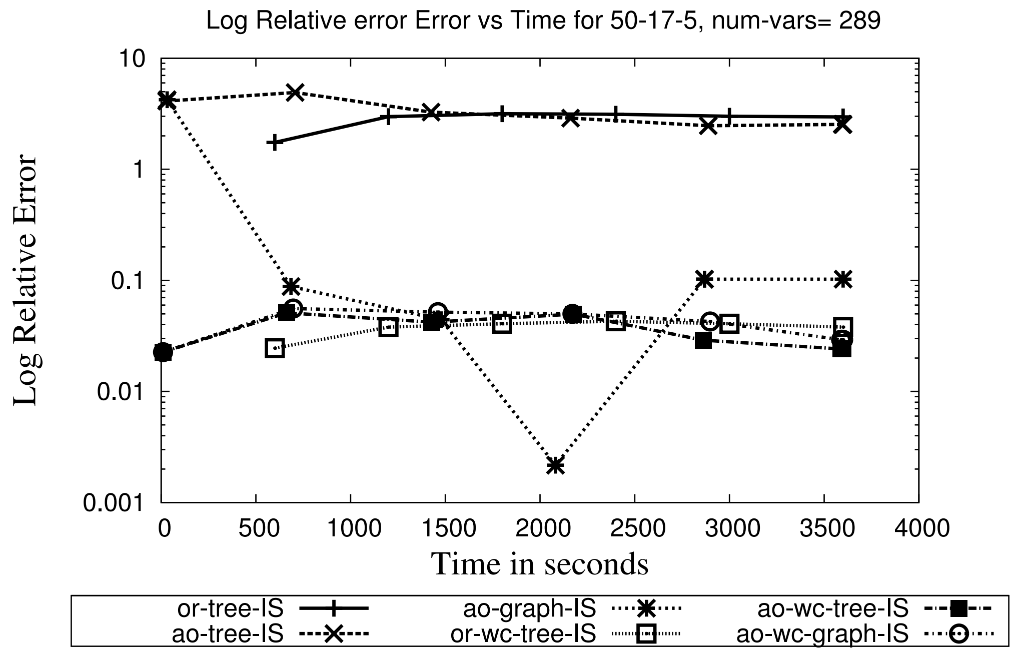
The grid networks are available from the authors of Cachet - a SAT model counter [119]. A grid Bayesian network is a $s \times s$ grid, where there are two directed edges from a node to its neighbors right and down. The upper-left node is a source, and the bottom-right node is a sink. The sink node is the evidence node whose marginal probability is to be determined. The deterministic ratio p is a parameter specifying the fraction of nodes that are deterministic, that is, whose values are determined given the values of their parents. The grid instances are designated as $p - s$. For example, the instance $50 - 18$ indicates a grid of size 18 in which 50% of the nodes are deterministic.

Table 6.1 shows the log-relative error computed by the various schemes after 1 hour of CPU time. Time versus log relative error plots for six sample grid instances are shown in Figures 6.12, 6.13 and 6.14 respectively.

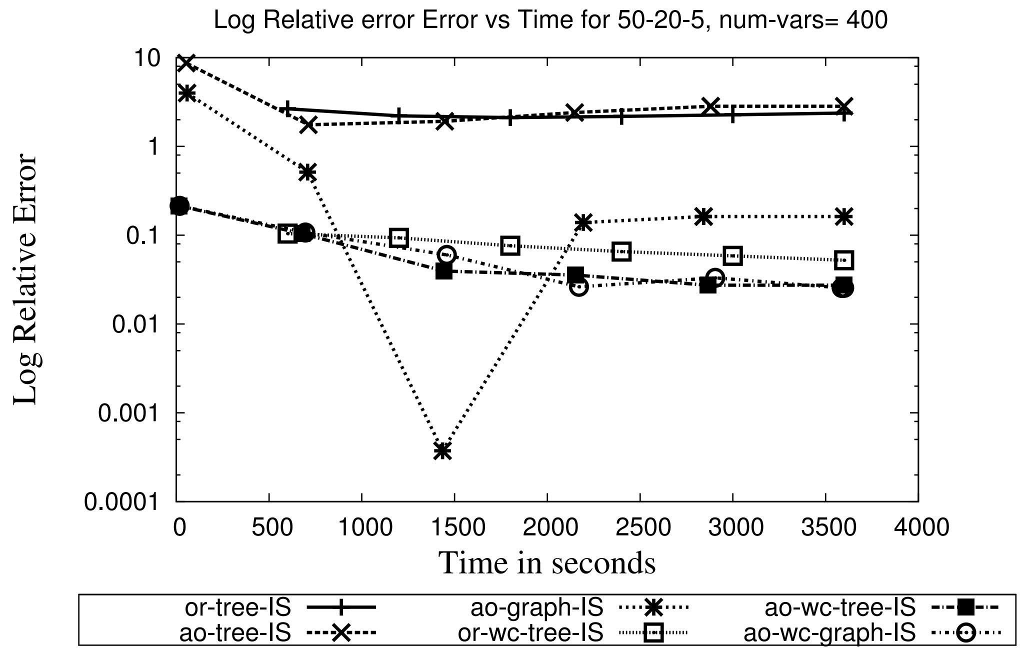
On grids with deterministic ratio of 50% (see Table 6.1 and Figures 6.12(a) and 6.12(b)),

Problem	$\langle n,k,E,t^*,c\rangle$	**Exact**	**or-tree-IS** Δ	**ao-tree-IS** Δ	**ao-graph-IS** Δ	**or-wc-tree-IS** Δ	**ao-wc-tree-IS** Δ	**ao-wc-graph-IS** Δ
50% Ratio								
50-17-5	$\langle 289,2,1,25,45\rangle$	7.71E-01	2.96E+00	2.54E+00	1.03E-01	3.80E-02	**2.41E-02**	2.92E-02
50-18-5	$\langle 324,2,1,27,53\rangle$	4.14E-01	3.49E-01	4.87E-01	2.19E-02	**1.91E-02**	1.98E-02	2.13E-02
50-19-5	$\langle 361,2,1,28,60\rangle$	2.21E-01	7.72E-02	1.06E-01	2.62E-02	**8.79E-03**	8.95E-03	9.68E-03
50-20-5	$\langle 400,2,1,30,70\rangle$	5.69E-01	2.38E+00	2.84E+00	1.62E-01	5.22E-02	2.74E-02	**2.55E-02**
75% Ratio								
75-16-5	$\langle 256,2,1,24,18\rangle$	6.16E-01	1.05E-01	4.22E-02	1.48E-02	3.77E-03	**3.14E-03**	5.96E-03
75-17-5	$\langle 289,2,1,25,23\rangle$	2.25E-02	3.00E-02	6.53E-03	2.60E-03	2.00E-03	**1.70E-03**	1.79E-03
75-18-5	$\langle 324,2,1,27,39\rangle$	1.89E-01	1.51E-01	6.77E-02	8.56E-03	3.80E-03	9.46E-04	**5.84E-04**
75-19-5	$\langle 361,2,1,28,38\rangle$	5.84E-01	9.55E-01	2.24E+00	1.24E-01	1.43E-02	5.77E-03	**1.90E-04**
75-20-5	$\langle 400,2,1,30,33\rangle$	9.93E-01	5.60E+01	2.26E+01	1.88E+00	1.59E+00	**1.44E+00**	1.51E+00
75-21-5	$\langle 441,2,1,32,43\rangle$	1.83E-01	1.19E-01	5.97E-02	6.82E-03	5.16E-02	**2.39E-03**	1.30E-02
75-22-5	$\langle 484,2,1,35,47\rangle$	4.38E-01	6.19E-01	6.55E-01	2.07E-01	3.74E-03	9.58E-03	**8.94E-04**
75-23-5	$\langle 529,2,1,35,65\rangle$	3.48E-01	6.90E-01	4.92E-01	1.36E-01	3.59E-02	7.40E-03	**3.83E-03**
75-26-5	$\langle 676,2,1,44,82\rangle$	2.64E-01	1.78E+00	1.22E+00	**1.54E-01**	1.75E-01	1.92E-01	1.75E-01
90% Ratio								
90-21-5	$\langle 441,2,1,32,7\rangle$	6.54E-01	1.21E-01	2.93E-02	1.87E-02	2.65E-03	**1.57E-03**	1.65E-03
90-22-5	$\langle 484,2,1,35,21\rangle$	8.31E-01	1.16E-01	2.10E-01	7.48E-02	3.65E-02	**3.37E-02**	3.58E-02
90-23-5	$\langle 529,2,1,35,13\rangle$	1.85E-01	2.20E-02	6.57E-03	1.90E-03	2.64E-04	3.13E-04	**4.69E-05**
90-24-5	$\langle 576,2,1,38,10\rangle$	2.62E-01	3.18E-02	2.04E-02	2.33E-02	2.29E-04	1.87E-04	**8.39E-05**
90-25-5	$\langle 625,2,1,39,7\rangle$	3.03E-01	2.39E-02	7.49E-02	1.38E-02	**3.73E-05**	1.70E-04	5.39E-05
90-26-5	$\langle 676,2,1,44,21\rangle$	3.33E-01	1.53E-01	4.61E-02	4.20E-02	1.18E-02	5.87E-03	**5.60E-03**
90-34-5	$\langle 1156,2,1,65,18\rangle$	8.59E-02	1.54E-01	3.88E-02	7.77E-02	9.95E-03	**5.51E-03**	7.88E-03
90-38-5	$\langle 1444,2,1,69,57\rangle$	1.41E-01	5.78E-01	1.77E-01	**5.02E-02**	2.96E-01	2.74E-01	3.73E-01
90-42-5	$\langle 1764,2,1,83,58\rangle$	6.55E-01	2.15E+00	2.60E-01	3.64E-02	**2.77E-04**	1.43E-02	3.90E-02
90-50-5	$\langle 2500,2,1,108,85\rangle$	1.91E-01	2.36E+00	7.20E-01	2.35E-01	2.00E-01	**1.36E-01**	1.60E-01

Table 6.1: Table showing the log-relative error Δ of importance sampling, w-cutset importance sampling and their AND/OR tree and graph variants for the Grid instances after 1 hour of CPU time.



(a)



(b)

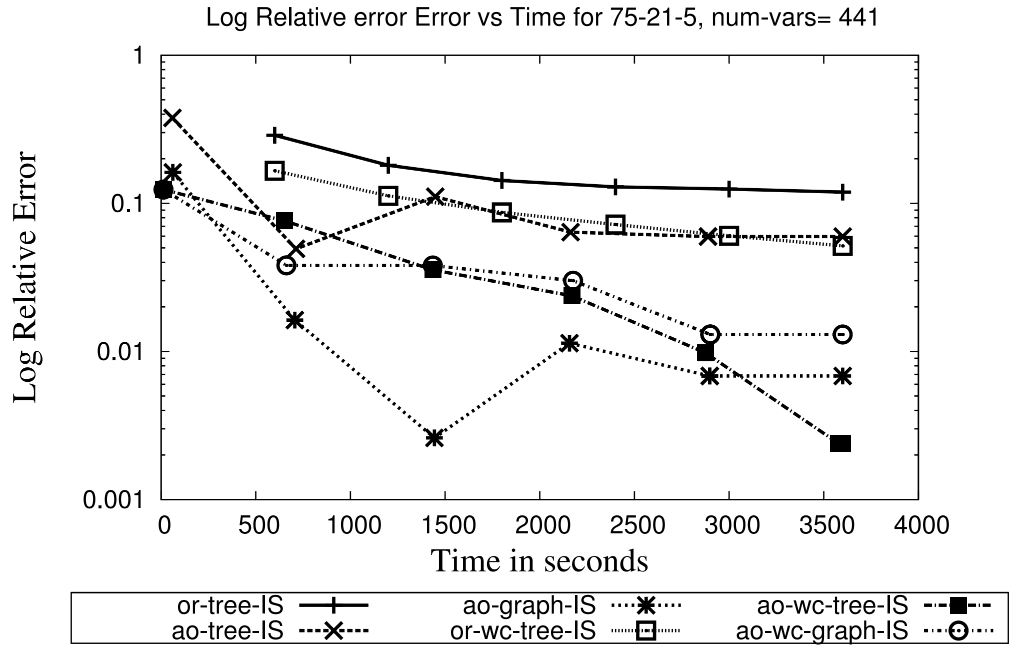
Figure 6.12: Time versus log relative error for importance sampling, w-cutset importance sampling and their AND/OR tree and graph generalizations for Grid instances with deterministic ratio = 50%.

schemes that employ w-cutset sampling (the last 3 columns in Table 6.1) are superior. We observe that the log relative error does not decrease when moving from OR tree sampling to AND/OR tree sampling. However, when moving from AND/OR tree sampling to AND/OR graph sampling, the log relative error decreases by one order of magnitude (see Figure 6.12(a) and (b)). Comparing w-cutset sampling schemes, we see that there is a very minor difference in their respective log relative errors. In other words, AND/OR w-cutset tree and AND/OR w-cutset graph sampling do not yield any additional variance reduction beyond what is achieved by OR w-cutset tree sampling.

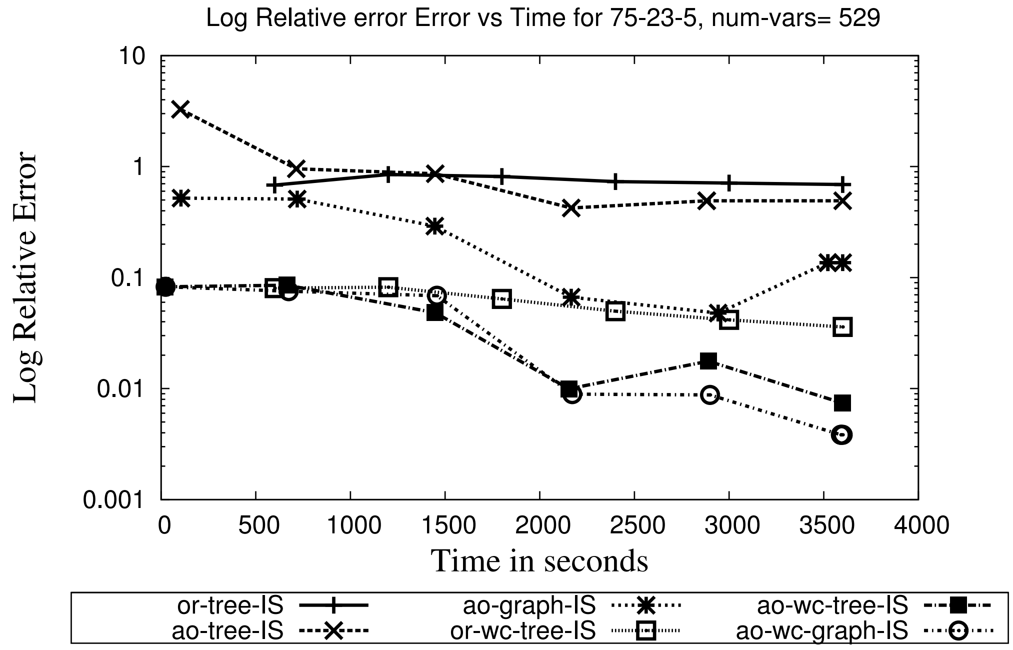
On grids with deterministic ratio of 75% (see Table 6.1 and Figures 6.13(a) and 6.13(b)), we see that on most instances ao-wc-tree-IS and ao-wc-graph-IS are slightly better than ao-graph-IS and or-wc-tree-IS. ao-tree-IS and or-tree-IS are the worst performing schemes. For example in Figures 6.13(a) and (b), ao-tree-IS and or-tree-IS are on the top indicating larger log relative error. Thus, moving from AND/OR tree sampling to graph sampling results in appreciable variance reduction but moving from OR and AND/OR w-cutset tree sampling to AND/OR w-cutset graph sampling does not yield additional variance reductions.

On grids with deterministic ratio of 90% (see Table 6.1 and Figures 6.14(a) and 6.14(b)), we see a similar picture to the grids with deterministic ratio of 75% in that ao-wc-tree-IS and ao-wc-graph-IS are slightly better than ao-graph-IS and or-wc-tree-IS schemes. ao-tree-IS and or-tree-IS schemes are the worst performing schemes.

Overall, we find that on the grids instances, ao-wc-graph-IS and ao-wc-tree-IS schemes are slightly better than or-wc-tree-IS and ao-graph-IS schemes and substantially better than or-tree-IS and ao-tree-IS schemes. Namely, moving from AND/OR tree to AND/OR graph pays off substantially in schemes which do not employ w-cutset sampling but not as much in schemes that use w-cutset sampling. w-cutset by itself has a significant impact.

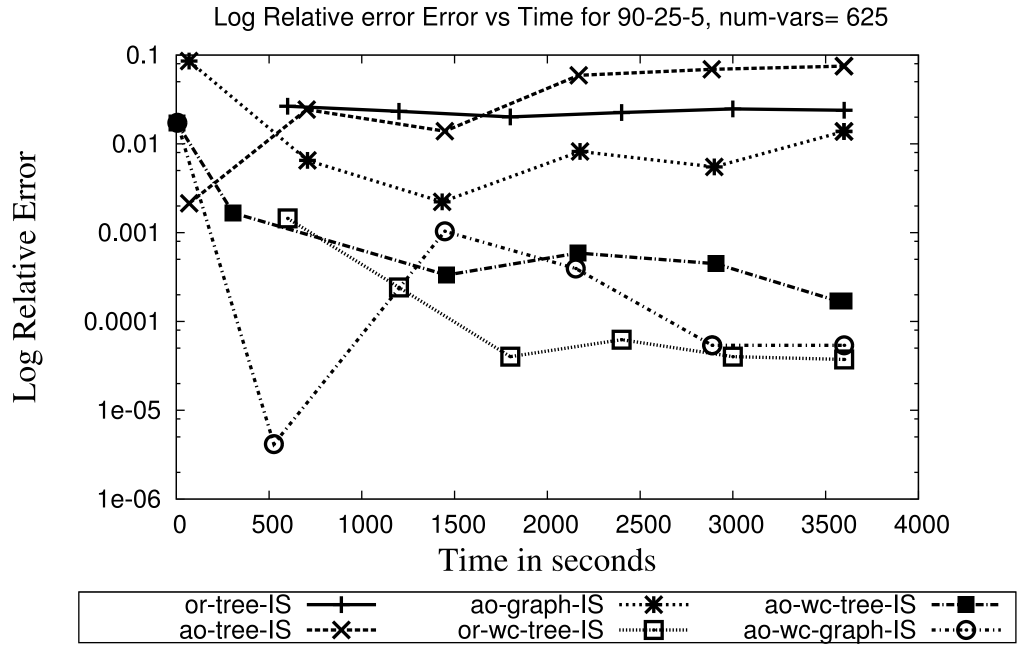


(a)

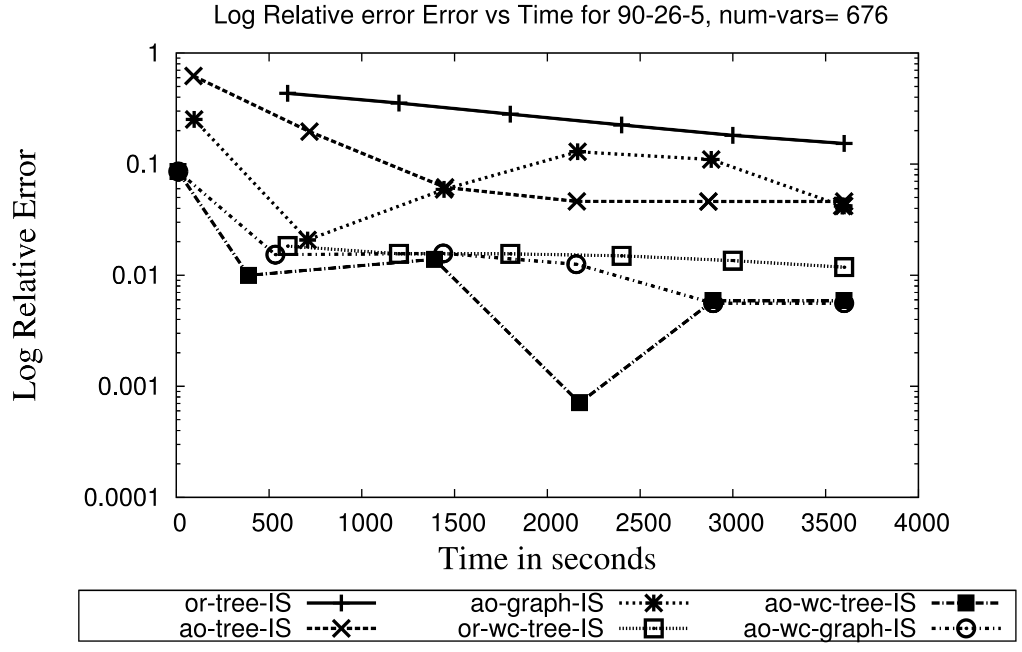


(b)

Figure 6.13: Time versus log relative error for importance sampling, w-cutset importance sampling and their AND/OR tree and graph generalizations for Grid instances with deterministic ratio = 75%.

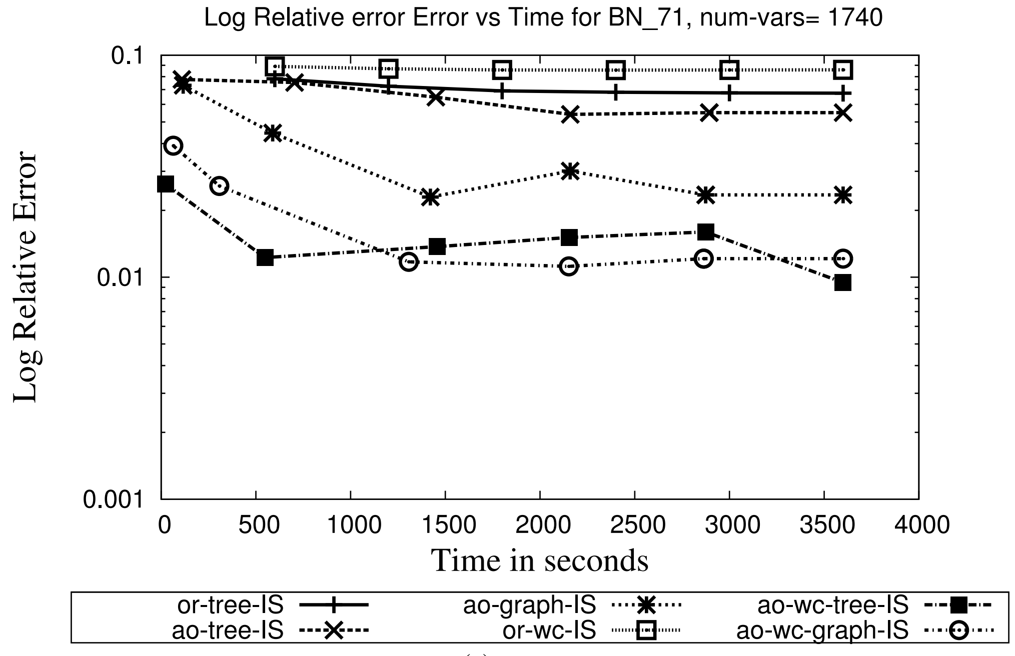


(a)

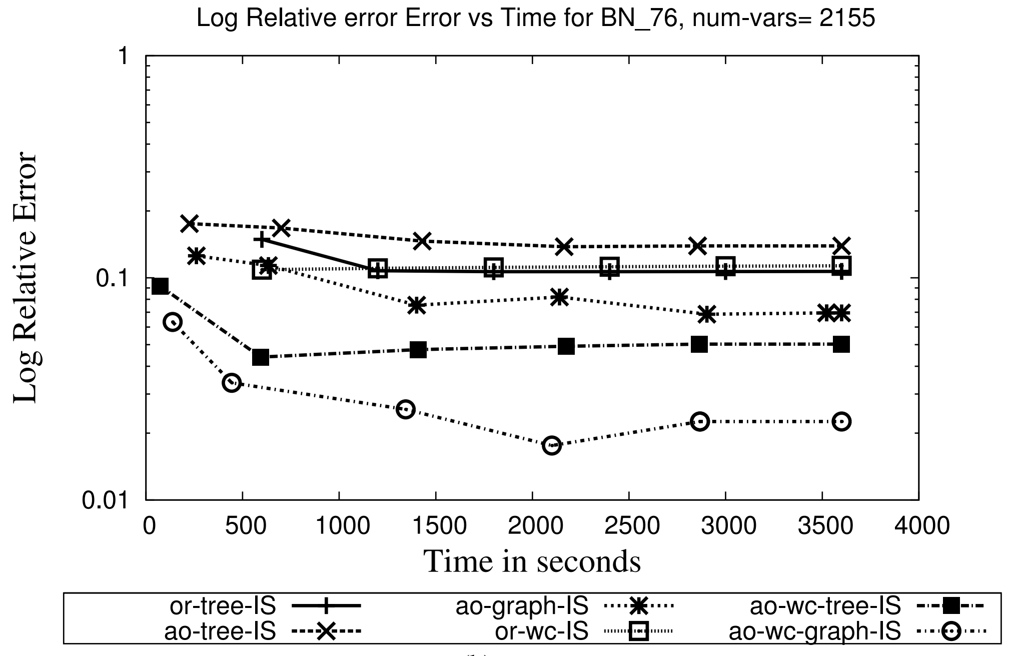


(b)

Figure 6.14: Time versus log relative error for importance sampling, w-cutset importance sampling and their AND/OR tree and graph generalizations for Grid instances with deterministic ratio = 90%.



(a)



(b)

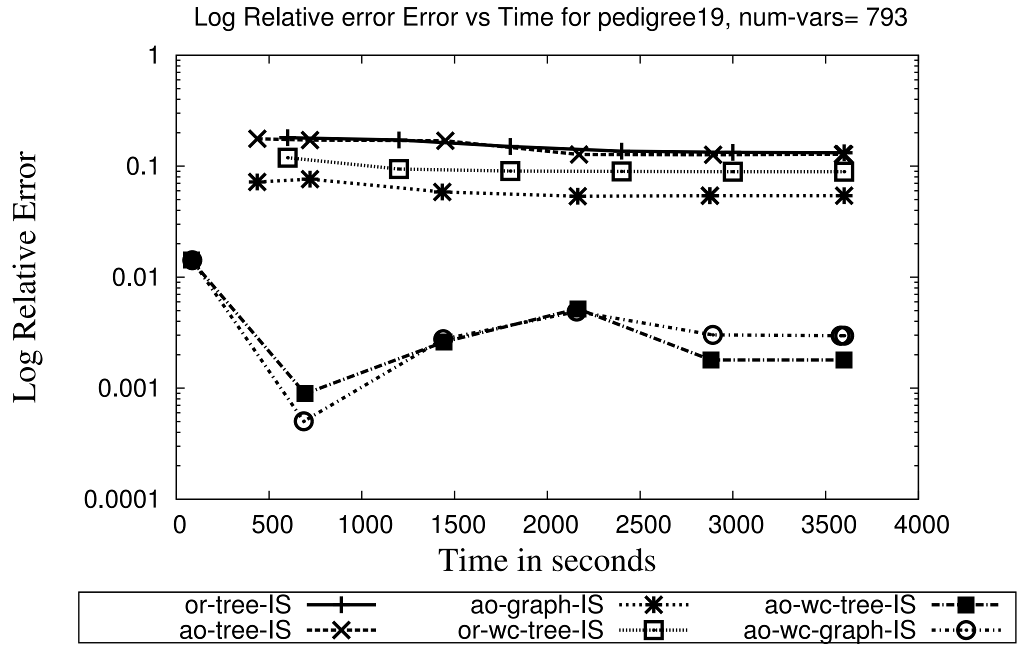
Figure 6.15: Time versus log relative error for importance sampling, w-cutset importance sampling and their AND/OR tree and graph generalizations for two sample Linkage instances from the UAI 2006 evaluation.

Problem	$\langle n, k, E, t^*, c\rangle$	**Exact**	**or-tree-IS** Δ	**ao-tree-IS** Δ	**ao-graph-IS** Δ	**or-wc-tree-IS** Δ	**ao-wc-tree-IS** Δ	**ao-wc-graph-IS** Δ
BN_69	$\langle 777, 7, 78, 47, 59\rangle$	5.28E-54	2.26E-02	2.46E-02	2.43E-02	2.42E-02	2.34E-02	**4.22E-03**
BN_70	$\langle 2315, 5, 159, 87, 98\rangle$	2.00E-71	6.32E-02	7.25E-02	5.12E-02	8.18E-02	5.36E-02	**2.62E-02**
BN_71	$\langle 1740, 6, 202, 70, 139\rangle$	5.12E-111	6.74E-02	5.51E-02	2.35E-02	8.58E-02	**9.46E-03**	1.21E-02
BN_72	$\langle 2155, 6, 252, 86, 88\rangle$	4.21E-150	3.19E-02	4.61E-02	2.46E-03	6.12E-02	**1.41E-03**	2.63E-03
BN_73	$\langle 2140, 5, 216, 101, 149\rangle$	2.26E-113	1.18E-01	1.12E-01	4.55E-02	1.58E-01	**3.54E-02**	3.95E-02
BN_74	$\langle 749, 6, 66, 45, 72\rangle$	3.75E-45	5.34E-02	4.31E-02	2.87E-02	8.08E-02	2.83E-02	**2.76E-02**
BN_75	$\langle 1820, 5, 155, 92, 131\rangle$	5.88E-91	4.47E-02	8.15E-02	4.73E-02	7.28E-02	4.20E-02	**7.60E-03**
BN_76	$\langle 2155, 7, 169, 64, 239\rangle$	4.93E-110	1.07E-01	1.39E-01	6.95E-02	1.13E-01	5.03E-02	**2.26E-02**
BN_77	$\langle 1020, 9, 135, 22, 97\rangle$	6.88E-79	1.06E-01	9.38E-02	8.26E-02	1.24E-01	6.75E-02	**3.27E-02**

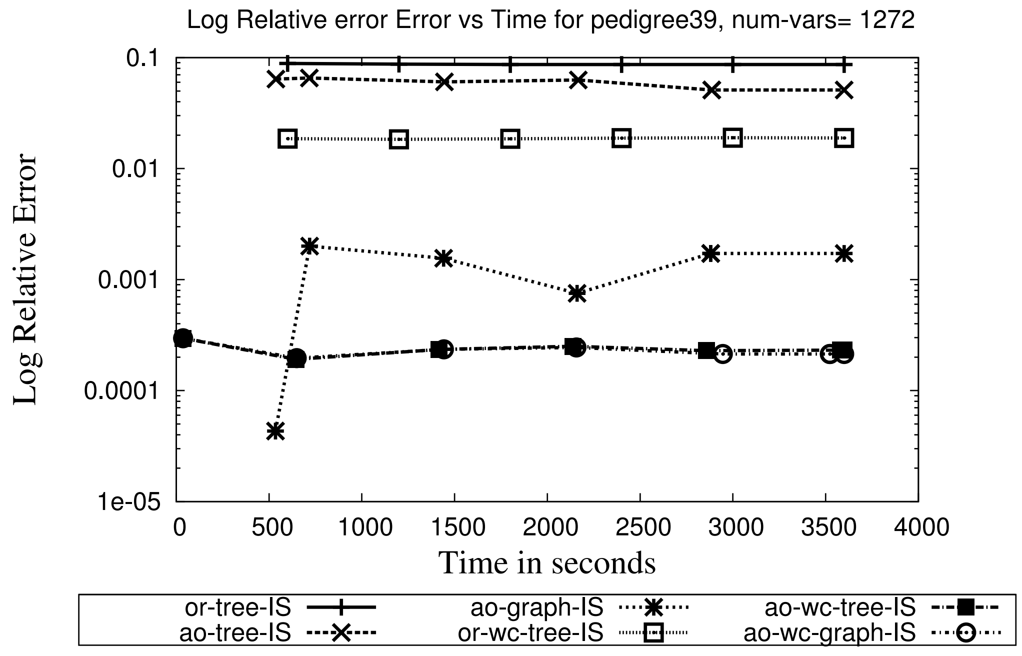
Table 6.2: Table showing the log relative error Δ of importance sampling, w-cutset importance sampling and their AND/OR tree and graph variants for the Linkage instances from UAI 2006 evaluation after 1 hour of CPU time.

6.7.4 Results on Linkage networks

The Linkage instances that we experimented with in Chapter 3 are generated by converting biological linkage analysis data into a Bayesian or Markov network. These networks have between 777-2315 nodes with an average domain size of 9 or less. The results of our experiments with these networks are shown in Table 6.2 and anytime performances for two sample linkage instances are shown in Figure 6.15. We observe that on the 6 out of the 9 instances, ao-wc-graph-IS is more accurate than ao-wc-tree-IS which in turn is substantially more accurate than or-wc-tree-IS. Comparing schemes which do not employ w-cutset sampling, ao-graph-IS is substantially better than ao-tree-IS and or-tree-IS schemes. Thus, we see that exploiting more problem decomposition using AND/OR graph sampling pays off more when used with conventional importance sampling than with w-cutset importance sampling. However, the scheme that utilizes the most decomposition - AND/OR w-cutset graph importance sampling consistently outperforms other schemes and whenever it is the second or the third best, the difference between it and the best scheme is very minor (for example see time versus log relative error plots for two sample Linkage networks given in Figure 6.15).



(a)



(b)

Figure 6.16: Time versus log relative error for importance sampling, w-cutset importance sampling and their AND/OR tree and graph generalizations for two sample Linkage instances from the UAI 2008 evaluation.

Problem	$\langle n, k, E, t^*, w\rangle$	Exact	or-tree-IS Δ	ao-tree-IS Δ	ao-graph-IS Δ	or-wc-tree-IS Δ	ao-wc-tree-IS Δ	ao-wc-graph-IS Δ
pedigree18	$\langle 1184, 1, 0, 26, 72\rangle$	4.19E-79	3.17E-02	3.44E-02	3.20E-03	4.30E-02	3.49E-04	**3.02E-04**
pedigree19	$\langle 793, 2, 0, 23, 102\rangle$	1.59E-60	1.32E-01	1.28E-01	5.41E-02	8.92E-02	**1.79E-03**	2.97E-03
pedigree1	$\langle 334, 2, 0, 20, 27\rangle$	7.81E-15	2.18E-03	1.90E-03	1.73E-04	3.15E-05	**7.61E-06**	1.13E-05
pedigree20	$\langle 437, 2, 0, 25, 33\rangle$	2.34E-30	1.52E-01	1.56E-01	2.12E-03	6.93E-02	**9.17E-04**	1.18E-03
pedigree23	$\langle 402, 1, 0, 26, 29\rangle$	2.00E-40	**2.62E-02**	2.74E-02	2.90E-02	2.82E-02	2.88E-02	2.88E-02
pedigree37	$\langle 1032, 1, 0, 25, 36\rangle$	2.63E-117	2.46E-02	3.50E-03	3.24E-03	1.45E-02	**3.00E-03**	3.01E-03
pedigree38	$\langle 724, 1, 0, 18, 45\rangle$	5.64E-55	4.08E-02	1.40E-02	1.25E-02	1.69E-02	8.91E-03	**8.79E-03**
pedigree39	$\langle 1272, 1, 0, 29, 42\rangle$	6.32E-103	8.67E-02	5.11E-02	1.72E-03	1.89E-02	2.31E-04	**2.13E-04**
pedigree42	$\langle 448, 2, 0, 23, 50\rangle$	1.73E-31	4.29E-03	1.94E-03	5.06E-04	1.11E-03	3.53E-05	**3.17E-05**
pedigree31	$\langle 1183, 2, 0, 45, 118\rangle$		1.09E-01	1.31E-01	4.15E-02	8.34E-02	**0.00E+00**	2.93E-04
pedigree34	$\langle 1160, 1, 0, 59, 104\rangle$		2.12E-01	1.47E-01	8.37E-02	8.09E-02	4.83E-04	**0.00E+00**
pedigree13	$\langle 1077, 1, 0, 51, 98\rangle$		3.93E-01	3.93E-01	5.66E-02	9.11E-02	1.51E-04	**0.00E+00**
pedigree41	$\langle 1062, 2, 0, 52, 95\rangle$		1.12E-01	5.06E-02	8.23E-04	5.04E-02	**0.00E+00**	3.15E-04
pedigree44	$\langle 811, 1, 0, 29, 64\rangle$		3.16E-02	3.08E-02	2.27E-03	1.90E-02	**0.00E+00**	4.63E-06
pedigree51	$\langle 1152, 1, 0, 51, 106\rangle$		9.22E-02	6.39E-02	2.26E-02	4.31E-02	9.35E-05	**0.00E+00**
pedigree7	$\langle 1068, 1, 0, 56, 90\rangle$		7.86E-02	9.98E-02	2.31E-02	4.61E-02	4.38E-04	**0.00E+00**
pedigree9	$\langle 1118, 2, 0, 41, 80\rangle$		3.29E-02	3.19E-02	**0.00E+00**	8.25E-02	9.74E-03	1.01E-02

Table 6.3: Table showing the log-relative error Δ of importance sampling, w-cutset importance sampling and their AND/OR tree and graph variants for the Markov Linkage instances from UAI 2008 evaluation after 1 hour of CPU time.

Markov Linkage networks

The Markov linkage instances were used in the UAI 2008 evaluation. These are linkage Bayesian networks in which evidence is instantiated yielding an un-normalized Bayesian network. These networks are quite hard and only a handful of problems were solved exactly in the UAI 2008 evaluation. As pointed earlier, because the exact value of probability of evidence is not available on some instances, we compare lower bounds and use the highest lower bound as a substitute for Z in Equation 5.29 for computing the log relative error.

Table 6.3 shows the log relative error of various schemes after 1 hour of CPU time. In Figure 6.16, we show time versus log relative error plots for two sample instances. We observe that ao-wc-graph-IS, ao-wc-tree-IS and ao-graph-IS schemes have lower log relative error than other schemes, often outperforming the competition by an order of magnitude. Of the three schemes that do not use w-cutset, we observe that ao-tree-IS is better than or-tree-IS on most instances while or-wc-tree-IS is usually better than ao-tree-IS and or-tree-IS.

Problem	$\langle n, k, E, t^*, c\rangle$	**Exact**	**or-tree-IS** Δ	**ao-tree-IS** Δ	**ao-graph-IS** Δ	**or-wc-tree-IS** Δ	**ao-wc-tree-IS** Δ	**ao-wc-graph-IS** Δ
4-coloring1	$\langle 400, 2, 0, 71, 309\rangle$		3.82E-03	4.05E-03	4.51E-03	6.00E-03	2.35E-03	**0.00E+00**
4-coloring2	$\langle 400, 2, 0, 95, 315\rangle$		1.23E-02	9.54E-03	7.64E-03	3.38E-02	3.63E-02	**0.00E+00**
4-coloring3	$\langle 800, 2, 0, 144, 617\rangle$		2.86E-03	4.58E-03	2.32E-03	2.41E-02	2.38E-02	**0.00E+00**
4-coloring4	$\langle 800, 2, 0, 191, 620\rangle$		2.13E-02	5.06E-03	2.19E-02	1.79E-02	4.69E-03	**0.00E+00**
4-coloring5	$\langle 1200, 2, 0, 304, 925\rangle$		2.98E-02	2.81E-02	5.85E-02	5.70E-02	3.89E-02	**0.00E+00**
4-coloring6	$\langle 1200, 2, 0, 338, 929\rangle$		3.43E-02	2.72E-02	2.63E-03	3.17E-03	2.09E-03	**0.00E+00**

Table 6.4: Table showing the log-relative error Δ of importance sampling, w-cutset importance sampling and their AND/OR tree and graph variants on Joseph Culberson's flat graph coloring instances after 1 hour of CPU time.

ao-wc-graph-IS yields the best approximation.

6.7.5 Results on 4-Coloring Problems

Our final domain is that of 4-coloring problems generated using Joseph Culberson's flat graph coloring generator [3]. Here, we are interested in counting the number of solutions of the graph coloring instance. Table 6.4 shows the log relative error of the estimate output by various schemes after 1 hour of CPU time. As described earlier, because exact results are not known, we compare lower bounds. We observe that AND/OR tree and graph sampling schemes yield estimates having lower log relative error than OR tree sampling schemes. Among the AND/OR schemes, AND/OR w-cutset graph sampling (ao-wc-graph-IS) yields the best lower bound on all instances.

[3]Available at http://www.cs.ualberta.ca/~joe/Coloring/

6.8 Discussion and Related work

6.8.1 Relation to other graph-based variance reduction schemes

The work presented here is related to the work by [66, 75, 27] who perform sampling based inference on a junction tree. The main idea in these papers is to perform message passing on a junction tree by substituting messages which are too hard to compute exactly by their sampling-based approximations. Kjærulff [75] and Dawid [27] use Gibbs sampling while Hernandez et al. [66] use importance sampling to approximate the messages. Similar to some recent works on Rao-Blackwellised sampling such as [7, 103, 53], variance reduction is achieved in these junction tree based sampling schemes because of some exact computations, as dictated by the Rao-Blackwell theorem. AND/OR estimation, however, does not require exact computations to achieve variance reduction. In fact, as we show, variance reduction due to Rao-Blackwellisation is orthogonal to the variance reduction achieved by AND/OR estimation and therefore the two could be combined to achieve more variance reduction. Also, unlike our work which focuses on probability of evidence or the weighted counting problem, the focus of these aforementioned papers was on the belief updating task.

6.8.2 Hoeffding's U-statistics

AND/OR-estimates are also closely related to *cross match estimates* [78] which are based on Hoeffding's U-statistics. To derive cross-match estimates, the original function over a set of variables is divided into several marginal functions which are defined only on a subset of variables. Then, each marginal function is sampled independently and the cross-match sample mean is derived by considering all possible combinations of the samples. For example, if there are k marginal functions and m samples are taken over each function, the cross match sample mean is computed over m^k combinations. It was shown in [78] that the cross match sample mean has lower variance than the conventional sample mean, similar to our work. The only caveat in *cross match estimates* is that it requires exponentially more time $O(m^k)$ to compute the estimates as compared to $O(m)$ for conventional estimates; making their direct application infeasible for large values of k. So the authors suggest resampling from the possible $O(m^k)$ samples with the hope that the estimates based on the resampled samples would have lower variance than the conventional one. Unlike *cross match estimates*, the most complex AND/OR estimates are only w^* times more expensive time wise, where w^* is the treewidth, as compared to the conventional estimates, and therefore do not require the extra resampling step.

6.8.3 Problem with large sample sizes

Given that the space complexity of the AND/OR graph based schemes is $O(nN)$, the reader may think that as more samples are drawn our algorithms would run out of memory. One can, however, perform multi-stage (adaptive) sampling to circumvent this problem. Here, at each stage we stop storing samples when a pre-specified memory limit is reached. Then AND/OR sample graph mean is computed from the stored samples and the samples are thrown away, repeating the process until the stipulated time bound expires or enough samples are drawn. The final sample mean is then simply the average of sample means computed at each stage. By linearity of expectation, the final sample mean is unbiased and obviously would have lower variance than the conventional sample mean.

6.9 Conclusion

The primary contribution of this chapter is in viewing importance sampling based estimation in the context of AND/OR search spaces rather than the usual OR search spaces for graphical models [37]. Specifically, we viewed sampling as a partial exploration of the full AND/OR search space yielding an AND/OR sample tree. We defined a new AND/OR sample mean on the samples organized on the AND/OR sample tree and showed that it is an unbiased estimate of the weighted counts.

We showed that the AND/OR sample tree mean has lower variance than the conventional OR sample tree mean because it yields more virtual samples by taking advantage of the conditional independencies in the graphical model. Furthermore, because the AND/OR sample tree mean has the same time complexity and only slightly more space overhead than the OR sample tree mean; it should always be preferred.

The AND/OR sample tree was extended into a graph by merging identical subtrees, which is analogous to extending AND/OR tree search to AND/OR graph search [37]. We showed that the AND/OR sample graph yields more virtual samples than the AND/OR sample tree and therefore reduces variance even further. However, computing the AND/OR sample graph mean requires more time and space and thus there is a tradeoff.

Subsequently, we showed that the AND/OR sampling scheme can be combined with the w-cutset sampling scheme [7] to reduce variance even further. The combination of AND/OR and w-cutset sampling yields a family of sample means which trade variance with time and space as summarized in Figures 6.10 and 6.11.

We focused our empirical investigation on the task of computing probability of evidence or partition function in a Bayesian and Markov network respectively. The main aim of our evaluation was to compare the impact of exploiting varying levels of graph decompositions via (a) OR tree (b) AND/OR tree, (c) AND/OR graph and (d) their w-cutset generalizations on the accuracy of the estimates. Our results show conclusively that (a) AND/OR tree sampling usually yields better estimates than (conventional) OR tree sampling, (b) AND/OR graph sampling is superior to AND/OR tree sampling and (c) w-cutset sampling (OR and AND/OR) is superior to non w-cutset sampling.

Chapter 7

Conclusions

In this chapter, we conclude the dissertation with a survey of our principal contributions and outline several promising avenues for further research.

Our research addresses the problems associated with performing sampling based approximate inference over mixed probabilistic and deterministic networks. In particular, because of determinism, importance sampling algorithms suffer from the rejection problem in that samples with zero weight are generated with probability arbitrarily close to one while Markov Chain Monte Carlo techniques do not converge at all yielding very poor performance.

7.1 Contributions

The first contribution of this dissertation is a scheme called IJGP-sampling which uses the output of a generalized belief propagation scheme called Iterative Join Graph Propagation (IJGP) [33] to construct a proposal distribution for importance sampling and uses the partial directional i-consistency power of IJGP to reduce rejection. We applied this algorithm to perform inference in a dynamic mixed network that modeled travel routines of individuals.

The second contribution is the SampleSearch scheme which combines systematic backtracking search with random sampling and completely eliminates the rejection problem. We showed that the bias introduced by mixing search with sampling can be characterized using the backtrack-free distribution and thus the samples generated by SampleSearch can be used for performing approximate inference with desirable statistical guarantees such as unbiasedness and asymptotic unbiasedness. We also showed that SampleSearch can be viewed as a systematic search technique whose value selection is stochastically guided by sampling from a distribution. This view enabled us to utilize state-of-the-art systematic SAT/CSP solvers such as minisat [125] for performing search within SampleSearch. Thus, advances in the systematic search community whose primary focus is solving "yes/no" type NP-complete problems can be leveraged through SampleSearch for approximating much harder #P-complete problems in Bayesian inference. We performed an extensive empirical evaluation on several benchmark graphical models and our results clearly demonstrate that SampleSearch is consistently superior to other state-of-the-art schemes on domains having a substantial amount of determinism.

The third contribution is two schemes of SampleSearch-MH and SampleSearch-SIR for sampling solutions from a uniform distribution over the solutions of a Boolean satisfiability problem. This solution sampling task is motivated by applications in fields such as functional/software verification [134, 31] and first order probabilistic models [113, 95]. SampleSearch-MH and SampleSearch-SIR combine SampleSearch with statistical techniques of Metropolis Hastings (MH) and Sampling / Importance Resampling (SIR) respectively and guarantee that in the limit of infinite samples the distribution over the generated solution samples is the uniform distribution over the solutions. Such convergence guarantees are not available for state-of-the-art schemes such as SampleSat [130] and XorSample [63]. Via a thorough empirical evaluation, we showed conclusively that our new schemes are superior to both SampleSat and XorSample in terms of accuracy as a function of time.

Our fourth contribution is a randomized approximation algorithm called *Markov-LB* for computing high confidence lower bounds on the weighted counting tasks such as probability of evidence and the partition function. Markov-LB is based on importance sampling and the Markov inequality. We argued that Markov inequality is quite weak (a fact well known in the statistics literature) because it is based on a single sample. To mitigate this deficiency, we proposed to extend the Markov inequality to multiple samples yielding three new schemes: one based on computing the sample average of the generated samples and two based on the martingale theory [10] which utilize the maximum statistics from the generated samples. We showed via a large-scale experimental evaluation that Markov-LB applied to IJGP-sampling and SampleSearch schemes presented in this thesis outperforms state-of-the-art schemes such as bound propagation [6], SampleCount [62], Relsat [115] and variable elimination and conditioning (VEC) and provides good quality high confidence lower bounds on most instances.

Our final contribution is a new sampling framework called AND/OR importance sampling which generalizes conventional OR space importance sampling which is impervious to problem decomposition to the AND/OR space [37] which is sensitive to problem decomposition. Our generalization yields a family of estimators which are based on the same set of samples and which trade variance (and therefore accuracy) with time and space by utilizing different levels of problem decomposition. At one end is the AND/OR tree estimator which has smaller variance than the conventional OR tree estimator and has the same time complexity. At the other end is the AND/OR graph estimator which has even lower variance than the AND/OR tree estimator but has higher time and space complexity. Subsequently, we showed that the AND/OR sampling scheme can be combined with another sampling technique which utilizes graph decomposition called w-cutset sampling [7] yielding further variance reduction. Via an extensive experimental evaluation, we showed that exploiting decomposition aggressively by combining AND/OR graph sampling and w-cutset sampling is well worth the extra computational complexity in that the combination usually yields more accurate estimates.

7.2 Directions for Future Research

Our investigation leaves plenty of room for additional improvements that can be pursued in the future.

Incremental SAT solving

To generate samples using SampleSearch, we solve the same SAT/CSP problem multiple times. Therefore, various goods and no-goods (i.e., knowledge about the problem space) discovered while generating one sample can be used to speed-up the search for a solution while generating the next sample. How to achieve this in a principled and structured way is an important theoretical and practical question. Some initial related research on solving a series of similar SAT problems which differ only slightly from one another has appeared in the bounded model checking community [43] and can be applied to improve SampleSearch's performance.

Adaptive Importance Sampling

A second line of future research is to use adaptive importance sampling [17, 101, 133, 97] within SampleSearch and AND/OR importance sampling. In adaptive importance sampling, one updates the proposal distribution based on the generated samples; so that with every update the proposal gets closer and closer to the desired posterior distribution. Because we already store the DFS traces of the generated samples in SampleSearch, one could use them to dynamically update and learn the proposal distribution. Also, because the optimal proposal distribution decomposes according to an AND/OR tree or graph, we can even utilize graph decompositions to facilitate learning.

Exploiting independencies uncovered while sampling

The AND/OR sampling framework utilizes decomposition or conditional independencies uncovered by the primal graph associated with a probabilistic model. It is well known that the graph captures only a fraction of the actual conditional independencies and therefore AND/OR sampling utilizes decomposition in a sub-optimal manner. New conditional independencies could be discovered while sampling through the AND/OR space; which could then be utilized to get better estimates as dictated by the AND/OR sampling theory. How to utilize these new independencies on the fly as well as how to guide sampling to look for such independencies is still an open question.

Approximate Compilation

When knowledge is represented as a graphical model, the representation requires only polynomial space but exact inference is time (and/or space) exponential. Compilation turns this problem upside down in that knowledge is represented by using as much space as possible so that it takes only polynomial time for typical inference tasks. Examples of compilation frameworks are Ordered Binary Decision Diagrams (OBDDs) [11], Decomposable Normal Negation form (DNNF) [25] and its variants and AND/OR multi-valued decision diagrams [93]. The problem with these exact compilation frameworks is that their space requirements are excessive and impractical. What we actually need is an approximate compilation framework which provides *good enough* answers based on the application.

SampleSearch and AND/OR sampling can be easily modified to approximately compile a graphical model. Obviously, some research issues that need to be addressed are whether we can provide guarantees on the quality and error of the estimates derived from the compiled structure.

Bibliography

[1] D. Allen and A. Darwiche. Rc_link: Genetic linkage analysis using Bayesian networks. *International Journal of Approximate Reasoning*, 48(2):499–525, 2008.

[2] S. Arnborg and A. Proskourowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete and Applied Mathematics*, 23:11–24, 1989.

[3] R. Bayardo and D. P. Miranker. On the space-time trade-off in solving constraint satisfaction problems. In *Fourteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 558–562, 1995.

[4] J. Bergeron. *Writing Testbenches: Functional Verification of HDL Models*. Kluwer Academic Publishers, 2000.

[5] B. Bidyuk and R. Dechter. An anytime scheme for bounding posterior beliefs. In *Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence*, pages 1095–1100, 2006.

[6] B. Bidyuk and R. Dechter. Improving bound propagation. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI)*, pages 342–346, 2006.

[7] B. Bidyuk and R. Dechter. Cutset sampling for Bayesian networks. *Journal of Artificial Intelligence Research (JAIR)*, 28:1–48, 2007.

[8] J. Bilmes and R. Dechter. Evaluation of Probabilistic Inference Systems of UAI'06. Available online at http://ssli.ee.washington.edu/ bilmes/uai06InferenceEvaluation/, 2006.

[9] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 33–42, 1998.

[10] L. Breiman. *Probability*. Addison-Wesley, 1968.

[11] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transancations on Computers*, 35(8):677–691, 1986.

[12] G. Casella and C. P. Robert. Rao-Blackwellisation of sampling schemes. *Biometrika*, 83(1):81–94, 1996.

[13] M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6–7):772–799, April 2008.

[14] M. D. Chavira, A. Darwiche, and M. Jaeger. Compiling relational Bayesian networks for exact inference. *International Journal of Approximate Reasoning*, 42(1-2):4–20, 2006.

[15] J. Cheng. Sampling algorithms for estimating the mean of bounded random variables. In *Computational Statistics*, pages 1–23. Volume 16(1), 2001.

[16] J. Cheng and M. J. Druzdzel. AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *Journal of Artificial Intelligence Research (JAIR)*, 13:155–188, 2000.

[17] J.-F. Cheng. *Iterative Decoding*. PhD thesis, California Institute of Technology (Electrical Engineering), 1997.

[18] A. Choi and A. Darwiche. An edge deletion semantics for belief propagation and its practical impact on approximation quality. In *Proceedings of The Twenty-First National Conference on Artificial Intelligence (AAAI)*, pages 1107–1114, 2006.

[19] M. C. Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41(1):89–95, 1990.

[20] R. G. Cowell, P. A. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems (Information Science and Statistics)*. Springer, New York, May 2003.

[21] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, 1993.

[22] P. Dagum and M. Luby. An optimal approximation algorithm for Bayesian inference. *Artificial Intelligence*, 93:1–27, 1997.

[23] A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001.

[24] A. Darwiche, R. Dechter, A. Choi, V. Gogate, and L. Otten. Results from the Probablistic Inference Evaluation of UAI'08. Available online at: http://graphmod.ics.uci.edu/uai08/Evaluation/Report, 2008.

[25] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research (JAIR)*, 17:229–264, 2002.

[26] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[27] A. P. Dawid, U. Kjaerulff, and S. L. Lauritzen. Hybrid propagation in junction trees. In *Advances in Intelligent Computing (IPMU)*, pages 85–97, 1994.

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

[29] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[30] R. Dechter, V. Gogate, L. Otten, R. Marinescu, and R. Mateescu. Graphical model algorithms at UC Irvine. website: http://graphmod.ics.uci.edu/group/Software, 2009.

[31] R. Dechter, K. Kask, E. Bin, and R. Emek. Generating random solutions for constraint satisfaction problems. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, pages 15–21, 2002.

[32] R. Dechter, K. Kask, and J. Larrosa. A general scheme for multiple lower-bound computation in constraint optimization. *Seventh International Conference on Principles and Practice of Constraint Programming (CP)*, pages 346–360, 2001.

[33] R. Dechter, K. Kask, and R. Mateescu. Iterative join graph propagation. In *Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 128–136. Morgan Kaufmann, August 2002.

[34] R. Dechter and D. Larkin. Hybrid processing of beliefs and constraints. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 112–119, 2001.

[35] R. Dechter and R. Mateescu. A simple insight into iterative belief propagation's success. *Proceedings of the 19th Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 175–183, 2003.

[36] R. Dechter and R. Mateescu. Mixtures of deterministic-probabilistic networks and their and/or search space. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 120–129, 2004.

[37] R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007.

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

[39] R. Dechter and I. Rish. Mini-buckets: A general scheme for bounded inference. *Journal of ACM*, 50(2):107–153, 2003.

[40] A. Doucet, N. de Freitas, K. P. Murphy, and S. J. Russell. Rao-blackwellised particle filtering for dynamic Bayesian networks. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 176–183, 2000.

[41] A. Doucet, S. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10(3):197–208, 2000.

[42] P. A. Dow and R. E. Korf. Best-first search for treewidth. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 1146–1151, 2007.

[43] N. Eén and N. Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003.

[44] S. Even. Graph algorithms. In *Computer Science Press*, 1979.

[45] P. W. Fieguth, W. C. Karl, and A. S. Willsky. Efficient multiresolution counterparts to variational methods for surface reconstruction. *Computer Vision and Image Understanding*, 70(2):157–176, 1998.

[46] M. Fishelson and D. Geiger. Optimizing exact genetic linkage computations. In *Proceedings of the seventh annual international conference on Research in computational molecular biology (RECOMB)*, pages 114–121, 2003.

[47] N. D. Freitas. Rao-blackwellised particle filtering for fault diagnosis. In *In IEEE Aerospace Conference*, pages 1767–1772, 2001.

[48] R. M. Fung and K.-C. Chang. Weighing and integrating evidence for stochastic simulation in Bayesian networks. In *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 209–220, 1990.

[49] A. Gelman, J. Carlin, H. Stern, and D. Rubin. *Bayesian Data Analysis*. Chapman & Hall/CRC, 2nd edition, June 1995.

[50] J. Geweke. Bayesian inference in econometric models using Monte Carlo integration. *Econometrica*, 57(6):1317–39, 1989.

[51] V. Gogate, B. Bidyuk, and R. Dechter. Studies in lower bounding probability of evidence using the Markov inequality. In *Proceedings of 23rd Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 141–148, 2007.

[52] V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In *Proceedings of 20th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 201–208, 2004.

[53] V. Gogate and R. Dechter. Approximate inference algorithms for hybrid Bayesian networks with discrete constraints. In *Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 209–216, 2005.

[54] V. Gogate and R. Dechter. A new algorithm for sampling CSP solutions uniformly at random. In *Proceedings of 12th International Conference on Principles and Practices of Constraint Programming (CP)*, pages 711–715, 2006.

[55] V. Gogate and R. Dechter. Approximate counting by sampling the backtrack-free search space. In *Proceedings of 22nd Conference on Artificial Intelligence (AAAI)*, pages 198–203, 2007.

[56] V. Gogate and R. Dechter. Samplesearch: A scheme that searches for consistent samples. *Proceedings of the 11th Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 147–154, 2007.

[57] V. Gogate and R. Dechter. AND/OR Importance Sampling. In *In 23rd Conference on Artificial Intelligence (AAAI)*, pages 212–219, 2008.

[58] V. Gogate and R. Dechter. Approximate solution sampling (and counting) on and/or spaces. In *Proceedings of 14th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 534–538, 2008.

[59] V. Gogate and R. Dechter. Studies in solution sampling. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI)*, pages 271–276, 2008.

[60] V. Gogate, R. Dechter, B. Bidyuk, C. Rindt, and J. Marca. Modeling transportation routines using hybrid dynamic mixed networks. In *Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 217–224, 2005.

[61] C. Gomes and D. Shmoys. Completing quasigroups or latin squares: A structured graph coloring problem. In *Proceedings of the Computational Symposium on Graph Coloring and Extensions*, 2002.

[62] C. P. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman. From sampling to model counting. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2293–2299, 2007.

[63] C. P. Gomes, A. Sabharwal, and B. Selman. Near-uniform sampling of combinatorial spaces using xor constraints. In *Advances in Neural Information Processing Systems (NIPS)*, pages 481–488, 2006.

[64] L. A. Goodman. On the exact variance of products. *Journal of the American Statistical Association*, 55(292):708–713, 1960.

[65] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, April 1970.

[66] L. D. Hernandez and S. Moral. Mixing exact and importance sampling propagation algorithms in dependence graphs. *International Journal of Approximate Reasoning*, 12(8):553–576, 1995.

[67] T. Heskes and O. Zoeter. Expectation propagation for approximate inference in dynamic Bayesian networks. In *Proceedings of 18th Conference of Uncertainty in Artificial Intelligence, UAI*, pages 216–223, 2002.

[68] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[69] H. Hoos and T. Stutzle. Satlib: An online resource for research on SAT. In H. van Maaren I. P. Gent and T. Walsh, editors, *SAT2000*, pages 283–292. IOS Press, 2000.

[70] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform. *Theoretical Computer Science*, 43(2-3):169–188, 1986.

[71] H. M. Kaplan. A method of one-sided nonparametric inference for the mean of a nonnegative population. *The American Statistician*, 41(2):157–158, 1987.

[72] K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying tree decompositions for reasoning in graphical models. *Artificial Intelligence*, 166(1-2):165–193, August 2005.

[73] G. Kitagawa. Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 5:1–25, 1996.

[74] U. Kjaerulff. Optimal decomposition of probabilistic networks by simulated annealing. *Statistics and Computing*, 2:7–17, 1992.

[75] U. Kjærulff. Hugs: Combining exact inference and Gibbs sampling in junction trees. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 368–375, 1995.

[76] T. Kloks. *Treewidth: Computations and Approximations*. Springer-Verlag New York, Incorporated, 1994.

[77] G. Kokolakis and P. Nanopoulos. Bayesian multivariate micro-aggregation under the hellinger distance criterion. *Research in official statistics*, 4:117–125, 2001.

[78] Kong, Augustine, Liu, Jun S., and Wong, Wing Hung. The properties of the cross-match estimate and split sampling. *The Annals of Statistics*, 25(6):2410–2432, dec 1997.

[79] A. M. Koster, H. L. Bodlaender, and S. P. M. van Hoesel. Treewidth: Computational experiments. Technical report, Universiteit Utrecht, 2001.

[80] D. Larkin and R. Dechter. Bayesian inference in the presence of determinism. In *Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2003.

[81] S. Lauritzen. Propagation of probabilities, means, and variances in mixed graphical association models. *Journal of the American Statistical Association*, 87(420):1098–1108, 1992.

[82] M. Leisink and B. Kappen. Bound propagation. *Journal of Artificial Intelligence Research*, 19:139–154, 2003.

[83] U. Lerner. *Hybrid Bayesian Networks for Reasoning about complex systems*. PhD thesis, Stanford University, 2002.

[84] R. Levine and G. Casella. Implementations of the Monte Carlo EM Algorithm. *Journal of Computational and Graphical Statistics*, 10:422–439, 2001.

[85] F. Li and P. Perona. A Bayesian hierarchical model for learning natural scene categories. In *Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 524–531, 2005.

[86] L. Liao, D. J. Patterson, D. Fox, and H. Kautz. Learning and inferring transportation routines. *Artificial Intelligence*, 171(5-6):311–331, 2007.

[87] Y. Lin and M. J. Druzdzel. Relevance-based incremental belief updating in Bayesian networks. *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, 13(2):285–295, 1999.

[88] J. Liu. *Monte-Carlo strategies in scientific computing*. Springer-Verlag, New York, 2001.

[89] Liu, Jun S., Liang, Faming, and Wong, Wing Hung. The multiple-try method and local optimization in metropolis sampling. *Journal of the American Statistical Association*, 95(449):121–134, mar 2000.

[90] A. W. Marshall. The use of multi-stage sampling schemes in Monte Carlo computations. *In Symposium on Monte Carlo Methods*, pages 123–140, 1956.

[91] R. Mateescu and R. Dechter. And/or cutset conditioning. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 230–235, 2005.

[92] R. Mateescu and R. Dechter. Mixed deterministic and probabilistic networks. *Annals of Mathematics and Artificial Intelligence (AMAI); Special Issue: Probabilistic Relational Learning (to appear)*, 2009.

[93] R. Mateescu, R. Dechter, and R. Marinescu. And/or multi-valued decision diagrams (aomdds) for graphical models. *Journal of Artificial Intelligence Research (JAIR)*, 33:465–519, 2008.

[94] B. Middleton, S. Michael Shwe M., D. Heckerman, D. Harold Lehmann M., and G. Cooper. Probabilistic diagnosis using a reformulation of the internist-1/qmr knowledge base. *Medicine*, 30:241–255, 1991.

[95] B. Milch, B. Marthi, S. J. Russell, D. Sontag, D. L. Ong, and A. Kolobov. Blog: Probabilistic models with unknown objects. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1352–1359, 2005.

[96] J. E. Miller. Langford's problem. *http://www.lclark.edu/$\sim$miller/langford.html*, 2006.

[97] S. Moral and A. Salmerón. Dynamic importance sampling in Bayesian networks based on probability trees. *International Journal of Approximate Reasoning*, 38(3):245–261, 2005.

[98] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California Berkeley, 2002.

[99] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 467–475, 1999.

[100] N. J. Nillson. *Principles of Artificial Intelligence*. Tioga, Palo Alto, Ca, 1980.

[101] L. Ortiz and L. Kaelbling. Adaptive importance sampling for estimation in structured domains. In *In Proceedings of the 16th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 446–454, 2000.

[102] J. Ott. *Analysis of Human Genetic Linkage*. The Johns Hopkins University Press, Baltimore, Maryland, 1999.

[103] M. A. Paskin. Sample propagation. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[104] D. J. Patterson, L. Liao, D. Fox, and H. Kautz. Inferring high-level behavior from low-level sensors. In *Proceedings of UBICOMP 2003: The Fifth International Conference on Ubiquitous Computing*, pages 73–89, 2003.

[105] V. Pavlovic, J. M. Rehg, T.-J. Cham, and K. P. Murphy. A dynamic Bayesian network approach to figure tracking using learned dynamic models. In *The Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 1, pages 94–101, 1999.

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

[107] S. Peeta and A. K. Zilaskopoulos. Foundations of dynamic traffic assignment: The past, the present and the future. *Networks and Spatial Economics*, 1:233–265, 2001.

[108] K. Pipatsrisawat and A. Darwiche. Rsat 2.0: SAT solver description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA, 2007.

[109] H. Poon and P. Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *The Twenty-First National Conference on Artificial Intelligence (AAAI)*, pages 458–463, 2006.

[110] M. Pradhan, G. Provan, B. Middleton, and M. Henrion. Knowledge engineering for large belief networks. In *Proceedings of the 10th Canadian Conference on Artificial Intelligence*, pages 484–490, 1994.

[111] C. Raphael. A hybrid graphical model for rhythmic parsing. *Artificial Intelligence*, 137(1-2):217–238, May 2002.

[112] W. Resource. SAT competitions. http://www.satcompetition.org/.

[113] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

[114] T. Ritter. Latin squares: A literature survey. *Available online at: http://www.ciphersbyritter.com/RES/LATSQ.HTM*, 2003.

[115] J. Roberto J. Bayardo and J. D. Pehoushek. Counting models using connected components. In *Proceedings of 17th National Conference on Artificial Intelligence (AAAI)*, pages 157–162, 2000.

[116] S. Roweis and Z. Ghahramani. A unifying review of linear Gaussian models. *Neural Computation*, 11:305–345, 1997.

[117] D. B. Rubin. The calculation of posterior distributions by data augmentation: Comment: A noniterative sampling/importance resampling alternative to the data augmentation algorithm for creating a few imputations when fractions of missing information are modest: The SIR Algorithm. *Jornal of the American Statistical Association*, 82(398):543–546, 1987.

[118] R. Y. Rubinstein. *Simulation and the Monte Carlo Method.* John Wiley & Sons Inc., 1981.

[119] T. Sang, P. Beame, and H. Kautz. Heuristics for fast exact model counting. In *Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 226–240, 2005.

[120] B. Selman, H. Kautz, and B. Cohen. Noise strategies for local search. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 337–343, 1994.

[121] R. D. Shachter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 221–234, 1990.

[122] H. D. Sherali, A. Narayanan, and R. Sivanandan. Estimation of origin-destination trip-tables based on a partial set of traffic link volumes. *Transportation Research, Part B: Methodological*, pages 815–836, 2003.

[123] L. Simon, D. L. Berre, and E. Hirsch. The SAT 2002 competition. *Annals of Mathematics and Artificial Intelligence(AMAI)*, 43:307–342, 2005.

[124] O. Skare, E. Bolviken, and L. Holden. Improved sampling-importance resampling and reduced bias importance sampling. *Scandinavian Journal of Statistics*, 30:719–737, 2003.

[125] N. Sorensson and N. Een. Minisat v1.13-a SAT solver with conflict-clause minimization. In *SAT 2005 competition*, 2005.

[126] L. G. Valiant. The complexity of enumeration and reliability problems. *Siam Journal of Computation*, 8(3):105–117, 1987.

[127] L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. In *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 458–463, New York, NY, USA, 1985.

[128] T. Walsh. SAT v CSP. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming*, pages 441–456, London, UK, 2000. Springer-Verlag.

[129] T. Walsh. Permutation problems and channelling constraints. In *Proceedings of the 8th International Conference on Logic Programming and Automated Reasoning (LPAR)*, pages 377–391, 2001.

[130] W. Wei, J. Erenrich, and B. Selman. Towards efficient sampling: Exploiting random walk strategies. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 670–676, 2004.

[131] W. Wei and B. Selman. A new approach to model counting. In *Proceedings of Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 324–339, 2005.

[132] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51:2282–2312, 2004.

[133] C. Yuan and M. J. Druzdzel. Importance sampling algorithms for Bayesian networks: Principles and performance. *Mathematical and Computer Modelling*, 43(9-10):1189–1207, 2006.

[134] J. Yuan, K. Shultz, C. Pixley, H. Miller, and A. Aziz. Modeling design constraints and biasing in simulation using BDDs. In *Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design (ICCAD)*, pages 584–590, 1999.