UNIVERSITY OF CALIFORNIA,
IRVINE

Anytime Approximate Inference in Graphical Models

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Qi Lou

2018

DEDICATION

To my grandfather in heaven.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

It would be an empty universe indeed if it were not for the people I love, and who love me.

Stephen Hawking, *Brief Answers to the Big Questions*

This PhD journey would surely not be possible without these people to be acknowledged here.

First and foremost, I would like to express my highest appreciation and gratitude to my amazing advisor – Prof. Alexander Ihler, for his guidance, support, and consideration during the course of my PhD. Alex has been such a fantastic mentor, teacher, and collaborator that what I have learned from him, probably just a very small portion of what he is capable of, has dramatically shaped my knowledge, skills and mindset. Over these years one fact that I have realized (and I wish I had realized earlier) is: the more I interact with him, the more research progress I will make – this was the reason that I vetoed last summer the switch to a much brighter and larger office which, however, is further away from Alex's. I was (and I still am now) constantly impressed by his critical thinking, his scientific writing, his deep understanding of many subjects, his integrity, his frankness, (maybe also infected by) his sense of humor, his walking speed, and his customer loyalty to Dr Pepper. Alex is the best advisor that I can think of, and I hope I can still resort to him for advice in the future.

I have benefited tremendously from interactions with Prof. Rina Dechter. Knowing her as an expert in the field, I approached Rina to seek collaboration in 2015, which has resulted in five papers accepted/published at several renowned AI/ML venues; these papers serve as the core of this dissertation. Rina challenged me on every sentence/notation that I wrote, which forced me to think deeper and make them more accessible to readers. I still remember her scrutiny led to a fix to a flawed proposition that was previously not noticed by anyone. I admire and appreciate her work attitude – feedbacks from her can be on holidays or at 2:00 a.m. in the morning. I thank her for always being very supportive to me, e.g., she decided to fly back to Irvine from the East Coast to attend my final defense.

I would like to thank Prof. Sameer Singh for being on my final thesis defense and advancement committees. I first met Sameer when he came to give a candidate talk in 2016. In the later years, I found he is so warm-hearted that he has never turned down any favor I asked for; I am very thankful to him. I also want to thank Prof. Charless Fowlkes and Prof. Zhaoxia Yu who served on my advancement committee. I am also really grateful to Prof. Xiaohui Xie for facilitating my collaboration with his students.

Outside UCI, I want to express my gratitude to Prof. Dhruv Batra at Georgia Tech, who offered me an invaluable opportunity to work with him at Virginia Tech, and recommended me to Alex when I applied to UCI. I would like to thank my mentors at Adobe Research, Dr.

Somdeb Sarkhel, Dr. Saayan Mitra, and Dr. Viswanathan Swaminathan, with whom I had a wonderful summer in 2017.

I am so fortunate to have interacted with a number of great peers (especially those in the ICS community) who made this journey more colorful than it could have been. Starting from my labmates, I would like to thank Qiang Liu who acted like a big brother when I first arrived in 2014, helping me with my research (I guess I could have been more productive if he had stayed longer at UCI) and my accommodation (in the past four years, I visited the same barber in Irvine initially recommended by him). Wei Ping and I spent many days and nights together in DBH 4051. He has helped me in many aspects such as paper writing, thesis formatting, and job hunting, for which I am very grateful. I also thank other labmates including Tiancheng Xu, Jonathan Stroud, Noble Kennamer, and Nick Gallo for creating a friendly environment for me to study. Rina's group are also quite close to me. In particular, I want to thank William Lam and Junkyu Lee from whom I received a lot of help. I appreciate others in Rina's lab including Filjor Broka, Bobak Pezeshki for being kind to me. Due to the tight schedule when I am writing this thesis, I hope my other friends can forgive me for not being able to mention their names here. It has been a luxury to have you in this journey.

To conclude, I want to thank my parents (especially my extremely hardworking mother) from the bottom of my heart for their love and support throughout the years. I feel truly blessed to have you in my life – Nathan, an endless source of joy, and Carrie, to whom I am forever indebted.

CURRICULUM VITAE

Qi Lou

EDUCATION

Doctor of Philosophy in Computer Science	**2014 – 2018**
University of California, Irvine	Irvine, California

INTERNSHIP

Research Intern	**06/2017 – 09/2017**
Big Data Experience Lab, Adobe Research	San Jose, California
Research Intern	**04/2014 – 08/2014**
Machine Learning & Perception Group, Virginia Tech	Blacksburg, Virginia

TEACHING

Teaching Assistant	**University of California, Irvine**
Introduction to Artificial Intelligence	Fall 2016
Machine Learning & Data Mining	Winter 2016
Introduction to Graphical Models	Fall 2015

ACADEMIC REVIEWING

Journal of Machine Learning Research (JMLR)	2018
Conference on Neural Information Processing Systems (NIPS)	2018
International Conference on Machine Learning (ICML)	2018
AAAI Conference on Artificial Intelligence (AAAI)	2018
Conference on Uncertainty in Artificial Intelligence (UAI)	2016 – 2018

PUBLICATIONS

Qi Lou, Rina Dechter, Alexander Ihler. Interleave variational optimization with Monte Carlo sampling: A tale of two approximate inference paradigms. *AAAI Conference on Artificial Intelligence (AAAI)*, 2019. To appear.

Tiehang Duan, **Qi Lou**, Sargur Srihari, Xiaohui Xie. Sequential embedding induced text clustering, a non-parametric Bayesian approach. 2018. Under review.

Qi Lou, Somdeb Sarkhel, Saayan Mitra, Viswanathan Swaminathan. Content-based effectiveness prediction of video advertisements. *IEEE International Symposium on Multimedia (ISM)*, 2018. Short paper. To appear.

Qi Lou, Rina Dechter, Alexander Ihler. Finite-sample bounds for marginal MAP. *Uncertainty in Artificial Intelligence (UAI)*, 2018.

Qi Lou, Rina Dechter, Alexander Ihler. Anytime anyspace AND/OR search for bounding marginal MAP. *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

Qi Lou, Rina Dechter, Alexander Ihler. Dynamic importance sampling for anytime bounds of the partition function. *Neural Information Processing Systems (NIPS)*, 2017.

Qi Lou, Rina Dechter, Alexander Ihler. Anytime anyspace AND/OR search for bounding the partition function. *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

Wentao Zhu, **Qi Lou**, Yeeleng Scott Vang, Xiaohui Xie. Deep multi-instance networks with sparse label assignment for whole mammogram classification. *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2017.

Qi Lou, Raviv Raich, Forrest Briggs, Xiaoli Fern. Novelty detection under multi-instance multi-label framework. *IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2013.

Forrest Briggs, Xiaoli Fern, Raviv Raich, **Qi Lou**. Instance annotation for multi-instance multi-label learning. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 7(3), 2013.

Raviv Raich, Forrest Briggs, **Qi Lou**, Xiaoli Fern, David Mellinger. A framework for analyzing bioacoustics audio recordings as multiple instance multiple label data. *Meetings on Acoustics*, 19(1), 2013.

Qi Lou, Ligang Liu. Curve intersection using hybrid clipping. *Computers & Graphics*, 36(5), 2012.

ABSTRACT OF THE DISSERTATION

Anytime Approximate Inference in Graphical Models

By

Qi Lou

Doctor of Philosophy in Computer Science

University of California, Irvine, 2018

Professor Alexander Ihler, Chair

Graphical models are a powerful framework for modeling interactions within complex systems. Reasoning over graphical models typically involves answering inference queries, such as computing the most likely configuration (*maximum a posteriori* or MAP) or evaluating the marginals or normalizing constant of a distribution (the *partition function*); a task called *marginal MAP* generalizes these two by maximizing over a subset of variables while marginalizing over the rest.

Exact computation of these queries is known to be intractable in general, leading to the development of many approximate schemes, the major categories of which are variational methods, search algorithms, and Monte Carlo sampling. Within these, *anytime* techniques that provide some guarantees on the correct value, and can be improved with more computational effort, are valued for quickly providing users with confidence intervals or certificates of accuracy and allow users to decide the desired balance of quality, time and memory.

In this dissertation, we develop a series of approximate inference algorithms for the partition function and marginal MAP with anytime properties by leveraging ideas and techniques from the three inference paradigms, and integrating them to provide hybrid solutions that inherit the strengths of all three approaches. We propose anytime anyspace best-first search algorithms that provide deterministic bounds on the partition function and marginal MAP. These best-first search schemes take advantage of both AND/OR tree search and optimized variational heuristics. We then extend this approach to give anytime probabilistic confidence bounds via a dynamic importance sampling algorithm, which interleaves importance sampling (using proposal distributions extracted from the variational bound) with our best-first search algorithm to refine the proposal. We also propose a framework for interleaving sampling with the optimization of the initial variational bound, which can automatically balance its computational effort between the two schemes. Overall, we show that our hybrid algorithms perform significantly better than existing methods, giving flexible approaches with excellent anytime confidence bounds.

Chapter 1

Introduction

Graphical models are a set of powerful frameworks for representing and reasoning in complex systems over a number of variables with probabilistic and deterministic information [Pearl, 1988, Lauritzen, 1996, Darwiche, 2009, Koller and Friedman, 2009, Dechter et al., 2010, Dechter, 2013]. Examples of graphical models include Bayesian networks, Markov random fields (MRFs), conditional random fields (CRFs), factor graphs, and influence diagrams. In the past few decades, they have been successfully applied to a wide range of domains including but not limited to computational biology [Friedman, 2004], computer vision [Szeliski, 2010], natural language processing [Jurafsky and Martin, 2008], reinforcement learning [Sutton and Barto, 1998].

Reasoning over graphical models typically involves (approximately) solving some combinatorial optimization problems or computing some fixed quantities such as *maximum a posteriori* (MAP) or *most probable explanation* (MPE), which aims to find a most likely configuration of all variables, the *partition function* that is the normalizing constant ensuring a proper probability measure over all variables, and marginal MAP (MMAP) [1] that generalizes the aforementioned ones by maximizing over a subset of variables with the remaining variables marginalized. The process of solving these problems is called *inference* in graphical models, and thus these problems are also known as inference queries or inference tasks. These inference

[1]In some literature, e.g., Park and Darwiche [2004], marginal MAP is simply called MAP.

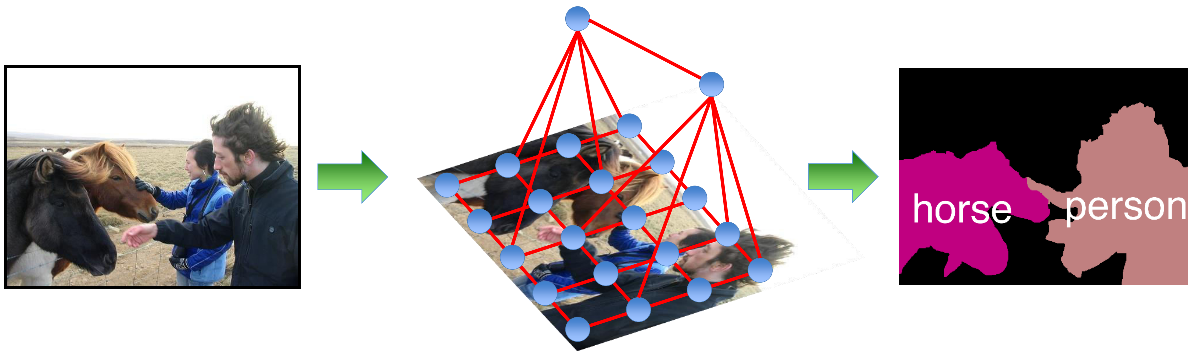


Figure 1.1: Semantic image segmentation as a MAP task associated with a conditional random field [Yadollahpour et al., 2013].

queries are widely used in many scenarios. For example, MAP plays an important role in structured prediction applications such as image denoising and semantic segmentation in computer vision [Nowozin and Lampert, 2011]. Figure 1.1 shows an example of formulating a semantic image segmentation task as a MAP query on a conditional random field.

Exact computation of these queries is known to be intractable in general, leading to the development of many approximate schemes, the major categories of which are variational methods [Wainwright and Jordan, 2008], search algorithms [Pearl, 1984], and Monte Carlo sampling [Liu, 2008]. Within these, techniques that provide some guarantees on the correct value, and can be improved with additional computation, are valued for providing users with concrete information or certificates of accuracy within a reasonable amount of time, and allowing users to decide the desired balance of quality versus time. Such an "anytime" property is crucial for tasks that are either not computationally feasible, or where it is not economical to compute the optimal answer [Zilberstein, 1996]. Moreover, algorithms that are able to run with limited memory resources are particularly useful because memory can be a bottleneck in practice; thus, an "anyspace" property is also important to users.

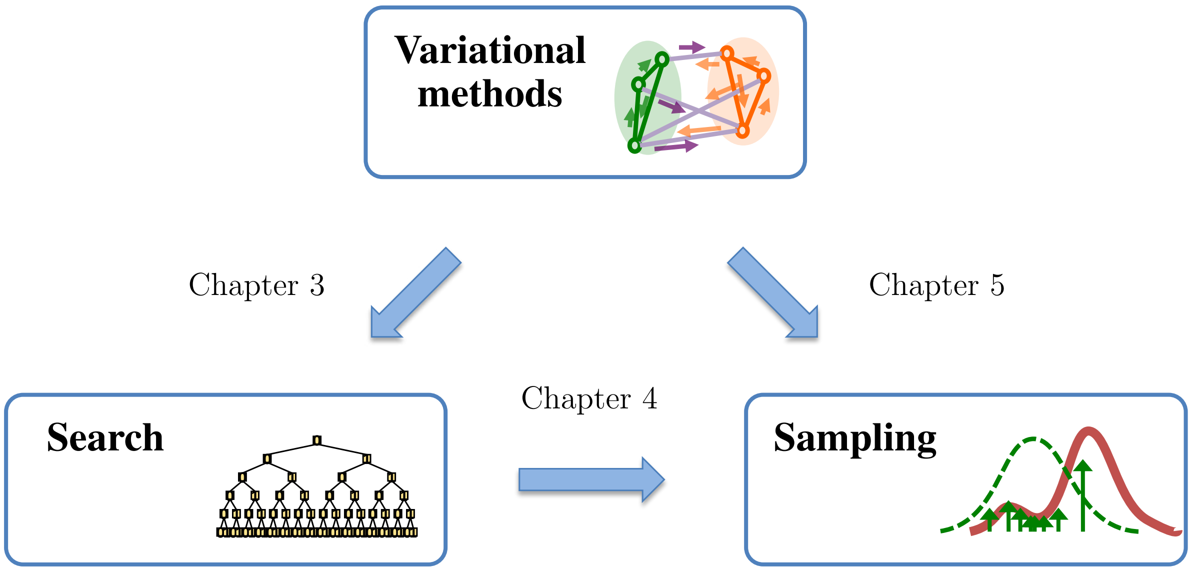


Figure 1.2: A pictorial illustration of how the three inference paradigms interact in the three chapters of the dissertation.

1.1 Main Contributions of This Dissertation

In this dissertation, we develop a series of approximate inference algorithms with anytime and anyspace properties by leveraging ideas and techniques from the aforementioned inference paradigms, and integrating them in a way that combines the relative strengths of each paradigm to provide approximations with strong bound guarantees. To the best of our knowledge, this is the first approach to combine all three major paradigms of approximate inference, and demonstrate the feasibility and potential of this unified perspective.

1.2 Outline of This Dissertation

A general outline of the rest of the dissertation is as follows: Chapter 2 introduces some background knowledge and notations that are used by the successive chapters. Chapters 3-5 present the main ideas and techniques developed by this thesis; Figure 1.2 depicts how the three inference paradigms interact in these three chapters. Chapter 6 discusses some future directions along this line of research and concludes the thesis. Some proofs and derivations omitted from the main text can be found in the appendices. To be more specific:

Chapter 2 first reviews some basics of graphical models, and formally defines the three inference tasks we consider. We then review some search terminology and algorithms, and show how search is applied on graphical models by introducing AND/OR search spaces along with some related concepts such as pseudo tree of the graph. We also review some key variational methods that will be used in the subsequent chapters, with a particular emphasis on elimination-based variational bounds (mini-bucket and similar methods) that will serve as building blocks for our own algorithms. Finally, we briefly describe the basics of Monte Carlo methods, introduce some concentration results, and conclude the chapter with a review of importance sampling.

Chapter 3 develops best-first search algorithms for deterministic bounds. We first review some related work on search based methods for inference queries and weighted model counting. We then present our approach, an anytime anyspace heuristic search algorithm that improves deterministic bounds for the partition function; this priority-driven best-first search scheme takes advantage of both AND/OR tree search and optimized variational heuristics, to efficiently reduce the bound gap on the partition function. We further extend this algorithm to bound marginal MAP by treating the maximization and summation operators in a unified way during search, using a carefully designed priority system that aims to drive down an upper bound on the MMAP optimum as quickly as possible. This chapter demonstrates how best-first search can be used to refine variational bounds in an anytime manner, as depicted in Figure 1.2. This chapter is partly based on our published work, Lou et al. [2017a] and Lou et al. [2018a].

Chapter 4 focuses on developing sampling algorithms to provide strong confidence intervals on the partition function and marginal MAP queries. We first propose a *dynamic importance* *sampling* (DIS) algorithm that provides anytime probabilistic bounds (i.e., they hold with probability $(1-\delta)$ for some user-selected confidence parameter δ) on the partition function. Our algorithm works by interleaving importance sampling with the variational bounds and best first search algorithm developed in Chapter 3, which acts to adapt and refine the proposal distribution of successive samples. We also generalize this idea to marginal MAP tasks by first converting the problem of bounding the MMAP optimum to a surrogate task defined by bounding a series of summation problems of an augmented graphical model, and then adapting DIS to provide finite-sample bounds for the surrogate task. These algorithms integrate ideas and techniques from all three inference paradigms. This chapter is partly based on our published work, Lou et al. [2017b] and Lou et al. [2018b].

Chapter 5 proposes an approach that interleaves Monte Carlo sampling with the initial optimization of the variational bounds, leading to faster improvement in confidence intervals early in the inference process. We propose an adaptive interleaving policy that can automatically balance the computational effort between these two schemes in an instance-dependent way. Our framework inherits the relative strengths of both schemes, leads to tighter anytime bounds and an unbiased estimate of the partition function, and allows flexible trade-offs between memory, time, and solution quality. This chapter is partly based on our paper, Lou et al. [2019].

Chapter 6 concludes the thesis. We summarize the dissertation, highlight its key contributions, and point out some directions to inspire future research.

Chapter 2

Background

In this chapter, we review some background knowledge and introduce notation that will be needed for the thesis. We organize the chapter as follows. Section 2.1 introduces some graphical model concepts used throughout the thesis, and formally defines the three primary inference tasks on graphical models. Sections 2.2, 2.3, and 2.4 describe the perspectives of and review some techniques from the three main approximate inference paradigms, consisting of variational bounds, search algorithms, and Monte Carlo methods, respectively. In particular, Section 2.2 reviews elimination-based variational bounds, including mini-bucket and weighted mini-bucket; Section 2.3 introduces heuristic search frameworks and AND/OR search spaces; and Section 2.4 reviews some Monte Carlo methods with a focus on importance sampling.

2.1 Graphical Models

Graphical models provide a formal mechanism for describing useful structures in large, complex problems. In particular, graphical models assume that a large complex function f, defined over some high-dimensional system, is actually an aggregation of smaller, more "local" functions that represent interactions among small portions of the whole. Then, the overall complexity of the system can be thought of as arising from the many, interdependent components of the function.

A	B	$f_1(A,B)$
0	0	0.24
0	1	0.56
1	0	1.1
1	1	1.2

$\cdots$

B	C	$f_3(B,C)$
0	0	0.12
0	1	0.36
1	0	0.3
1	1	1.8

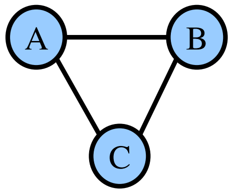


Figure 2.1: A pairwise model over three binary variables. Factors $f_1(A, B), f_2(A, C), f_3(B, C)$ are represented by tables.

More formally, a graphical model is a collection of variables, domains, and nonnegative functions or factors. Let $X = (X_1, \ldots, X_M)$ be a vector of random variables, where each X_i takes values in a domain $\mathcal{X}_i$; variable domains can be either discrete or continuous. This thesis only focuses on the discrete case unless otherwise stated, i.e., we assume $|\mathcal{X}_i|$ finite. We use lower case letters, e.g., $x_i \in \mathcal{X}_i$, to indicate a value of X_i, and x to indicate an assignment of X. A graphical model over X consists of a set of factors $\mathcal{F} = \{f_\alpha(X_\alpha) \mid \alpha \in \mathcal{I}\}$, where each factor f_α is defined on a subset $X_\alpha = \{X_i \mid i \in \alpha\}$ of X, called its scope. We can interpret $\mathcal{F}$ as an unnormalized probability measure, so that

$$f(x) = \prod_{\alpha \in \mathcal{I}} f_\alpha(x_\alpha). \tag{2.1}$$

We associate an undirected graph $\mathcal{G} = (V, E)$ (called the *primal graph*) with $\mathcal{F}$, where each node $i \in V$ corresponds to a variable X_i and we connect two nodes, $(i, j) \in E$, if and only if $\{i, j\} \subseteq \alpha$ for some α. The elements of $\mathcal{I}$ then correspond to cliques (fully connected subsets) of the graph $\mathcal{G}$.

Example 2.1. Figure 2.1 shows a pairwise model over three binary variables. There are three local functions $f_1(A, B), f_2(A, C), f_3(B, C)$, whose product defines an overall function $f(A, B, C)$ over variables A, B, C. Each local function induces an edge between variables in its scope. Those local functions are usually represented by tables.

2.1.1 Inference Tasks in Graphical Models

Given a graphical model, *inference* refers to answering probabilistic queries about the model. There are three typical inference queries that we formally define here.

MAP

The *maximum a posteriori* (MAP) or *most probable explanation* (MPE) task aims to find the maximum value of f, and a configuration (assignment of values to variables) achieving that value, i.e.,

$$x^\star = \operatorname*{argmax}_x f(x) \qquad\qquad f^\star = f(x^\star) = \max_x f(x)$$

Note that it is possible that there are multiple optimal solutions; since we usually care about finding one, we use $x^\star$ to denote one optimal assignment instead of the set of all optimal assignments. The MAP task is also called the optimization or max inference task in graphical models. This task is known to be NP-hard [Shimony, 1994] in general.

The Partition Function

Another common task in graphical models is to compute the normalizing constant defined as

$$Z = \sum_x \prod_{\alpha \in \mathcal{I}} f_\alpha(x_\alpha), \tag{2.2}$$

which is often called the *partition function*. Z ensures

$$p(x) = f(x)/Z \tag{2.3}$$

to be a proper probability distribution, where $p(x)$ is called the underlying distribution of the model. Computing Z is often a key task in evaluating the probability of observed data, model selection, or computing predictive probabilities. This task is also known as the sum inference task, and closely related to weighted model counting in constraint satisfaction problems (CSPs). It is #P-hard [Valiant, 1979].

Marginal MAP

Marginal MAP (MMAP) generalizes the aforementioned two tasks by maximizing over a subset of variables with the remaining variables marginalized. Let X_{M} be a subset of X called MAX variables, and $X_{\mathrm{S}} = X \setminus X_{\mathrm{M}}$ SUM variables. The MMAP task seeks an assignment $x^\star_{\mathrm{M}}$ of X_{M} with the largest marginal probability, and its value:

$$x^\star_{\mathrm{M}} = \operatorname*{argmax}_{x_{\mathrm{M}}} \sum_{x_{\mathrm{S}}} f(x) \qquad\qquad \pi^\star = \pi(x^\star_{\mathrm{M}}) = \sum_{x_{\mathrm{S}}} f(x^\star_{\mathrm{M}}, x_{\mathrm{S}}) \tag{2.4}$$

Note that, if X_{M} is an empty set, the value $\pi^\star$ is simply the partition function; if X_{S} is empty, MMAP reduces to the standard MAP inference task. MMAP is known to extremely hard; its decision version is $\mathrm{NP}^{\mathrm{PP}}$-complete [Park, 2002]), and it is commonly believed to be harder than either pure max inference or sum inference, and can even be intractable on tree-structured models, for which max or sum inference is efficient.

2.2 Variational Bounds

Variational algorithms are a class of deterministic approximations to the inference problems. They cast the problem of computing a quantity of interest (say, $f^\star$ or Z) into a continuous optimization problem, and then solve an approximate, simplified version of this optimization problem, for example by applying some form of relaxation [Wainwright and Jordan, 2008].

A number of variational algorithms have been developed over the past few decades, including loopy belief propagation [Pearl, 1988], mean field algorithms [Opper and Saad, 2001], expectation propagation [Minka, 2001], and many others. We refer readers to Wainwright and Jordan [2008] for a detailed survey of the framework. For this thesis, we focus on approximate elimination approaches, which are a subset of variational methods that provide bound guarantees.

Elimination-based algorithms solve inference tasks by *eliminating* variables one at a time with respect to a prescribed variable ordering, where "eliminating a variable" means that we apply some operator (e.g., "sum" or "max") to a set of candidate functions to generate new functions whose scopes do not contain that variable. In what follows, we first introduce bucket elimination (BE) Dechter [1999] which is an exact algorithm that directly eliminates variables in sequence, then introduce mini-bucket elimination (MBE) [Dechter and Rish, 2003] which alleviates the computational complexity of BE by eliminating variables approximately, and finally introduce weighted mini-bucket (WMB) [Liu and Ihler, 2011] which generalizes both BE and MBE. For the sake of convenience we consider the partition function task when describing these three algorithms; only minor modifications are required for these algorithms to be applied to the other two tasks, which we describe at the end of the section.

■ 2.2.1 Bucket Elimination (BE)

Given a variable elimination ordering, BE processes variables one by one with respect to this elimination ordering. For any variable, say, X_i, from the original model, BE first collects those not-yet-processed factors with X_i in their scopes; this collection of factors, together with factors generated by processing variables prior to X_i in the ordering that also have X_i in their scopes, forms a set of factors, termed the *bucket* of X_i and denoted $\mathcal{B}_i$. BE then marginalizes X_i out from the product of factors in $\mathcal{B}_i$, resulting in a new factor; if this new factor is non-constant, we denote it $\lambda_{i\to\pi_i}$, and call it a (forward) message from X_i to X_{π_i}:

$$\lambda_{i\to\pi_i} = \sum_{x_i} \prod_{f_\alpha \in \mathcal{B}_i} f_\alpha, \tag{2.5}$$

where X_{π_i} is the earliest uneliminated variable in the scope of this new factor. The message $\lambda_{i\to\pi_i}$ is then placed in the bucket of X_{π_i} for later processing. After all variables are processed, BE gives the exact value of Z by taking the product of all constant factors generated during the elimination process.

Both the time and space complexity of BE are exponential in the *induced width* of the graph and order, which intuitively corresponds to the largest number of variables in the scope of any message computed during the elimination. Thus, BE becomes impractical in the case that the induced width is large.

■ 2.2.2 Mini-bucket Elimination (MBE)

To alleviate the computational burden of BE, MBE introduces a user-specified parameter called the *ibound*, and ensures that no message generated by MBE has a scope size larger than *ibound*. To be more specific, when processing a bucket, say, $\mathcal{B}_i$, MBE partitions it into several disjoint subsets $\{\mathcal{B}_i^j\}$ called *mini-buckets* such that each mini-bucket involves no more than (*ibound* +1) variables. MBE then selects one mini-bucket, say (without loss of generality) the first mini-bucket $\mathcal{B}_i^1$, for marginalization, and the rest for maximization, which leads to the following inequality:

$$\sum_{x_i} \prod_{f_\alpha \in \mathcal{B}_i} f_\alpha \leq \prod_j \lambda_{i\to\pi_{ij}},$$

where

$$\lambda_{i \to \pi_{ij}} = \begin{cases} \sum_{x_i} \prod_{f_\alpha \in \mathcal{B}_i^1} f_\alpha, & \text{if } j = 1; \\ \max_{x_i} \prod_{f_\alpha \in \mathcal{B}_i^j} f_\alpha, & \text{otherwise.} \end{cases} \tag{2.6}$$

Analogous to BE, the message $\lambda_{i \to \pi_{ij}}$ will be placed in the bucket of $X_{\pi_{ij}}$, where $X_{\pi_{ij}}$ is the earliest uneliminated variable in the scope of $\lambda_{i \to \pi_{ij}}$. Taking the product of the constant factors generated during this process provides an upper bound on Z. A similar procedure can also be used to compute a lower bound on Z, by replacing the "max" operator with the "min" operator in the mini-bucket elimination step (2.6).

Compared to BE, MBE ensures the overall complexity will be exponential in the user-controlled parameter *ibound*, rather than in the induced width of the graph, which may be very high. Selecting a larger *ibound* usually provides a better bound; when the *ibound* is larger than the graph's induced width, MBE is identical to BE and thus gives exact answers. Thus, MBE trades off bound quality with computational complexity via the *ibound*, but at a cost that increases exponentially in both time and memory.

Bounds from MBE can be improved via moment matching [Flerova et al., 2011]. Partitioning of mini-buckets can also affect the quality of bounds; we refer readers to Rollon and Dechter [2010], Rollon et al. [2013] and Forouzan and Ihler [2015] for a more detailed treatment on this topic.

■ 2.2.3 Weighted Mini-bucket (WMB)

Weighted mini-bucket (WMB) generalizes its predecessors, BE and MBE, by replacing the exact summation with a "power sum" operator during variable elimination.

More specifically, WMB is analogous to MBE in that it partitions bucket $\mathcal{B}_i$ of X_i into disjoint mini-buckets $\{\mathcal{B}_i^j\}$ with their scope sizes bounded by (*ibound* +1). WMB then assigns a "weight" ρ_{ij} to each $\mathcal{B}_i^j$, and eliminates X_i in factors of $\mathcal{B}_i^j$ using the power sum (see (2.7) for a self-explanatory definition), which generates a message $\lambda_{i\to\pi_{ij}}$:

$$\lambda_{i\to\pi_{ij}} = (\sum_{x_i} \prod_{f_\alpha \in \mathcal{B}_i^j} f_\alpha^{\frac{1}{\rho_{ij}}})^{\rho_{ij}}. \tag{2.7}$$

When the weights ρ_{ij} are positive and sum to one, i.e.,

$$\rho_{ij} > 0, \qquad \sum_j \rho_{ij} = 1, \tag{2.8}$$

Hölder's inequality [Hardy et al., 1952] guarantees that the product of the messages $\lambda_{i\to\pi_{ij}}$ is an upper bound of the sum:

$$\sum_{x_i} \prod_{f_\alpha \in \mathcal{B}_i} f_\alpha \leq \prod_j \lambda_{i\to\pi_{ij}}. \tag{2.9}$$

Therefore, WMB provides an upper bound of Z if (2.8) is satisfied for all variables. Liu and Ihler [2011] shows that the resulting upper bound is equivalent to a class of bounds based on tree reweighted (TRW) belief propagation [Wainwright et al., 2005], or more generally, conditional entropy decomposition [Globerson and Jaakkola, 2007].

Note that, like MBE, WMB can also be used to compute a lower bound when the factors are strictly positive; this is obtained by replacing the non-negativity condition on the weights in (2.8) with a condition that exactly one weight, say, ρ_{ik} is positive, and the rest are negative:

$$\rho_{ik} > 0, \qquad \rho_{ij} < 0 \text{ if } j \neq k, \qquad \sum_j \rho_{ij} = 1. \tag{2.10}$$

The WMB bounds can be tightened by *cost shifting* (or *reparameterization*) and weight optimization, which can be implemented in a forward-backward message passing procedure.

We refer readers to Liu and Ihler [2011], Liu [2014], Ping et al. [2015] for more details.

Example 2.2. We illustrate how BE, MBE and WMB behave on the model shown in Figure 2.1, assuming the elimination ordering is A, B, C, and setting $ibound = 1$. We use lowercase letters a, b, c to denote an instantiation of A, B, C. For MBE and WMB, we use the upper bound variant of the algorithms.

- BE first creates bucket $\mathcal{B}_A = \{f_1, f_2\}$ for A by collecting all factors involving A, and then eliminates A to generate a message

  $$\lambda_{A\to B}(b,c) = \sum_a f_1(a,b) f_2(a,c)$$

  which is placed into bucket $\mathcal{B}_B$ for B, resulting in $\mathcal{B}_B = \{f_3, \lambda_{A\to B}\}$. BE next eliminates B and generates a message:

  $$\lambda_{B\to C}(c) = \sum_b \lambda_{A\to B}(b,c) f_3(b,c).$$

  Finally, BE eliminates C from bucket $\mathcal{B}_C = \{\lambda_{B\to C}\}$, giving the exact value of Z:

  $$Z = \sum_c \lambda_{B\to C}(c).$$

  Note that the time and memory complexity of these operations is dominated by the construction of $\lambda_{A\to B}(b,c)$; this function (table) takes memory quadratic in the domain size of the variables, for all pairs of values (b,c), and is cubic in its computation, iterating over all values of a for each (b,c).

- Given $ibound = 1$, MBE cannot construct a message over b and c jointly, and so instead must partition factors involving A into two mini-buckets, say, $\mathcal{B}_A^1 = \{f_1\}$ and $\mathcal{B}_A^2 = \{f_2\}$, and process them separately. We sum over mini-bucket $\mathcal{B}_A^1$ and maximize over $\mathcal{B}_A^2$:

$$\lambda_{A\to B}(b) = \sum_a f_1(a,b),$$
$$\lambda_{A\to C}(c) = \max_a f_2(a,c).$$

MBE then eliminates B, C consecutively (no further mini-bucket partitions are required), and returns an upper bound U_{mbe} of the partition function:

$$\lambda_{B\to C}(c) = \sum_b \lambda_{A\to B}(b) f_3(b,c),$$
$$U_{mbe} = \sum_c \lambda_{A\to C}(c)\lambda_{B\to C}(c).$$

- Like MBE, WMB creates two mini-buckets for A, then assigns weights ρ_{A1} and ρ_{A2} satisfying (2.8) to them respectively, and eliminates A using the power sum:

$$\lambda_{A\to B}(b) = \Big(\sum_a f_1^{\frac{1}{\rho_{A1}}}(a,b)\Big)^{\rho_{A1}},$$
$$\lambda_{A\to C}(c) = \Big(\sum_a f_2^{\frac{1}{\rho_{A2}}}(a,c)\Big)^{\rho_{A2}}.$$

  As with MBE, no further partitioning is necessary for B and C, so WMB eliminates them exactly, to give an upper bound U_{wmb}:

$$U_{wmb} = \sum_c \lambda_{B\to C}(c)\lambda_{A\to C}(c) = \sum_c \Big(\sum_b \lambda_{A\to B}(b) f_3(b,c)\Big)\lambda_{A\to C}(c)$$

Remark 2.1. Some conditions on the weights of WMB can be slightly relaxed without loss of the boundedness property of WMB. We take the upper bound case below for example; the lower bound case is analogous. The positiveness condition in (2.8) can be relaxed to nonnegativeness by defining

$$\Big(\sum_{x_i} \prod_{f_\alpha \in \mathcal{B}_i^j} f_\alpha^{\frac{1}{0}}\Big)^0 = \max_{x_i} \prod_{f_\alpha \in \mathcal{B}_i^j} f_\alpha.$$

This extension is continuous from the right side due to the following fact:

$$\max_{x_i} \prod_{f_\alpha \in \mathcal{B}_i^j} f_\alpha = \lim_{\rho_{ij} \to 0^+} \Big(\sum_{x_i} \prod_{f_\alpha \in \mathcal{B}_i^j} f_\alpha^{\frac{1}{\rho_{ij}}}\Big)^{\rho_{ij}}. \tag{2.11}$$

In light of (2.6), we can see that the MBE upper bound is a special case of the WMB upper bound, in which one of the mini-buckets is assigned weight one, and the others weight zero.

Remark 2.2. The three elimination algorithms, BE, MBE, and WMB, can easily be modified to compute or bound the other inference tasks:

- For MAP, to compute or bound the MAP value, BE and MBE simply replace the "sum" operator with the "max" operator in (2.5) and (2.6) respectively. BE then gives an optimal MAP value, while MBE gives an upper bound. In this setting, WMB can set all ρ_{ij}'s in (2.7) to be positive and near zero to give an equivalent upper bound. To decode a MAP solution for BE, MBE or WMB, we can compute assignments to each variable X_i sequentially along the *reverse* elimination ordering: at each X_i, we find the value x_i that maximizes the product of the factors in bucket $\mathcal{B}_i$, conditioned on the partial solution we have decoded so far (an assignment to X_{π_i}). Note that the solution decoded by BE will be optimal, but may be suboptimal for MBE and WMB in general.
- For MMAP, we first require a restricted elimination ordering that ensures all SUM variables X_{S} are eliminated before any MAX variables X_{M}. Then, when eliminating a SUM variable, the three algorithms operate the same way as if computing the partition function, while using the MAP variant forms when eliminating a MAX variable. Decoding can then be performed analogously to that for MAP.

2.3 Search in Graphical Models

Search is a general framework for problem solving and has a variety of applications [Russell and Norvig, 2009]. In this section, we introduce some search terminologies, several heuristic search algorithms, and how to formulate an inference task in a graphical model as a search problem on the so-called AND/OR space [Pearl, 1984].

2.3.1 Heuristic Search

A search problem is defined by a search space, typically a search tree or more generally a search graph, where *nodes* represent encodings of subproblems, and edges represents actions that are available to the problem solver. Each edge is associated with a *cost*. A unique node in the search space is designated as the *root*, and a (possibly empty) set of nodes are set as *goal* nodes. Search starts from the root, and traverses the search space by repeatedly *expanding* nodes according to some *search strategy*. Expansion means to generate all successors of a given node; this node is called the *parent* node while those successors are called its *child* nodes. The set of nodes at the frontier that are available for expansion is called the *open list* and denoted $OPEN$. We say that a portion of the search space has been *explored* if its nodes have been visited.

A common search objective is to find a least cost path from the root to one of the goal nodes, where the path cost is defined by a combination of the edge costs, typically by the sum; we assume this setting when we present a number of popular search algorithms in the sequel. In general, whether search is efficient for a problem of interest depends critically on how we formulate the problem as a search problem, and what search strategy we adopt to explore the search space.

A well-known category of search strategies is called *heuristic search*, or *informed search*, which exploits problem-dependent information beyond the definition of the problem itself; this is in contrast to so-called *blind* or *uninformed search* algorithms such as depth-first search (DFS) and breadth-first search (BFS). *Best-first search*, as an instance of heuristic search, has an evaluation function that incorporates domain-specific knowledge in some way, and that is defined at each node and measures how valuable a node is for expansion; this evaluation function greatly determines the performance of a best-first search algorithm, and thus requires careful crafting.

▢ 2.3.2 A* and SMA*

The A* search algorithm [Hart et al., 1968] is a widely used best-first search algorithm that combines two functions to define its evaluation function. The first is a cost function $g(n)$ that measures the path cost from the root to a node n, and the other is a heuristic function $h(n)$ that estimates the cheapest path cost from n to a goal node. The A* search algorithm then expands a frontier node with minimal value $g + h$ at each step.

The A* search algorithm has several nice properties, e.g., whenever A* selects a node for expansion, the optimal path to that node has been found, given some conditions on the heuristic function [Russell and Norvig, 2009]. A broad discussion of those properties can be found in Dechter and Pearl [1985].

However, the A* search algorithm has to maintain a (possibly implicit) priority queue of the frontier nodes, the size of which can grow very fast and makes A* memory-intensive in practice. To overcome this memory bottleneck, a series of memory-bounded variants of A* have been developed, including iterative-deepening A* (IDA*) [Korf, 1985], simplified memory bounded A* (SMA*) [Russell, 1992], and recursive best-first search [Korf, 1993], just to name a few.

Since SMA* plays an important role in developing our own search algorithms in Chapter 3, we present its essential components here. Before a user-specified memory limit is full, SMA* behaves the same as A*. When SMA* reaches the memory limit, since it cannot continue to add more search nodes to the frontier, it starts to delete some frontier nodes that it deems unpromising to free memory for the new additions. More precisely, when necessary it deletes the “worst” frontier node, i.e., the frontier node where the evaluation function achieves the highest value. In deleting the node, we lose information about the value (evaluation function) of that node, and so this lost information is propagated up to its parent, ensuring that the ancestor of a removed subtree is aware of the most current information on the best path that could include it. SMA* will then regenerate this path if it turns out to be more promising than any other path it has seen so far.

■ 2.3.3 AND/OR Search for Graphical Models

This section introduces OR search spaces and AND/OR search spaces for graphical models, which are useful tools for us to solve inference tasks via search.

OR Search Spaces

A typical way to do perform search for graphical models is to view search nodes as partial assignments of variables, and expansion as extending this partial assignment by assigning a value to a single variable at a time, following some variable ordering. In the simplest case, this process defines a search tree, called the *OR search tree*, in which the root is no assignment, and each leaf (or equivalently, path from root to a leaf, called a “solution path”) corresponds to a full assignment of the graphical model. Let M be the number of variables and d be the maximum domain size; then it is easy to see that the size of the full OR search tree is $O(d^M)$, which can be very large.

Example 2.3. Figure 2.2(b) shows a full OR search tree for the graphical model depicted in Figure 2.2(a). One solution path, marked red, corresponds to a complete configuration of all

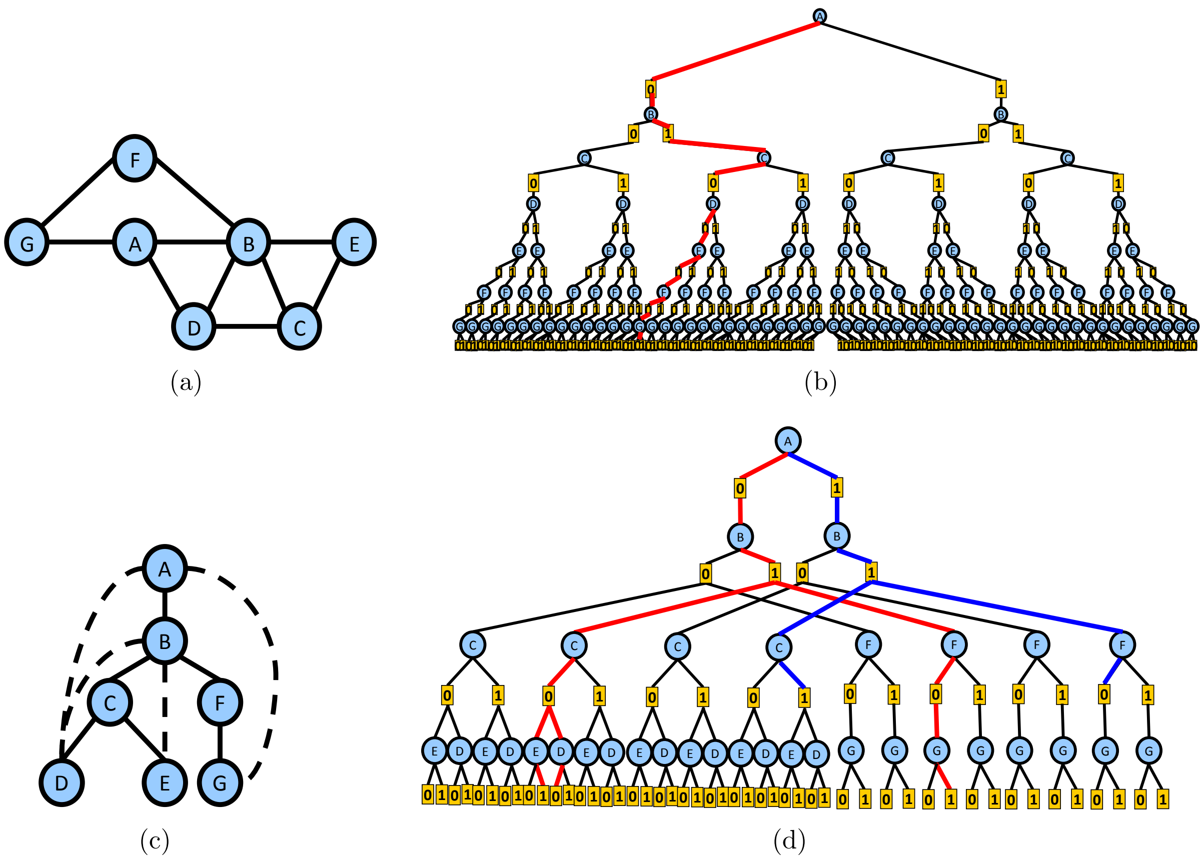


Figure 2.2: (a) A primal graph of a graphical model over 7 variables. (b) A full OR search tree for the model. One solution path is marked red. (c) A pseudo tree for the primal graph. (d) The full AND/OR search tree guided by the pseudo tree. One (full) solution tree is marked red. One partial solution tree is marked blue.

variables.

AND/OR Search Spaces

An AND/OR search space is a generalization of the aforementioned OR search space that enables us to exploit conditional independence structures during search [Dechter and Mateescu, 2007]. The AND/OR search space for a graphical model is defined relative to a *pseudo tree* that captures problem decomposition along a fixed search order.

Definition 2.1 (pseudo tree). A pseudo tree of an undirected graph $\mathcal{G} = (V, E)$ is a directed tree $\mathfrak{T} = (V, E')$ sharing the same set of nodes as $\mathcal{G}$. The tree edges E' form a subset of E, and we require that each edge $(i, j) \in E \setminus E'$ be a "back edge", i.e., the path from the root of $\mathfrak{T}$ to j passes through i (denoted $i \leq j$). $\mathcal{G}$ is called the primal graph of $\mathfrak{T}$.

Guided by a pseudo tree, we can construct an AND/OR search tree consisting of alternating levels of OR and AND nodes for a graphical model. Each OR node s is associated with a variable, which we lightly abuse notation to denote X_s; the children of s, $ch(s)$, are AND nodes corresponding to the possible assignments of X_s. The root $\emptyset$ of the AND/OR search tree corresponds to the root of the pseudo tree. Let $pa(c)$ indicate the parent of c, and $an(c) = \{n \mid n \leq c\}$ indicate the ancestors of c (including itself) in the tree.

In an AND/OR tree, any AND node c corresponds to a partial configuration $x_{\leq c}$ of X, defined by its assignment and that of its ancestors: $x_{\leq c} = x_{\leq p} \cup \{X_s = x_c\}$, where $s = pa(c)$, $p = pa(s)$. The corresponding variables of $x_{\leq c}$ is denoted as $X_{\leq c}$. For completeness, we also define $x_{\leq s}$ for any OR node s, which is the same as that of its AND parent, i.e., $x_{\leq s} = x_{\leq pa(s)}$. For any node n, the corresponding variables of $x_{\leq n}$ is denoted as $X_{\leq n}$. Let $de(X_n)$ be all variables below X_n in the pseudo tree; we define $X_{>n} = de(X_n)$ if n is an AND node; $X_{>n} = de(X_n) \cup \{X_n\}$ if n is an OR node. An instantiation of $X_{>n}$ is denoted $x_{>n}$.

A complete configuration X of the model corresponds to a subtree called a *solution tree*:

Definition 2.2 (solution tree). A solution tree T of an AND/OR search tree $\mathcal{T}$ is a subtree satisfying three conditions: (1) T contains the root of $\mathcal{T}$; (2) if an OR node is in T, exactly one of its children is in T; (3) if an AND node is in T, all of its children are in T.

We can see that there is a one-to-one correspondence between a complete assignment of X and a solution tree. Thus, for any full configuration x, we use T_x to denote its corresponding solution tree; for any solution tree T, we use x_T to denote its corresponding full configuration.

We generalize the notion of a "solution tree" to that of a "partial solution tree" in order to relate partial configurations of X to some subtrees in the AND/OR search tree:

Definition 2.3 (partial solution tree). A partial solution tree T of an AND/OR search tree $\mathcal{T}$ is a subtree satisfying three conditions: (1) T contains the root of $\mathcal{T}$; (2) if an OR node is in T, **at most** one of its children is in T; (3) if an AND node is in T, all of its children **or** none of its children are in T.

From the above definition, we can see any partial solution tree T defines a partial configuration x_T of X, where $x_T = \cup_{n \in T} x_{\leq n}$; we use $X_T \subset X$ to denote the corresponding variables of x_T. Note that two partial solution trees may induce an identical partial configuration, which is different from the case of solution trees. In the sequel, we may refer a solution tree as a *full* solution tree to avoid possible ambiguity.

The following property captures the size of a full AND/OR search tree. A full AND/OR search tree usually has a much smaller size than that of a full OR search tree.

Proposition 2.1. A full AND/OR tree has size of $O(b^h d^h)$, where h is the pseudo tree height, b is the max branching factor of the pseudo tree, and d is the max variable domain size. See Dechter and Mateescu [2007].

Example 2.4. Figure 2.2(d) shows a full AND/OR search tree for the model in Figure 2.2(a). A solution tree is marked red in the AND/OR search tree. According to the definition, a solution tree corresponds to a full instantiation of all the variables.

Inference Tasks as Search Problems on AND/OR Search Trees

We associate an arc cost w_c with each OR-AND arc (an edge connecting an OR parent node and one of its AND children), indexed by the AND child c, defined to be the product[1]of all

[1]Typical search formulations seek to *minimize* an *additive cost*, for which it is more convenient to use an arc cost that is the negative log of the one defined here, i.e., $-\log w_c$. However, when defining summation problems on search trees, we find it more convenient to adopt the product form described here, in which w_c is not precisely a "cost" but simply a partial function value.

factors f_α that are instantiated at c but not before:

$$w_c = \prod_{\alpha \in \mathcal{I}_c} f_\alpha(x_\alpha), \qquad\qquad \mathcal{I}_c = \{\alpha \mid X_{pa(c)} \in X_\alpha \subseteq X_{an(c)}\}$$

We define $w_s = 1$ for any AND-OR arc, indexed by the OR child s. It is then easy to see that the product of all arc costs on a path from the root,

$$g_c = \prod_{a \leq c} w_a,$$

defined as the cost of the path, equals the value of the factors whose scope is fully instantiated at c. We again lightly abuse the notation to use g as a function of $x_{\leq c}$ such that

$$g(x_{\leq c}) = g_c. \tag{2.12}$$

Note that this definition of function g is proper because it is easy to verify that any two nodes induce an identical partial assignment of X must have the same path cost. Analogously, we can define a cost g_T for any partial solution T as the product of all arc costs on T:

$$g_T = \prod_{a \in T} w_a. \tag{2.13}$$

Again, we define a functional version of g_T,

$$g(x_T) = g_T, \tag{2.14}$$

corresponding to the view that g_T is the product of all factors fully instantiated by x_T. As a special case, the cost of a full solution tree equals the value $f(x)$, where x is the corresponding complete configuration.

Finally, the purpose of the search tree is to compute some inference quantity for the model, such as the optimum $f^\star = \max_x f(x)$ or the partition function Z (see (2.2)). To this end, we associate an exact "value" v_n with each node n in the AND/OR search tree, which represents the inference task's value on the unexpanded portion of the search space below node n. The value v_n can be defined recursively in terms of its children and grandchildren as follows. We first define $v_n = 1$ for any leaf (since no part of the model remains uninstantiated). For maximization tasks, we have

$$\text{Max:} \qquad v_n = \begin{cases} \prod_{c \in ch(n)} v_c, & \text{if } n \text{ is an AND node,} \\ \max_{c \in ch(n)} w_c v_c, & \text{if } n \text{ is an OR node;} \end{cases} \tag{2.15}$$

while in the case of summation, the recursion defining v_n for node n is

$$\text{Sum:} \qquad v_n = \begin{cases} \prod_{c \in ch(n)} v_c, & \text{if } n \text{ is an AND node,} \\ \sum_{c \in ch(n)} w_c v_c, & \text{if } n \text{ is an OR node.} \end{cases} \tag{2.16}$$

Any search algorithm for reasoning about the model can be thought of as maintaining estimators (typically, upper or lower bounds) on these quantities at each node. In particular, for heuristic search, we assume that we have a heuristic function h_c^+ that gives upper bounds (or upper and lower bounds, h_c^+ and h_c^-) on v_c. These heuristics typically are more accurate deeper in the search tree, and therefore their updates can be propagated upwards to the root to yield tighter bounds to the overall inference value. Any search algorithm is then defined by the order of expansion of the search tree.

Remark 2.3. An even more compact search space for a graphical model, called an AND/OR search graph [Dechter and Mateescu, 2007], can be obtained by merging identical subproblems in the AND/OR search tree. However, graph search algorithms are harder to design and implement than tree search ones.

Remark 2.4. For maximization tasks, since our goal is to find a configuration with the highest value, goal nodes correspond to leaf nodes of an optimal solution tree in the AND/OR search tree. While for summation tasks, there is *no* goal node because in theory we have to traverse the whole AND/OR space to accumulate all the mass.

2.4 Monte Carlo Methods

Monte Carlo methods constitute another major framework for approximate inference in graphical models. In this section, we review some background about Monte Carlo methods with an emphasis on importance sampling.

Monte Carlo methods use random samples to create estimates of the quantities of interest. They typically formulate their goal as estimating an expectation of some function $u(x)$:

$$\mathrm{E}_t[u(x)] = \int_{\Omega} u(x)t(x)\,\mathrm{d}x, \tag{2.17}$$

where $t(x)$ is called the target distribution, defined on Ω, a measurable subset of some Euclidean space. When $u(x)$ and $t(x)$ are properly chosen, many inference tasks, such as the calculation of marginal probabilities and the partition function, can be expressed in this way [Andrieu et al., 2003]. The expectation in (2.17) can then be approximated using a sample mean $\widehat{u}$:

$$\mathrm{E}_t[u(x)] \approx \widehat{u} = \frac{1}{N}\sum_{i=1}^{N} u(x^i), \tag{2.18}$$

where samples $\{x^i\}_{i=1}^N$ are independently drawn from $t(x)$. The sample mean $\widehat{u}$ in (2.18) is an *unbiased* estimate of the expectation in (2.17):

$$\mathrm{E}_t[\widehat{u}] = \mathrm{E}_t[u(x)],$$

and converges to the expectation almost surely as N goes to infinity:

$$\widehat{u} \xrightarrow{a.s.} \mathrm{E}_t[u(x)].$$

If the variance $\mathrm{Var}_t[u(x)]$ is finite, the estimator $\widehat{u}$ is asymptotically Gaussian due to the fact that $\sqrt{N}\big(\widehat{u} - \mathrm{E}_t[u(x)]\big)$ converges in distribution to $\mathcal{N}\big(0, \mathrm{Var}_t[u(x)]\big)$ according to the central limit theorem:

$$\sqrt{N}\big(\widehat{u} - \mathrm{E}_t[u(x)]\big) \xrightarrow{d} \mathcal{N}\big(0, \mathrm{Var}_t[u(x)]\big);$$

this implies the probability mass concentrates rapidly around $\mathrm{E}_t[u(x)]$ as N increases.

2.4.1 Some Concentration Results

Concentration bounds use the fact the probability mass of random averages, such as $\hat{u}$, concentrate around their expected values to provide confidence intervals that hold with high probability. Many concentration results for Monte Carlo estimators have been developed over the years. We review several of these, since they will be used as key components of our proposed algorithms in subsequent chapters. We refer readers to Bousquet [2002], Boucheron et al. [2004] for a more thorough treatment. For notational convenience, we use the shorthand u_i for $u(x^i)$ and u for $u(x)$ interchangeably. We also select δ to be a confidence parameter, with $\delta \in (0, 1)$.

Markov's Inequality

A fundamental building block of most concentration bounds is Markov's inequality, which states:

Theorem 2.1 (Markov's inequality). *Assuming $u(x) \geq 0$, then $\forall \epsilon > 0$, we have*

$$\Pr[\widehat{u} \geq \epsilon] \leq \frac{\mathrm{E}_t[u]}{\epsilon}.$$

By setting $\epsilon = \mathrm{E}_t[u]/\delta$, we can derive a confidence interval from Markov's inequality:

$$\Pr\left[\ \mathrm{E}_t[u] \geq \delta \widehat{u}\ \right] \geq 1 - \delta, \tag{2.19}$$

which states that, for non-negative $u(x)$, $\delta\widehat{u}$ is a lower bound of the true expectation $\mathrm{E}_t[u]$ with probability at least $1 - \delta$.

Hoeffding's Inequality

Building on Markov's inequality, Hoeffding showed that averages of bounded random variables have rapidly decaying distribution tails.

Theorem 2.2 (Hoeffding's inequality [Hoeffding, 1963]). *Assuming $u(x) \in [0, 1]$, for $\forall \epsilon > 0$, we have*

$$\Pr[\mathrm{E}_t[u] \geq \widehat{u} + \epsilon] \leq e^{-2N\epsilon^2},$$
$$\Pr[\mathrm{E}_t[u] \leq \widehat{u} - \epsilon] \leq e^{-2N\epsilon^2},$$

hold respectively.

We can again derive confidence intervals from these inequalities:

$$\Pr\left[\mathrm{E}_t[u] \leq \widehat{u} + \sqrt{-\frac{\log \delta}{2N}}\right] \geq 1 - \delta,$$
$$\Pr\left[\mathrm{E}_t[u] \geq \widehat{u} - \sqrt{-\frac{\log \delta}{2N}}\right] \geq 1 - \delta.$$

Empirical Bernstein Bounds

A more sophisticated analysis can be used to derive a class of bounds that is often tighter in practice:

Theorem 2.3 (Empirical Bernstein bounds [Maurer and Pontil, 2009]). *Assuming* $u(x) \in [0, 1]$, *we have*

$$\Pr[\mathrm{E}_t[u] \le \widehat{u} + \sqrt{\frac{2\widehat{\mathrm{Var}}\log(2/\delta)}{N}} + \frac{7\log(2/\delta)}{3(N-1)}] \ge 1-\delta,$$
$$\Pr[\mathrm{E}_t[u] \ge \widehat{u} - \sqrt{\frac{2\widehat{\mathrm{Var}}\log(2/\delta)}{N}} - \frac{7\log(2/\delta)}{3(N-1)}] \ge 1-\delta,$$

hold respectively, where

$$\widehat{\mathrm{Var}} \quad = \quad \frac{1}{N-1}\sum_{i}^{N}(u_i - \widehat{u})^2 \quad = \quad \frac{\sum_{i=1}^N u_i^2}{N-1} - \frac{N\widehat{u}^2}{N-1} \tag{2.20}$$

is the sample variance of $\{u_i\}_{i=1}^N$, *corresponding to an unbiased estimate of the variance, i.e.,* $\mathrm{E}_t[\widehat{\mathrm{Var}}] = \mathrm{Var}_t[u]$.

Compared to Hoeffding's bounds, the empirical Bernstein bounds allow us to exploit second-order information (the estimated variance) and thus are typically tighter. Moreover, the empirical Bernstein bounds remain easy to compute since they only involve some simple statistics of the samples u_i; this is in contrast to some other bounds, such as Bennett's bounds [Bennett, 1962], that require access to potentially unknown quantities such as the true variance.

Remark 2.5. Both the Hoeffding and Empirical Bernstein concentration bounds require boundedness, ($u(x) \in [0, 1]$), unbiasedness ($\mathrm{E}_t[u_i] = \mathrm{E}_t[u]$), and independence of the samples u_i, but do not require that the u_i be identically distributed.

■ 2.4.2 Importance Sampling

Direct sampling from the target distribution $t(x)$ might be difficult or even impossible in many cases. Importance sampling (IS) offers an alternative approach, by introducing a *proposal distribution* $q(x)$ from which we are able to draw samples efficiently. The support of $q(x)$ should be large enough:

$$\{x \mid u(x)t(x) \neq 0\} \subseteq \{x \mid q(x) > 0\},$$

such that the ratio $u(x)t(x)/q(x)$ is finite whenever $u(x)t(x)$ is nonzero. Then, the target mean can be approximated using samples from the proposal distribution:

$$\begin{aligned} \mathrm{E}_t[u(x)] &= \mathrm{E}_q\left[\frac{u(x)t(x)}{q(x)}\right] \\ &\approx \frac{1}{N}\sum_{i=1}^{N}\frac{u(x^i)t(x^i)}{q(x^i)} \end{aligned} \tag{2.21}$$

where $\{x^i\}_{i=1}^N$ are drawn independently from $q(x)$. The variance of $u(x)t(x)/q(x)$ is

$$\begin{aligned} \mathrm{Var}_q\left[\frac{u(x)t(x)}{q(x)}\right] &= \mathrm{E}_q\left[\Big(\frac{u(x)t(x)}{q(x)} - \mathrm{E}_t[u(x)]\Big)^2\right] \\ &= \mathrm{E}_q\left[\Big(\frac{u(x)t(x)}{q(x)}\Big)^2\right] - \Big(\mathrm{E}_t[u(x)]\Big)^2. \end{aligned} \tag{2.22}$$

Ideally, we want to have a proposal distribution that leads to an estimate with the minimal variance. The following proposition (whose proof is given in Appendix A) claims that the proposal distribution $q^\star(x)$ proportional to $|u(x)|\,t(x)$ is optimal in terms of variance under very mild conditions.

Proposition 2.2. If we assume $\mathrm{E}_t[|u(x)|] > 0$, and let

$$q^\star(x) = |u(x)|t(x)/\,\mathrm{E}_t[|u(x)|], \tag{2.23}$$

then for any proposal distribution $q(x)$, we have

$$\mathrm{Var}_{q^\star}\left[\frac{u(x)t(x)}{q^\star(x)}\right] \leq \mathrm{Var}_q\left[\frac{u(x)t(x)}{q(x)}\right].$$

Two special cases of $q^\star(x)$ are worth mentioning here. When $u(x)$ is nonnegative, by substituting (2.23) into (2.22), we can see that $q^\star(x)$ achieves zero variance. Another special case is that when $u(x)$ is a constant, in which case $q^\star(x)$ is exactly the target distribution $t(x)$.

The effectiveness of IS highly depends on the proposal distribution. If there is a large discrepancy between $q(x)$ and $q^\star(x)$, the estimate in (2.21) can be very misleading [MacKay, 1998]. Finding a high-quality proposal distribution is difficult in general, which is often a major practical concern about the application of IS.

Estimating Z via importance sampling

IS is directly applicable to estimating the partition function Z by setting $u(x) = Z$ and $t(x) = p(x)$ where $p(x)$ is the underlying distribution of a graphical model $f(x)$ (see (2.1)–(2.3)). Thus, Z can be approximated by its IS estimate $\widehat{Z}$ defined as follows:

$$\begin{aligned} Z &= \mathrm{E}_q\left[\frac{f(x)}{q(x)}\right] \\ &\approx \widehat{Z} = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x^i)}{q(x^i)}. \end{aligned}$$

By following our previous arguments about the optimal proposal, we see that one guiding principle for designing a proposal distribution is to find one that is close to $p(x)$.

There is a lot of existing work using IS and its extensions for estimating the partition function [e.g., Salakhutdinov and Murray, 2008, Sohl-Dickstein and Culpepper, 2012, Ma et al., 2013, Naesseth et al., 2014, Burda et al., 2015, Liu et al., 2015a,b]. However, most of these methods do not provide any certificate on the accuracy of their estimates. One key exception is Liu et al. [2015a], which provides finite-sample bounds on the partition function; we review this approach in detail in Chapter 4.

Chapter 3

Best-first Search Aided by Variational Heuristics

In this chapter, we develop anytime anyspace best-first search algorithms for bounding the partition function and marginal MAP, taking advantage of the AND/OR tree structure and optimized variational heuristics to tighten deterministic bounds.

In Section 3.1, we study how our priority-driven best-first search scheme can improve on state-of-the-art variational bounds in an anytime way within limited memory resources, as well as the effect of the AND/OR framework to exploit conditional independence structure within the search process within the context of summation. We compare our resulting bounds to a number of existing methods, and show that our approach offers a number of advantages on real-world problem instances taken from recent UAI competitions.

In Section 3.2, we propose a best-first search algorithm that provides anytime upper bounds for marginal MAP. It folds the computation of external maximization and internal summation into an AND/OR tree search framework, and solves them simultaneously using a unified best-first search algorithm. The algorithm avoids some unnecessary computation of summation sub-problems associated with MAP assignments, and thus yields significant time savings. Furthermore, our algorithm is able to operate within limited memory. Empirical evaluation on three challenging benchmarks demonstrates that our unified best-first search algorithm using pre-compiled variational heuristics often provides tighter anytime upper bounds compared to those state-of-the-art baselines.

■ 3.1 Anytime Anyspace Best-first Search for The Partition Function

In this section, we develop an anytime anyspace best-first search algorithm that improves bounds for the partition function by exploring the AND/OR search space, using pre-compiled variational heuristics. We define and study the effect of several priority functions, which determine the order in which the search space is explored. Empirical results demonstrate that our approach, used with heuristics extracted from WMB (see Section 2.2) is almost always superior to several state-of-the search algorithms on various benchmark-memory settings.

■ 3.1.1 Introduction

As discussed in the background chapter, exact computation of the partition function is intractable in general, leading to the development of a broad array of approximate schemes. Particularly useful are schemes that provide guarantees, such as a confidence interval (upper and lower bounds), and can also be improved in an anytime and anyspace manner.

The approximate elimination methods and variational bounds described in Section 2.2 provide deterministic guarantees on the partition function. However, these bounds are not anytime; their quality depends critically on the amount of memory available, and they do not improve indefinitely without additional memory. On the other hand, Monte Carlo methods, such as those based on importance sampling [e.g., Liu et al., 2015a], or approximate hash-based counting for weighted SAT [e.g., Chakraborty et al., 2016] can smoothly trade time for quality, but provide only probabilistic bounds (e.g., they hold with probability $1 - \delta$ for some confidence parameter δ), and can be slow to provide tight intervals.

Historically, search techniques are well studied in graphical models for optimization (e.g., MAP and weighted CSP tasks) [Shimony and Charniak, 1991, Santos, 1991, Kask and Dechter, 2001, Bacchus et al., 2003a] and exact summation [Darwiche, 2001, Chavira and Darwiche, 2008, Bacchus et al., 2003b, Sang et al., 2005, Darwiche, 2009]. However, there has been relatively little use of search on approximating summation problems such as the partition function. One exception is Viricel et al. [2016], which adapts a depth-first branch-and-bound search scheme to provide deterministic upper and lower bounds on the partition function.

As we can see in Section 2.3.3, AND/OR search spaces provide an elegant framework that exploits conditional, possibly context-specific independence structure during search. In contrast to methods such as recursive conditioning or "clamping" [Darwiche, 2001, Dechter and Mateescu, 2007, Weller and Domke, 2016] and recent work on knowledge compilation [Kisa et al., 2014], that can also explore the AND/OR search space, most explicit AND/OR search algorithms were used for optimization, employing a fixed search order that restricts the search but enables the use of strong, pre-compiled heuristic functions that can lead to faster exploration and better early pruning [Marinescu and Dechter, 2009b,a, Otten et al., 2011, Otten and Dechter, 2012, Marinescu et al., 2014, 2015].

Other related approaches for summation queries include cutset-conditioning for exact solutions [Pearl, 1988, Dechter, 2013] or approximation with sampling [Bidyuk and Dechter, 2007]. Bidyuk et al. [2010] used conditioning to combine bound intervals on marginal probabilities.

3.1.2 AND/OR Best-first Search (AOBFS)

We explore the power of AND/OR search with precompiled heuristics for bounding sum inference tasks. In this section, we introduce our main algorithm; we first present a simplified version that is A*-like, and then generalize to an SMA*-like version to operate with limited memory.

Beginning with only the root $\emptyset$, we expand the search tree according to the pseudo tree structure and some control strategy (depth-first, breadth-first, etc.) With best-first search, we assign a priority (i.e., an evaluation value) to each frontier node on the search tree, then expand the top priority frontier node in each iteration.

More precisely, we maintain an explicit AND/OR search tree of visited nodes, denoted $\mathcal{S}$, whose frontier nodes are denoted *OPEN*. The remaining nodes of $\mathcal{S}$ are called *internal* nodes. For an internal node n, we denote *OPEN(n)* as the set of descendants of n that are in *OPEN*. A frontier AND node of $\mathcal{S}$ is called *solved* if it corresponds to a leaf node in the pseudo tree. An internal node of $\mathcal{S}$ is solved only if all its children are solved.

Recall from (2.16) that, for the summation problem, the *value* of a node n is defined recursively in terms of the edge weights w_c and values v_c of its children:

$$v_n = \begin{cases} \prod_{c \in ch(n)} v_c, & \text{if } n \text{ is an AND node,} \\ \sum_{c \in ch(n)} w_c v_c, & \text{if } n \text{ is an OR node.} \end{cases} \tag{3.1}$$

During search, we maintain upper and lower bounds on the values of explored nodes, denoted u_n and l_n, respectively. When a search node n is first generated (i.e., when it is in *OPEN*), these bounds can be initialized via the pre-compiled heuristics, $l_n = h_n^- \leq v_n \leq h_n^+ = u_n$, and then subsequently refined as n and its descendants are expanded during search. For internal nodes, given the currently expanded tree $\mathcal{S}$, we update the bounds u_n and l_n using information propagated from the frontier:

$$u_n = \begin{cases} \prod_{c \in ch(n)} u_c, & \text{if } n \text{ is an AND node,} \\ \sum_{c \in ch(n)} w_c u_c, & \text{if } n \text{ is an OR node,} \end{cases}$$
$$l_n = \begin{cases} \prod_{c \in ch(n)} l_c, & \text{if } n \text{ is an AND node,} \\ \sum_{c \in ch(n)} w_c l_c, & \text{if } n \text{ is an OR node.} \end{cases} \tag{3.2}$$

Note that these values depend implicitly on the search tree $\mathcal{S}$.

It will be useful to define a quantity at each node, that represents the total sum conditioned on the path from the root to that node. Let $branch(p, q)$ be all OR nodes that are siblings of some node s on the path $p < s \leq q$; these are "other" children of AND-node ancestors of q. For shorthand we use $branch(q) = branch(\emptyset, q)$ for paths from the root. Then, define

$$V_n = g_n \, v_n \prod_{s \in branch(n)} v_s, \tag{3.3}$$

where the branch terms correspond to incorporating the exact values for conditionally independent subproblems for sibling subtrees at AND nodes. As a special case, this quantity at the root $V_\emptyset$ equals the partition function Z. It will also be useful to define two node-specific quantities that are upper and lower bound analogues of V_n:

$$U_n = g_n \, u_n \prod_{s \in branch(n)} u_s, \tag{3.4}$$
$$L_n = g_n \, l_n \prod_{s \in branch(n)} l_s.$$

These quantities combine the path cost and heuristic in the usual way, e.g., the path cost g_n times the current heuristic bound, u_n. At the root, the dynamically updated bounds $U = U_\emptyset = u_\emptyset$, $L = L_\emptyset = l_\emptyset$ serve as anytime bounds on Z. Thus, we can interpret U_n and L_n

as the contributions of n to the global bounds U and L. Moreover, we show in Appendix B,

Proposition 3.1. The quantity V_n represents the total weighted sum over all solution trees that include the path from the root to node n. Given a current search tree $\mathcal{S}$, for all n, we have $L_n \le V_n \le U_n$.

Priority Types

A critical element of best-first search is the priority value for each frontier node, which determines which node will be expanded next. However, some complications arise in AND/OR search, and particularly for summation queries, which we discuss here.

In typical A* search, we select the frontier node with the lowest cost incurred so far, plus an optimistic heuristic estimate of the cost to go. In our notation and for maximization, this corresponds to selecting $s \in \mathit{OPEN}$ to maximize U_s; this choice will tighten the current global upper bound U, and we refer to it as the "upper" priority. However, for summation problems it may be more natural to tighten the current gap, $U - L$, which suggests selecting s to maximize $U_s - L_s$ (the "gap" priority); more precisely, in Appendix B we show:

Proposition 3.2. Given a current search tree $\mathcal{S}$, fully solving the summation problem below a frontier node s will tighten the bound difference $U - L$ by

$$g_s\,(u_s - v_s) \prod_{t \in branch(s)} u_t + g_s\,(v_s - l_s) \prod_{t \in branch(s)} l_t$$

which is upper bounded by $gap(s) = U_s - L_s$.

Thus, $gap(s)$ serves as an optimistic estimate of the effect of expanding s on the global bound difference. For OR search, this essentially reduces to the priority proposed in Henrion [1991].

However, in AND/OR search, tracking the highest priority node can be difficult. In particular,

Algorithm 3.1 AND/OR Best-first Search (AOBFS) for anytime summation bounds.

```
1: Initialize $\mathcal{S} \leftarrow \{\emptyset\}$ with the root $\emptyset$.
2: while termination conditions not met
3:     ExpandBest($\emptyset$, 1, 1)                          // find best frontier, from root
4: end while
5: function ExpandBest($n$, $U_n$, $L_n$)
6:     if  $n \notin OPEN$                                  // not frontier; recurse down:
7:         Find $c^+$ via (3.5) or (3.6)
8:         ExpandBest($c^+$, $U_{c^+}$, $L_{c^+}$)
9:     else                                                   // expand frontier node:
10:        Generate children of $n$; $u_c = h_c^+$, $l_c = h_c^-$.
11:        Mark any leaves (or $u_c = l_c$) as $SOLVED$.
12:    end if
13:    Update $u_n$, $l_n$ via (3.2)
14:    if all children $c \in ch(n)$ are $SOLVED$
15:        Remove $ch(n)$ from $\mathcal{S}$; add $n$ to $SOLVED$.
16:    end if
17:    Find $c^+$ and update $U_n^+$, $L_n^+$ by (3.5) or (3.6)-(3.7).
18: end function
```

both the upper and gap priorities are *non-static*: after expanding a node in $OPEN$, the priority of other nodes may change their values and relative orders. Consider two nodes which share an AND ancestor; this means that the nodes correspond to conditionally independent subproblems, and their contributions are multiplied in the recursive value definition (3.1). Since *both* branches must be solved, proving that one branch is unpromising tells us not to explore the other branch, and reduces its priority.

For the upper bound heuristic at a node n, it is easy to show that, if no descendant of n is updated, the relative order of n's children does not change. To see this, write $U_s = U_n(U_s/U_n)$ for any frontier node s descended from n, $s \geq n$. Then, the second term, U_s/U_n, will involve only quantities defined on descendants of n; thus, changes to ancestors of n can only affect U_n and hence cannot change the relative order of n's descendants. Then, by maintaining at each node n the relative value of the highest priority descendant,

$$U_n^+ = \max_{s \geq n, s \in OPEN} U_s / U_n$$

we can always identify the highest-priority child of n, and propagate U_c^+ upward, by computing

$$c^+ = \operatorname*{argmax}_{c \in ch(n)} U_c U_c^+ \qquad \text{and} \qquad U_n^+ = U_{c^+} U_{c^+}^+ / U_n. \tag{3.5}$$

Unfortunately, the gap heuristic is less well-behaved. At a node n, the descendant gap $U_n(U_s/U_n) - L_n(L_s/L_n)$ *can* change the relative order if n's ancestors are updated and U_n and L_n change values. For this reason, we elect to track the gap priority approximately, by computing the child with the highest score,

$$c^+ = \operatorname*{argmax}_{c \in ch(n)} \; U_c U_c^+ - L_c L_c^+, \tag{3.6}$$

and updating n's priority terms as

$$U_n^+ = U_{c^+} U_{c^+}^+ / U_n \qquad \text{and} \qquad L_n^+ = L_{c^+} L_{c^+}^+ / L_n. \tag{3.7}$$

This is not guaranteed to find the current highest gap priority node at each step, but will become more accurate as the search tree grows and the U_n, L_n become more accurate (and thus stable).

The overall algorithm is given in Algorithm 3.1 and named AOBFS. Each iteration recurses from the root to the best frontier node, scoring n's children and computing U_n, L_n. At the frontier, we expand a node, and recurse back up the tree, updating the bounds u_n, l_n and descendant priority quantities U_n^+, L_n^+. We can show:

Proposition 3.3. The time complexity of each node expansion and update in Algorithm 3.1

Algorithm 3.2 Memory-limited AOBFS for anytime summation bounds.

```
1: Initialize $\mathcal{S} \leftarrow \{\emptyset\}$ with the root $\emptyset$.
2: while termination conditions not met
3:     if memory OK:    $n \leftarrow$ExpandBest($\emptyset$, 1, 1)
4:     else   $n \leftarrow$RemoveWorst($\emptyset$, 1, 1)
5:     end if
6: end while
7: function RemoveWorst($n$, $U_n$, $L_n$)
8:     if $n$'s children all in $OPEN$                      // worst removable node
9:         Remove $ch(n)$ from $\mathcal{S}$; mark $n$ in $OPEN$
10:    else                                                 // or recurse toward worst
11:        Find worst non-$OPEN$ child $c^-$ via (3.8) or (3.9)
12:        RemoveWorst($c^-$, $U_{c^-}$, $L_{c^-}$)
13:    end if
14:    Update $c^-$ and $U_n^-$, $L_n^-$ via (3.8) or (3.9)–(3.10)
15: end function
16: function ExpandBest($n$, $U_n$, $L_n$)
17:    // As in Algorithm 3.1, except:
18:    Ensure $u_n$, $l_n$, $U_n^+$, $L_n^+$ updated monotonically
19:    Update $c^-$ and $U_n^-$, $L_n^-$ via (3.8) or (3.9)–(3.10)
20: end function
```

is bounded by $O(h(b+d))$ where h is the height of the pseudo tree, b is the max branching factor of the pseudo tree, and d is the max variable domain size.

3.1.3 Memory-limited AOBFS

As is typical for best-first search, memory usage can quickly become a major bottleneck. To continue improving in memory-limited settings, we could switch to some low-memory strategy such as depth-first search (DFS). However, this is often slow to tighten the bounds.

Instead, we apply a variant of SMA* [Russell, 1992], so that near the memory limit, we continue expanding nodes in a best-first way, but remove low-priority nodes from $\mathcal{S}$, in such a way that they will be re-generated once the high-priority subtrees are tightened or solved. We simply modify our updates in two ways: (1) at each node n, we also track the lowest-priority *removable* descendant of n; and (2) we force u_n, l_n, and the node priority quantities U_n^+, L_n^+

to be updated monotonically, to avoid worsening the bounds or overestimating priority when subtrees are removed and later re-generated. The resulting memory-limited AOBFS is shown in Algorithm 3.2.

For convenience, we define a node as *removable* if its children are all in *OPEN*, and "remove" it by deleting its children and re-adding it to *OPEN*; this notion simplifies tracking and re-expanding removed nodes. We keep track of the smallest priority value of any removable node below n; for the upper heuristic,

$$c^- = \underset{c \in rm(n)}{\operatorname{argmin}} U_c U_c^- \qquad \text{and} \qquad U_n^- = U_{c^-} U_{c^-}^- / U_n \tag{3.8}$$

with $rm(n) = ch(n) \backslash OPEN$, i.e., the children of n not in *OPEN*, and $U_n^- = U_n^+$ at removable nodes. For the gap priority, we again track only approximately, using

$$c^- = \underset{c \in rm(n)}{\operatorname{argmin}} \ U_c U_c^- - L_c L_c^- \tag{3.9}$$

and

$$\begin{aligned} U_n^- &= U_{c^-} U_{c^-}^- / U_n, \\ L_n^- &= L_{c^-} L_{c^-}^- / L_n \end{aligned} \tag{3.10}$$

Then, to remove a node, we search downward along the worst children c^-, and remove n when its children are all in *OPEN*.

■ 3.1.4 Experimental Settings

To evaluate the effectiveness of memory-limited AOBFS (Algorithm 3.2), we compare it to a number of existing methods on four benchmarks and under three different memory budgets.

Benchmarks

We evaluate our method's performance on several benchmark instance sets: CPD, a set of computational protein design problems from Viricel et al. [2016]; PIC'11, a benchmark subset of 23 instances selected by Viricel et al. [2016] from the 2012 UAI competition; BN, a set of Bayesian networks from the 2006 competition[1]; and Protein, made from the "small" protein side-chains of Yanover and Weiss [2002]. For CPD, BN, and Protein, we evaluate on randomly selected subsets of size 100, 50, and 50, respectively. Table 3.1 gives statistics on the composition of these benchmarks. Note that CPD and PIC'11 are used in order to facilitate comparison with the baseline Z_ε^*. BN and Protein are chosen to reflect diverse problem characteristics; as we can observe from Table 3.1, BN instances have large number of variables and functions on average, while Protein instances typically have large domain sizes.

Methods

Our memory-limited AOBFS (Algorithm 3.2) is tested in two variants: using the "gap" priority (denoted **A-G**), and using the "upper" priority (**A-U**). To measure whether errors in tracking and finding the highest gap priority affect quality, we also compare an OR search variant, **O-G**; this can be done by simply selecting a chain-structured pseudo tree, and ensures that the highest gap priority node is found at each step, at the cost of not exploiting the AND/OR problem decomposition. For our experiments, we use WMB heuristics whose memory use is roughly controlled by the *ibound*. For a given memory budget, we first compute the largest *ibound* that fits the memory budget, and then use the remaining memory for search. A more carefully chosen allocation strategy might be able to yield better performance; however, we leave this for future work.

$\boldsymbol{Z_\varepsilon^*}$ [Viricel et al., 2016] is a recent algorithm that provides lower and upper bounds on the

[1]`http://melodi.ee.washington.edu/~bilmes/uai06InferenceEvaluation/`

partition function. For a given ε, it returns bounds on $\log Z$ whose gap is at most $\log(1+\varepsilon)$. We adopted the parameter setup suggested by the authors: upper bound set to Ub_1, enforcing VAC at the root, and using EDAC during the rest of the search.

VEC [Dechter, 1999, 2013] (Variable Elimination with Conditioning, also known as cutset conditioning) is a classical method for trading off memory with time when the induced width of the problem is too high. The algorithm determines the highest induced width for which it can run the variable elimination algorithm, denoted w, and then searches over a w-cutset set of nodes, applying variable elimination to each assignment of the cutset.

MMAP-DFS [Marinescu et al., 2014] (abbreviated **M-D**) is a state-of-the-art method for marginal MAP using AND/OR search, which solves the internal summation problem exactly using depth-first search aided by weighted mini-bucket heuristics. We use it to compute the partition function by treating all variables as summation variables. Since it also uses weighted mini-bucket, we use the same *ibound* selected for the heuristics in our algorithm.

Viricel et al. [2016] also compared several other solvers, minic2d [Oztok and Darwiche, 2015], ace [Chavira and Darwiche, 2008], and cachet [Sang et al., 2005], but found that even with significant memory (60GB), these solvers did not complete a significant fraction of the CPD problems: minic2d solved about 13%, ace solved about 5%, and cachet solved 7% of the instances. For this reason, we elected not to include them in our evaluation.

Implementations of all methods are in C/C++ by the original authors. We allowed each solver a maximum time of 1 hour per instance, and tested under three different memory budgets: 1GB, 4GB and 16GB.

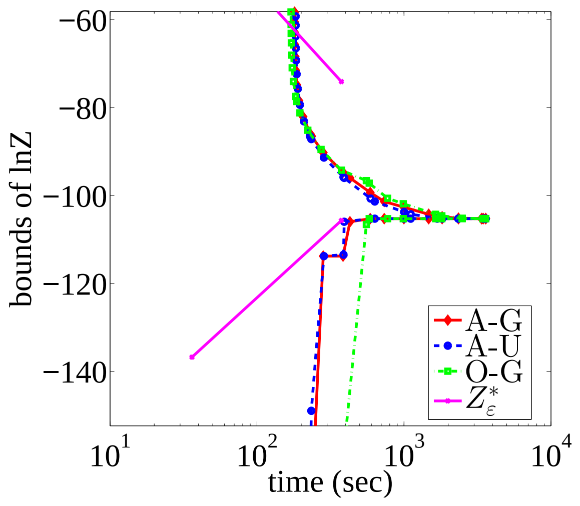
Table 3.1: Statistics of the four evaluated benchmark sets.

	PIC'11	Protein	BN	CPD
# instances	23	50	50	100
avg. # variable	104.04	99.96	838.60	15.68
avg. # factor	409.09	355.84	838.60	135.32
avg. max domain size	2.91	77.94	12.44	32.54
avg. # evidence	0	0	96.04	0
avg. induced width	16.35	11.24	32.78	14.68
avg. pseudotree depth	26.09	27.66	112.46	12.27

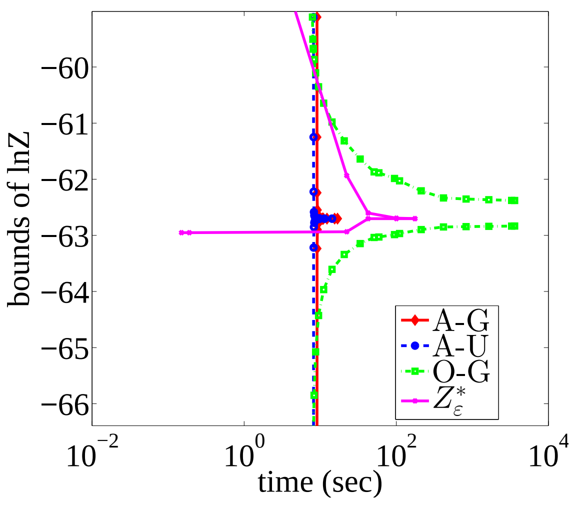
3.1.5 Empirical Results

The main goal of our algorithm is to provide continuously improving, anytime bounds. While Z^*_ε is able to reduce its quality requirement (by increasing ε), it is not anytime in this sense. To simulate anytime-like behavior, we vary ε across the range $\log(\varepsilon + 1) \in [10^{-3}, 10^2]$ to provide a sequence of quality/time pairs. Figure 3.1 shows the anytime behavior of our algorithm along with these Z^*_ε runs on one instance per benchmark. Generally speaking, our best-first search process tightens the bound much more quickly and smoothly than Z^*_ε. For example, in Figure 3.1(a), only very coarse bounds are produced within the time limit; in (b), the Z^*_ε bounds are loose until our proposed algorithm has essentially solved the problem (with heuristic construction comprising most of the total time); in (c) Z^*_ε did not produce any useful bound within the time limit.

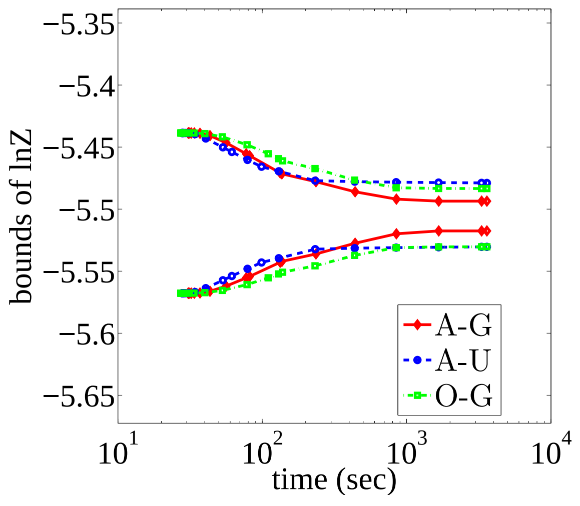
Within our algorithm's variants, we see that both AND/OR variants perform similarly, with the OR variant similar or slower (e.g., Figure 3.1(b)). We find that our gap-based priority can be slower initially, but usually improves to do better than upper priority after enough time; for example, this behavior is visible in Figure 3.1(c).



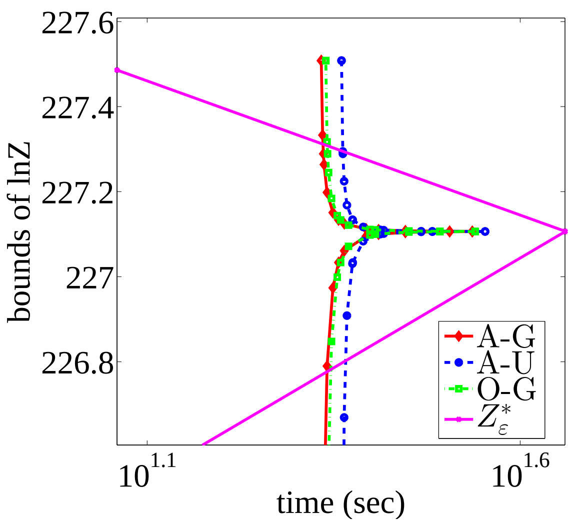
(a) PIC'11/queen5_5_4



(b) Protein/1g6x



(c) BN/BN_8

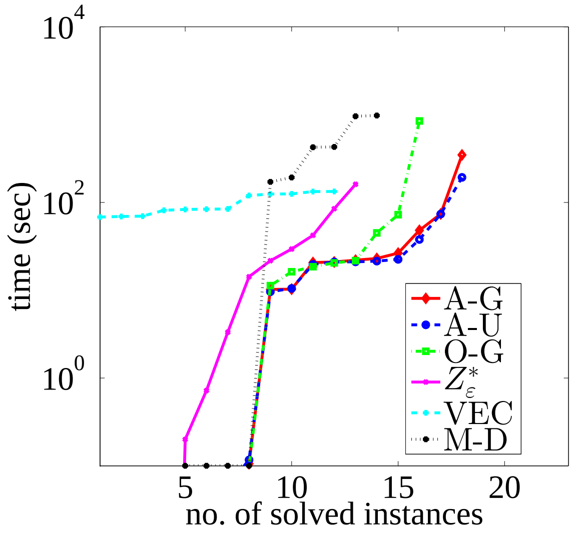


(d) CPD/237_SER

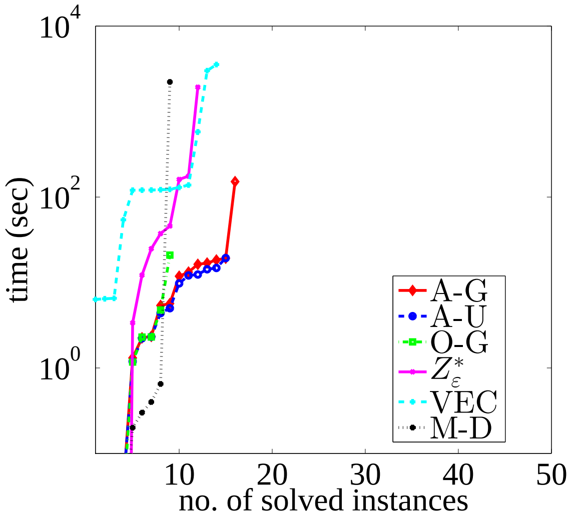
Figure 3.1: Anytime behavior of AOBFS on one instance per benchmark. For **A-G**, **A-U**, and **O-G**, anytime bounds on $\log Z$ are truncated at $\log U - \log L = 10^{-3}$. Curves for Z^*_ε may be missing since it may not be able to produce bounds or bounds are not in a reasonable scope for $\log(\varepsilon+1) \in [10^{-3}, 10^2]$.

Solving to fixed tolerance

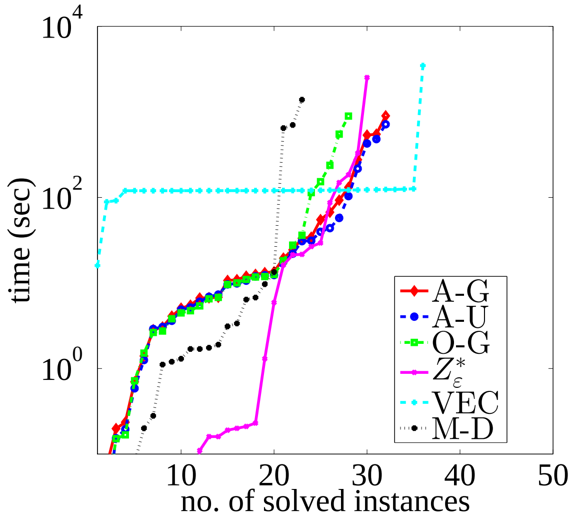
We evaluate the number of instances in each benchmark that can be solved to tolerance (so that the gap between the upper and lower bound is less than ϵ) within the memory and time constraints. Table 3.2 shows the performance of each algorithm at two thresholds, "loose" ($\epsilon = 1.0$) and "tight" ($\epsilon = 10^{-3}$). Exact methods (VEC, M-D) are included with the tight



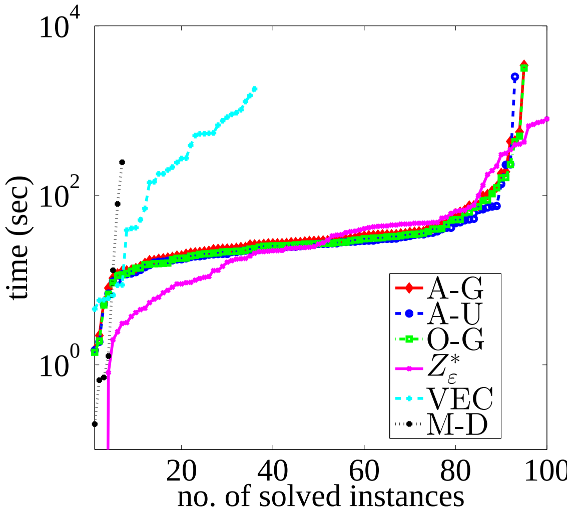
(a) PIC'11



(b) Protein



(c) BN

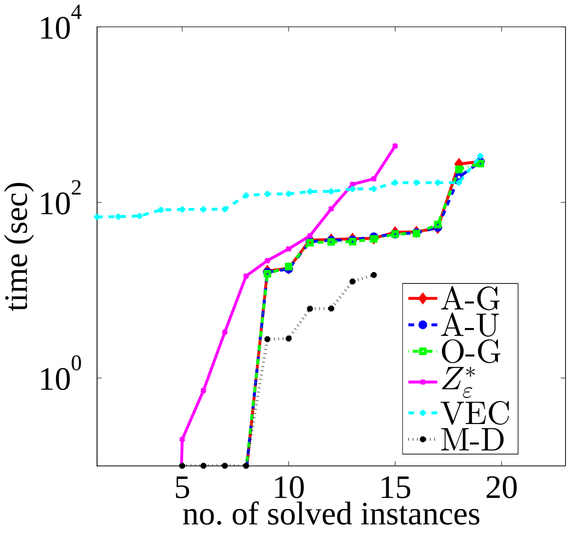


(d) CPD

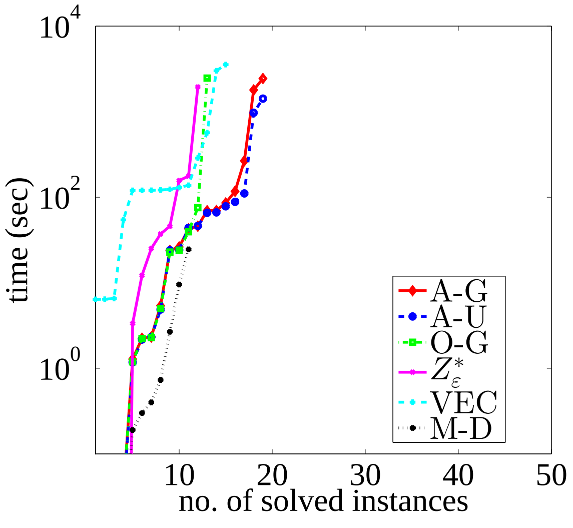
Figure 3.2: Time required to solve the n-th fastest instance in the benchmark (to tolerance 10^{-3}), with memory 1GB. Maximum time budget is 1 hour. The total benchmark sizes (from left to right) are 23, 50, 50 and 100 instances, respectively. Here, lower curves are better, indicating less time required to solve those problems.

tolerance interval.

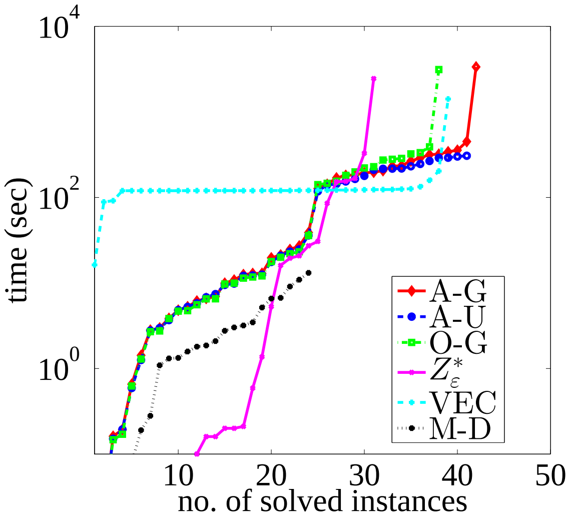
From Table 3.2, we can see that **A-G** solves the most instances to tight tolerance in 9 of 12 settings; **A-U** is only slightly worse, and also improves over the previous approaches. (Again, we see that **O-G** solves fewer instances, emphasizing the usefulness of exploiting conditional



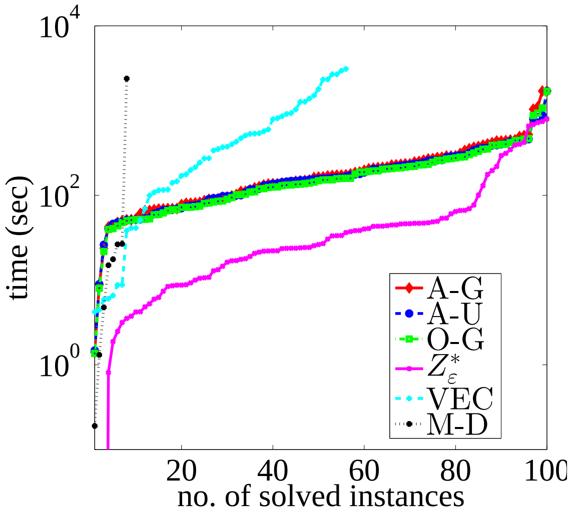
(a) PIC'11



(b) Protein



(c) BN



(d) CPD

Figure 3.3: Time required to solve the n-th fastest instance in the benchmark (to tolerance 10^{-3}), with memory 16GB. Maximum time budget is 1 hour. The total benchmark sizes (from left to right) are 23, 50, 50 and 100 instances, respectively. Again, lower curves are better, indicating faster solutions.

independence in the AND/OR space.) An exception is on CPD, which Z_ε^* performs slightly better at lower memory bounds. When looser bounds are acceptable, Z_ε^* solves only one additional problem at 16GB, and two additional problems at lower memory; it has difficulty taking advantage of anytime tradeoffs. In contrast, the proposed search algorithms solve significantly more problems to loose tolerance.

Table 3.2: Number of instances solved to tolerance interval for each benchmark & memory setting. Each entry contains the number of solved instances in 1 hour with memory budget 1GB, 4GB, and 16GB from left to right. The highest (most solved) for each setting is bolded. Exact methods (VEC, M-D) are compared to the "tight" tolerance 10^{-3}.

"Loose": $\log U - \log L < 1.0$

	PIC'11	Protein	BN	CPD
#inst.	23	50	50	100
Memory: 1GB/4GB/16GB				
A-G	**20/20/21**	**23/29/30**	**39/42/43**	**100/100/100**
A-U	**20/20/21**	**23/29/30**	**39/42/43**	**100/100/100**
O-G	19/**20/21**	12/13/16	35/37/40	**100/100/100**
Z^*_ε	14/14/15	13/13/13	30/31/31	**100/100/100**

"Tight": $\log U - \log L < 10^{-3}$

	PIC'11	Protein	BN	CPD
#inst.	23	50	50	100
Memory: 1GB/4GB/16GB				
A-G	**18/18/19**	**16/17/19**	32/**40/42**	95/98/**100**
A-U	**18/18/19**	15/**17/19**	32/**40**/41	93/95/**100**
O-G	16/**18/19**	9/12/13	28/36/38	95/98/**100**
Z^*_ε	13/13/15	12/12/12	30/31/31	**100/100/100**
VEC	12/14/**19**	14/15/15	**36**/38/39	36/52/56
M-D	14/14/14	9/9/11	23/23/24	7/7/8

Figure 3.2 and Figure 3.3 show the time required to solve (to tolerance 10^{-3}) the instances in each benchmark with memory 1GB and 16GB respectively. Again, we find that Z^*_ε is usually faster on CPD problems, but slower than our proposed method on PIC'11 and Protein, and faster on only the easiest of the BN instances. Among exact methods, **M-D** is often faster on simple models, but then fails to solve harder models, while **VEC** is usually both slower and solves fewer models than our best-first search. We also observe that more memory often leads to better performance for all methods. Our algorithm and **VEC** generally improve more compared to other baselines with more memory.

Search can sometimes effectively solve instances with very high width, if there are relatively few high-weight configurations of the model. For example, in the Protein instances, **A-G** solves instance “1who” with an induced width 10 in only 12 seconds and 1GB memory, while the corresponding junction tree requires about 150GB memory; even more extreme, instance “2fcr” with an induced width 17 is solved in 21 minutes and 16GB memory, while junction tree would require approximately 3.5PB.

A-G versus A-U

As we observed from Figure 3.1, **A-U** may outperform **A-G** early on in search, but **A-G** usually catches up. One possible explanation is that, early on, the lower bound heuristics from weighted mini-bucket are often significantly worse than the corresponding upper bounds, and thus the gap, $U - L$, is very close to U, and the priorities are similar. However, our approximate procedure may not identify the top gap-priority node, and moreover, **A-U** priority requires slightly less computation than that of **A-G**, making it more efficient during early search. However, as search proceeds and the heuristics become more accurate, focusing on nodes that have high gap, rather than merely high value, pays off; for example, **A-G** solves slightly more instances than **A-U** to tolerance in Table 3.2.

3.1.6 Summary of AOBFS

In this section, we develop an anytime anyspace search algorithm for bounding the partition function. It is a priority-driven best-first search scheme on the AND/OR search tree based on precompiled heuristics from state-of-the-art variational bounds, and is able to improve on these bounds in an anytime fashion within limited memory resources. The AND/OR search tree enables exploiting conditional independence during search, while the heuristics and priority guide it toward critical subtrees of the search space. In experimental comparisons, our best-first algorithm outperforms existing, state-of-the-art baseline methods on multiple standard benchmarks and memory limits.

3.2 Anytime Anyspace Best-first Search for MMAP

We next extend our anytime anyspace AND/OR best-first search algorithm to provide improving deterministic upper bounds for marginal MAP (MMAP). Our algorithm unifies max and sum inference within a specifically designed priority system that aims to reduce upper bound of the optimal solution(s) as quickly as possible, which generalizes AOBFS for the pure summation task. Our approach avoids some unnecessary exact evaluation of conditional summation problems, yielding significant computational benefits in many cases, as demonstrated by empirical results on three challenging benchmarks compared to existing state-of-the-art baselines on anytime upper bounds.

3.2.1 Introduction

Some early works [Park and Darwiche, 2003, Yuan and Hansen, 2009] solve MMAP exactly based on depth-first branch and bound. Marinescu et al. [2014] outperforms the predecessors by using AND/OR search spaces and high-quality variational heuristics. Later, a best-first search variant of Marinescu et al. [2014] was proposed and shown empirically superior to the depth-first one given enough memory [Marinescu et al., 2015]. Because best-first search is memory-intensive, a recursive best-first search algorithm [Marinescu et al., 2015] was proposed to that operate within a limited memory budget.

Because of the inherent difficulty of MMAP, searching for exact solutions is generally unpromising. Recent works on MMAP often focus on approximate schemes. Approximate elimination methods (see Section 2.2) and closely related variational bounds [Liu and Ihler, 2013] provide deterministic guarantees. However, as noted before in the case of partition function, these bounds are not anytime; their quality often depends on the memory available, and does not improve without additional memory. The decomposition bounds of Ping et al. [2015] operate similarly; while the bounds improve over time, they cannot be guaranteed to find optimal solutions unless given enough memory. Some Monte Carlo methods such as one based on random hashing [Xue et al., 2016] provide probabilistic bounds only, while some may not even have this property, e.g., those based on Markov chain Monte Carlo [Doucet et al., 2002, Yuan et al., 2004]. Another algorithm that provides anytime deterministic bounds for MMAP is based on factor set elimination [Mauá and de Campos, 2012], but the factor sets it maintains tend to grow very large and thus limit its practical use to problems with relatively small induced widths [Marinescu et al., 2017].

Recent search-based algorithms improve upon their exact predecessors [Otten and Dechter, 2012, Marinescu et al., 2014, 2015] by introducing weighted heuristics in best-first search [Lee et al., 2016b] or by combining best-first search with depth-first search [Marinescu et al., 2017] to improve their anytime performance, and these comprise the current state of the art. However, these algorithms [Marinescu et al., 2014, 2015, Lee et al., 2016b, Marinescu et al., 2017] treat the max and sum inference separately, and require solving exactly a number of conditional summation problems, which is generally intractable. Although this strategy works reasonably at anytime improvement of lower bounds, it is often slow to improve the associated upper bound. Moreover, it is not always necessary to fully solve an internal summation problem to prove a configuration's sub-optimality. In contrast, we focus our solver on quickly reducing its MMAP upper bound via best-first search, with computation of the internal summation problems integrated in a best-first way as well. When the search encounters an internal summation problem, instead of fully solving it immediately we gradually instantiate its summation variables, which acts to reduce the upper bound of the current candidate MAP (sub-) configuration. This "pure" best-first behavior can potentially solve fewer summation problems to certify the optimal MAP configuration.

■ 3.2.2 AND/OR Search Spaces for MMAP

To present an AND/OR search algorithm for MMAP, we first have to adapt and extend some notions introduced in Section 2.3.3 such as "pseudo tree" (see Definition 2.1) and "partial solution tree" (see Definition 2.3) to this particular task.

If a tree node of a pseudo tree corresponds to a MAX variable in the associated graphical model of the pseudo tree, we call it a MAX node, otherwise we call it a SUM node. A pseudo tree is called *valid* for an MMAP task if there is *no* MAX variable that is descendant of some SUM variable. Thus, all MAX variables of a valid pseudo tree form a subtree (assuming a dummy MAX root) that contains the root. We assume valid pseudo trees in the sequel.

Example 3.1. Figure 3.4(a) shows the primal graph of a pairwise model, with A,B,C being MAX variables, and the rest SUM. Figure 3.4(b) shows one valid pseudo tree of the model.

If an OR node is associated with some MAX variable, it is called an OR-MAX node. Notions of OR-SUM, AND-MAX, AND-SUM nodes are defined analogously. We introduce a concept called *partial MAP solution tree* below.

Definition 3.1 (partial MAP solution tree). A partial MAP solution tree T of an AND/OR search tree $\mathcal{T}$ is a partial solution tree (see Definition 2.3) that satisfies the following condition: If an OR-SUM node is in T, this node must have OR-MAX sibling(s); T contains all its siblings and meanwhile T does not contain any of its children.

The associated configuration $x_{T^{\mathrm{M}}}$ of a partial MAP solution tree T^{M} is a partial assignment of X_{M} (all the MAX variables, see Section 2.1.1). If $x_{T^{\mathrm{M}}}$ is a complete assignment of X_{M}, we call it a full MAP solution tree. We let $\mathbb{T}$ denote the set of all full solution trees (see Definition 2.2). and $\mathbb{T}^{\mathrm{M}}$ denote the set of all full MAP solution trees.

Example 3.2. A full solution tree is marked red in the AND/OR search tree shown in Figure 3.4(c). According to the definitions, a full solution tree (see Definition 2.2) corresponds

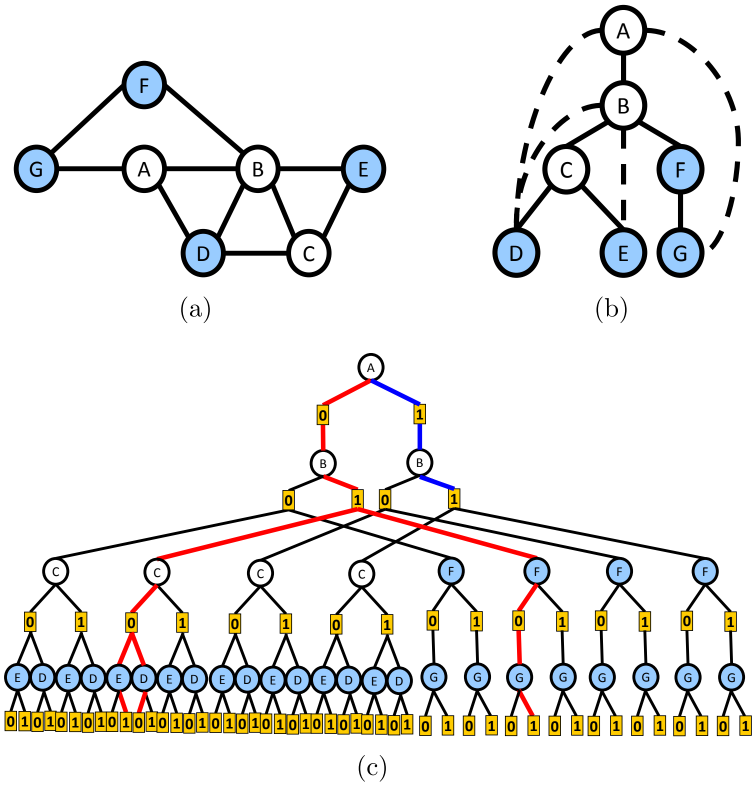


Figure 3.4: (a) A primal graph of a graphical model over 7 variables (A, B, C are MAX variables and D, E, F, G are SUM variables). (b) A valid pseudo tree for the primal graph. (c) AND/OR search tree guided by the valid pseudo tree. One full solution tree is marked red. One partial solution tree that is also a partial MAP solution tree is marked blue.

to a complete assignment of all variables. A partial solution tree (see Definition 2.3) that happens to be a partial MAP solution tree is marked blue in Figure 3.4(c).

We extend the definitions (2.15)–(2.16) to the mixed MAX/SUM of the MMAP task. First, if n is an AND node that corresponds to a leaf node of the pseudo tree, we define $v_n = 1$ as usual; and for the rest, we have

$$\begin{aligned} &\text{AND node } n\text{: } v_n = \prod_{s \in ch(n)} v_s \\ &\text{OR node } n\text{:} \quad v_n = \begin{cases} \max\limits_{s \in ch(n)} w_s v_s, & \text{if MAX node } n \\ \sum\limits_{s \in ch(n)} w_s v_s, & \text{if SUM node } n \end{cases} \end{aligned} \tag{3.11}$$

Thus, the value $v_\emptyset$ at the root is the optimum of the MMAP task, i.e., $v_\emptyset = \max_{x_\mathrm{M}} \sum_{x_\mathrm{S}} f(x)$.

For each node n, we lightly abuse notation to define V_n as the MMAP value of the model conditioned on $x_{\leq n}$, expressed explicitly as:

$$V_n = g_n \, v_n \prod_{s \in branch(n)} v_s. \tag{3.12}$$

Note that the above has the same form as that of (3.3) for the summation task. In general, for any partial solution tree T, let V_T denote the optimum of the MMAP task for the model conditioned on x_T, we have

$$V_T = g_T \prod_{n \in leaf(T)} v_n \tag{3.13}$$

where $leaf(T)$ is the set of all leaf nodes of T, and g_T is defined in (2.13). Let T_n^M be the optimal full MAP solution that contains or extends to n, i.e., $T_n^\mathrm{M} = \mathrm{argmax}_{\{T^\mathrm{M} \in \mathbb{T}^\mathrm{M} | T^\mathrm{M} \cup \{n\} \subset T, T \in \mathbb{T}\}} V_{T^\mathrm{M}}$. The following proposition reveals the fact that V_n is the total sum of conditioned task values over those full solution trees that contain both T_n^M and n:

Proposition 3.4.

$$V_n \quad = \sum_{\{T \in \mathbb{T} | T_n^\mathrm{M} \cup \{n\} \subset T\}} V_T \tag{3.14}$$

These quantities and their relations will help us develop the priority function defined for our algorithm in the sequel.

■ 3.2.3 Unified Best-first Search (UBFS)

In this section, we will present our main algorithm by first introducing the double-priority system which leads to a simplified version that is A*-like, and then generalizing it to an SMA*-like version to operate with limited memory. If the MMAP task corresponds to a pure summation, our algorithm will reduce to AOBFS using the "upper" priority.

Beginning with only the root $\emptyset$, we expand the search tree respecting the pseudo tree structure. As a best-first scheme, we assign a priority to each frontier node on the search tree, then expand the top priority frontier node in each iteration. More precisely, we maintain an explicit AND/OR search tree of visited nodes, denoted $\mathcal{S}$, whose frontier nodes are denoted *OPEN*. Without loss of generality, we assume *OPEN* only contains AND nodes. A frontier AND node of $\mathcal{S}$ is solved if it corresponds to a leaf node in the pseudo tree. An internal node of $\mathcal{S}$ is solved if all its children are solved. For an internal OR-MAX node, it suffices to conclude that it is solved if its "best" child $\text{argmax}_{s \in ch(n)} w_s v_s$ is solved.

For each node n in the AND/OR search tree, we make u_n an upper bound of v_n, initialized via pre-compiled heuristic h_n^+ s.t. $v_n \leq h_n^+$, and subsequently tightened during search. Given the currently expanded tree $\mathcal{S}$, we update u_n using information propagated from the frontier, analogous to (3.11):

$$\begin{aligned} &\text{AND node } n: \quad u_n = \prod_{s \in ch(n)} u_s \\ &\text{OR node } n: \quad u_n = \begin{cases} \max\limits_{s \in ch(n)} w_s u_s, & \text{if MAX node } n \\ \sum\limits_{s \in ch(n)} w_s u_s, & \text{if SUM node } n \end{cases} \end{aligned} \tag{3.15}$$

These values depend implicitly on the search tree $\mathcal{S}$. Thus, the dynamically updated bounds $U = u_\emptyset$ at the root serve as an anytime bound on the optimum of the MMAP task.

Priority

Our goal is to drive down the global upper bound U as quickly as possible. Intuitively, this can be achieved by expanding the frontier node that affects U most at each iteration. Following this intuition, we will first show how to make connection between a frontier node n

and the global upper bound U, which is bridged by the current "best" partial MAP solution tree in $\mathcal{S}$ that contains or extends to n; we will then establish a double-priority system that marks the most "influential" frontier node as the top priority one.

Analogously to V_n and V_T (see (3.12), (3.13)), we define U_n for any node $n \in \mathcal{S}$ and U_T for any partial solution tree $T \subset \mathcal{S}$:

$$U_n = g_n \, u_n \prod_{s \in branch(n)} u_s, \qquad\qquad U_T = g_T \prod_{n \in leaf(T)} u_n \tag{3.16}$$

These two quantities are upper bounds of V_n and V_T respectively. Note that U_n and U_T depend on the current search tree $\mathcal{S}$ while V_n and V_T do not. The relation between U_n and U_T is also analogous to that of V_n and V_T stated in (3.14):

$$U_n = \sum_{\{T \in \mathbb{T}_\mathcal{S} \mid T^{\mathrm{M}}_\mathcal{S}(n) \cup \{n\} \subset T\}} U_T \tag{3.17}$$

where $\mathbb{T}_\mathcal{S} = \{T \cap \mathcal{S} \mid T \in \mathbb{T}\}$ is the set of the partial solution trees formed by projections of all full solution trees on $\mathcal{S}$, $\mathbb{T}^{\mathrm{M}}_\mathcal{S} = \{T^{\mathrm{M}} \cap \mathcal{S} \mid T^{\mathrm{M}} \in \mathbb{T}^{\mathrm{M}}\}$ is the set of partial MAP solution trees formed by projections of all full MAP solution trees on $\mathcal{S}$, and $T^{\mathrm{M}}_\mathcal{S}(n) = \mathrm{argmax}_{\{T^{\mathrm{M}} \in \mathbb{T}^{\mathrm{M}}_\mathcal{S} \mid T^{\mathrm{M}} \cup \{n\} \subset T, T \in \mathbb{T}_\mathcal{S}\}} U_{T^{\mathrm{M}}}$ is the partial MAP solution tree in $\mathbb{T}^{\mathrm{M}}_\mathcal{S}$ with the highest upper bound among those in $\mathbb{T}^{\mathrm{M}}_\mathcal{S}$ that contain or extend to n.

In lieu of (3.17), we can derive the following proposition that implies U_n quantifies the contribution of node n to the upper bound of $T^{\mathrm{M}}_\mathcal{S}(n)$:

Proposition 3.5.

$$U_{T^{\mathrm{M}}_\mathcal{S}(n)} = U_n + \sum_{\{T \in \mathbb{T}_\mathcal{S} \mid T^{\mathrm{M}}_\mathcal{S}(n) \subset T, n \notin T\}} U_T \tag{3.18}$$

Moreover, it is intuitively clear that the global upper bound U is determined by the current most "promising" partial MAP solution tree, $T^{\mathrm{M}}_{\mathcal{S}} = \mathrm{argmax}_{T^{\mathrm{M}} \in \mathbb{T}^{\mathrm{M}}_{\mathcal{S}}} U_{T^{\mathrm{M}}}$:

Proposition 3.6.

$$U = \max_{T^{\mathrm{M}} \in \mathbb{T}^{\mathrm{M}}_{\mathcal{S}}} U_{T^{\mathrm{M}}} \tag{3.19}$$

From Proposition 3.5 and 3.6, we see that among all the frontier nodes, only those contained in or reachable by $T^{\mathrm{M}}_{\mathcal{S}}$ eventually contribute to U, which is very different from the pure summation case where all the frontier nodes contribute to the global bound. This group of frontier nodes, denoted $\mathit{OPEN}(T^{\mathrm{M}}_{\mathcal{S}})$, can be expressed explicitly as:

$$\mathit{OPEN}(T^{\mathrm{M}}_{\mathcal{S}}) = \{n \in \mathit{OPEN} \mid T^{\mathrm{M}}_{\mathcal{S}} \cup \{n\} \subset T, T \in \mathbb{T}_{\mathcal{S}}\} \tag{3.20}$$

It is obvious that $T^{\mathrm{M}}_{\mathcal{S}} = T^{\mathrm{M}}_{\mathcal{S}}(n)$ for any $n \in \mathit{OPEN}(T^{\mathrm{M}}_{\mathcal{S}})$. We thus prefer to expand $n^\star = \mathrm{argmax}_{n \in \mathit{OPEN}(T^{\mathrm{M}}_{\mathcal{S}})} U_n$, the node that contributes most to U, because such expansion is likely to decrease U most.

Double-priority system. The above discussion also suggests a double-priority system that helps to identify $n^\star$ in each iteration. For any $n \in \mathit{OPEN}$, a primary priority defined by the upper bound $U_{T^{\mathrm{M}}_{\mathcal{S}}(n)}$, indicates the potential of $T^{\mathrm{M}}_{\mathcal{S}}(n)$ to be the tree that that determines the overall upper bound U; a secondary priority given by U_n quantifies the contribution of n to $U_{T^{\mathrm{M}}_{\mathcal{S}}(n)}$. Note that these two priorities are equal for any MAX node.

However, tracking the highest priority node can be difficult in AND/OR search. In particular, both these primary and secondary priorities are *non-static*: after expanding a node in OPEN, the priority of other nodes in OPEN may change their values and relative orders. A similar effect occurred in the pure summation case. The double-priority system does preserve some local order invariance:

Proposition 3.7. For any internal node $n \in \mathcal{S}$, expansion of any node in $\mathit{OPEN} \setminus \mathit{OPEN}(n)$

will not change the relative order of nodes in $OPEN(n)$ for either the primary or secondary priority.

The above local order-preserving property allows us to design an implicit priority queue for nodes in $OPEN$. To be more specific, first, for any $s \in OPEN$, we define $U^\star_s = u_s$; then, for each internal node n, we maintain several quantities during search:

$$c^\star = \begin{cases} \operatorname*{argmax}_{c \in ch(n)} U^\star_c / u_c, & \text{if AND node } n \\ \operatorname*{argmax}_{c \in ch(n)} (w_c u_c, w_c U^\star_c), & \text{if OR-MAX node } n \\ \operatorname*{argmax}_{c \in ch(n)} w_c U^\star_c, & \text{if OR-SUM node } n \end{cases} \tag{3.21}$$

$$U^\star_n = \begin{cases} U^\star_{c^\star} u_n / u_{c^\star}, & \text{if AND node } n \\ w_{c^\star} U^\star_{c^\star}, & \text{if OR node } n \end{cases} \tag{3.22}$$

In the above, "argmax" over pairs "$(,)$" means that we compute the argmax over the first component and use the second component to break ties; this reflects the primary and secondary priority structure. It is still possible for two frontier nodes to have the same priority: if those two nodes have the same node type, e.g., both are MAX nodes, we break ties randomly for simplicity; if they have different node types, we favor the MAX node over the SUM node because we prefer to reach a full MAP solution tree as early as possible.

The overall algorithm is presented in Algorithm 3.3. Note that the current best partial MAP configuration $x_{T^{\mathrm{M}}_S}$ serves as an anytime approximation, and can be extended into a full MAP configuration in some greedy way if required.

Proposition 3.8. In each iteration, Algorithm 3.3 finds a top-priority frontier node to expand.

See Appendix B for proof.

Algorithm 3.3 Unified Best-first Search (UBFS) for anytime upper bound of MMAP

```
1: Initialize $\mathcal{S} \leftarrow \{\emptyset\}$ with the root $\emptyset$.
2: while termination conditions not met
3:     ExpandBest($\emptyset$)                          // find best frontier, from root
4: end while
5: function ExpandBest($n$)
6:     if  $n \notin OPEN$                               // not frontier; recurse down:
7:         ExpandBest($c^\star$)
8:     else                                              // expand frontier node:
9:         Generate children of $n$; $U_c^\star = u_c = h_c^+$.
10:        Mark any leaves as SOLVED.
11:    end if
12:    Update $u_n$ via (3.15).
13:    Find $c^\star$ and update $U_n^\star$ via (3.21)-(3.22).
14:    if all children $c \in ch(n)$ are SOLVED
15:      or $n$ is an OR-MAX node and $c^\star$ is SOLVED
16:        Remove $ch(n)$ from $\mathcal{S}$; add $n$ to SOLVED.
17:    end if
18: end function
```

Proposition 3.9. The time complexity of each node expansion and update in Algorithm 3.3 is bounded by $O(h(b+d))$ where h is the pseudo tree height, b is the max branching factor of the pseudo tree, and d is the max variable domain size.

3.2.4 Memory-limited UBFS

As memory usage can quickly become a major bottleneck for a best-first search algorithm, we apply a variant of SMA* similar to that in Algorithm 3.2, so that near the memory limit, we continue expanding nodes in a best-first way, but remove low-priority nodes from $\mathcal{S}$, in such a way that they will be re-generated once the high-priority subtrees are tightened or solved. We simply modify our updates in two ways: (1) at each node n, we also track the lowest-priority *removable* descendant of n; and (2) we force $(u_n, U_n^\star)$ to be updated monotonically, to avoid worsening the bounds or overestimating the priority when subtrees are removed and later re-generated. The resulting memory-limited best-first algorithm is shown in Algorithm 3.2.

Algorithm 3.4 Memory-limited UBFS for anytime upper bound of MMAP

```
1: Initialize $\mathcal{S} \leftarrow \{\emptyset\}$ with the root $\emptyset$.
2: while termination conditions not met
3:     if memory OK:    $n \leftarrow$ExpandBest($\emptyset$)
4:     else   $n \leftarrow$RemoveWorst($\emptyset$)
5:     end if
6: end while
7: function RemoveWorst($n$)
8:     if $ch(n) \subset OPEN$                                  // worst removable node
9:         Remove $ch(n)$ from $\mathcal{S}$; mark $n$ in $OPEN$.
10:    else                                                     // or recurse toward worst
11:        RemoveWorst($c^-$)
12:    end if
13:    Update $c^-$, $u_n^-$ and $U_n^-$ via (3.23)-(3.25).
14: end function
15: function ExpandBest($n$)
16:    // As in Algorithm 3.3, except:
17:    Ensure $(u_n, U_n^\star)$ updated monotonically
18:    Update $c^-$, $u_n^-$ and $U_n^-$ via (3.23)-(3.25).
19: end function
```

For convenience, we define a node as removable if its children are all in $OPEN$, and "remove" it by deleting its children and re-adding it to $OPEN$; this simplifies tracking and re-expanding removed nodes. To do so, we introduce two quantities u_n^- and U_n^- that are defined for internal nodes of $\mathcal{S}$. For each n that is removable, we set $U_n^- = U_n^\star$, $u_n^- = u_n$ if n is a MAX node and $u_n^- = U_n^\star$ if n is a SUM node. The bottom-up recursion is as follows:

$$c^- = \begin{cases} \operatorname*{argmin}_{c \in rm(n)} (u_c^-/u_c, U_c^-/u_c), & \text{if AND node } n \\ \operatorname*{argmin}_{c \in rm(n)} (w_c u_c^-, w_c U_c^-), & \text{if OR node } n \end{cases} \tag{3.23}$$

$$u_n^- = \begin{cases} u_n, & \text{if AND-MAX } n \text{ \& OR-SUM } c^- \\ u_{c^-}^- u_n / u_{c^-}, & \text{if AND-MAX } n \text{ \& OR-MAX } c^- \\ w_{c^-} u_{c^-}^-, & \text{if OR node } n \end{cases} \tag{3.24}$$

$$U_n^- = \begin{cases} U_{c^-}^- u_n / u_{c^-}, & \text{if AND node } n \\ w_{c^-} U_{c^-}^-, & \text{if OR node } n \end{cases} \tag{3.25}$$

where $rm(n) = ch(n) \setminus OPEN$, i.e., the children of n not in $OPEN$. Then, to remove a node, we search downward along the worst children c^-, and remove n when its children are all in $OPEN$.

3.2.5 Empirical Evaluation

We evaluate the anytime performance of our proposed algorithm UBFS, in Algorithm 3.4, against three baselines which can also provide anytime bounds, on three benchmark problem sets. The baselines include AAOBF [Marinescu et al., 2017], a state-of-the-art search algorithm, XOR_MMAP [Xue et al., 2016], a recent random hashing based approach, and AFSE [Mauá and de Campos, 2012], a factor-set elimination scheme. AAOBF and AFSE can provide anytime deterministic bounds, while XOR_MMAP provides anytime stochastic bounds. Generally, the baselines are chosen to cover recent anytime bounding schemes from different categories.

The benchmark set includes three problem domains: grid networks (`grid`), medical diagnosis expert systems (`promedas`), and `protein`, made from the "small" protein side-chains of Yanover and Weiss [2002]. Our `grid` set is a subset of the grid dataset used in Marinescu et al. [2017], with "small" instances (less than 324 variables) excluded because they can be solved exactly during the heuristic construction and so provide little insight into the search

Table 3.3: Statistics of the three evaluated benchmark sets. "avg. induced width" and "avg. pseudotree depth" are computed given 50% of variables randomly selected as MAX variables.

	`grid`	`promedas`	`protein`
# instances	100	100	50
avg. # variables	764.48	1063.84	99.96
avg. # of factors	764.48	1076.84	355.84
avg. max domain size	2.00	2.00	77.94
avg. max scope	3.00	3.00	2.00
avg. induced width	190.59	136.72	35.28
avg. pseudotree depth	218.23	170.50	43.72

process. The `promedas` set is the same dataset as that used in Marinescu et al. [2017]. We tested two settings of MAX variables. In the first setting, 50% of the variables are randomly selected as MAX variables, which is the same setting used in Marinescu et al. [2017]. To compare the algorithms on instances with relatively hard internal summation problems, we also decrease the percentage of MAX variables to 10% for the second setting. The statistics of the benchmarks in Table 3.3 show these instances are very challenging.

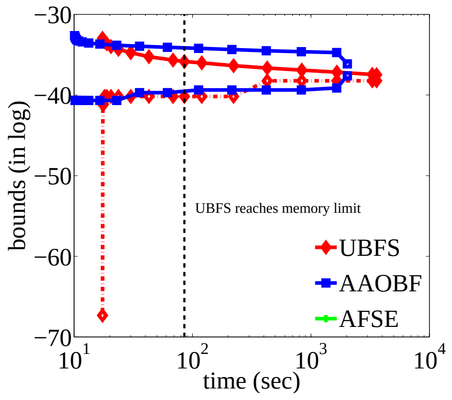
The time budget is set to 1 hour for all experiments. We allot 4GB memory to all algorithms, with 1GB extra memory to AAOBF for caching. For our heuristic search methods, we use WMB heuristics, whose memory usage is roughly controlled by the *ibound*. For a given memory budget, we first compute the largest *ibound* that fits in memory, then use the remaining memory for search. Since AAOBF also uses weighted mini-bucket heuristics, the same *ibound* is shared during heuristic construction between our proposed algorithm and AAOBF. Implementations of all methods are in C/C++ by the original authors except AFSE, implemented by the authors of Marinescu et al. [2017]. The step size used by AFSE to control the partitioning of the factor sets is set to 1 since little difference in performance was observed for larger values [Marinescu et al., 2017]. For XOR_MMAP, we adopt parameter values suggested by its authors. Unfortunately, XOR_MMAP failed to produce valid bounds on many of our instances in the allotted time, perhaps because it maintains multiple copies of the problem instance and must solve internal NP-hard parity constraint problems. Despite its promising solutions and bounds on small instances [Xue et al., 2016], it does not appear to scale well to our harder benchmarks.

Individual Results

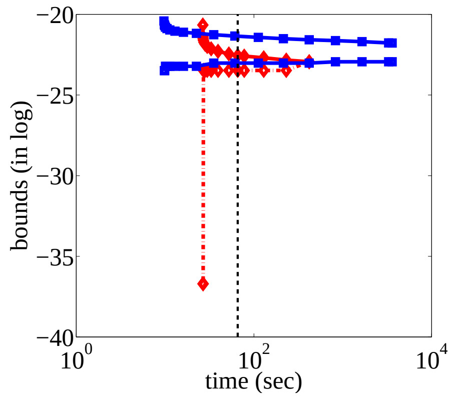
Our algorithm is designed to improve upper bounds in an anytime fashion. Figure 3.5 shows the methods' anytime behavior on individual instances from each benchmark. The lower bounds from UBFS in those plots, corresponding to the best solutions found so far by UBFS at the respective timestamps, are computed offline and are intended only for reference, to give a sense of the quality of the anytime MAP configurations predicted by UBFS.

From Figure 3.5, we observe that only UBFS and AAOBFS are able to provide valid (upper) bounds on all the 6 instances. AFSE runs out of memory on all but one instance before providing any bounds, which was fairly typical; for comparison, UBFS and AAOBF are able to solve this instance optimally. We commonly observed that when AFSE is able to produce bounds, UBFS and AAOBF usually produce better bounds or even find optimal solutions. XOR_MMAP failed to provide any solutions or bounds for these instances.

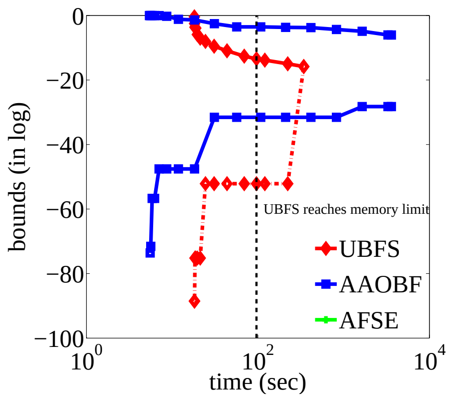
When UBFS reaches the memory limit, it keeps improving the upper bounds and optimally solves some of the problems (e.g., Figures 3.5(b), 3.5(c), and 3.5(d)). As expected, memory is a bottleneck and UBFS usually reaches the 4GB limit (vertical lines in the plots) within a few minutes of search, but continues to improve its bound by pruning low-priority nodes. In contrast, AAOBF terminates when memory is used up (e.g., Figure 3.5(a)). Note that UBFS typically runs into the memory limit faster than AAOBF; in Figure 3.5(a) UBFS uses up its memory in 100 seconds, while AAOBF reaches the limit after 1000 seconds. This is because UBFS is a pure best-first scheme, while AAOBF is a hybrid scheme that solves internal summation problems using depth-first search; if the DFS step is slow (hard summation



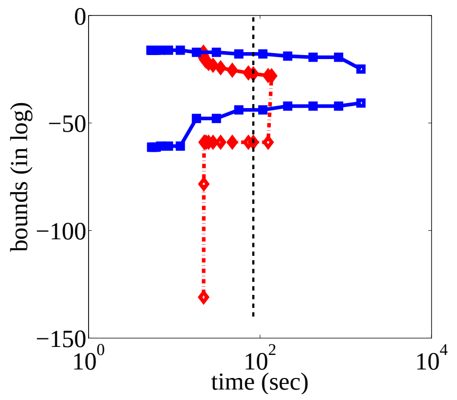
(a) `grid`/75-26-5



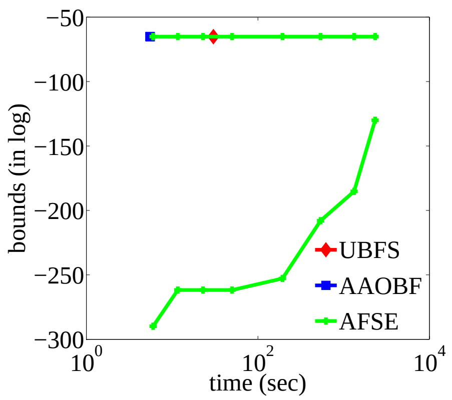
(b) `grid`/75-20-5



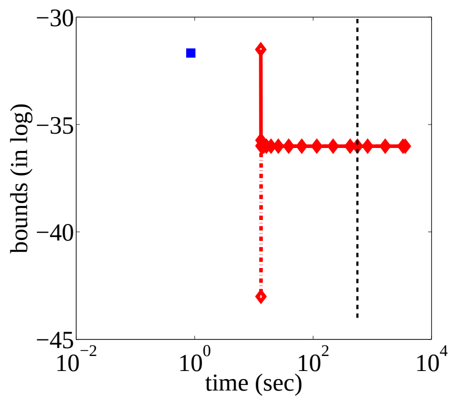
(c) `promedas`/or_chain_50.fg



(d) `promedas`/or_chain_4.fg



(e) `protein`/1noa



(f) `protein`/1r69

Figure 3.5: Anytime bounds for two instances per benchmark, with 50% MAX variables. AFSE is missing if it ran out of memory before producing bounds; XOR_MMAP failed to produce bounds on these instances. UBFS lower bounds are computed offline and shown only for reference. Black dotted lines mark UBFS reaching the 4GB memory limit; time budget 1 hour.

Table 3.4: Number of instances with non-trivial bounds at three times (1 min, 10 min, and 1 hour resp.) for each benchmark. 50% MAX variables. The highest for each setting is bolded.

	`grid`	`promedas`	`protein`
# instances	100	100	50
	Timestamp: 1min/10min/1hr		
UBFS	**100/100/100**	**100/100/100**	**46/50/50**
AAOBF	98/**100/100**	**100/100/100**	23/31/34
AFSE	0/0/0	25/27/27	7/7/7

problems), the best-first portion of the search will proceed slowly. Figure 3.5(f) shows an example where evaluation of this summation is so slow that AAOBF fails to provide any bounds beyond its inital heuristic upper bound; in contrast, UBFS quickly finds a much higher-quality upper bound (albeit without a corresponding online lower bound).

Collective Results

We evaluate some statistics across algorithms to compare their performance on the benchmarks. Since XOR_MMAP generally fails to provide bounds, we exclude it from the remaining discussion.

Responsiveness. Responsiveness characterizes how fast an algorithm produces non-trivial bounds. For UBFS and AAOBF, we require them to produce bounds other than the initial heuristic bound. From Table 3.4, we can see that UBFS responds quickly on almost all the instances within 1 minute, except for 4 `protein` instances that require more time to process. AAOBF is responsive on most `grid` and `promedas` instances, but performs worse on a number of `protein` instances due to their very hard internal summation problems. AFSE is not competitive and fails to produce any bounds on the `grid` instances.

Bound quality. We compare anytime upper bounds among the algorithms. Table 3.5 shows the number of instances for which an algorithm achieves the tightest upper bounds at each of

Table 3.5: Number of instances that an algorithm achieves the best upper bounds at each timestamp (1 min, 10 min, and 1 hour) for each benchmark. **50%** MAX variables. The best for each setting is bolded.

	`grid`	`promedas`	`protein`
# instances	100	100	50
Timestamp: 1min/10min/1hr			
UBFS	**85/84/89**	**83/86/87**	**46/50/50**
AAOBF	33/46/44	47/47/47	17/16/18
AFSE	0/0/0	0/0/0	5/5/5

Table 3.6: Number of instances that an algorithm achieves best upper bounds at each given timestamp (1 min, 10 min, and 1 hour) for each benchmark. **10%** MAX variables. The best for each setting is bolded.

	`grid`	`promedas`	`protein`
# instances	100	100	50
Timestamp: 1min/10min/1hr			
UBFS	**99/100/100**	**88/99/99**	**43/50/50**
AAOBF	1/1/3	17/12/17	15/9/9
AFSE	0/0/0	10/8/10	7/7/7

three timestamp. From Table 3.5, we see that our algorithm does best on this task across all benchmark/timestamp settings, by a large margin. Again, the advantage is most significant on the `protein` instances, and again, AAOBFS performs better than AFSE.

Harder summation. UBFS avoids some unnecessary evaluation of the internal summation problems, thus it is expected to be suitable for those MMAP problems with hard internal summation problems. To assess this setting, we decrease the percentage of MAX variables from 50% to 10%. Table 3.6 shows the relative upper bound quality. By comparing Table 3.5 and Table 3.6, we can see that UBFS gains a larger advantage when the search space of MAX variables is smaller, while the summation tasks are relatively harder.

▢ 3.2.6 Summary of UBFS

In this section, we present an anytime anyspace AND/OR best-first search algorithm to improve upper bounds for MMAP problems. We cast max and sum inference into one best-first search framework. The specially designed double-priority system allows our algorithm to identify the current most promising partial MAP solution tree and expand the frontier node that contributes most to the upper bound of that partial MAP solution tree, which often leads to quick reduction on the global upper bound. Our algorithm avoids some unnecessary exact computation of internal conditional summation problems and thus has a significant advantage especially on those instances with hard internal summation problems. Empirical results demonstrate that our approach with heuristics extracted from weighted mini-bucket is superior to those state-of-the-art baselines on various settings.

Chapter 4

Sampling Enhanced by Best-first Search

In this chapter, we develop approximate inference algorithms involving importance sampling, heuristic search, and variational bounds, for the tasks of bounding the partition function and MMAP respectively.

In Section 4.1, we propose a dynamic importance sampling (DIS) scheme that provides anytime finite-sample bounds for the partition function. Our algorithm balances the advantages of the three major inference strategies, Monte Carlo sampling, heuristic search, and variational bounds, blending sampling with search to refine a variationally defined proposal. Our algorithm combines and generalizes work on anytime search (Algorithm 3.1) and probabilistic bounds [Liu et al., 2015a] of the partition function. By using an intelligently chosen weighted average over the samples, we construct an unbiased estimator of the partition function with strong finite-sample confidence intervals that inherit both the rapid early improvement rate of sampling and the long-term benefits of an improved proposal from search. This gives significantly improved anytime behavior, and more flexible trade-offs between memory, time, and solution quality.

In Section 4.2, we propose a mixed dynamic importance sampling (MDIS) algorithm that blends heuristic search and importance sampling to provide anytime finite-sample bounds for marginal MAP, along with predicted MMAP configurations. We convert bounding the marginal MAP solution to a surrogate task of bounding a series of summation problems of an augmented graphical model, and then adapt DIS to provide finite-sample bounds for the surrogate task. Those bounds are guaranteed to become tight given enough time, and the values of the predicted MMAP solutions will converge to the optimum. Our algorithm runs in an anytime/anyspace manner, which gives flexible trade-offs between memory, time, and solution quality. We demonstrate the effectiveness of our approach empirically by comparing to a number of state-of-the-art search algorithms.

4.1 Dynamic Importance Sampling

In this section, we present a dynamic importance sampling (DIS) algorithm that provides anytime probabilistic bounds (i.e., they hold with probability $(1-\delta)$ for some confidence parameter δ) for the partition function. DIS interleaves importance sampling with best-first search (AOBFS, see Algorithm 3.1) which is used to refine the proposal distribution of successive samples. In practice, DIS enjoys both the rapid bound improvement characteristic of importance sampling [Liu et al., 2015a], while also benefiting significantly from search on problems where search is relatively effective, or when given enough computational resources, even when these points are not known in advance. Since our samples are drawn from a sequence of different, improving proposals, we devise a weighted average estimator that upweights higher-quality samples, giving excellent anytime behavior.

4.1.1 Introduction

As we mentioned in Chapter 3, search algorithms [Henrion, 1991, Viricel et al., 2016] explicitly enumerate over the space of configurations and eventually provide an exact answer; however, while some problems are well-suited to search, others only improve their quality very slowly with more computation. Importance sampling [e.g., Dagum and Luby, 1997, Liu et al., 2015a] gives probabilistic bounds that improve with more samples at a predictable rate; in practice this means bounds that improve rapidly at first, but are slow to become very tight. Several algorithms combine two strategies: approximate hash-based counting combines sampling (of hash functions) with CSP-based search [e.g., Chakraborty et al., 2014, 2016] or other MAP queries [e.g., Ermon et al., 2013, 2014], although these are not typically formulated to provide anytime behavior. Most closely related to this work are AOBFS and WMB-IS [Liu et al., 2015a], which perform search and sampling, respectively, guided by variational bounds.

A Motivating Example

We illustrate the focus and contributions of our work on an example problem instance (Figure 4.1). Search (see Algorithm 3.2) provides strict bounds (gray) but may not improve rapidly, particularly once memory is exhausted; on the other hand, sampling using WMB-IS [Liu et al., 2015a] provides probabilistic bounds (green) that improve at a predictable rate, but require more and more samples to become tight. We first describe a "two stage" sampling process that uses a search tree to improve the baseline bound from which importance sampling starts (blue), greatly improving its long-term performance, then present our dynamic importance sampling (DIS) algorithm, which interleaves the search and sampling processes (sampling from a sequence of proposal distributions) to give bounds that are strong in an anytime sense.

4.1.2 WMB-IS

Since WMB-IS [Liu et al., 2015a] is closely related to our algorithm, we present it here for completeness. In the sequel, we assume all the weights of WMB satisfy the convex condition in (2.8), i.e., all are non-negative and sum to one. We use "E" to denote the expectation operator (without a subscript to indicate the distribution when it is obvious).

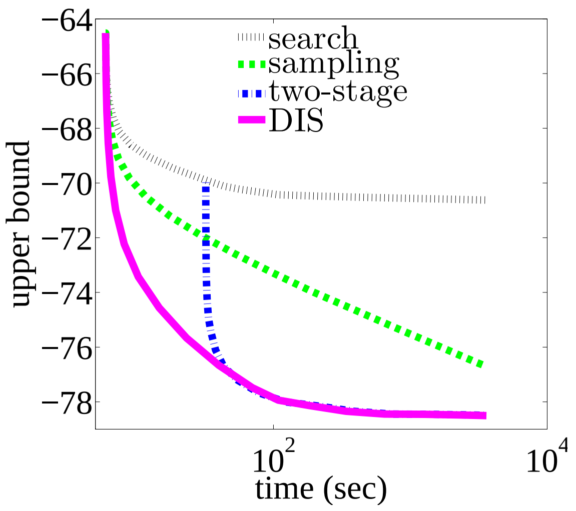


Figure 4.1: Example: upper bounds on $\log Z$ for `protein` instance "1bgc".

Recall that when WMB processes a variable, say, X_i, it first partitions all unprocessed factors (including those intermediately generated ones) that contain X_i into several disjoint subsets called mini-buckets. WMB then eliminates X_i from those mini-buckets, which generates new factors called messages (see (2.7) for a precise definition). Those messages along with their corresponding factors can be used to define conditional distributions over X_i. For example, the conditional distribution q_{ij} derived from the j-th mini-bucket $\mathcal{B}_i^j$ of X_i and its induced message $\lambda_{i\to\pi_{ij}}$ is as follows:

$$q_{ij}(x_i|x_{an_j(i)}) = \left(\frac{\prod_{f_\alpha \in \mathcal{B}_i^j} f_\alpha}{\lambda_{i\to\pi_{ij}}}\right)^{\frac{1}{\rho_{ij}}}.$$

where $X_{an_j(i)}$ are the variables included in the mini-bucket $\mathcal{B}_i^j$ other than X_i. Note that $q_{ij}(x_i|x_{an_j(i)})$ is a proper probability measure because it is non-negative and normalized:

$$\sum_{x_i} q_{ij}(x_i|x_{an_j(i)}) = \frac{\sum_{x_i} \prod_{f_\alpha \in \mathcal{B}_i^j} f_\alpha^{\frac{1}{\rho_{ij}}}}{\left(\lambda_{i\to\pi_{ij}}\right)^{\frac{1}{\rho_{ij}}}} = 1.$$

The last equation is derived by simply applying the definition of $\lambda_{i\to\pi_{ij}}$:

$$\lambda_{i\to\pi_{ij}} = \left(\sum_{x_i} \prod_{f_\alpha \in \mathcal{B}_i^j} f_\alpha^{\frac{1}{\rho_{ij}}} \right)^{\rho_{ij}} .$$

If we raise each q_{ij} to the power of ρ_{ij} and then take the product:

$$\begin{aligned} \prod_i \prod_j q_{ij}^{\rho_{ij}}(x_i | x_{an_j(i)}) &= \prod_i \prod_j \frac{\prod_{f_\alpha \in \mathcal{B}_i^j} f_\alpha}{\lambda_{i\to\pi_{ij}}} \\ &= \frac{\prod_{\alpha\in\mathcal{I}} f_\alpha}{U} \\ &= \frac{f(x)}{U}. \end{aligned} \tag{4.1}$$

In the preceding equation, the middle equality holds because of the following: any intermediate non-constant message $\lambda_{i\to\pi_{ij}}$ appears exactly once in the denominator and numerator respectively, corresponding to the bucket i in which it is generated and the parent bucket π_{ij} in which it is placed, and thus is canceled out. This leaves only the original factors f_α and the product of all constant messages (i.e., the WMB upper bound U) remaining in the quotient. Thus, Eq. (4.1) gives a reparametrization of the model:

$$f(x) = U \prod_i \prod_j q_{ij}^{\rho_{ij}}(x_i | x_{an_j(i)}) \tag{4.2}$$

We can then use the q_{ij} to define the WMB-IS proposal $q(x)$:

$$q(x) = \prod_i \sum_j \rho_{ij} q_{ij}(x_i | x_{an_j(i)}). \tag{4.3}$$

Notice that $q(x)$ takes the form of a product of conditional distributions over each X_i, where each conditional distribution is a weighted average (mixture distribution) of the conditionals q_{ij}, which condition on variables eliminated after X_i in the order (ancestors of the mini-buckets

$\mathcal{B}_i^j$). It is thus easy to verify that $q(x)$ is a valid distribution.

Moreover, for each i, the terms in (4.2) and (4.3) can be seen to correspond to a geometric mean and arithmetic mean, respectively, of the q_{ij}. Thus, applying the arithmetic-geometric mean (AM-GM) inequality, we have:

$$\prod_j q_{ij}^{\rho_{ij}}(x_i | x_{an_j(i)}) \leq \sum_j \rho_{ij} q_{ij}(x_i | x_{an_j(i)}),$$

from which we can immediately see that the importance weights $f(x)/q(x)$ generated by samples from this choice of $q(x)$ are guaranteed to be bounded:

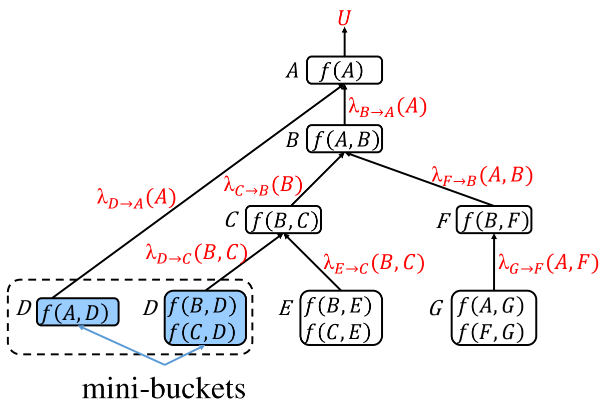
$$\frac{f(x)}{q(x)} \leq U. \tag{4.4}$$

A basic property of importance sampling is that it provides an unbiased estimate of Z (see Section 2.4.2):

$$\mathrm{E}\left[\frac{f(x)}{q(x)}\right] = Z. \tag{4.5}$$

Together, the boundedness and unbiasedness properties can be used to obtain finite-sample bounds on the partition function. Specifically, applying the empirical Bernstein form of concentration bound (see Theorem 2.3) gives:

Proposition 4.1 (Corollary 3.2 of Liu et al. [2015a]). Let $\{x^i\}_{i=1}^N$ be i.i.d. samples drawn from $q(x)$ as defined in (4.3), $\widehat{\mathrm{Var}}$ be the empirical variance of $\{f(x^i)/q(x^i)\}_{i=1}^N$ (see



(a)

Extract a proposal distribution $q(x)$:

$\tilde{a} \sim q(a) = \lambda_{D\to A}(a)\lambda_{B\to A}(a)f(a)/U$

$\vdots$

$\tilde{d} \sim \rho_1 q_1(d|\tilde{a}) + \rho_2 q_2(d|\tilde{b},\tilde{c})$

Weighted mixture:
use mini-bucket 1 with probability ρ_1, or, mini -bucket 2 with probability $\rho_2 = 1 - \rho_1$, where
$q_1(d|\tilde{a}) = \left[\frac{f(\tilde{a},d)}{\lambda_{D\to A}(\tilde{a})}\right]^{\frac{1}{\rho_1}}, q_2(d|\tilde{b},\tilde{c}) = \left[\frac{f(\tilde{b},d)f(\tilde{c},d)}{\lambda_{D\to C}(\tilde{b},\tilde{c})}\right]^{\frac{1}{\rho_2}}.$

(b)

Figure 4.2: (a) Given an elimination order G, F, E, D, C, B, A, and *ibound* $= 2$, WMB runs on the model depicted in Figure 2.2(a) to produce an upper bound of the partition function. (b) The same relaxation from (a) gives a proposal distribution with properties present in (4.4) and (4.5).

(2.20) for reference), and

$$\widehat{Z} = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x^i)}{q(x^i)},$$
$$\Delta = \sqrt{\frac{2\widehat{\mathrm{Var}}\log(2/\delta)}{N}} + \frac{7U\log(2/\delta)}{3(N-1)}.$$

For any $\delta \in (0,1)$, we have that $\widehat{Z} + \Delta$ and $\widehat{Z} - \Delta$ are upper and lower bounds on Z with probability at least $(1-\delta)$ respectively:

$$\Pr[Z \le \widehat{Z} + \Delta] \ge 1 - \delta,$$
$$\Pr[Z \ge \widehat{Z} - \Delta] \ge 1 - \delta. \tag{4.6}$$

When this bound is worse than the deterministic bound U, it can be simply replaced by U; however, it usually becomes significantly tighter than U very quickly as the number of samples N increases.

Example 4.1. Figure 4.2 shows an example of running WMB on the pairwise graphical model from Example 2.3 to produce an upper bound on the partition function as well as the proposal distribution $q(\cdot)$, assuming the elimination order G, F, E, D, C, B, A and imposing $ibound = 2$.

4.1.3 AOBFS Revisited

Our algorithm requires a closer look at AOBFS developed in Section 3.1 for bounding the partition function. First we introduce a *conditional cost* function $g(x_{>n}|x_{\leq n})$ at a node n of an AND/OR search tree to denote the quotient $g([x_{\leq n}, x_{>n}])/g(x_{\leq n})$ (see (2.12)):

$$g(x_{>n}|x_{\leq n}) = \frac{g([x_{\leq n}, x_{>n}])}{g(x_{\leq n})}.$$

where $x_{>n}$ is any assignment of $X_{>n}$, the variables below n in the search tree. Thus, the value v_n as defined in (2.16) for each node n can be re-interpreted as the total conditional cost of all configurations below n:

$$v_n = \sum_{x_{>n}} g(x_{>n}|x_{\leq n}). \tag{4.7}$$

The value of the root is simply the partition function, $v_\emptyset = Z$.

AOBFS (Algorithm 3.1) expands the search tree in a best-first manner, and maintains an explicit AND/OR search tree $\mathcal{S}$ of visited nodes. For each node n in the AND/OR search tree, AOBFS maintains an upper bound on v_n, denoted u_n (see (3.2)), which is initialized via our pre-compiled heuristic $v_n \leq h_n^+$, and subsequently updated during search using information propagated from the frontier. Thus, the upper bound at the root, $u_\emptyset$, is an anytime deterministic upper bound of Z. Note that this upper bound depends on the current search tree $\mathcal{S}$, so we write $U^{\mathcal{S}} = u_\emptyset$.

If all nodes below n have been visited, then $u_n = v_n$; we remove the subtree below n from memory because n is solved. Hence, we can partition the frontier nodes into two sets[1]: solved frontier nodes, *SOLVED*$(\mathcal{S})$, and unsolved ones, *OPEN*$(\mathcal{S})$. AOBFS assigns a priority to each node and expands a top-priority node from *OPEN*$(\mathcal{S})$ at each iteration. Here we use the "upper priority" U_n (defined in (3.4)) which quantifies n's contribution to the global bound $U^{\mathcal{S}}$; AOBFS with this priority attempts to reduce the upper bound on Z as quickly as possible. We can also interpret our bound $U^{\mathcal{S}}$ as a sum of bounds on each of the partial configurations covered by $\mathcal{S}$. Concretely, let $\mathbb{T}^{\mathcal{S}}$ be the set of projections of full solution trees on $\mathcal{S}$ (in other words, $\mathbb{T}^{\mathcal{S}}$ are partial solution trees whose leaves are frontier nodes of $\mathcal{S}$); then,

$$U^{\mathcal{S}} = \sum_{T \in \mathbb{T}^{\mathcal{S}}} U_T, \tag{4.8}$$

where

$$U_T = g(T) \prod_{s \in leaf(T)} u_s. \tag{4.9}$$

$leaf(T)$ are the leaf nodes of the partial solution tree T; $g(T)$ is defined as $g(x_T)$ (see (2.14)). We will lightly abuse notation by using $g(T)$ and $g(x_T)$ interchangeably.

■ 4.1.4 WMB for Heuristics and Sampling

WMB can not only provide upper bound heuristics for AOBFS (as detailed in Section 3.1), but also offers a proposal distribution with the same relaxation that yields finite-sample bounds as we see in WMB-IS. The strong connection between WMB heuristics and the mixture proposal distribution in (4.3) lays the foundation for development of our algorithms.

Let n be any node in the search tree; analogous to the derivation of (4.2), one can show that

[1] Note that this partitioning is proper because we apply only the plain AOBFS in Algorithm 3.1, not the memory-limited variant in Algorithm 3.2.

WMB yields the following reparametrization of the conditional cost below n:

$$g(x_{>n}|x_{\leq n}) = h_n^+ \prod_k \prod_j q_{ij}(x_i|x_{an_j(i)})^{\rho_{ij}}, \quad X_i \in X_{>n} \tag{4.10}$$

Suppose that we define a conditional distribution $q(x_{>n}|x_{\leq n})$ by replacing the geometric mean over the q_{ij} in (4.10) with their arithmetic mean:

$$q(x_{>n}|x_{\leq n}) = \prod_i \sum_j \rho_{ij} q_{ij}(x_i|x_{an_j(i)}) \tag{4.11}$$

Applying the arithmetic-geometric mean inequality, we see that

$$\frac{g(x_{>n}|x_{\leq n})}{h_n^+} \leq q(x_{>n}|x_{\leq n}).$$

Summing over $x_{>n}$ shows that h_n^+ is a valid upper bound heuristic for v_n:

$$v_n = \sum_{x_{>n}} g(x_{>n}|x_{\leq n}) \leq h_n^+$$

The mixture distribution $q(x_{>n}|x_{\leq n})$ can be also used as a proposal for importance sampling, by drawing samples from $q(x_{>n}|x_{\leq n})$ and averaging the importance weights. For any node n, we have that the importance weight $g(x_{>n}|x_{\leq n})/q(x_{>n}|x_{\leq n})$ is an unbiased and bounded estimate of v_n:

$$\underset{\text{(bounded weights)}}{\frac{g(x_{>n}|x_{\leq n})}{q(x_{>n}|x_{\leq n})} \leq h_n^+} \qquad \text{and} \qquad \underset{\text{(unbiased estimator)}}{\mathrm{E}\left[\frac{g(x_{>n}|x_{\leq n})}{q(x_{>n}|x_{\leq n})}\right] = v_n.} \tag{4.12}$$

In particular, we recover the WMB-IS proposal in (4.3) by setting n to the root, together with properties (4.4) and (4.5) of its importance weights.

▩ 4.1.5 Two-step Sampling

The finite-sample bound (4.6) suggests that improvements to the upper bound on Z may be translatable into improvements in the probabilistic, sampling bound. In particular, if we define a proposal that uses the search tree $\mathcal{S}$ and its bound $U^{\mathcal{S}}$, we can improve our sample-based bound as well. This motivates us to design a *two-step* sampling scheme that exploits the refined upper bound constructed using search; it is a top-down procedure starting from the root:

Step 1 For an internal node n: if it is an AND node, all its children are selected; if n is an OR node, one child $c \in ch(n)$ is randomly selected with probability $w_c u_c / u_n$.

Step 2 When a frontier node n is reached, if it is unsolved, draw a sample of $X_{>n}$ from $q(x_{>n}|x_{\le n})$; if it is solved, quit.

The behavior of **Step 1** can be understood by the following proposition:

Proposition 4.2. Step 1 returns a partial solution tree $T \in \mathbb{T}^{\mathcal{S}}$ with probability $U_T / U^{\mathcal{S}}$. Any frontier node of $\mathcal{S}$ will be reached with probability proportional to its upper priority defined in (3.4).

Note that at **Step 2**, although the sampling process terminates when a solved node n is reached, we associate every configuration $x_{>n}$ of $X_{>n}$ with probability $g(x_{>n}|x_{\le n})/v_n$ which is appropriate in lieu of (4.7). Thus, we can show that this two-step sampling scheme induces a proposal distribution, denoted $q^{\mathcal{S}}(x)$, which can be expressed as:

$$q^{\mathcal{S}}(x) = \prod_{n \in AND(T_x \cap \mathcal{S})} w_n u_n / u_{pa(n)} \prod_{n' \in OPEN(\mathcal{S}) \cap T_x} q(x_{>n'}|x_{\le n'}) \prod_{n'' \in SOLVED(\mathcal{S}) \cap T_x} g(x_{>n''}|x_{\le n''})/v_{n''}$$

where $AND(T_x \cap \mathcal{S})$ is the set of all AND nodes of the partial solution tree $T_x \cap \mathcal{S}$. By applying the upper bound recursion in (3.2), and noticing that the upper bound is the initial heuristic for any node in $\mathit{OPEN}(\mathcal{S})$ and is exact at any solved node, we re-write $q^{\mathcal{S}}(x)$ as

$$q^{\mathcal{S}}(x) = \frac{g(T_x \cap \mathcal{S})}{U^{\mathcal{S}}} \prod_{n' \in \mathit{OPEN}(\mathcal{S}) \cap T_x} h^{+}_{n'}\, q(x_{>n'}|x_{\le n'}) \prod_{n'' \in \mathit{SOLVED}(\mathcal{S}) \cap T_x} g(x_{>n''}|x_{\le n''}). \tag{4.13}$$

The search tree proposal $q^{\mathcal{S}}(x)$ can be shown to provide bounded importance weights that use the refined upper bound $U^{\mathcal{S}}$:

Proposition 4.3. Importance weights from $q^{\mathcal{S}}(x)$ are bounded by the upper bound of $\mathcal{S}$, and are unbiased estimators of Z, i.e.,

$$\frac{f(x)}{q^{\mathcal{S}}(x)} \le U^{\mathcal{S}} \qquad \text{and} \qquad \mathrm{E}\left[\frac{f(x)}{q^{\mathcal{S}}(x)}\right] = Z.$$

Proof. Note that $f(x)$ can be written as

$$f(x) = g(T_x \cap \mathcal{S}) \prod_{n' \in \mathit{OPEN}(\mathcal{S}) \cap T_x} g(x_{>n'}|x_{\le n'}) \prod_{n'' \in \mathit{SOLVED}(\mathcal{S}) \cap T_x} g(x_{>n''}|x_{\le n''})$$

Noticing that for any $n' \in \mathit{OPEN}(\mathcal{S})$, $g(x_{>n'}|x_{\le n'}) \le h^{+}_{n'}\, q(x_{>n'}|x_{\le n'})$ by (4.12), and comparing with (4.13), we see $f(x)/q^{\mathcal{S}}(x)$ is bounded by $U^{\mathcal{S}}$. Its unbiasedness is trivial. □

Thus, importance weights resulting from our two-step sampling can enjoy the same type of bounds described in (4.6). Moreover, note that at any solved node, our sampling procedure incorporates the “exact” value of that node into the importance weights, which serves as Rao-Blackwellisation [Casella and Robert, 1996] and can potentially reduce variance.

We can see that if $\mathcal{S} = \emptyset$ (before search), $q^{\mathcal{S}}(x)$ is the WMB-IS proposal in (4.3); as search proceeds, the quality of the proposal distribution improves, gradually approaching the underlying distribution $f(x)/Z$ as $\mathcal{S}$ approaches the complete search tree. If we perform search first, up to some memory limit, and then sample, which we refer to as *two-stage* sampling, our probabilistic bounds will proceed from an improved baseline, giving better

Algorithm 4.1 Dynamic Importance Sampling (DIS)

```
Require: Control parameters $N_d$, $N_l$; memory budget, time budget.
Ensure: $N$, $\mathrm{HM}(\boldsymbol{U})$, $\widehat{\mathrm{Var}}(\{\widehat{Z}_i/U_i\}_{i=1}^N)$, $\widehat{Z}$, $\Delta$.
 1: Initialize $\mathcal{S} \leftarrow \{\emptyset\}$ with the root $\emptyset$.
 2: while within the time budget
 3:     if within the memory budget          // update $\mathcal{S}$ and its associated upper bound $U^{\mathcal{S}}$
 4:         Expand $N_d$ nodes via AOBFS (Algorithm 3.1) with the upper priority (see (3.4)).
 5:     end if
 6:     Draw $N_l$ samples via TwoStepSampling($\mathcal{S}$).
 7:     After drawing each sample:
 8:         Update $N$, $\mathrm{HM}(\boldsymbol{U})$, $\widehat{\mathrm{Var}}(\{\widehat{Z}_i/U_i\}_{i=1}^N)$.
 9:         Update $\widehat{Z}$, $\Delta$ via (4.14), (4.15).
10: end while
11: function TwoStepSampling($\mathcal{S}$)
12:     Start from the root of the search tree $\mathcal{S}$:
13:         For an internal node $n$: select all its children if it is an AND node; select exactly
14:         one child $c \in ch(n)$ with probability $w_c u_c / u_n$ if it is an OR node.
15:         At any unsolved frontier node $n$, draw one sample from $q(x_{>n}|x_{\le n})$ in (4.11).
16: end function
```

bounds at moderate to long computation times. However, doing so sacrifices the quick improvement early on given by basic importance sampling. In the next section, we describe our dynamic importance sampling procedure, which balances these two properties.

4.1.6 Main Algorithm

To provide good anytime behavior, we would like to do both sampling and search, so that early samples can improve the bound quickly, while later samples obtain the benefits of the search tree's improved proposal. To do so, we define a dynamic importance sampling (DIS) scheme, presented in Algorithm 4.1, which interleaves drawing samples and expanding the search tree. Figure 4.3 illustrates the interleaving process of DIS.

One complication of such an approach is that each sample comes from a different proposal distribution, and thus has a different bound value entering into the concentration inequality. Moreover, each sample is of a different quality – later samples should have lower variance, since they come from an improved proposal. To this end, we construct an estimator of Z that upweights higher-quality samples. Let $\{x^i\}_{i=1}^N$ be a series of samples drawn via Algorithm 4.1, with $\{\widehat{Z}_i = f(x^i)/q^{\mathcal{S}_i}(x^i)\}_{i=1}^N$ the corresponding importance weights, and $\{U_i = U^{\mathcal{S}_i}\}_{i=1}^N$ the corresponding upper bounds on the importance weights respectively. We introduce an estimate $\widehat{Z}$ of Z:

$$\widehat{Z} = \frac{\mathrm{HM}(\boldsymbol{U})}{N} \sum_{i=1}^{N} \frac{\widehat{Z}_i}{U_i}, \qquad \text{where} \qquad \mathrm{HM}(\boldsymbol{U}) = \left[\frac{1}{N} \sum_{i=1}^{N} \frac{1}{U_i} \right]^{-1}. \tag{4.14}$$

Here, $\mathrm{HM}(\boldsymbol{U})$ is the harmonic mean of the upper bounds U_i. It is easy to see that $\widehat{Z}$ is an unbiased estimate of Z, since it is a weighted average of independent, unbiased estimators. Additionally, since $Z/\mathrm{HM}(\boldsymbol{U})$, $\widehat{Z}/\mathrm{HM}(\boldsymbol{U})$, and $\widehat{Z}_i/U_i$ are all within the interval $[0, 1]$, we can apply the empirical Bernstein bounds (see Theorem 2.3) to derive finite-sample bounds:

Theorem 4.1. *Define the deviation term*

$$\Delta = \mathrm{HM}(\boldsymbol{U}) \left(\sqrt{\frac{2\widehat{\mathrm{Var}}(\{\widehat{Z}_i/U_i\}_{i=1}^N)\log(2/\delta)}{N}} + \frac{7\log(2/\delta)}{3(N-1)} \right) \tag{4.15}$$

where $\widehat{\mathrm{Var}}(\{\widehat{Z}_i/U_i\}_{i=1}^N)$ is the unbiased empirical variance of $\{\widehat{Z}_i/U_i\}_{i=1}^N$ (see (2.20) *for reference). Then $\widehat{Z} + \Delta$ and $\widehat{Z} - \Delta$ are upper and lower bounds of Z with probability at least $1 - \delta$, respectively, i.e.,*

$$\Pr[Z \le \widehat{Z} + \Delta] \ge 1 - \delta,$$
$$\Pr[Z \ge \widehat{Z} - \Delta] \ge 1 - \delta.$$

It is possible that $\widehat{Z} - \Delta < 0$ at first; if so, we may replace $\widehat{Z} - \Delta$ with any non-trivial lower bound of Z. In the experiments, we use $\widehat{Z}\delta$, a $(1 - \delta)$ probabilistic bound by Markov's inequality (see (2.19)). We can also replace $\widehat{Z} + \Delta$ with the current deterministic upper

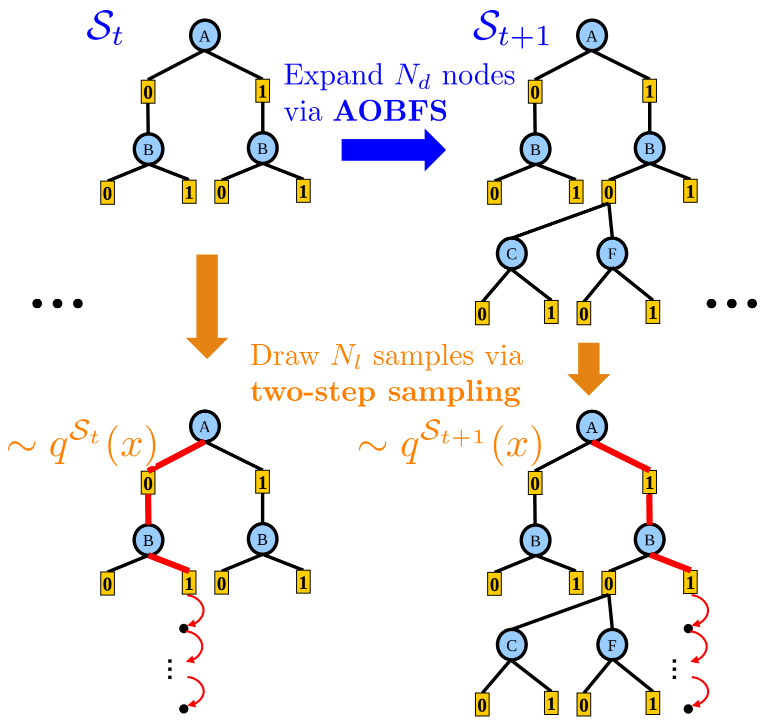

$\mathcal{S}_t$
$\mathcal{S}_{t+1}$
Expand N_d nodes via **AOBFS**
Draw N_l samples via **two-step sampling**
$\sim q^{\mathcal{S}_t}(x)$
$\sim q^{\mathcal{S}_{t+1}}(x)$
A
B
C
F
0
1

Figure 4.3: An illustration of the interleaving process of DIS in Algorithm 4.1.

bound if the latter is tighter.

Intuitively, our DIS algorithm is similar to Monte Carlo tree search (MCTS) [Browne et al., 2012], which also grows an explicit search tree while sampling. However, in MCTS, the sampling procedure is used to grow the tree, while DIS uses a classic search priority. This ensures that the DIS samples are independent, since samples do not influence the proposal distribution of later samples. This also distinguishes DIS from methods such as adaptive importance sampling (AIS) [Oh and Berger, 1992].

4.1.7 Empirical Evaluation

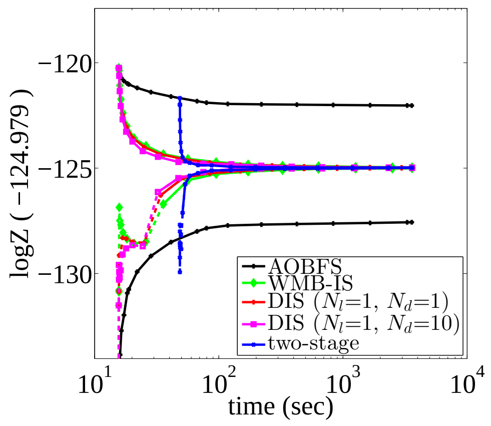
We evaluate DIS against AOBFS (search) and WMB-IS (sampling) on several benchmarks of real-world problem instances from recent UAI competitions. Our benchmarks include `pedigree`, 22 genetic linkage instances from the UAI'08 inference challenge; `protein`, 50 randomly selected instances made from the "small" protein side-chains of [Yanover and Weiss, 2002]; and BN, 50 randomly selected Bayesian networks from the UAI'06 competition[2]. These three sets are selected to illustrate different problem characteristics; for example `protein` instances are relatively small ($M = 100$ variables on average, and average induced width 11.2) but high cardinality (average $\max |\mathcal{X}_i| = 77.9$), while `pedigree` and `BN` have more variables and higher induced width (average M 917.1 and 838.6, average width 25.5 and 32.8), but lower cardinality (average $\max |\mathcal{X}_i|$ 5.6 and 12.4).

We alloted 1GB memory to all methods, first computing the largest *ibound* that fits the memory budget, and using the remaining memory for search. All the algorithms used the same upper bound heuristics, and thus DIS and AOBFS had the same amount of memory available for search. For AOBFS, we use the memory-limited version (see Algorithm 3.2) with the upper priority, which continues improving its bounds past the memory limit. Additionally, we let AOBFS access a lower bound heuristic for no cost, to facilitate comparison between DIS and AOBFS. We show DIS for two settings, (N_l=1, N_d=1) and (N_l=1, N_d=10), balancing the effort between search and sampling. Note that WMB-IS can be viewed as DIS with (N_l=Inf, N_d=0), i.e., it runs pure sampling without any search, and two-stage sampling viewed as DIS with (N_l=1, N_d=Inf), i.e., it searches up to the memory limit and then samples. We set $\delta = 0.025$ and ran each algorithm for 1 hour. All implementations are in C/C++.

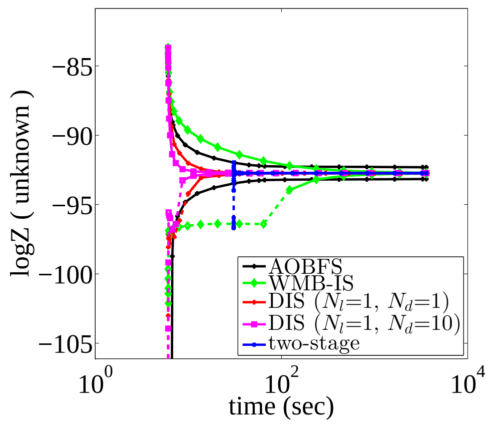
Anytime Bounds for Individual Instances

Figure 4.4 shows the anytime behavior of all methods on two instances from each benchmark. We observe that compared to WMB-IS, DIS provides better upper and lower bounds on all instances. In 4.4(d)–(f), WMB-IS is not able to produce tight bounds within 1 hour, but DIS quickly closes the gap. Compared to AOBFS, in 4.4(a)–(c),(e), DIS improves much faster, and in (d),(f) it remains nearly as fast as search. Note that four of these examples are sufficiently hard to be unsolved by a variable elimination-based exact solver, even with

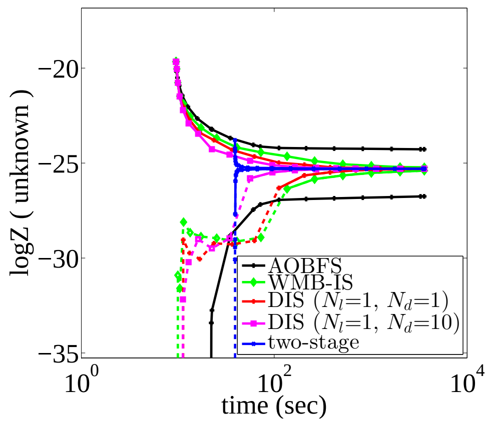
[2] `http://melodi.ee.washington.edu/~bilmes/uai06InferenceEvaluation/`



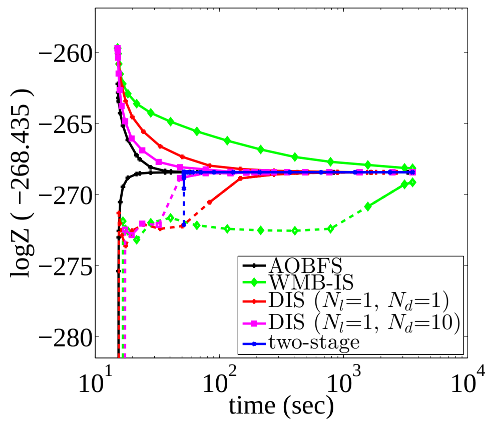
(a) `pedigree`/pedigree33



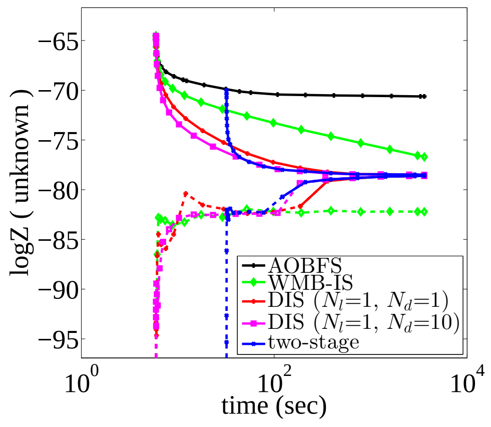
(b) `protein`/1co6



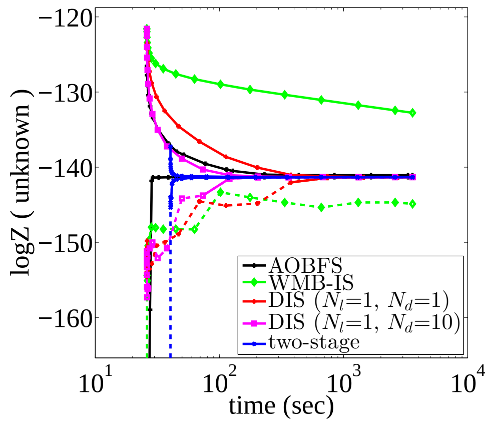
(c) `BN`/BN_30



(d) `pedigree`/pedigree37



(e) `protein`/1bgc



(f) `BN`/BN_129

Figure 4.4: Anytime bounds on $\log Z$ for two instances per benchmark. Dotted line sections on some curves indicate Markov lower bounds. In examples where search is very effective (d,f), or where sampling is very effective (a), DIS is equal or nearly so, while in (b,c,e) DIS is better than either.

Table 4.1: Mean area between upper and lower bounds of log$Z$, normalized by WMB-IS, for each benchmark. Smaller numbers indicate better anytime bounds. The best for each benchmark is bolded.

	AOBFS	WMB-IS	DIS (N_l=1, N_d=1)	DIS (N_l=1, N_d=10)	two-stage
`pedigree`	16.638	1	0.711	**0.585**	1.321
`protein`	1.576	1	0.110	**0.095**	2.511
`BN`	0.233	1	0.340	**0.162**	0.865

several orders of magnitude more computational resources (200GB memory, 24 hour time limit).

DIS provides excellent anytime behavior; in particular, (N_l=1, N_d=10) seems to work well, perhaps because expanding the search tree is slightly faster than drawing a sample (since the tree depth is less than the number of variables). On the other hand, two-stage sampling gives weaker early bounds, but is often excellent at longer time settings.

Aggregated Results across The Benchmarks

To quantify anytime performance of the methods in each benchmark, we introduce a measure based on the area between the upper and lower bound of log$Z$. For each instance and method, we compute the area of the interval between the upper and lower bound of log$Z$ for that instance and method. To avoid vacuous lower bounds, we provide each algorithm with an initial lower bound on log$Z$ from WMB. To facilitate comparison, we normalize the area of each method by that of WMB-IS on each instance, then report the geometric mean of the normalized areas across each benchmark in Table 4.1; this shows the average relative quality compared to WMB-IS; smaller values indicate tighter anytime bounds. We see that on average, search is more effective than sampling on the `BN` instances, but much less effective on `pedigree`. Across all three benchmarks, DIS (N_l=1, N_d=10) produces the best result by a significant margin, while DIS (N_l=1, N_d=1) is also very competitive, and two-stage sampling does somewhat less well.

4.2 Mixed Dynamic Importance Sampling

Since the probabilistic bounds produced by sampling, and DIS in particular, are often much faster to improve than deterministic bounds, we would like to apply them to other inference tasks, such as marginal MAP, as well. However, sampling methods are built around approximating expectations, which makes them difficult to apply directly to maximization queries such as MAP and marginal MAP. In this section, we describe a generalization to DIS that allows it to provide anytime, finite-sample probabilistic bounds on MMAP.

Briefly speaking, we follow Doucet et al. [2002] to construct an augmented graphical model from the original model by replicating the marginalized variables and potential functions. From this augmented model, we derive a sequence of decreasing summation objectives that bound the MMAP optimum raised to some fixed power. Then, we adapt DIS to bound these summation objectives and provide finite-sample bounds of the MMAP optimum. Our framework has several key advantages: 1) it provides anytime probabilistic upper and lower bounds that are guaranteed to be tight given enough time. 2) it is able to predict high-quality MAP solutions whose values converge to the optimum; the exploration-exploitation trade-off of searching MAP solutions is controlled by the number of replicates of the marginalized variables. 3) it runs in an anytime/anyspace manner, which gives flexible trade-offs between memory, time, and solution quality.

4.2.1 Introduction

As we discussed in Section 3.2, those search methods (e.g., Marinescu et al. [2017]) for bounds of MMAP typically require regular evaluation of internal summation problems when traversing the MAP space; when these internal sums are difficult, the search process may stall completely.

One way to avoid this issue is to unify the summation with the MAP search in a single, best-first search framework (UBFS), which allows the bounds to improve as the summation is performed, and switch to other MAP configurations when appropriate.

Another promising route is to make use of probabilistic bounds (e.g., DIS), which hold with a user-selected probability, and can be significantly faster and tighter than deterministic bounds. However, since each MAP configuration is associated with an independent summation problem, comparing MAP configurations using probabilistic bounds must compensate for the presence of many uncertain tests (in effect, a multiple hypothesis testing problem), and is thus non-trivial to adapt to the MAP search, which may contain exponentially many such configurations.

To some extent, the intrinsic hardness of MMAP arises from the non-commutativity of the sum and max operations. One natural idea to alleviate this issue is to convert the mixed inference task to a pure sum or a pure max one first. For example, Cheng et al. [2012] constructs an explicit factorized approximation of the marginalized distribution using a form of approximate variable elimination, which results in a structured MAP problem.

Following this line of thoughts, we design an algorithm taking advantage of both search and sampling, the general idea of which is to first bound the mixed inference objective with a series of sum inference objectives whose finite-sample bounds can be established by generalizing DIS, and then translate the bounds back to those of the original objective in which we are interested.

■ 4.2.2 An Augmented Graphical Model

We first introduce an augmented graphical model which connects the MMAP optimum to a series of summation tasks. The augmented graphical model is built from the original model by replicating the SUM variables and the factors. Note that the idea of introducing an

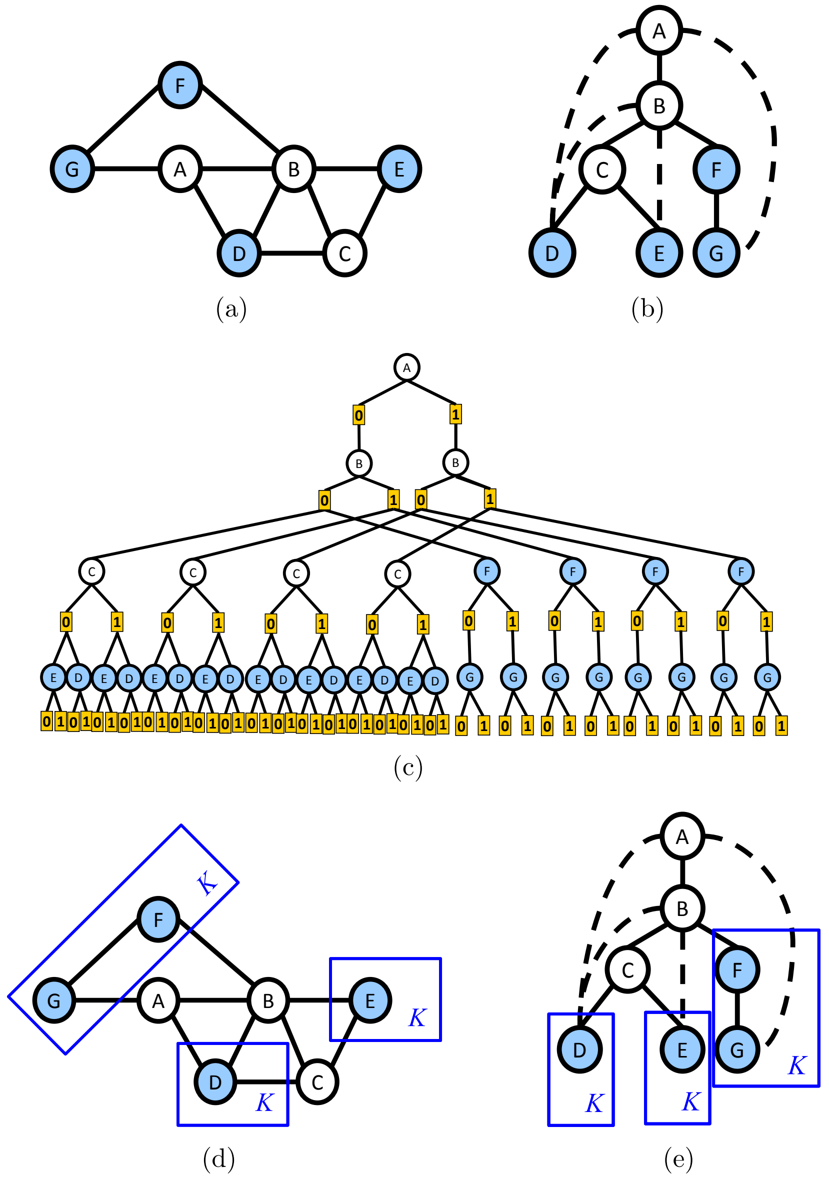


Figure 4.5: (a) A primal graph of a graphical model over 7 variables. A, B, C are MAX variables and D, E, F, G are SUM variables. (b) A valid pseudo tree for the MMAP task on the model in (a). (c) An AND/OR search tree guided by the pseudo tree in (b). (d) An augmented model created by replicating SUM variables and factors of the original model in (a). Plate notations used here. (e) A valid pseudo tree for the augmented model.

augmented space on which we perform inference is adopted from Doucet et al. [2002].

Let $X_{\text{aug}} = (X_{\text{M}}, X_{\text{S}}^1, \ldots, X_{\text{S}}^K)$ be all the variables of the augmented model where $X_{\text{S}}^1, \ldots, X_{\text{S}}^K$ are K replicates of the SUM variables X_{S}. The overall function f_{aug} of the augmented model is defined as

$$f_{\text{aug}}(x_{\text{aug}}) = \prod_{k=1}^{K} f(x_{\text{M}}, x_{\text{S}}^{k}).$$

Thus, the partition function of the augmented model is

$$\begin{aligned} Z_{\text{aug}} &= \sum_{x_{\text{M}}, x_{\text{S}}^{1}, \ldots, x_{\text{S}}^{K}} \prod_{k=1}^{K} f(x_{\text{M}}, x_{\text{S}}^{k}) \\ &= \sum_{x_{\text{M}}} \pi^{K}(x_{\text{M}}) \end{aligned}$$

where

$$\pi(x_{\text{M}}) = \sum_{x_{\text{S}}} f(x).$$

Considering that $\pi^{K}(x_{\text{M}}^{\star})$ (where $x_{\text{M}}^{\star}$ is the optimal MAP solution, see (2.4)) is the largest term in the sum on the R.H.S., we have

$$\frac{Z_{\text{aug}}}{|\mathcal{X}_{\text{M}}|} \leq \pi^{K}(x_{\text{M}}^{\star}) \leq Z_{\text{aug}}, \tag{4.16}$$

that is to say,

$$\Big(\frac{Z_{\text{aug}}}{|\mathcal{X}_{\text{M}}|}\Big)^{1/K} \leq \pi(x_{\text{M}}^{\star}) \leq Z_{\text{aug}}^{1/K},$$

where $|\mathcal{X}_{\text{M}}|$ is the size of the MAP space. The above inequalities are actually well-known boundedness relations between the ∞-norm and p-norms of the Euclidean space $\mathbb{R}^{|\mathcal{X}_{\text{M}}|}$. These bounds are monotonic in K, i.e., they improve as K increases, and become tight as K goes to infinity. In other words, K acts as a "reverse temperature" parameter. The lower bound is negatively impacted by the domain sizes of the MAX variables, which can be quite loose if $|\mathcal{X}_{\text{M}}|$ is large compared to the scale of K.

The significance of (4.16) is that it connects the MMAP optimum to a summation quantity Z_{aug} that can be easily approximated using Monte Carlo methods such as importance sampling.

Example 4.2. Figure 4.5(d) shows an augmented graphical model created from the model of Figure 4.5(a). Figure 4.5(e) shows one valid pseudo tree for the augmented model.

Main Algorithm

A straightforward idea is to apply DIS to bound Z_{aug} whose finite-sample bounds can then be translated to those of $\pi(x_{\text{M}}^{\star})$. However, several key issues remain to be addressed for this idea to work well.

The first issue is about how to adapt DIS to the augmented model in an efficient manner. Since the augmented model might have many more variables compared to the original model, a naïve construction of AND/OR trees leads to an excessively large search space. Note that any X_{S}^{k} in the augmented model is an identical copy of X_{S}; we thus do not necessarily distinguish those X_{S} copies during search. That is to say, when search instantiates those factors involving SUM variables, it behaves as usual but takes into account the effect of replication when using information propagated from SUM nodes. We can also apply an analogous idea to construct WMB heuristics to ensure that they are still compatible with the new search process. In a nutshell, search for the augmented model can enjoy the same complexity as that for the original model.

Meanwhile, the proposal distribution $q_{\text{aug}}^{\mathcal{S}}(x_{\text{aug}})$ associated with a search tree $\mathcal{S}$ has a decomposition property:

$$q_{\text{aug}}^{\mathcal{S}}(x_{\text{aug}}) = q_{\text{aug}}^{\mathcal{S}}(x_{\text{M}}) \prod_{k=1}^{K} q_{\text{aug}}^{\mathcal{S}}(x_{\text{S}}^{k}|x_{\text{M}}), \tag{4.17}$$

with $q_{\text{aug}}^{\mathcal{S}}(x_{\text{S}}^k|x_{\text{M}})$ are identical conditional distributions. Its importance weights also share the boundedness property:

$$\frac{f_{\text{aug}}(x_{\text{aug}})}{q_{\text{aug}}^{\mathcal{S}}(x_{\text{aug}})} \leq U_{\text{aug}}^{\mathcal{S}},$$

where $U_{\text{aug}}^{\mathcal{S}}$ is the upper bound associated with $\mathcal{S}$. Note that sampling from $q_{\text{aug}}^{\mathcal{S}}$ can also be done via a two-step sampling procedure analogous to that in DIS.

One point worth mentioning is that

$$\pi(x_{\text{M}}) = \text{E}\left[\frac{f(x_{\text{M}}, x_{\text{S}}^k)}{q_{\text{aug}}^{\mathcal{S}}(x_{\text{S}}^k|x_{\text{M}})}\right]$$

implies that we can estimate the value of each sampled MAP configuration x_{M} along the way.

Another issue is that if Z_{aug} is much larger than $\pi^K(x_{\text{M}}^\star)$, even high-quality bounds of Z_{aug} might not result in reasonably good bounds of $\pi(x_{\text{M}}^\star)$, let alone those bounds will never be tight in general for $\pi(x_{\text{M}}^\star)$ with a finite K. One way to alleviate this issue is based on the following key observation: for any subset $\mathcal{A}$ of $\mathcal{X}_{\text{M}}$ that contains $x_{\text{M}}^\star$, we have

$$\frac{Z_{\text{aug}}^{\mathcal{A}}}{|\mathcal{A}|} \leq \pi^K(x_{\text{M}}^\star) \leq Z_{\text{aug}}^{\mathcal{A}}, \tag{4.18}$$

where

$$Z_{\text{aug}}^{\mathcal{A}} = \sum_{x_{\text{M}} \in \mathcal{A}} \pi^K(x_{\text{M}}).$$

The above inequalities tell us that if we know an instantiation of X_{M} is not optimal, we can mute its contribution to Z_{aug} and use the resulting smaller summation quantity to bound $\pi^K(x_{\text{M}}^\star)$.

This observation enables pruning during search: any node ruled out from being associated

Algorithm 4.2 Mixed Dynamic Importance Sampling (MDIS)

```
Require: Control parameters $K$, $N_d$, $N_l$; confidence parameter $\delta$; memory budget, time budget.
Ensure: $\widehat{Z}_{\text{aug}}$, $\Delta$, $\text{HM}(\boldsymbol{U})$, $\text{HM}(\boldsymbol{U}/|\boldsymbol{\mathcal{A}}|)$.
 1: Construct WMB heuristics for the augmented model.
 2: Initialize $\mathcal{S} \leftarrow \{\emptyset\}$ with the root $\emptyset$.
 3: while within the time budget
 4:     if within the memory budget                // update $\mathcal{S}$, $U^{\mathcal{S}}$, $\mathcal{A}^{\mathcal{S}}$ during search.
 5:         Expand $N_d$ nodes via AOBFS (see Algorithm 3.1) with its upper priority.
 6:     else
 7:         Expand $N_d$ nodes via depth-first search.
 8:     end if
 9:     Draw $N_l$ samples from $q^{\mathcal{S}}_{\text{aug}}$ (see (4.17)).
10:     After drawing each sample:
11:         Update $N$, $\widehat{Z}_{\text{aug}}$, $\text{HM}(\boldsymbol{U})$, $\text{HM}(\boldsymbol{U}/|\boldsymbol{\mathcal{A}}|)$, $\widehat{\text{Var}}$, $\Delta$ via (4.19), (4.20), (4.25), (4.26).
12: end while
```

with the optimal configuration can be removed from memory. Such pruning is particularly useful to prune MAX nodes: for any AND-MAX node with its sub-problem beneath solved, if it holds the highest value among its siblings, all its siblings (solved or not) and their descendants can be pruned immediately. Thus, as pruning proceeds along with search, $\mathcal{A}$ shrinks towards $\{x^\star_{\text{M}}\}$. We use $\mathcal{A}^{\mathcal{S}}$ to denote the remaining MAP space associated with the search tree $\mathcal{S}$.

Note that when we approach the memory limit, we switch the default best-first search to a depth-first search (DFS) that is also compatible with the sampling procedure, and leads to a complete search algorithm with the capability to identify $x^\star_{\text{M}}$ and its value given enough time. By interleaving search and sampling, we derive our mixed dynamic importance sampling (MDIS) algorithm and present it in Algorithm 4.2.

Remarks on Algorithm 4.2

1) K as the number of replicates of the SUM variables controls the exploration-exploitation trade-off. When K is small, we draw a small number of samples for the SUM variables in

each iteration, which allows us to evaluate each sampled MAP configuration fast, however introduces more randomness when assessing the MAP configuration; when K is large, we have more accurate estimate of a MAP configuration being sampled, but also slow down exploration of the MAP space.

2) To predict MAP solutions in an anytime manner, one can simply choose the one with the highest estimated value among those configurations that have been sampled.

Finite-sample Bounds for Marginal MAP

In MDIS, each sample not only comes from a different proposal distribution but also gives importance weights corresponding to a different expectation, which is more complicated than in DIS.

Let $\{x^i_{\text{aug}}\}_{i=1}^N$ be a series of samples drawn via Algorithm 4.2, with $\{\mathcal{S}_i\}$ the corresponding search trees, $\{\widehat{Z}^i_{\text{aug}} = f_{\text{aug}}(x^i_{\text{aug}})/q^{\mathcal{S}_i}_{\text{aug}}(x^i_{\text{aug}})\}_{i=1}^N$ the corresponding importance weights, and $\{U_i = U^{\mathcal{S}_i}_{\text{aug}}\}_{i=1}^N$ the corresponding upper bounds associated with those search trees respectively. We denote $\mathcal{A}_i$ as the MAP space preserved in $\mathcal{S}_i$. Thus,

$$\mathrm{E}\left[\widehat{Z}^i_{\text{aug}}\right] = Z^{\mathcal{A}_i}_{\text{aug}}.$$

That is to say, the importance weights have different (in fact, decreasing) expectations; this differs from the case of DIS where any importance weight has the same expectation (the partition function). We propose an estimate $\widehat{Z}_{\text{aug}}$ whose expectation is again an upper bound of $\pi^K(x^\star_{\text{M}})$ in the following way:

$$\widehat{Z}_{\text{aug}} = \frac{\text{HM}(\boldsymbol{U})}{N} \sum_{i=1}^{N} \frac{\widehat{Z}^i_{\text{aug}}}{U_i}, \tag{4.19}$$

where

$$\mathrm{HM}(\boldsymbol{U}) = \left[\frac{1}{N}\sum_{i=1}^{N}\frac{1}{U_i}\right]^{-1} \tag{4.20}$$

is the harmonic mean of the upper bounds $\{U_i\}_{i=1}^N$. Thus, $\widehat{Z}_{\text{aug}}$ upweights the terms $\widehat{Z}_{\text{aug}}^i$ whose expectations are closer to $\pi^K(x_{\text{M}}^\star)$. The expectation of $\widehat{Z}_{\text{aug}}$ is

$$\mathrm{E}\left[\widehat{Z}_{\text{aug}}\right] = \frac{\mathrm{HM}(\boldsymbol{U})}{N}\sum_{i=1}^{N}\frac{Z_{\text{aug}}^{\mathcal{A}_i}}{U_i}.$$

$\mathrm{E}\left[\widehat{Z}_{\text{aug}}\right]$ is a convex combination of $\{Z_{\text{aug}}^{\mathcal{A}_i}\}_{i=1}^N$ with coefficients $\{\frac{\mathrm{HM}(\boldsymbol{U})}{NU_i}\}_{i=1}^N$, shrinking towards $\pi^K(x_{\text{M}}^\star)$ as search proceeds.

According to (4.18), since $\pi^K(x_{\text{M}}^\star) \leq Z_{\text{aug}}^{\mathcal{A}_i}$, we know

$$\pi^K(x_{\text{M}}^\star) \leq \mathrm{E}\left[\widehat{Z}_{\text{aug}}\right], \tag{4.21}$$

and from $Z_{\text{aug}}^{\mathcal{A}_i} \leq |\mathcal{A}_i|\pi^K(x_{\text{M}}^\star)$, we know

$$\mathrm{E}\left[\widehat{Z}_{\text{aug}}\right] \leq \pi^K(x_{\text{M}}^\star)\frac{\mathrm{HM}(\boldsymbol{U})}{N}\sum_{i=1}^{N}\frac{|\mathcal{A}_i|}{U_i}. \tag{4.22}$$

By combining (4.20), (4.21), and (4.22), we derive two-sided bounds for $\pi^K(x_{\text{M}}^\star)$ involving $\mathrm{E}\left[\widehat{Z}_{\text{aug}}\right]$:

$$\frac{\sum_{i=1}^N 1/U_i}{\sum_{i=1}^N |\mathcal{A}_i|/U_i}\,\mathrm{E}\left[\widehat{Z}_{\text{aug}}\right] \leq \pi^K(x_{\text{M}}^\star) \leq \mathrm{E}\left[\widehat{Z}_{\text{aug}}\right]. \tag{4.23}$$

From the above, we can see that the bounds get tight only when $\mathcal{A}_i$ approaches $\{x_{\text{M}}^\star\}$. To be concise, we re-arrange the L.H.S. of (4.23) to derive:

$$\frac{\mathrm{HM}(\boldsymbol{U}/|\boldsymbol{\mathcal{A}}|)}{\mathrm{HM}(\boldsymbol{U})}\,\mathrm{E}\left[\widehat{Z}_{\mathrm{aug}}\right] \leq \pi^{K}(x^{\star}_{\mathrm{M}}) \leq \mathrm{E}\left[\widehat{Z}_{\mathrm{aug}}\right], \tag{4.24}$$

where

$$\mathrm{HM}(\boldsymbol{U}/|\boldsymbol{\mathcal{A}}|) = \left[\frac{1}{N}\sum_{i=1}^{N}\frac{|\mathcal{A}_i|}{U_i}\right]^{-1} \tag{4.25}$$

is the harmonic mean of $\{U_i/|\mathcal{A}_i|\}_{i=1}^{N}$.

Considering $\widehat{Z}^{i}_{\mathrm{aug}}$ are independent, and $\mathrm{E}\,\widehat{Z}_{\mathrm{aug}}/\,\mathrm{HM}(\boldsymbol{U})$, $\widehat{Z}_{\mathrm{aug}}/\,\mathrm{HM}(\boldsymbol{U})$, $\widehat{Z}^{i}_{\mathrm{aug}}/U_i$ are all within the interval $[0,1]$, we can apply the empirical Bernstein bounds (see Theorem 2.3) to derive finite-sample bounds on $\mathrm{E}\,\widehat{Z}_{\mathrm{aug}}$ and translate those bounds to $\pi(x^{\star}_{\mathrm{M}})$ based on (4.24).

Theorem 4.2. *For any $\delta \in (0,1)$, we define*

$$\Delta = \mathrm{HM}(\boldsymbol{U})\left(\sqrt{\frac{2\widehat{\mathrm{Var}}\log(2/\delta)}{N}} + \frac{7\log(2/\delta)}{3(N-1)}\right), \tag{4.26}$$

where $\widehat{\mathrm{Var}}$ is the unbiased empirical variance of $\{\widehat{Z}^{i}_{aug}/U_i\}_{i=1}^{N}$. Then, the following probabilistic bounds hold for $\pi(x^{\star}_{M})$:

$$\Pr\left[\pi(x^{\star}_{M}) \leq (\widehat{Z}_{aug}+\Delta)^{\frac{1}{K}}\right] \geq 1-\delta,$$

$$\Pr\left[\pi(x^{\star}_{M}) \geq \left(\frac{(\widehat{Z}_{aug}-\Delta)\,\mathrm{HM}(\boldsymbol{U}/|\boldsymbol{\mathcal{A}}|)}{\mathrm{HM}(\boldsymbol{U})}\right)^{\frac{1}{K}}\right] \geq 1-\delta,$$

i.e., $(\widehat{Z}_{aug}+\Delta)^{\frac{1}{K}}$ and $\left(\frac{(\widehat{Z}_{aug}-\Delta)\,\mathrm{HM}(\boldsymbol{U}/|\boldsymbol{\mathcal{A}}|)}{\mathrm{HM}(\boldsymbol{U})}\right)^{\frac{1}{K}}$ are upper and lower bounds of $\pi(x^{\star}_{M})$ with probability at least $(1-\delta)$, respectively.

Note that it is possible that $\widehat{Z}_{\mathrm{aug}} - \Delta < 0$ early on; if so, we may replace $\widehat{Z}_{\mathrm{aug}} - \Delta$ with any non-trivial lower bound of Z_{aug}. In the experiments, we use $\delta\widehat{Z}_{\text{aug}}$, a $(1-\delta)$ probabilistic bound by the Markov's inequality (see (2.19)). We can replace $\widehat{Z}_{\text{aug}} + \Delta$ with the current deterministic upper bound if the latter is tighter.

4.2.3 Experiments

We evaluate our proposed approach (MDIS) against two baseline methods on five benchmarks. The baselines include UBFS (see Algorithm 3.4), a unified best-first search algorithm that emphasizes rapidly tightening the upper bound, and AAOBF [Marinescu et al., 2017], a best-first/depth-first hybrid search algorithm that balances upper bound quality with generating and evaluating potential solutions. These two are state-of-the-art algorithms for anytime upper and lower bounds respectively. We do not compare to XOR_MMAP [Xue et al., 2016] and AFSE [Mauá and de Campos, 2012] due to their limitations to relatively easy problem instances as shown in Section 3.2.

Three benchmarks are formed by problem instances from recent UAI competitions: `grid`, 50 grid networks with size no smaller than 25 by 25; `promedas`, 50 medical diagnosis expert systems; `protein`, 44 instances made from the "small" protein side-chains of [Yanover and Weiss, 2002]. Since the original UAI instances are pure MAP tasks, we generate MMAP instances by randomly selecting 10% of the variables as MAP variables. The fourth benchmark is `planning`, formed by 15 instances from probabilistic conformant planning with a finite-time horizon [Lee et al., 2016a]. On these four benchmarks, we compare anytime bounds. Some statistics of the four benchmarks are shown in Table 4.2. These benchmarks are selected to illustrate different problem characteristics; for example, `protein` instances are relatively small but high cardinality, while `planning` instances have more variables and higher induced width, but lower cardinality. The fifth benchmark, which we will describe in detail later, is created from an image denoising model in order to evaluate quality of the predicted MAP solutions.

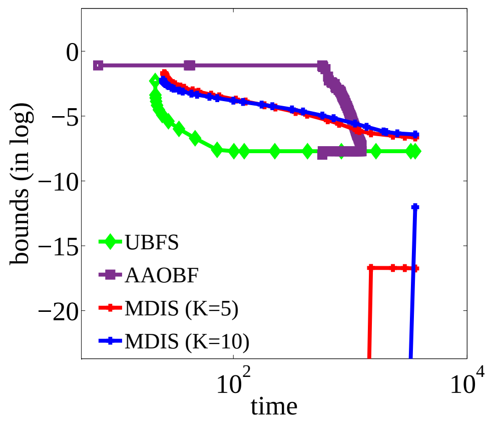
Table 4.2: Statistics of the four evaluated benchmarks. The first three benchmarks are formed by problem instances from recent UAI competitions, where 10% of variables are randomly selected as MAX variables. "avg. ind. width of sum" in the last row stands for the average induced width of the internal summation problems.

	`grid`	`promedas`	`protein`	`planning`
# instances	50	50	44	15
avg. # variables	1248.20	982.10	109.55	1122.33
avg. % of MAX vars	10%	10%	10%	12%
avg. # of factors	1248.20	994.76	394.64	1127.67
avg. max domain size	2.00	2.00	81.00	3.00
avg. max scope	3.00	3.00	2.00	5.00
avg. induced width	124.82	108.14	15.84	165.00
avg. pseudo tree depth	228.92	158.78	33.52	799.33
avg. ind. width of sum	43.44	40.32	10.20	49.67

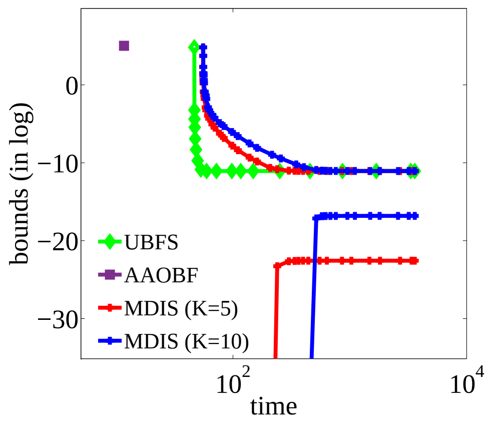
The time budget is set to 1 hour for the experiments on the first four benchmarks. We allot 4GB memory to all algorithms, with 1GB extra memory to AAOBF for caching. For our experiments, we use the WMB heuristics; whose memory usage is roughly controlled by the *ibound*. For a given memory budget, we first compute the largest *ibound* that fits in memory, then use the remaining memory for search. Since all the competing algorithms use weighted mini-bucket heuristics, the same *ibound* is shared during heuristic construction. We set $N_d = 100$ and $N_l = 1$ (see Algorithm 4.2) as suggested by the experimental results in Section 4.1. We set $\delta = 0.025$. All implementations are in C/C++ courtesy of the original authors.

Anytime Bounds for Individual Instances

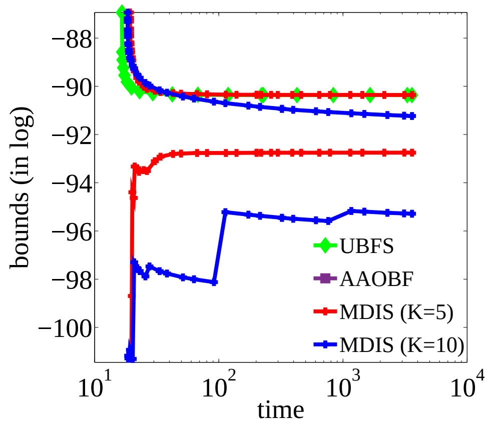
Figure 4.6 shows the anytime behavior of all the methods on instances from four benchmarks. In terms of lower bounds, our approach can always provide decent lower bounds even when the internal summation problems are quite challenging, while AAOBF may not work well since it relies on exact evaluation of those internal summation problems, e.g., on those shown in Figure 4.6(b)-4.6(d). When the internal summation problems are relatively easy, their



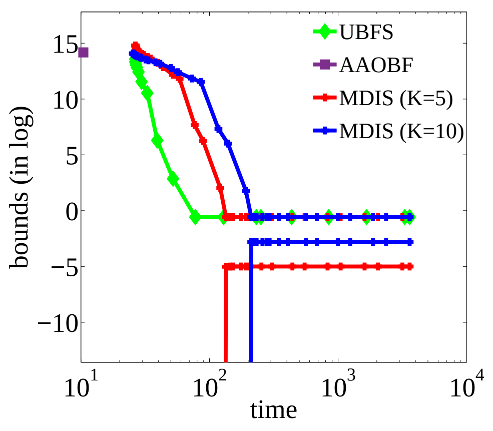
(a) `grid`/90-25-5



(b) `promedas`/or_chain_48.fg



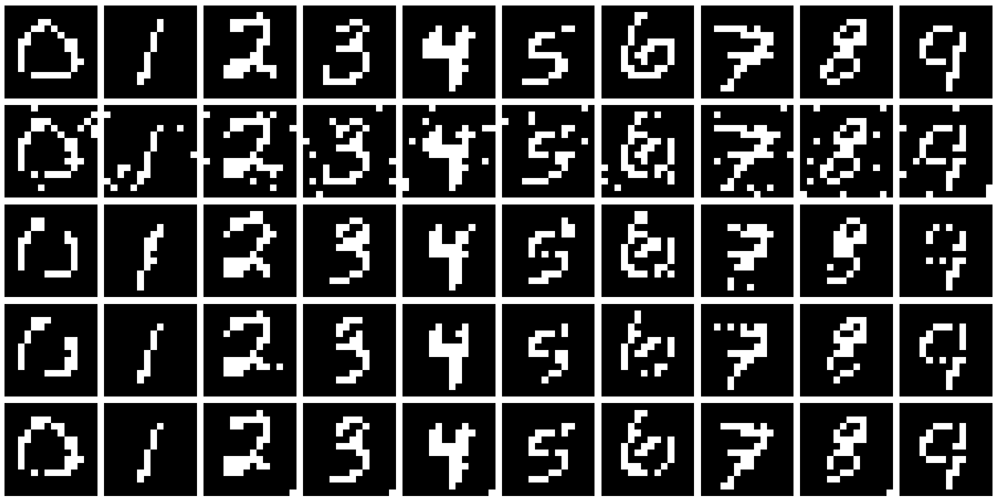
(c) `protein`/1skz

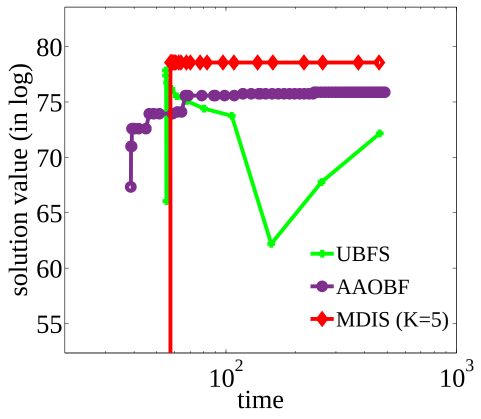


(d) `planning`/bw2_2_4_4

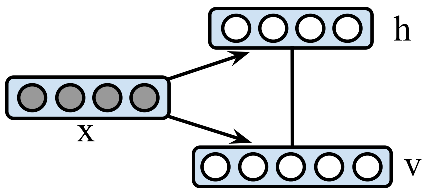
Figure 4.6: Anytime bounds for MMAP on instances from four benchmarks. The max domain sizes of those instances from (a)-(d) are 2, 2, 81, 3 respectively, and the induced widths of the internal summation problems are 25, 28, 8, 24 respectively. Curves for some bounds may be (partially) missing because they are not in a reasonable scope. UBFS only provides upper bounds. The time limit is 1 hour.

exact evaluation is cheap; thus AAOBF might perform better than ours. Figure 4.6(a) gives a typical example. In terms of upper bounds, our bound quality is often eventually comparable to UBFS, e.g., Figure 4.6(b)-4.6(d). UBFS typically performs better than MDIS early on, while MDIS quickly catches up and becomes comparable. Improvement in AAOBF on upper bounds also requires fast exact evaluation of the internal summation problems, which might not be possible in many cases. So, AAOBF is usually not as competitive as the other two methods on upper bounds.


(a)



(b)



(c) CRBM

Figure 4.7: (a) Image denoising results for one instance per digit. The first row is for the ground truth images. The second row is for the noisy inputs created from the ground truth by randomly flipping 5% pixels. Below the first two rows are denoised images from UBFS, AAOBF, MDIS (K=5) respectively. (b) An example on MAP solution quality comparison. (c) Illustration of the conditional restricted Boltzmann machine (CRBM) model used for the image denoising task. When conditioned on an input "X", this model has a bipartite graph structure between hidden units "h" (SUM variables) and visible units "v" (MAX variables).

Anytime Bounds across Benchmarks

We present the anytime performance across the four benchmarks in Table 4.3 and 4.4 where we compare anytime bounds at three different timestamps: 1 minute, 10 minutes and 1 hour. From Table 4.3, we can observe that MDIS with K=5 is dominant at any of these timestamp/benchmark combinations for lower bounds. MDIS with K=10 performs less well,

Table 4.3: Number of instances that an algorithm achieves the best *lower bounds* at each timestamp (1 min, 10 min, and 1 hour) for each benchmark. The best for each setting is bolded. Entries for UBFS are blank because UBFS does not provide lower bounds.

	`grid`	`promedas`	`protein`	`planning`
# instances	50	50	44	15
		Timestamp: 1min/10min/1hr		
MDIS (K=5)	**47/44/45**	**32/34/31**	**31/27/28**	**14/13/13**
MDIS (K=10)	3/2/1	4/5/6	11/13/14	1/2/2
UBFS	-/-/-	-/-/-	-/-/-	-/-/-
AAOBF	0/4/4	16/21/24	2/4/4	0/0/0

Table 4.4: Number of instances that an algorithm achieves the best *upper bounds* at each timestamp (1 min, 10 min, and 1 hour) for each benchmark. The best for each setting is bolded.

	`grid`	`promedas`	`protein`	`planning`
# instances	50	50	44	15
		Timestamp: 1min/10min/1hr		
MDIS (K=5)	0/0/0	9/12/13	5/9/15	1/1/1
MDIS (K=10)	0/0/0	10/13/14	9/10/13	1/2/3
UBFS	**50/50/50**	**50/50/50**	**36/32/26**	**14/14/13**
AAOBF	0/0/1	2/4/6	2/2/2	1/1/1

perhaps because it requires more time to draw one full sample compared to when K=5, leading the empirical Bernstein lower bounds to kick in relatively late; this phenomenon can be also observed in all the plots in Figure 4.6. UBFS provides the best upper bounds as shown in Table 4.4. However, our algorithm generally performs better than AAOBF in terms of upper bounds.

Empirical Evaluation of Solution Quality

To evaluate the MAP solution quality predicted by our algorithm, we create an image denoising task from the MNIST database[3] of handwritten digits [LeCun et al., 1998]. We binarize each image, resize it to 14 by 14, and then randomly flip 5% of the pixels to generate a corrupt one. We train a conditional restricted Boltzmann machine (CRBM) [Mnih et al., 2011] model with 64 hidden units and 196 visible units using mixed-product BP [Ping and Ihler, 2017, Liu and Ihler, 2013] for the denoising task. The resulting graphical model thus has 64 SUM variables and 196 MAX variables. Figure 4.7(c) gives an illustration of this model. The advantage of this model is that we can easily evaluate any MAP configuration since the internal summation problem only contains singleton potentials; thus this model favors AAOBF since AAOBF is able to evaluate MAP configurations at a very low cost. We set K to 5 and runtime to 10 minutes for convenience. We test on 100 images with 10 images per digit. Figure 4.7(a) compares the denoising results among all the algorithms for one instance per digit. Figure 4.7(b) gives an example of the quality of the predicted MAP solutions of our algorithm. In general, the quality of predicted MAP solutions for our algorithm are better than the other two baselines in 51 of 100 instances, which is generally as good as AAOBF (47/100) despite the model being well-suited to AAOBF. A possible reason is that our algorithm is able to traverse the MAP space very quickly and get cheap stochastic estimates of the most promising MAP solutions.

4.3 Conclusion

In this chapter we developed a set of fast, anytime and anyspace probabilistic bounds based on importance sampling, by leveraging both AND/OR search and variationally optimized bounds, for the partition function and marginal MAP objectives. Our dynamic importance

[3]`http://yann.lecun.com/exdb/mnist/`

sampling algorithm follows an AOBFS search process that improves the proposal distribution over time, while our choice of weighted average for the importance weights gives the resulting estimator rapidly tightening finite-sample bounds. This line of work also opens up several avenues for future research, including investigating different weighting schemes for the samples, more flexible balances between search and sampling (for example, changing over time), and more closely integrating the variational optimization process into the anytime behavior.

We then leveraged the dynamic importance sampling procedure to develop anytime finite-sample upper and lower bounds for marginal MAP, giving a technique for this query that enjoys the merits of both heuristic search and importance sampling. Our approach is particularly useful for problem instances whose internal summation problems are challenging. It predicts high-quality MAP solutions along with their estimated values and runs in an anytime and anyspace manner, allowing the user to take advantage of flexible trade-offs between memory, time, and solution quality.

Chapter 5

Interleaving Variational Optimization with Sampling

5.1 Introduction

In the preceding chapter, we described an approach that first built a pre-compiled variational bound, optimizing it with cost shifting or message passing, before using this bound as the basis for subsequent search (as a heuristic function) and sampling (as a proposal). While this proved quite effective, particularly over long time periods (minutes to hours), it has the drawback that, early in the process, only the variational bound is available, which does not give the best possible bounds at faster time scales. The best amount of effort to put into constructing this initial bound depends on the problem: failing to build a high-quality variational bound can dramatically slow down the subsequent progress of search or sampling on more difficult problems, but for easier problems that would be solvable with a less optimized heuristic or proposal, solution time is dominated by this initial build time. Unfortunately it is difficult to know beforehand into which regime a particular problem instance will fall.

Our Contributions

In this chapter, we propose a general framework that interleaves optimization of TRW [Wainwright et al., 2005] and WMB variational bounds (via message passing) with Monte Carlo importance sampling, giving faster initial bound improvement without sacrificing long-term performance. We also propose an adaptive interleaving policy that can automatically balance the computational effort between these two schemes in an instance-dependent way, which provides our framework with the strengths of both schemes, leads to tighter anytime bounds and an unbiased estimate of the partition function, and allows flexible trade-offs between memory, time, and solution quality. Our experiments on real-world problems demonstrate that our interleaving framework with the adaptive policy is superior to several non-interleaving baselines and interleaving baselines with simple static policies in terms of anytime performance, gives competitive final bound quality, and is relatively insensitive to its hyperparameter, making it easy to use and automate in practice.

5.2 Main Algorithm

In this section, we present a general scheme that interleaves optimization of the variational upper bound with importance sampling, derive finite-sample bounds for this scheme, and discuss interleaving policies.

5.2.1 A General Interleaving Framework

Our general scheme is quite simple: we interleave the two processes according to some policy. Since optimization of variational bounds is typically through a message passing procedure, we consider interleaving message passing with importance sampling in our algorithm. Procedurally, we first build up an initial bound within a given memory budget, and then interleave

Algorithm 5.1 A General Interleaving Framework

```
Require: memory budget, time budget, confidence parameter $\delta$, interleaving policy $P$.
Ensure: $N$, $U$, $\widehat{Z}$, $\mathrm{HM}(\boldsymbol{U})$, $\Delta$, $\widehat{\mathrm{Var}}$.
 1: Build an initial variational upper bound that fits the memory budget.
 2: while within the time budget
 3:     Generate $(R, S)$ via policy $P$. // R steps of message passing and S steps of sampling
 4:     for $r \leftarrow 1$ to $R$
 5:         Run one round of message passing.
 6:         Update $U$.
 7:     end for
 8:     for $s \leftarrow 1$ to $S$
 9:         Draw one sample from the current proposal.
10:         Update $N$, $\widehat{Z}$, $\mathrm{HM}(\boldsymbol{U})$, $\Delta$, $\widehat{\mathrm{Var}}$ via (5.1), (5.2), (5.4)
11:         and (5.5) respectively.
12:     end for
13:     Update policy $P$ if necessary.
14: end while
```

the two processes following a policy that we will discuss in the sequel. Algorithm 5.1 presents details of our framework.

Our framework actually serves as a "meta-algorithm" from which any optimizable proposal with properties in (4.4) and (4.5) can benefit; in particular this includes convex variational bounds such as WMB and TRW (see Liu et al. [2015a] for more details on why this holds for TRW).

Interleaving these two processes leads to rapid early bound improvement, since the improvement provided by the probabilistic bounds is not delayed until after variational optimization converges or is otherwise terminated (e.g., in WMB-IS and DIS); this results in significant improvements in anytime behavior as we demonstrate in the empirical evaluation. A critical point is that, by interleaving updates to the proposal, our samples are no longer identically distributed; in fact, their variance is decreasing as the proposal improves, and for best results we should up-weight these more accurate samples in our estimates. To this end, and analogous to that of DIS, we define a weighted average estimator $\hat{Z}$ and apply the empirical Bernstein bounds (see Theorem 2.3) to derive finite-sample bounds on the error between Z and $\hat{Z}$ based on independent samples from a sequence of proposals satisfying (4.4) and (4.5):

Theorem 5.1. *Let $\{x^i\}_{i=1}^N$ be a series of samples drawn from proposal distributions $\{q_i(x)\}_{i=1}^N$ respectively via Algorithm 5.1, with $\{\widehat{Z}_i = f(x^i)/q_i(x^i)\}_{i=1}^N$ the corresponding importance weights, and $\{U_i\}_{i=1}^N$ the corresponding variational upper bounds respectively. Let*

$$\widehat{Z} = \frac{\mathrm{HM}(\boldsymbol{U})}{N} \sum_{i=1}^{N} \frac{\widehat{Z}_i}{U_i}, \tag{5.1}$$

where

$$\mathrm{HM}(\boldsymbol{U}) = \left[\frac{1}{N} \sum_{i=1}^{N} \frac{1}{U_i} \right]^{-1} \tag{5.2}$$

is the harmonic mean of $\{U_i\}_{i=1}^N$. then, $\widehat{Z}$ is an unbiased estimator of Z, i.e.,

$$\mathrm{E}[\,\widehat{Z}\,] = Z. \tag{5.3}$$

Define a deviation term

$$\Delta = \mathrm{HM}(\boldsymbol{U}) \left(\sqrt{\frac{2\widehat{\mathrm{Var}} \log(2/\delta)}{N}} + \frac{7 \log(2/\delta)}{3(N-1)} \right), \tag{5.4}$$

where

$$\widehat{\mathrm{Var}} = \frac{1}{N-1} \sum_{i=1}^{N} \left(\frac{\widehat{Z}_i}{U_i} - \frac{\widehat{Z}}{\mathrm{HM}(\boldsymbol{U})} \right)^2 \tag{5.5}$$

is the unbiased empirical variance (see (2.20)) of $\{\widehat{Z}_i/U_i\}_{i=1}^N$. Since the boundedness of each

$\widehat{Z}_i$ guarantees that $\widehat{Z}_i/U_i \le 1$ for all i, applying standard empirical Bernstein results we have

$$\begin{aligned} \Pr[Z \le \widehat{Z} + \Delta] &\ge 1 - \delta, \\ \Pr[Z \ge \widehat{Z} - \Delta] &\ge 1 - \delta, \end{aligned} \tag{5.6}$$

i.e., $\widehat{Z} + \Delta$ and $\widehat{Z} - \Delta$ are upper and lower bounds of Z with probability at least $(1-\delta)$, respectively.

5.2.2 Interleaving Policies

The key for our framework to work well is its interleaving policy. Our goal is to design a policy balancing the effort between message passing and sampling, so that the probabilistic bounds in (5.6) can improve as quickly as possible. Algorithm 5.1 controls this balance by alternating between R steps of message passing, and S steps of sampling, within each iteration.

Optimize-first Policy

In most prior work, the variational bound is optimized first, as a separate pre-processing step. Within the framework of Algorithm 5.1, this takes the form of setting $(R, S) = (1, 0)$ (all message passing) until some convergence or time-out criteria are satisfied, then changing the policy P so that $(R, S) = (0, 1)$ (all sampling). A simple strategy is to switch over after some fixed time. As noted previously, this approach suffers from several drawbacks. The deterministic variational bounds are considerably weaker than those provided by sampling, and the early work serves mainly to improve the quality of later sampling, so that quality does not improve in a smooth, anytime way. A related point is that this makes it difficult to know how much effort to put into the optimization process; too little, and the probabilistic bounds will improve only slowly; too much, and we waste time that could have been used for sampling.

Static Policy

Alternatively, we can interleave updates and samples using a simple static policy, fixing (R, S) in Algorithm 5.1 to some constants. While this choice will alternate between update types, giving smoother performance, it is also non-trivial to automate for different types of problems. In particular, it is difficult to know how any given (R, S) will perform a priori; the characteristics of the problem instance may make message updates more or less expensive and more or less effective, changing the desired balance between R and S.

Adaptive Policy

The main issue with static policies is that they are "blind" to the current status and behavior of the inference process on a particular problem instance. We propose an adaptive policy that is able to adjust its behavior based on information available on the fly. The basic idea of our adaptive policy is to select the action that is projected to have larger unit contribution to improving the probabilistic upper bound in each iteration. Details follow.

Suppose we have already drawn N samples so far. The deviation term Δ in (5.4) can be roughly approximated by Δ':

$$\Delta \approx \Delta' = \mathrm{HM}(\boldsymbol{U}) \left(\sqrt{\frac{\log(2/\delta)}{2N}} + \frac{7\log(2/\delta)}{3(N-1)} \right). \tag{5.7}$$

Δ' is derived from Δ by substituting the empirical variance $\widehat{\mathrm{Var}}$ (as defined in (5.5)) with 1/4, which is an upper bound of the empirical variance in expectation according to Popoviciu's inequality [Popoviciu, 1935]. This means that Δ' is actually an upper bound of the expectation of Δ thanks to the concavity of the square root function. Therefore, the probabilistic upper bound $\widehat{Z} + \Delta$ in (5.6) can be approximated by $Z + \Delta'$ since $\widehat{Z}$ is an unbiased estimate of Z.

Thus, we define a gain function that approximates the improvement in the probabilistic upper bound that we expect after drawing N' more samples, parameterized with respect to the value of the variational upper bound:

$$\begin{aligned} gain(N, U, N') &= (Z + \Delta') - (Z + \Delta'_{N'}) \\ &= \Delta' - \Delta'_{N'} \end{aligned}$$

where

$$\Delta'_{N'} = \left[\frac{1}{N+N'} (\frac{N'}{U} + \sum_{i=1}^{N} \frac{1}{U_i}) \right]^{-1} \left(\sqrt{\frac{\log(2/\delta)}{2(N+N')}} + \frac{7\log(2/\delta)}{3(N+N'-1)} \right).$$

Assuming that one sampling step takes time t_{is}, we can easily define the unit gain $gain_{is}$ of drawing one sample:

$$gain_{is} = gain(N, U, 1)/t_{is} \tag{5.8}$$

However, to define the gain of running one message passing step is more complicated: message passing does not directly contribute to the current probabilistic bound, but rather, affects the deterministic upper bound and hence the quality of all later samples. We also do not know precisely how much the bound will improve by executing a single message passing step. To address these issues, we first predict the improved value of the deterministic bound based on the improvements from past message updates (in our experiments, we use a simple linear interpolation). Denoting this improved bound value by U', we estimate the resulting improvement in the probabilistic bound after N_s more samples are drawn using this improved proposal. Then, the unit gain $gain_{msg}$ of one message passing step is:

$$gain_{msg} = gain(N, U', N_s)/t_{msg} \tag{5.9}$$

where t_{msg} is the time it takes to complete one message passing step.

Finally, our adaptive policy compares $gain_{is}$ and $gain_{msg}$ and takes the action with larger unit gain in the following step. Note that neither $gain_{is}$ nor $gain_{msg}$ involves information from the samples themselves, which ensures that their independence is preserved and that Theorem 5.1 still applies.

5.3 Empirical Evaluation

In this section, we present empirical results to demonstrate the usefulness of our framework and the effectiveness of our adaptive policy.

We evaluated on three benchmarks of real-world problem instances from recent UAI competitions. Our benchmarks include: `pedigree`, 22 genetic linkage instances from the UAI'08 inference challenge; `protein`, 50 instances made from the "small" protein side-chains of [Yanover and Weiss, 2002]; `promedas`, 50 medical diagnosis expert systems [Wemmenhove et al., 2007]. These three sets are selected to illustrate different problem characteristics. Table 5.1 shows some summary statistics of these benchmarks.

We adopted WMB as our variational bound, and allocated a maximum of 512MB memory, using the largest *ibound* that fit our memory budget. We set a maximum time budget of 600 seconds. The confidence parameter δ for our probabilistic bounds is set to 0.025. In the experiments, we also used $\widehat{Z}\delta$, a $(1-\delta)$ probabilistic lower bound by the Markov's inequality (see (2.19)), and switched to our lower bound $\widehat{Z} - \Delta$ when it becomes non-trivial. We also replaced $\widehat{Z} + \Delta$ with the best deterministic upper bound reached so far if the latter is tighter.

Table 5.2 explains the evaluated algorithms, all of which share the same initial WMB structure. We test our adaptive approach against several non-interleaved strategies ("fixed-X") as well as statically interleaved strategies ("static-X"). The "equal time" strategy corresponds to "static $(1, t_{msg}/t_{is})$" instead of "static $(t_{is}/t_{msg}, 1)$" because message passing is usually much

Table 5.1: Statistics of the three evaluated benchmark sets.

	`pedigree`	`protein`	`promedas`
# instances	22	50	50
avg. # variable	917.14	99.96	682.12
avg. # of factor	917.14	355.84	682.12
avg. max domain size	4.95	77.94	2.00
avg. max scope	4.45	2.00	3.00
avg. induced width	25.50	11.24	25.76

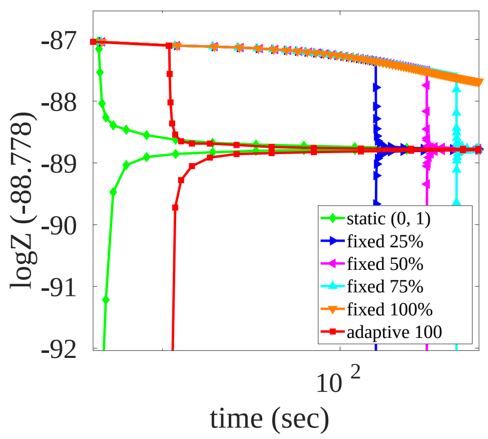
Table 5.2: Notations and abbreviations used in figures and tables for the evaluated algorithms.

fixed $p\%$	optimize-first policy with the first $p\%$ of time for message passing, and the rest for sampling.
static (R, S)	static policy with some given (R, S).
equal time	static policy with $(1, t_{msg}/t_{is})$.
adaptive N_s	adaptive policy with N_s pseudo samples.

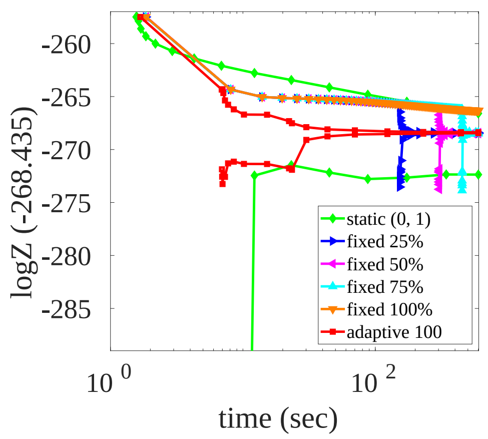
more expensive than sampling (typically, $t_{msg}/t_{is} > 10^3$). Note that "static (0, 1)" can also be viewed as an optimize-first (non-interleaving) strategy "fixed 0%", because it does not spend any time improving the variational bound after the initial bound construction. During one round of message passing, i.e., one forward-backward pass as defined in Liu and Ihler [2011], we do cost-shifting and weight optimization simultaneously. All implementations are in C/C++.

5.3.1 Interleaving versus Non-interleaving

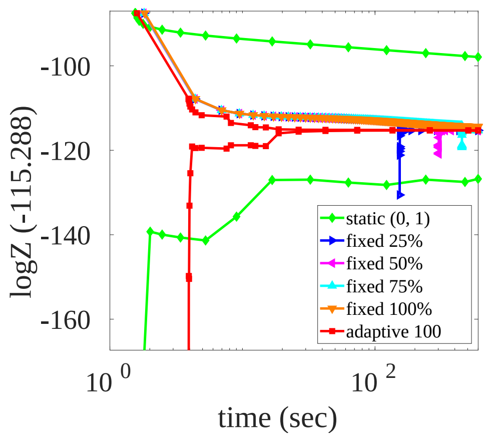
Figure 5.1 shows anytime bounds from some typical instances of each benchmark for interleaving and non-interleaving strategies. We can observe from Figure 5.1 that those non-interleaving strategies except "static (0, 1)" lack good anytime behavior compared to our adaptive interleaving strategy: they are unable to compute a lower bound until they quit message passing and start sampling; their early upper bounds correspond to the deterministic variational bounds, which typically do not improve as fast as the probabilistic upper bound



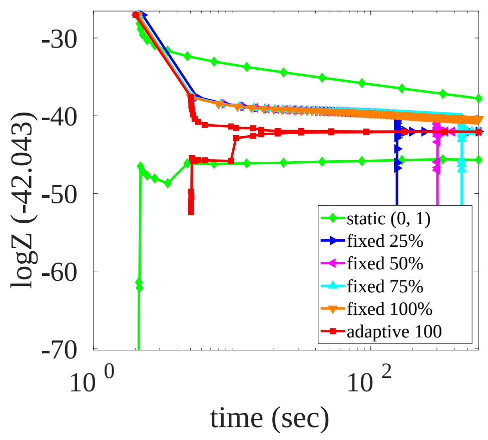
(a) **pedigree**/pedigree23



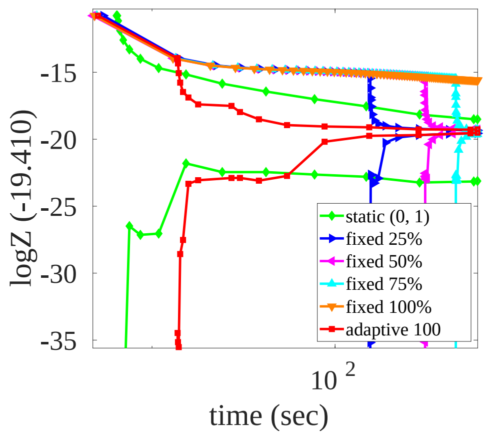
(b) **pedigree**/pedigree37



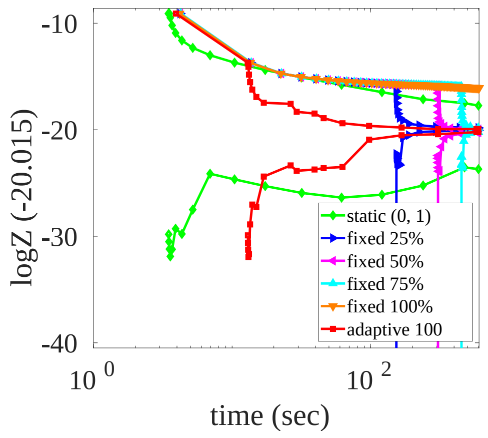
(c) **protein**/1qsq



(d) **protein**/1whi

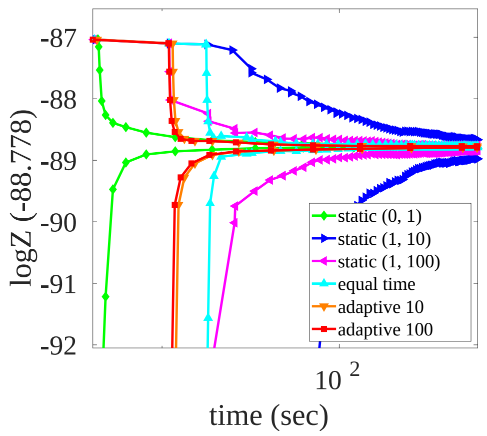


(e) **promedas**/or_chain_132.fg

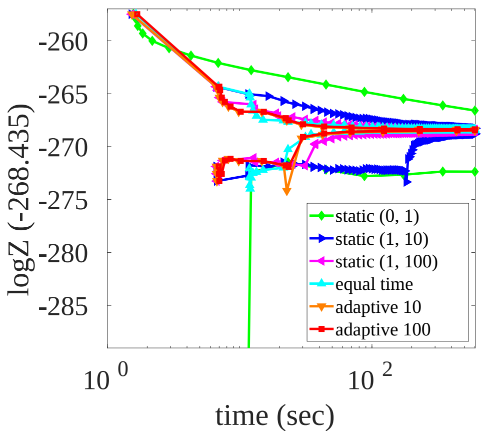


(f) **promedas**/or_chain_153.fg

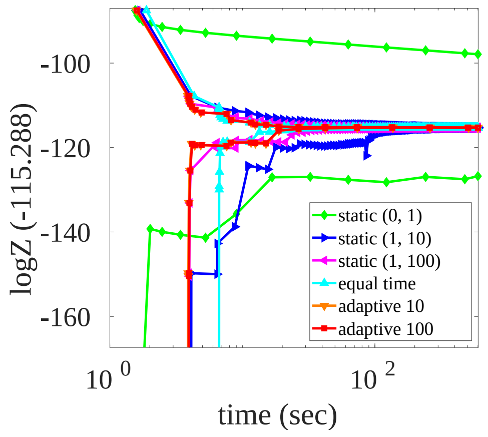
Figure 5.1: Anytime bounds on $\log Z$ for two instances per benchmark, comparing our adaptive interleaving strategy with non-interleaved strategies, spending various fractions of the time (0% through 100%) optimizing the deterministic bound first (see Table 5.2; "static(0,1)" is equivalent to "fixed 0%"). The adaptive strategy strikes a balance between responsiveness (giving tighter bounds early) and long-term performance (tight bounds later).



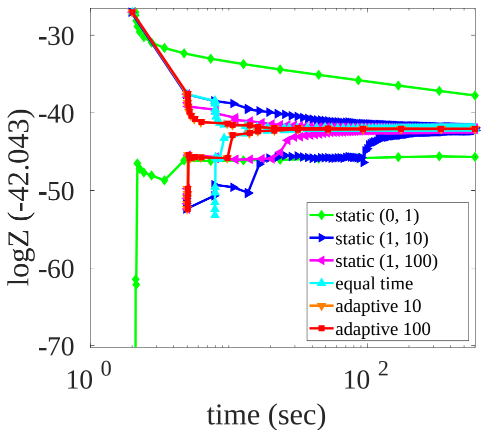
(a) `pedigree`/pedigree23



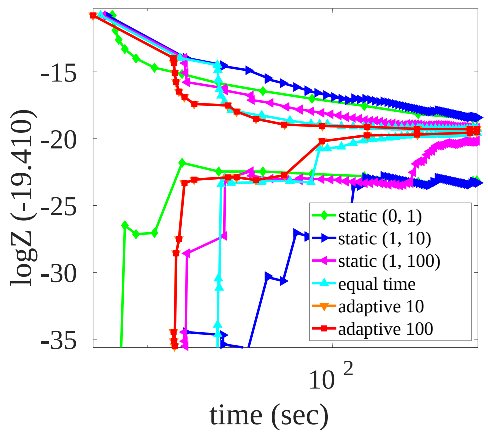
(b) `pedigree`/pedigree37



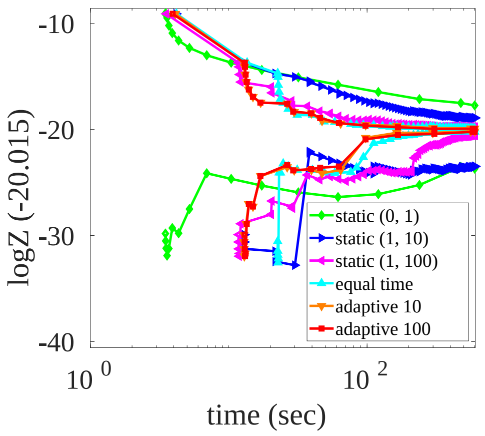
(c) `protein`/1qsq



(d) `protein`/1whi



(e) `promedas`/or_chain_132.fg



(f) `promedas`/or_chain_153.fg

Figure 5.2: Anytime bounds on $\log Z$ for two instances per benchmark, comparing our adaptive strategy with statically interleaved strategies balancing message updates and sampling (see Table 5.2). The best fixed strategy is typically "equal time", but adaptivity is usually slightly better, as it is able to change its behavior dynamically based on the observed improvement in bounds from message updates.

of our adaptive variant. "static (0, 1)" responds quickly but often gives looser results at longer time scales; see Figure 5.1(c) and 5.1(f) for example. It performs very well when the initial bound is already close to the ground truth (see Figure 5.1(a)), but such information is usually not available to users beforehand.

To quantify the anytime performance of the methods in each benchmark, we use two measures: one is the area between the upper bound of $\log Z$ and the (estimated) ground truth of $\log Z$; the other is the area between the upper and lower bound of $\log Z$ as introduced in Section 4.1.7. The first facilitates comparison with those methods that do not provide lower bounds early on. These quantities are computed for each instance and method, and then normalized by those of "static (0, 1)". Finally, we take the (geometric) mean over these scores across each benchmark.

From Table 5.3, we observe that our adaptive policy performs significantly better in terms of anytime upper bound than any of the non-interleaving variants across all the benchmarks. We can also see differences in performance stemming from the different problem characteristics in the benchmarks: for example, the performance of the non-interleaved strategies degrades with higher time used for message passing on the `pedigree` benchmark, while this is less true of the other two benchmarks; this indicates the difficulty of deciding when to quit message passing and start sampling for good anytime behavior, especially when we do not know the time limit in advance. In contrast, our interleaving framework does not have such limitations.

We also examine performance at a fixed time limit, as opposed to anytime behavior. Table 5.5 shows the mean final gap between the upper bound and the (estimated) $\log Z$. We can see that our adaptive variants perform almost as well as the best method for each benchmark, which implies that although our algorithm is designed for a more responsive, anytime behavior, it does not sacrifice much in terms of long-term bound quality.

Table 5.3: Mean area between upper bounds and (estimated) ground truth log$Z$, normalized by that of "static (0, 1)", for each benchmark. Smaller numbers indicate better anytime upper bounds. The best for each benchmark is bolded.

	static (0,1)	static (1,10)	static (1,100)	fixed 25%	fixed 50%
`pedigree`	1	2.436	1.262	2.084	3.027
`protein`	1	0.145	0.077	0.161	0.231
`promedas`	1	1.217	0.615	0.898	1.408
	fixed 75%	**fixed 100%**	**equal time**	**adaptive 10**	**adaptive 100**
`pedigree`	3.904	4.655	0.947	**0.781**	0.786
`protein`	0.285	0.329	0.054	0.051	**0.050**
`promedas`	1.855	2.268	0.414	0.354	**0.349**

Table 5.4: Mean area between upper and lower bounds of log$Z$, normalized by that of "static (0, 1)", for each benchmark. Entries for some non-interleaved strategies are missing because they do not give lower bounds early on. Smaller numbers indicate better anytime bounds. The best for each benchmark is bolded.

	static (0,1)	static (1,10)	static (1,100)	fixed 25%	fixed 50%
`pedigree`	**1**	4.624	2.213	-	-
`protein`	1	0.299	0.118	-	-
`promedas`	1	2.674	1.203	-	-
	fixed 75%	**fixed 100%**	**equal time**	**adaptive 10**	**adaptive 100**
`pedigree`	-	-	2.061	1.544	1.529
`protein`	-	-	0.104	0.080	**0.078**
`promedas`	-	-	1.271	0.728	**0.726**

5.3.2 Adaptive versus Static

We next compare our adaptive interleaving strategy with several statically interleaved approaches. Figure 5.2 shows anytime bounds of two instances per benchmark for various interleaving settings. From Figure 5.2, we observe that in general, the "equal time" variant performs better than a static choice of interleaving rate, since it is able to take into account how computationally expensive the message updates are. Even so, both of our adaptive variants (corresponding to a shorter or longer horizon when estimating the impact of a

Table 5.5: Mean final gap between upper bound and (estimated) ground truth log$Z$, normalized by that of "static (0, 1)", for each benchmark. Smaller numbers are better; the best method for each benchmark is bolded.

	static (0,1)	static (1,10)	static (1,100)	fixed 25%	fixed 50%
`pedigree`	1	2.113	0.804	**0.468**	0.480
`protein`	1	0.045	0.016	**0.004**	**0.004**
`promedas`	1	0.879	0.308	**0.121**	0.133

	fixed 75%	fixed 100%	equal time	adaptive 10	adaptive 100
`pedigree`	0.581	6.576	0.534	0.554	0.572
`protein`	0.005	0.208	0.005	0.006	0.005
`promedas`	0.185	2.519	0.165	0.150	0.148

message update) perform better most of the time; they are able to make more informed decisions about the current benefits of sampling versus message updates. These observations are also supported by the statistical results in Table 5.3 and Table 5.4. From Figure 5.2, Table 5.3, and Table 5.4, we can also see that the two adaptive variants perform fairly similarly. In fact, we found in our experiments that our adaptive policy works reasonably well within a wide range of N_s, i.e., it is not very sensitive to the hyperparameter value, which simplifies its use in practice.

5.4 Conclusion

In this chapter, we propose a general framework that interleaves optimization of a variational upper bound with importance sampling based on its associated proposal, to obtain high-quality anytime bounds and estimates of the partition function. This framework can be viewed as a meta-algorithm for convex variational bounds such as TRW and WMB, giving them more responsive bounds without sacrificing long-term quality. Our proposed adaptive policy, which selects the action with larger unit gain for improving the probabilistic upper bound at each iteration, leads to excellent empirical anytime performance, both in comparison to simple non-interleaved baselines as well as simpler interleaved policies within our framework. Our approach is easy to use in practice since it does not appear to be sensitive to its hyperparameter in our experiments. For future work, one may consider incorporating importance sampling in the initial bound construction as well, to further boost the anytime performance.

Chapter 6

Conclusion and Future Directions

In this dissertation, we studied inference problems that arise from reasoning over graphical models. By leveraging ideas and techniques from three major inference paradigms, i.e., variational methods, heuristic search, and Monte Carlo sampling, we developed a series of anytime anyspace inference algorithms for the partition function and marginal MAP with deterministic or probabilistic bound guarantees.

6.1 Our Contributions

In Chapter 3, we proposed anytime anyspace best-first search algorithms for bounding the partition function and marginal MAP respectively, taking advantage of the AND/OR tree structure and optimized variational heuristics to tighten deterministic bounds. In particular, we first presented our AOBFS algorithm for the partition function, which can improve on state-of-the-art variational bounds in an anytime way within limited memory resources. We then generalized this idea to marginal MAP by introducing UBFS that unifies max and sum inference within a specifically designed priority system, that aims to reduce upper bound of the optimal solution(s) as quickly as possible. Its effectiveness is also confirmed by other researchers [Marinescu et al., 2018].

In Chapter 4, we developed approximate inference algorithms that integrate importance sampling, best-first search, and variational upper bounds, that provide anytime finite-sample bounds for the partition function and marginal MAP respectively. Specially, for the partition function, we proposed DIS that interleaves importance sampling with best-first search, that in practice enjoys both the rapid bound improvement characteristic of sampling while also benefiting significantly from search on problems where search is relatively effective, or when given enough computational resources, even when these points are not known in advance. We then generalized DIS to MDIS which gives anytime finite-sample bounds for marginal MAP with predicted MAP solutions. MDIS first converts bounding marginal MAP to a surrogate task of bounding a series of summation problems of an augmented graphical model, and then adapt DIS to provide finite-sample bounds for the surrogate task. Given enough time, those bounds are guaranteed to be tight, and the values of the predicted MAP solutions are also guaranteed to converge to the optimum. MDIS shares the same anyspace property as DIS.

In Chapter 5, we presented a general inference framework that interleaves optimization of variational upper bounds with importance sampling. Our adaptive interleaving policy can automatically balance the computational effort between these two schemes in a problem-dependent way, which equips our framework with the strengths of both schemes, leads to tighter anytime finite-sample bounds and an unbiased estimate of the partition function, and allows flexible trade-offs between memory, time, and solution quality.

6.2 Future Directions

This dissertation opens up several directions for future research. An important angle one can think of is how to further explore ideas/techniques from the three major inference paradigms and integrate them in a way that new algorithms can share strengths from all of the paradigms and thus can be deployed in a broad spectrum of real-world scenarios. This holistic view is

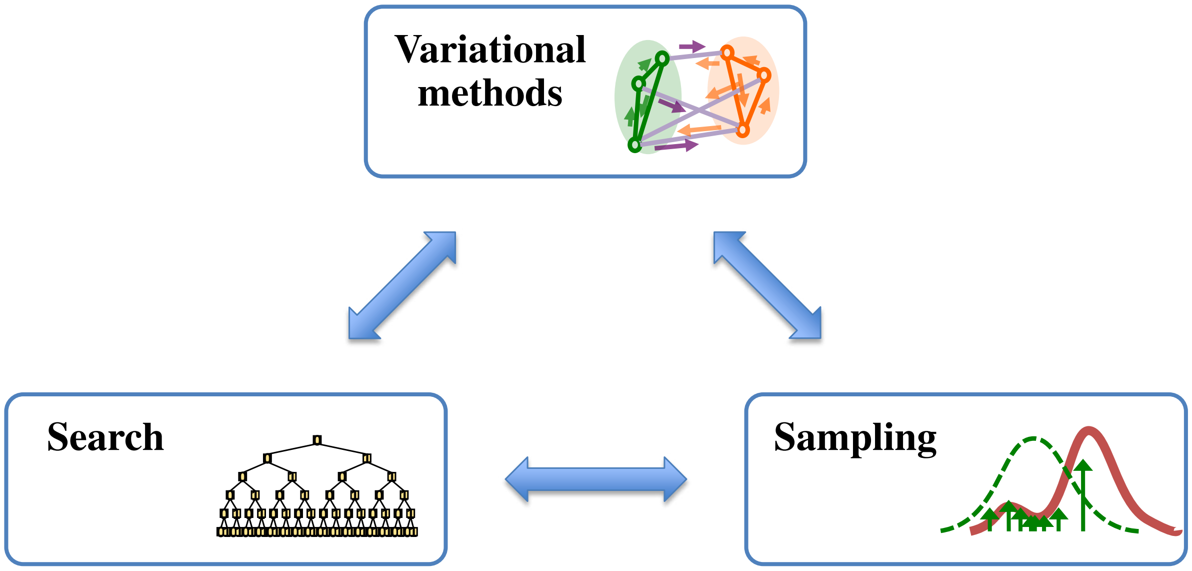


Figure 6.1: A holistic view of future work by exploring ideas/techniques from the three inference paradigms.

illustrated in Figure 6.1.

In this thesis, we utilized variational bounds to provide heuristics for search, and proposal distributions for sampling; we can in turn investigate how search and sampling can help construct high-quality variational bounds. One possible perspective is to use samples to decide what regions to add in order to tighten some relaxation [Sontag et al., 2008] because samples from a proposal extracted from the variational bound may carry valuable information to help make decision. This serves as a form of stochastic optimization, and can potentially be faster than deterministic optimization approaches (e.g., Forouzan and Ihler [2015]).

Another point worth mentioning is about variable ordering. In our algorithms, e.g., AOBFS, we imposed a fixed variable ordering, which is not necessarily optimal in many settings. Instead, one may wish to consider what variable to assign "on the fly" – this dynamic variable ordering strategy would alleviate the burden of finding a good variable ordering *a priori*, and can possibly lead to improved performance, as has been suggested by research on other domains [Bacchus and Van Run, 1995].

Bibliography

C. Andrieu, N. De Freitas, A. Doucet, and M. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50(1-2):5–43, 2003.

F. Bacchus and P. Van Run. Dynamic variable ordering in CSPs. In *Proceedings of the 1st International Conference on Principles and Practice of Constraint Programming*, CP'95, pages 258–275, 1995.

F. Bacchus, S. Dalmo, and T. Piassi. Value elimination: Bayesian inference via backtracking search. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence*, UAI'03, 2003a.

F. Bacchus, S. Dalmo, and T. Piassi. Algorithms and complexity results for #SAT and Bayesian inference. In *Proceedings of the 44th Symposium on Foundations of Computer Science*, FOCS'03, 2003b.

G. Bennett. Probability inequalities for the sum of independent random variables. *Journal of the American Statistical Association*, 57(297):33–45, 1962.

B. Bidyuk and R. Dechter. Cutset sampling for Bayesian networks. *Journal of Artificial Intelligence Research*, 28:1–48, 2007.

B. Bidyuk, R. Dechter, and E. Rollon. Active tuples-based scheme for bounding posterior beliefs. *Journal of Artificial Intelligence Research*, 39:335, 2010.

S. Boucheron, G. Lugosi, and O. Bousquet. *Concentration Inequalities*, pages 208–240. Springer Berlin Heidelberg, 2004.

O. Bousquet. *Concentration inequalities and empirical processes theory applied to the analysis of learning algorithms*. PhD thesis, Biologische Kybernetik, 2002.

C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Y. Burda, R. Grosse, and R. Salakhutdinov. Accurate and conservative estimates of MRF log-likelihood using reverse annealing. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, AISTATS'15, pages 102–110, 2015.

G. Casella and C. Robert. Rao-Blackwellisation of sampling schemes. *Biometrika*, 83(1): 81–94, 1996.

S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi. Distribution-aware sampling and weighted model counting for SAT. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, AAAI'14, pages 1722–1730, 2014.

S. Chakraborty, K. S. Meel, and M. Y. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 3569–3576, 2016.

M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6):772–799, 2008.

Q. Cheng, F. Chen, J. Dong, W. Xu, and A. Ihler. Approximating the sum operation for marginal-MAP inference. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, AAAI'12, pages 1882–1887, 2012.

P. Dagum and M. Luby. An optimal approximation algorithm for Bayesian inference. *Artificial Intelligence*, 93(1-2):1–27, 1997.

A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1):5–41, 2001.

A. Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge University Press, 2009.

R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

R. Dechter. Reasoning with probabilistic and deterministic graphical models: Exact algorithms. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 7(3):1–191, 2013.

R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2–3):73–106, 2007.

R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3):505–536, 1985.

R. Dechter and I. Rish. Mini-buckets: A general scheme of approximating inference. *Journal of the ACM*, 50(2):107–153, 2003.

R. Dechter, H. Geffner, and J. Y. Halpern. *Heuristics, Probability and Causality. A Tribute to Judea Pearl*. College Publications, 2010.

A. Doucet, S. J. Godsill, and C. P. Robert. Marginal maximum a posteriori estimation using Markov chain Monte Carlo. *Statistics and Computing*, 12(1):77–84, 2002.

S. Ermon, C. Gomes, A. Sabharwal, and B. Selman. Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *Proceedings of the 30th International Conference on International Conference on Machine Learning*, ICML'13, pages 334–342, 2013.

S. Ermon, C. Gomes, A. Sabharwal, and B. Selman. Low-density parity constraints for hashing-based discrete integration. In *Proceedings of the 31st International Conference on International Conference on Machine Learning*, ICML'14, pages 271–279, 2014.

N. Flerova, A. Ihler, R. Dechter, and L. Otten. Mini-bucket elimination with moment matching. In *Proceedings of NIPS Workshop on Discrete and Combinatorial Problems in Machine Learning*, DISCML'11, 2011.

S. Forouzan and A. Ihler. Incremental region selection for mini-bucket elimination bounds. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*, UAI'15, pages 268–277, 2015.

N. Friedman. Inferring cellular networks using probabilistic graphical models. *Science*, 303 (5659):799–805, 2004.

A. Globerson and T. Jaakkola. Approximate inference using conditional entropy decompositions. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics*, AISTATS'07, pages 131–138, 2007.

G. Hardy, J. Littlewood, and G. Pólya. *Inequalities.* Cambridge Mathematical Library. Cambridge University Press, 1952.

P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

M. Henrion. Search-based methods to bound diagnostic probabilities in very large belief nets. In *Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence*, UAI'91, pages 142–150, 1991.

W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

D. Jurafsky and J. Martin. *Speech and Language Processing.* Prentice Hall, 2008.

K. Kask and R. Dechter. A general scheme for automatic generation of search heuristics from specification dependencies. *Artificial Intelligence*, 129(1):91–131, 2001.

D. Kisa, G. Van den Broeck, A. Choi, and A. Darwiche. Probabilistic sentential decision diagrams. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning*, KR'14, 2014.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques.* MIT Press, 2009.

R. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78, 1993.

S. Lauritzen. *Graphical models*, volume 17. Clarendon Press, 1996.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

J. Lee, R. Marinescu, and R. Dechter. Applying search based probabilistic inference algorithms to probabilistic conformant planning: Preliminary results. In *Proceedings of the 14th International Symposium on Artificial Intelligence and Mathematics*, ISAIM'16, 2016a.

J. Lee, R. Marinescu, R. Dechter, and A. Ihler. From exact to anytime solutions for marginal MAP. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, AAAI'16, pages 3255–3262, 2016b.

J. S. Liu. *Monte Carlo strategies in scientific computing.* Springer Science & Business Media, 2008.

Q. Liu. *Reasoning and Decisions in Probabilistic Graphical Models – A Unified Framework.* PhD thesis, University of California, Irvine, 2014.

Q. Liu and A. Ihler. Bounding the partition function using Hölder's inequality. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 849–856, 2011.

Q. Liu and A. Ihler. Variational algorithms for marginal MAP. *Journal of Machine Learning Research*, 14(1):3165–3200, 2013.

Q. Liu, J. W. Fisher, III, and A. Ihler. Probabilistic variational bounds for graphical models. In *Proceedings of the 29th Conference on Neural Information Processing Systems*, NIPS'15, pages 1432–1440, 2015a.

Q. Liu, J. Peng, A. Ihler, and J. Fisher III. Estimating the partition function by discriminance sampling. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*, UAI'15, pages 514–522, 2015b.

Q. Lou, R. Dechter, and A. Ihler. Anytime anyspace AND/OR search for bounding the partition function. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, AAAI'17, pages 860–867, 2017a.

Q. Lou, R. Dechter, and A. Ihler. Dynamic importance sampling for anytime bounds of the partition function. In *Proceedings of the 31st Conference on Neural Information Processing Systems*, NIPS'17, pages 3198–3206, 2017b.

Q. Lou, R. Dechter, and A. Ihler. Anytime anyspace AND/OR best-first search for bounding marginal MAP. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, AAAI'18, 2018a.

Q. Lou, R. Dechter, and A. Ihler. Finite-sample bounds for marginal MAP. In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence*, UAI'18, 2018b.

Q. Lou, R. Dechter, and A. Ihler. Interleave variational optimization with Monte Carlo sampling: A tale of two approximate inference paradigms. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, AAAI'19, 2019. To appear.

J. Ma, J. Peng, S. Wang, and J. Xu. Estimating the partition function of graphical models using Langevin importance sampling. In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics*, AISTATS'13, pages 433–441, 2013.

D. MacKay. Introduction to Monte Carlo methods. In *Learning in Graphical Models*, pages 175–204. Springer, 1998.

R. Marinescu and R. Dechter. AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16):1457–1491, 2009a.

R. Marinescu and R. Dechter. Memory intensive AND/OR search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16-17):1492–1524, 2009b.

R. Marinescu, R. Dechter, and A. Ihler. AND/OR search for marginal MAP. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence*, UAI'14, pages 563–572, 2014.

R. Marinescu, R. Dechter, and A. Ihler. Pushing forward marginal MAP with best-first search. In *Proceedings of the 24th International Joint Conferences on Artifical Intelligence*, IJCAI'15, pages 696–702, 2015.

R. Marinescu, J. Lee, A. Ihler, and R. Dechter. Anytime best+ depth-first search for bounding marginal MAP. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, AAAI'17, pages 3775–3782, 2017.

R. Marinescu, R. Dechter, and A. Ihler. Stochastic anytime search for bounding marginal MAP. In *Proceedings of the 27th International Joint Conferences on Artifical Intelligence*, IJCAI'18, pages 5074–5081, 2018.

D. Mauá and C. de Campos. Anytime marginal maximum a posteriori inference. In *Proceedings of the 29th International Conference on International Conference on Machine Learning*, ICML'12, pages 1395–1402, 2012.

A. Maurer and M. Pontil. Empirical Bernstein bounds and sample variance penalization. In *Proceedings of the 22nd Conference on Learning Theory*, COLT'09, 2009.

T. Minka. Expectation propagation for approximate Bayesian inference. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, UAI'01, pages 362–369, 2001.

V. Mnih, H. Larochelle, and G. Hinton. Conditional restricted Boltzmann machines for structured output prediction. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, UAI'11, pages 514–522, 2011.

C. A. Naesseth, F. Lindsten, and T. Schön. Sequential Monte Carlo for graphical models. In *Proceedings of the 28th Conference on Neural Information Processing Systems*, NIPS'14, pages 1862–1870, 2014.

S. Nowozin and C. Lampert. Structured learning and prediction in computer vision. *Foundations and Trends® in Computer Graphics and Vision*, 6(3–4):185–365, 2011.

M.-S. Oh and J. Berger. Adaptive importance sampling in Monte Carlo integration. *Journal of Statistical Computation and Simulation*, 41(3-4):143–168, 1992.

M. Opper and D. Saad, editors. *Advanced Mean Field Methods: Theory and Practice*. MIT, 2001.

L. Otten and R. Dechter. Anytime AND/OR depth-first search for combinatorial optimization. *AI Communications*, 25(3):211–227, 2012.

L. Otten, A. Ihler, K. Kask, and R. Dechter. Winning the PASCAL 2011 MAP challenge with enhanced AND/OR branch-and-bound. In *Proceedings of NIPS Workshop on Discrete and Combinatorial Problems in Machine Learning*, DISCML'11, 2011.

U. Oztok and A. Darwiche. A top-down compiler for sentential decision diagrams. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, IJCAI'15, pages 3141–3148, 2015.

J. Park. MAP complexity results and approximation methods. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, UAI'02, pages 388–396, 2002.

J. Park and A. Darwiche. Solving MAP exactly using systematic search. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence*, UAI'03, pages 459–468, 2003.

J. Park and A. Darwiche. Complexity results and approximation strategies for MAP explanations. *Journal of Artificial Intelligence Research*, 21:101–133, 2004.

J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley, 1984.

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

W. Ping and A. Ihler. Belief propagation in conditional RBMs for structured prediction. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, AISTATS'17, pages 1141–1149, 2017.

W. Ping, Q. Liu, and A. Ihler. Decomposition bounds for marginal MAP. In *Proceedings of the 29th Conference on Neural Information Processing Systems*, NIPS'15, pages 3267–3275, 2015.

T. Popoviciu. Sur les équations algébriques ayant toutes leurs racines réelles. *Mathematica*, 9:129–145, 1935.

E. Rollon and R. Dechter. New mini-bucket partitioning heuristics for bounding the probability of evidence. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, AAAI'10, pages 1199–1204, 2010.

E. Rollon, J. Larrosa, and R. Dechter. Semiring-based mini-bucket partitioning schemes. In *Proceedings of the 23rd international joint conference on Artificial Intelligence*, IJCAI'13, pages 644–650, 2013.

S. Russell. Efficient memory-bounded search methods. In *Proceedings of the 10th European Conference on Artificial Intelligence*, ECAI'92, pages 1–5, 1992.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Press, 2009.

R. Salakhutdinov and I. Murray. On the quantitative analysis of deep Belief networks. In *Proceedings of the 25th International Conference on Machine learning*, ICML'08, pages 872–879, 2008.

T. Sang, P. Beame, and H. Kautz. Solving Bayesian networks by weighted model counting. In *Proceedings of the 20th National Conference on Artificial Intelligence*, AAAI'05, pages 475–482, 2005.

E. Santos. On the generation of alternative explanations with implications for belief revision. In *Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence*, UAI'91, pages 339–347, 1991.

S. Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68(2): 399–410, 1994.

S. Shimony and E. Charniak. A new algorithm for finding MAP assignments to belief networks. In *Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence*, UAI'91, pages 185–193, 1991.

J. Sohl-Dickstein and B. Culpepper. Hamiltonian annealed importance sampling for partition function estimation. *arXiv preprint arXiv:1205.1925*, 2012.

D. Sontag, T. Meltzer, A. Globerson, T. Jaakkola, and Y. Weiss. Tightening LP relaxations for MAP using message passing. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, UAI'08, pages 503–510, 2008.

R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* MIT press, 1998.

R. Szeliski. *Computer Vision: Algorithms and Applications.* Springer Science & Business Media, 2010.

L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8 (2):189–201, 1979.

C. Viricel, D. Simoncini, S. Barbe, and T. Schiex. Guaranteed weighted counting for affinity computation: Beyond determinism and structure. In *Proceedings of the 22nd International Conference on Principles and Practice of Constraint Programming*, CP'16, pages 733–750. Springer, 2016.

M. Wainwright and M. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2):1–305, 2008.

M. Wainwright, T. Jaakkola, and A. Willsky. A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory*, 51(7):2313–2335, July 2005.

A. Weller and J. Domke. Clamping improves TRW and mean field approximations. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, AISTATS'16, pages 38–46, 2016.

B. Wemmenhove, J. M. Mooij, W. Wiegerinck, M. Leisink, H. J. Kappen, and J. P. Neijt. Inference in the promedas medical expert system. In *Conference on Artificial Intelligence in Medicine in Europe*, pages 456–460. Springer, 2007.

Y. Xue, Z. Li, S. Ermon, C. Gomes, and B. Selman. Solving marginal MAP problems with NP oracles and parity constraints. In *Proceedings of the 30th Conference on Neural Information Processing Systems*, NIPS'16, pages 1127–1135, 2016.

P. Yadollahpour, D. Batra, and G. Shakhnarovich. Discriminative re-ranking of diverse segmentations. In *Proceedings of the 26th IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'13, pages 1923–1930, 2013.

C. Yanover and Y. Weiss. Approximate inference and protein-folding. In *Proceedings of the 16th Conference on Neural Information Processing Systems*, NIPS'02, pages 1457–1464, 2002.

C. Yuan and E. Hansen. Efficient computation of jointree bounds for systematic MAP search. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, pages 1982–1989, 2009.

C. Yuan, T.-C. Lu, and M. Druzdzel. Annealed MAP. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI'04, pages 628–635, 2004.

S. Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73, 1996.

Appendix A

Proofs for Propositions in Chapter 2

A.1 Proof of Proposition 2.2

Proposition 2.2. If we assume $\mathrm{E}_t[|u(x)|] > 0$, and let

$$q^\star(x) = |u(x)|t(x)/\,\mathrm{E}_t[|u(x)|], \tag{2.23}$$

then for any proposal distribution $q(x)$, we have

$$\mathrm{Var}_{q^\star}\left[\frac{u(x)t(x)}{q^\star(x)}\right] \le \mathrm{Var}_q\left[\frac{u(x)t(x)}{q(x)}\right].$$

Proof. Since

$$\begin{aligned}
\mathrm{E}_{q^\star}\left[\Big(\frac{u(x)t(x)}{q^\star(x)}\Big)^2\right] &= \int_\Omega \frac{u^2(x)t^2(x)}{q^\star(x)}\,\mathrm{d}x \\
&= \int_\Omega \frac{u^2(x)t^2(x)}{|u(x)|t(x)/\,\mathrm{E}_t[|u(x)|]}\,\mathrm{d}x \\
&= \mathrm{E}_t\left[|u(x)|\right]\int_\Omega |u(x)|t(x)\,\mathrm{d}x \\
&= \left(\int_\Omega |u(x)|t(x)\,\mathrm{d}x\right)^2
\end{aligned}$$

$$
\begin{aligned}
&= \Big(\int_\Omega \frac{|u(x)|t(x)}{q(x)} q(x)\,\mathrm{d}x \Big)^2 \\
&\le \int_\Omega \Big(\frac{|u(x)|t(x)}{q(x)} \Big)^2 q(x)\,\mathrm{d}x \qquad \text{(Cauchy-Schwarz inequality)} \\
&= \mathrm{E}_q \left[\Big(\frac{u(x)t(x)}{q(x)} \Big)^2 \right],
\end{aligned}
$$

i.e.,

$$
\mathrm{E}_{q^\star} \left[\Big(\frac{u(x)t(x)}{q^\star(x)} \Big)^2 \right] \le \mathrm{E}_q \left[\Big(\frac{u(x)t(x)}{q(x)} \Big)^2 \right].
$$

We know

$$
\begin{aligned}
\mathrm{Var}_{q^\star} \left[\frac{u(x)t(x)}{q^\star(x)} \right] &= \mathrm{E}_{q^\star} \left[(\frac{u(x)t(x)}{q^\star(x)})^2 \right] - \Big(\mathrm{E}_t[u(x)] \Big)^2 \qquad \text{(see (2.22))} \\
&\le \mathrm{E}_q \left[(\frac{u(x)t(x)}{q(x)})^2 \right] - \Big(\mathrm{E}_t[u(x)] \Big)^2 \\
&= \mathrm{Var}_q \left[\frac{u(x)t(x)}{q(x)} \right],
\end{aligned}
$$

i.e.,

$$
\mathrm{Var}_{q^\star} \left[\frac{u(x)t(x)}{q^\star(x)} \right] \le \mathrm{Var}_q \left[\frac{u(x)t(x)}{q(x)} \right],
$$

which proves the proposition.

Appendix B

Proofs for Propositions in Chapter 3

B.1 Proof of Proposition 3.1

Proposition 3.1. The quantity V_n represents the total weighted sum over all solution trees that include the path from the root to node n. Given a current search tree $\mathcal{S}$, for all n, we have $L_n \leq V_n \leq U_n$.

Proof. First, it is obvious that U_n and L_n are upper and lower bounds of V_n respectively. For a *solution* tree $T \in \mathbb{T}(\mathcal{S})$ where $\mathbb{T}(\mathcal{S})$ denotes the set of all solution trees of $\mathcal{S}$, its mass is

$$v_T = \prod_{c \in T} w_c \prod_{s \in T \cap OPEN} v_s$$

T also defines an upper bound of v_T denoted as u_T where

$$u_T = \prod_{c \in T} w_c \prod_{s \in T \cap OPEN} u_s$$

By taking the sum of masses over all the solution trees of $\mathcal{S}$, we obtain the exact partition function. Analogously, by taking the sum of upper bounds over all its solution trees, we can upper bound the partition function. Namely,

$$U = \sum_{T \in \mathbb{T}(\mathcal{S})} u_T$$

Now, for any solution tree T that contains n, g_n contributes to v_T as a multiplicative factor. The rest of v_T comes from subtrees underneath nodes in $\{n\} \bigcup branch(n)$. Thus, it is easy to verify that

$$V_n = \sum_{\{T \in \mathbb{T}(\mathcal{S}) \ | \ n \in T\}} v_T$$

i.e., V_n is the total mass of all solution trees that contain n.

B.2 Proof of Proposition 3.2

Proposition 3.2. Given a current search tree $\mathcal{S}$, fully solving the summation problem below a frontier node s will tighten the bound difference $U - L$ by

$$g_s\left(u_s - v_s\right) \prod_{t \in branch(s)} u_t + g_s\left(v_s - l_s\right) \prod_{t \in branch(s)} l_t$$

which is upper bounded by $gap(s) = U_s - L_s$.

Proof. We can see that fully solving the subproblem underneath s will decrease U_s to

$$g_s \, v_s \prod_{t \in branch(s)} u_t$$

This will improve U by

$$g_s\left(u_s - v_s\right) \prod_{t \in branch(s)} u_t$$

By applying the same argument to lower bound, we know the bound difference $U - L$ will be reduced by

$$g_s\left(u_s - v_s\right) \prod_{t \in branch(s)} u_t + g_s\left(v_s - l_s\right) \prod_{t \in branch(s)} l_t$$

which is obviously a lower bound of $gap(s) = U_s - L_s$.

B.3 Proof of Proposition 3.8

Proposition 3.8. In each iteration, Algorithm 3.3 finds a top-priority frontier node to expand.

Proof. We will prove the following claims from which we can easily derive the proposition.

For any internal node $n \in \mathcal{S}$, we claim: First, its best child $c^\star$ found via (3.21) is an ancestor of a top-priority descendant of n in $OPEN$.

Second,

$$U_n^\star g_n \prod_{s \in branch(n)} u_s \tag{B.1}$$

is the secondary priority value of n' top-priority descendant in $OPEN$.

We will prove the above claims via induction on the height of nodes in $\mathcal{S}$, where the height for a node is defined as the distance from that node to its furthest descendant in $OPEN$. For example, a frontier node has height 0; a node has all its children in *OPEN* has height 1.

To begin with, the first claim is vacuously true for all frontier nodes, and the second claim is true for all frontier nodes by definition. Then, suppose the claims hold for those nodes of height no greater than some $h \geq 0$, we will show they also hold for nodes of height $h+1$. Now, let $n \in \mathcal{S}$ be a node of height $h+1$; all its children have height no greater than h and thus the claims apply to them.

If n is an AND node, for any $c_1, c_2 \in ch(n)$, it is easy to see that their top-priority frontier descendants have the same primary priority. Thus, we only have to compare the secondary priority of their top-priority frontier descendants. Since

$$U^{\star}_{c_1} g_{c_1} \prod_{s \in branch(c_1)} u_s$$

is the secondary priority value of c_1's top-priority frontier descendant according to (B.1). By applying the facts that $g_n = g_{c_1}$ and $u_n = \prod_{c \in ch(n)} u_c$ to the above quantity, we have

$$U^{\star}_{c_1} / u_{c_1} u_n g_n \prod_{s \in branch(n)} u_s$$

Thus, (3.21) finds $c^{\star}$ that leads to a top-priority descendant of AND node n, which has the secondary priority

$$U^{\star}_{c^{\star}} / u_{c^{\star}} u_n g_n \prod_{s \in branch(n)} u_s$$

Combining the above and (3.22), we know the second claim is true for n as well.

If n is an OR-MAX node, for any $c_1 \in ch(n)$, it is easy to see that

$$g_{c_1} u_{c_1} \prod_{s \in branch(c_1)} u_s$$

is the primary priority value of c_1's top-priority frontier descendant, which can then be re-written as

$$w_{c_1} u_{c_1} g_n \prod_{s \in branch(n)} u_s$$

by considering the facts that $g_{c_1} = w_{c_1} g_n$ and $branch(c_1) = branch(n)$. According to the second claim and $g_{c_1} = w_{c_1} g_n$, the secondary priority value of c_1's top-priority frontier descendant can be re-written as

$$w_{c_1} U^{\star}_{c_1} g_n \prod_{s \in branch(n)} u_s$$

Thus, $c^{\star}$ found via (3.21) leads to a top-priority descendant. Also, the second claim holds for n in lieu of (3.22).

If n is an OR-SUM node, we know all its frontier descendants share the same primary priority. The analysis on the secondary priority is the same as that of the OR-MAX case.

All in all, we can see that the two claims hold for a node of height $h+1$. By induction, we know these claims hold for all internal nodes of $\mathcal{S}$, which implies Proposition 3.8.