UNIVERSITY OF CALIFORNIA,
IRVINE

Advancing AND/OR Abstraction Sampling and AND/OR Search-Based Computational Protein Design

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Bobak Pezeshki

2024

DEDICATION

To my mother who taught me the importance of following one's heart
also with compassion for others.

In memory of Filjor Broka.

"...You can use logic to justify just about anything;
that's its power and its flaw."
– Rick Berman, 1995, *Star Trek: Voyager*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Algorithms

Page

ACKNOWLEDGMENTS

First and foremost, I want to express my heartfelt thank you to Professor Rina Dechter. Thank you for being such a great advisor and always kindly and patiently, yet sternly, pushing me to grow. Thank you for recognizing my strengths and helping me improve on my weaknesses, both in computer science and as a person. Your kindness, passion for your work, and commitment —- including working late into the night ... *many nights* -— have been deeply inspiring. We often joke that we hope to have as much passion for our careers as you do for yours. I'm especially grateful for how involved you've been in me and all of your students, and for always providing us genuine feedback. Your dedication, heart, and wisdom have left a permanent mark on me.

I'm also deeply grateful to Professor Alexander Ihler. Thank you for patiently supporting me while offering your unique and intuitive insights to complex concepts. You have a special way of breaking things down that makes them more understandable (despite the fact that it takes me a few tries to get it – thank you for being patient with me!) You're always encouraging the explorative side of my mind and expanding my motivation in meaningful directions. Your dedication, support, and helpful advice have meant a lot to me throughout this journey. I especially appreciate all the time you took out of your busy schedule to meet with me in person; it truly means a lot to me.

I want to extend a special thank you to Junkyu Lee, for all your incredible support and dedication. You're always willing to help, always putting more on your shoulders to help the rest of us. I'm amazed by how fast you respond to emails no matter what time it is. Sometimes I wonder if you ever sleep! I'll never forget the fun memories of taking the generative models class together grinding on that last project together. I especially admire how you manage to balance all your work while being a wonderful husband and dad. I am forever grateful to you.

Another special thank you is to Filjor Broka. Thank you for believing that I can transfer my personality and passion into my work and the hours you stayed late to explain concepts to me. I'll never forget the one time we spent hours on the white board discussing concepts of probability, both from a mathematical and philosophical perspective. It really highlighted your spirit and love. I'll miss our chats.

I want to express so much gratitude to Radu Marinescu for working with me when I was still learning the ropes. I can't not say this: your code is written so clearly and organized so well! I truly appreciate your willingness to join Zoom meetings late nights to guide me through complex topics and making sure I was ok moving forward. Your positive and encouraging attitude has always been energizing and motivating.

Special thanks to Professor Kalev Kask for guiding me on my academic journey and helping me transition into my first research project with Abstraction Sampling. You were my first AI instructor, and your class ignited my passion for AI and reinforced it for teaching. Your insights and willingness to explain complex concepts has been so helpful. Thank you for

your mentorship, both for specific research projects and for supporting my development as a PhD student and future instructor.

I want to give a big thank you to Professor Erik Sudderth for being part of my defense committee and also as a pillar of the AI/ML group. You're always around to help and answer questions, whether it's supporting the department, AI group, or running seminars. Also your Learning in Graphical Models class was one of the best classes that I've taken. I also want to thank Professor Sameer Singh for always having your door open and constantly working to improve our department. The PhD students see you as a warm and friendly father figure thanks to your care for us.

I would like to acknowledge my partners (in crime?) in the lab, Nicholas Cohen and Annie Raichev. Thank you Nick for always pushing our group to work together more. You paved the way for our lunches, walks, and talks (research, philosophical, and otherwise), and always have such interesting insights. I look forward to many more sessions to come! Thank you Annie for being the burst of energy in our group. We can always expect to get a text from you before long, and it'll always have cat pics included. PS. I really admire the way you give talks. You're so good at breaking things down, showing examples, and just having good flow. Inspiring.

Thank you, Rodrigo de Salvo Braz. Working with you two summers on symbolic probabilistic languages at SRI was such a memorable experience. I was always so excited to work on our project, and to come and discuss with you. Your calm and peaceful demeanor is so enjoyable and motivating. There is a complex care I feel from you towards people? the world? that is touching and inspiring. I really appreciate your mentorship and providing me my first glimpse of insight into industry research. You made each day a learning experience.

Thank you, Shufeng Kong, Yasaman Razeghi, Jiapeng Zhao, and Sakshi Agarwal. You were all so motivating and I truly valued the opportunity to interact with each of you. Our discussions, even in the short time we had, were enjoyable and enriching. Yasaman, thank you for pushing me to build confidence. Sakshi, thank you for your endless amount of happy energy. Shufeng, thank you for the many late nights at DBH and words of wisdom as I started my PhD. I hope that we can stay in touch. (And Jiapeng, I look forward to working with you more in the Fall!)

I also want to acknowledge the professors who have offered enriching courses that I've taken. In addition to the myriad of AI courses taught by Professors Dechter, Ihler, Sudderth, and Kask, the courses by Professor Padhraic Smyth for Probabilistic Learning, Professor Stephen Mandt for Deep Generative Models, Professor Weining Shen for Statistics, and Professor Michael Dillencourt for Algorithms have been amazing. Your teachings have brought excitement to my eductaion and have been fundamental in shaping my understanding and skills in these areas.

Thank you to the professors whom I've TA'ed for and/or have helped me evolve instructionally: Professors Jennifer Wong-Ma, Raymond Klefstad, Bob Pelayo. Thank you also to

Professor Xiohui Xie for being part of my advancement committee.

I would like to acknowledge Bruce Donald and the members of his lab, as well as Thomas Schiex and his group from the INRAE Centre, for their crucial role in jump-starting the research on computational protein design.

To department staff – Kris, Naima, Mariko, Lumen, Majde, and many others! – thank you for your support with logistics, paperwork, and various other tasks. We would all be in shambles without your help.

Thank you the OIT Helpdesk for managing our computer cluster and always responding immediately to aid requests. Your fast responses – sometimes even outside of normal hours! – have been such a great help.

I would like to thank everyone in my cohort for experiencing parts of this journey with me – Claudio, Kyu-Seon, Wonnie, Madina, Gabe, John, Akshay, AK, Nile, and Siwei. I look forward to our next reunion. And to my late-night DBH study group – Ethel, Andrew, and Momoko – thank you for all our late night (early morning?) study sessions.

Finally, I would like to extend my heartfelt thanks to my family and friends for being in my life throughout all of its colors.

To Stephen, who is always checking in on me and making sure our group from high school stays connected, you have a dear and permanent place in my heart. To Michael and Jerry, thank you for making sure I did not go through these many years alone. Your visits, our video chats, and your votes of confidence helped keep me grounded. Thank you, Debbie Jih, for spending so many of your hours to keep me company through my work this heavy summer. Your love, kindness, and positivity brightens my day.

To my mom, Mehrnaz Dadgar, you are the number one inspiration for whom I strive to be. Its amazing that I grew up with my deepest best friend and only realized it until I got older. I just hope you know how much you mean to me, and how much of you I am. Or at least that I hope to be. Your endless love, and care of others, has been a guiding light in my life.

I am also deeply grateful to my dad and mom, Ardeshir and Shirin, my siblings, Amir, Payam, Natasha, and Kristina, and to my niece and nephews, Kayden, Roman, Jackson and Sienna, for their endless love and support. Thank you for pulling me out of my PhD den from time to time, reminding me of the rest of life. You are always checking in on me, supporting me to succeed (in your own ways ;), and I look up to each and every one of you. I look forward to our future together as we continue to grow as a peaceful loving family.

To end, I would like to express sincere thanks for the funding sources that made my graduate studies possible, including NSF grants IIS-2008516 and CNS-2321786, as well as travel funds from the Department of Computer Science.

CURRICULUM VITAE

Bobak Pezeshki

EDUCATION

Doctor of Philosophy in Computer Science **2024**
University of California, Irvine *Irvine, California*

Master of Science in Computer Science **2022**
University of California, Irvine *Irvine, California*

Department of Computer Information Systems **2017**
De Anza College *Cupertino, California*

Bachelor of Science in Molecular and Cell Biology, and Integrative Biology **2007**
University of California, Berkeley *Berkeley, California*

ACADEMIC RESEARCH EXPERIENCE

Graduate Research Assistant **2017 – Present**
University of California, Irvine *Irvine, California*

Research Assistant **2007 – 2008**
University of California, San Francisco *San Francisco, California*

TEACHING EXPERIENCE

Teaching Assistant **2017 – Present**
University of California, Irvine *Irvine, California*

AP Chemistry and Physics Teacher **2010 – 2016**
Skyline High School *Oakland, California*

Chemistry-P Instructor **2006 – 2007**
Summer Bridge Program, University of California, Berkeley *Berkeley, California*

Chemistry Study-Group Instructor **2002 – 2006**
Student Learning Center, University of California, Berkeley *Berkeley, California*

PROFESSIONAL RESEARCH EXPERIENCE

Symbolic Probabilistic Systems Intern **2018, 2019**
Stanford Research Institute (SRI) International *Menlo Park, California*

Research Assistant **2009**
Novartis Institutes for Biomedical Research *Emeryville, California*

PUBLICATIONS

Bobak Pezeshki, Kalev Kask, Alexander Ihler, and Rina Dechter. **Value-based abstraction functions for abstraction sampling.** In *Proceedings of the Fortieth Conference on Uncertainty in Artificial Intelligence (UAI)*, to appear in Proceedings of Machine Learning Research (PMLR). PMLR, 2024.

Bobak Pezeshki, Radu Marinescu, Alexander Ihler, and Rina Dechter. **Boosting AND/OR-based computational protein design: dynamic heuristics and generalizable UFO.** In *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 216 of *Proceedings of Machine Learning Research (PMLR)*, pages 1662–1672. PMLR, 2023.

Bobak Pezeshki, Radu Marinescu, Alexander Ihler, and Rina Dechter. **AND/OR branch-and-bound for computational protein design optimizing K^*.** In *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 180 of *Proceedings of Machine Learning Research (PMLR)*, pages 1602–1612. PMLR, 2022.

Kalev Kask, Bobak Pezeshki, Filjor Broka, Alexander Ihler, and Rina Dechter. **Scaling up AND/OR abstraction sampling.** In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4266–4274. International Joint Conferences on Artificial Intelligence Organization, 2020.

Franz Gruswitz, Sarika Chaudhary, Joseph D Ho, Avner Schlessinger, Bobak Pezeshki, Chi-Min Ho, Andrej Sali, Connie M Westhoff, and Robert M Stroud. **Function of human rh based on structure of rhcg at 2.1 Å.** *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 107(21):9638–43, May 25 2010. PMC2906887.

WORKSHOP PRESENTATIONS

Bobak Pezeshki, Radu Marinescu, Alexander Ihler, and Rina Dechter. **Boosting AND/OR-based computational protein design: dynamic heuristics and generalizable UFO.** In *Workshop on Tractable Probabilistic Modeling (TPM) at the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*, 2023.

Bobak Pezeshki, Radu Marinescu, Alexander Ihler, and Rina Dechter. **AND/OR branch-and-bound for computational protein design optimizing K***. In *Workshop on Tractable Probabilistic Modeling (TPM) at the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, 2022. ***Best paper award***.

Bobak Pezeshki, Radu Marinescu, Alexander Ihler, and Rina Dechter. **AND/OR branch-and-bound for computational protein design optimizing K***. In *Workshop on AI to Accelerate Science and Engineering (AI2ASE) at the Thirty-Sixth AAAI Conference on Artificial Intelligence*, 2022.

ABSTRACT OF THE DISSERTATION

Advancing AND/OR Abstraction Sampling
and
AND/OR Search-Based Computational Protein Design

By

Bobak Pezeshki

Doctor of Philosophy in Computer Science

University of California, Irvine, 2024

Prof. Rina Dechter, Co-Chair
Prof. Alexander Ihler, Co-Chair

Graphical models are a powerful framework for efficiently representing and reasoning about complex systems. They can be used to answer probabilistic queries, facilitate planning, and enable automated design across various fields including in business, medicine, and physics. Reasoning within graphical models typically involves optimization tasks, like finding the most probable configuration, or summation tasks, like computing beliefs over variables, or a combination of both, such as Marginal MAP where we maximize over a subset of variables while summing over the rest. These tasks are computationally challenging, requiring the use of approximation algorithms typically implemented through search, sampling, or variational inference techniques.

This dissertation presents advances in graphical model schemes in two main directions:

Several advancements to a recently developed Monte Carlo sampling method called Abstraction Sampling are presented. Abstraction Sampling is an unbiased stratified importance sampling-like scheme that leverages abstractions (similar to stratification) to solve summation queries such as determining beliefs about random variables or calculating the probability of evidence. This dissertation presents AOAS, an Abstraction Sampling algorithm tailored for compact AND/OR search spaces, explores a diverse range of abstraction functions, provides a theoretical analysis of the properties of Abstraction Sampling, and conducts an extensive empirical evaluation demonstrating Abstraction Sampling's superior performance compared to well-known methods including Importance Sampling, Weighted Mini-Bucket Importance Sampling, IJGP-SampleSearch, and Dynamic Importance Sampling.

Also presented is more applied research adapting graphical model frameworks for Computational Protein Design, specifically focusing on the automated redesign of proteins. This redesign is formulated as an optimization problem to maximize K^*, an approximation of binding affinity. Included are two novel formulations of this task within a graphical model framework, as well as introduction of wMBE-K*, a message-passing scheme based on weighted Mini-Bucket Elimination for estimating and bounding K^*. Additionally, a range of algorithms derived from adapting Marginal MAP algorithms on AND/OR search spaces to address the protein redesign problem is presented. Results, demonstrated on real protein benchmarks, show superior performance compared to the state-of-the-art algorithm BBK* for small and medium-sized problems. Finally, a technique for infusing determinism into graphical models that emerged from this work, which significantly speeds up inference, is presented.

Chapter 1

Introduction

Graphical models provide a robust framework for efficiently representing and reasoning about complex systems. They are used within a wide variety of fields ranging from business, to autonomous transportation, robotics, medicine, biology, chemistry, physics, and many more. Graphical models are used to answer probabilistic queries such as to evaluate the probability of an unusual event, for recognition and labeling of media such as annotating medical images, in planning for example to map out robotic movements, to enable automated design such as the creation of new molecules, and to infer about causal influences and the effects of interventions. Reasoning within graphical models often involves solving optimization problems such as finding the most probable assignment to a set of variables, or performing summation tasks such as computing "beliefs," or posterior probabilities, over variables given observations. Sometimes, tasks may involve combining both approaches, so that we maximize over a subset of variables while marginalizing (summing over) the remainder. These tasks can be computationally demanding in large models and often require approximation algorithms, which are usually implemented through search, sampling, or variational inference methods. In the spirit of advancing graphical models research, this dissertation provides new graphical model frameworks and algorithms for both general and domain-specific tasks.

We begin in Chapter 2 by providing some background on graphical models, reviewing common inference tasks for which they are typically used, and discuss paradigms for solving these tasks. Then in Chapter 3, we explore a recently introduced Monte Carlo stratified importance sampling-like method called Abstraction Sampling for answering summation tasks over graphical models. We analyze Abstraction Sampling's theoretical properties, introduce several powerful extensions, and demonstrate evidence of its compelling performance through an extensive experimental evaluation. Next, Chapter 4 describes, analyzes, and demonstrates the potential of a new framework called Underflow-threshold Optimization (UFO) for infusing artificial determinism into models. We show the potential of the UFO framework for empowering graphical model algorithms to take advantage of constraint processing to increase their efficiency, analyzing its properties and demonstrating its effectiveness in experiments comparing to state-of-the-art solvers. Finally, in Chapter 5 we focus on the applied task of computational protein design, providing a graphical model framework and accompanying algorithms for protein redesign to improve binding affinity. We demonstrate our methods' performance experimentally over real protein benchmarks.

We outline some of the novel contributions included in this dissertation, organized by chapter:

Chapter 3: Advancing Abstraction Sampling Contributions:

1. Based on work from Broka et al. [2018], we introduce algorithm `ORAS` for performing Abstraction Sampling on classical OR state space search trees.
2. Based on work from Broka et al. [2018], we give a theoretical analysis on OR Abstraction Sampling properties, including variance reduction conditions and a proof of unbiasedness of `ORAS`.
3. We propose a new Abstraction Sampling algorithm, `AOAS`, designed for compact AND/OR search spaces. By freeing `AOAS` from upholding a restrictive property known as "properness", our new scheme significantly enhances the scalability and performance of Abstraction Sampling for AND/OR spaces.
4. We give theoretical analysis on AND/OR Abstraction Sampling properties, in-

cluding variance reduction conditions and a proof of unbiasedness that lifts the "properness" restriction.

5. We propose three classes of abstraction functions that guide Abstraction Sampling's stratification process. The first, based on work from Broka et al. [2018], uses a graph notion of "context"; a second is based on partitioning nodes based on positive real number values associated with them; and lastly a purely randomized abstraction scheme is introduced. Overall, over twenty-four distinct abstraction functions were tested, each with the ability to vary granularity of their abstractions.

6. We perform an extensive empirical evaluation on over 400 problems from five well known summation benchmarks, comparing our Abstraction Sampling frameworks against each other and to competing schemes. Our experiments illustrate the properties of Abstraction Sampling, its strength at estimating summation queries, and allow us to illuminate a few particularly powerful Abstraction Sampling setups.

Chapter 4: UFO: Underflow-Threshold Optimization Contributions:

1. We propose a new scheme called `UFO` for infusing artificial determinism into graphical models, which empowers graphical model algorithms to exploit constraint processing for increased efficiency, for example by early pruning of inconsistent branches within a search algorithm.

2. We derive theoretical properties of introducing such artificial determinism, including boundedness for common graphical model tasks and tractable bounds on the error for the well known MMAP task.

3. We evaluate UFO performance with a vanilla AND/OR branch-and-bound algorithm empowered with constraint propagation on 100 problems used in the 2022 UAI Inference Competition. We compare the performance of UFO to the six

solvers used in the original competition, highlighting the UFO method's potential.

Chapter 5: AND/OR Search-Based Computational Protein Design Contributions:

1. We give two formulations of the protein redesign problem as a graphical model applied to the inference task of optimizing an objective called K^*, which acts as an approximation to the protein's binding affinity.
2. We propose `wMBE-K*`, an adaptation of Weighted Mini-Bucket Elimination for use in bounding K^*. The method bounds the maximum K^*, but in addition (and more importantly) can be used to efficiently generate heuristics for search over AND/OR search spaces.
3. We develop an array of `AOBB-K*` algorithms – specifically, anytime depth-first branch-and-bound algorithms over AND/OR search spaces for protein redesign maximizing K^*. Variants include augmentation with weighted heuristic search, the use of dynamic heuristics, and incorporating the `UFO` technique.
4. We provide empirical analysis on real protein benchmarks, comparing our schemes to the state-of-the-art algorithm `BBK*` [Ojewole et al., 2018], which is used as part of a long-standing computational protein design software called OSPREY [Hallen et al., 2018]. Our methods show strong performance in our comparisons.

In Chapter 6, we conclude the thesis, highlighting some potential extensions of the thesis' research topics as possible directions for future work.

Chapter 2

Background

▢ 2.1 Probabilistic Graphical Models

Graphical Models, such as Bayesian or Markov networks [Pearl, 1988, Darwiche, 2009, Dechter, 2013], are mathematical tools for modeling complex systems, each composed of a set of variables with defined domains and functions defined over subsets of the variables. The functions capture local dependencies of the subset of variables over which they are defined, known as the function's **scope**. The functions of a graphical model often represent a factorization of a global function over all the variables. An assignment to all of the variables (referred to as a **full configuration**) represents a possible state of the modeled system.

Graphical models are constructed not only to model a system, but also to provide a means of efficiently answering specific queries of interest via exploitation of the model's structure. Some common computational tasks are:

- determination of the **partition function**: a normalization constant necessary for computing probabilistic quantities.
- determination of the **MAP** (**maximum a posteriori**): the most probable full configuration of the model, sometimes given a **partial configuration** (assignments to a

subset of the variables) known as observations or evidence.

- determination of the **MMAP** (**marginal maximum a posteriori**) configuration: the configuration of a target subset of variables that maximizes their marginal likelihood, i.e., the probability of the configuration after summing over any variables that are not in the target subset.

More details about common graphical model queries are provided in Section 2.2: Common Graphical Model Queries.

■ 2.1.1 Discrete Graphical Models

In this thesis, we mainly consider models defined over a discrete space. A discrete graphical model can be defined as a 3-tuple $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$, where:

- $\mathbf{X}$ is the set of variables over which the model is defined.
- $\mathbf{D} = \{D_X : X \in \mathbf{X}\}$ is a set of finite domains, one for each $X \in \mathbf{X}$, defining the possible values each X can be assigned.
- Each $f_\alpha \in \mathbf{F}$ (sometimes denoted $f \in \mathbf{F}$ for simplicity) is a real-valued function defined over a subset of the model's variables $\alpha \subseteq \mathbf{X}$, known as the function's **scope**, for which the function defines local interactions. More concretely, if we let D_α denote the Cartesian product of the domains of the variables in α, then $f_\alpha : D_\alpha \rightarrow \mathbb{R}^{\geq 0}$. These functions can be expressed as tables for which there is a non-negative real valued output associated with every possible input $d_\alpha \in D_\alpha$ (i.e., every possible joint assignment – or **configuration** – to all of the variables in α).

2.1.2 Notation

We use capital letters (X) to represent variables and lowercase letters (x) to represent their values. Boldfaced capital letters ($\mathbf{X}$) denote a collection of variables, $|\mathbf{X}|$ its cardinality, $D_{\boldsymbol{X}}$ their joint domains, and $\boldsymbol{x}$ a particular realization in that joint domain. Abusing notation slightly, we denote operations on sets of variables $\bigoplus_{\boldsymbol{X}}$ (for example, summation $\sum_{\boldsymbol{X}}$) to indicate,

$$\begin{aligned}\bigoplus_{\boldsymbol{X}} f(\boldsymbol{X}) &= \bigoplus_{\boldsymbol{x} \in D_{\boldsymbol{X}}} f(\boldsymbol{x}) \\ &= \bigoplus_{x_1 \in D_{X_1}} \bigoplus_{x_2 \in D_{X_2}} \cdots \bigoplus_{x_{|\boldsymbol{X}|} \in D_{X_{|\boldsymbol{X}|}}} f(x_1, x_2, \ldots, x_{|\boldsymbol{X}|})\end{aligned} \tag{2.1}$$

Furthermore, given a function f_α with scope α, a super-set of variables β s.t. $\alpha \subseteq \beta$, a particular configuration $\mathbf{b}$ of β, we define

$$f_\alpha(\mathbf{b}) := f_\alpha(\mathbf{b}_\alpha), \tag{2.2}$$

where $\mathbf{b}_\alpha$ is the restriction of the assignments in b to those variables that are in set α.

Many of the algorithms we describe in the sequel operate sequentially along an ***ordering*** o over the variables. Different types of orderings may be useful for different types of algorithms, and are discussed further in the relevant sections. We say that $j >_o i$ if i comes earlier than j in the ordering o.

A Simple Running Example. Consider a simple model that relates temperature and humidity to the chance of rain, and temperature and elevation to the chance of oxygen levels. We select binary variables $\mathbf{X} = \{T, H, R, E, O\}$ to represent these different levels and construct a corresponding graphical model $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$ where:

- T has domain $D_T = \{high, low\}$ representing high and low temperature
- H has domain $D_H = \{high, low\}$ representing high and low humidity
- R has domain $D_R = \{yes, no\}$ representing the presence or absence of rain
- E has domain $D_E = \{high, low\}$ representing the high or low elevation
- O has domain $D_O = \{high, low\}$ representing the high or low oxygen levels.

Our model has five functions:

- $f_T(T)$ representing the probability of the temperature being high or low, $p(T)$,
- $f_H(H)$ representing the probability of the humidity being high or low, $p(H)$,
- $f_E(E)$ representing the probability of the elevation being high or low, $p(E)$,
- $f_{T,H,R}(T, H, R)$ representing the conditional probability of rain given levels of humidity and temperature, $p(R \mid T, H)$,
- $f_{T,E,O}(T, E, O)$ representing the conditional probability of high vs. low oxygen concentrations given the temperature and elevation levels, $p(O \mid T, E)$.

These functions are defined by the following tables, respectively:

T	$p(T)$
high	0.40
low	0.60

H	$p(H)$
high	0.25
low	0.75

E	$p(E)$
high	0.20
low	0.80

T	H	R	$p(R \mid T, H)$
high	*high*	*yes*	0.40
high	*high*	*no*	0.60
high	*low*	*yes*	0.05
high	*low*	*no*	0.95
low	*high*	*yes*	0.80
low	*high*	*no*	0.20
low	*low*	*yes*	0.10
low	*low*	*no*	0.90

T	E	O	$p(O \mid T, E)$
high	*high*	*high*	0.20
high	*high*	*low*	0.80
high	*low*	*high*	0.40
high	*low*	*low*	0.60
low	*high*	*high*	0.25
low	*high*	*low*	0.75
low	*low*	*high*	0.70
low	*low*	*low*	0.30

and we make independence assumptions that allow the joint distribution $P(T, H, R, E, O)$ to factorize into the defined probability functions, i.e.,

$$\begin{aligned} p(T, H, E, R, O) &= p(T) \cdot p(H \mid T) \cdot p(E \mid T, H) \cdot p(R \mid T, H, E) \cdot p(O \mid T, H, E, R) \\ &= p(T) \cdot p(H) \cdot p(E) \cdot p(R \mid T, H) \cdot p(O \mid T, E) \quad \textit{(due to independence assumptions)} \\ &= f_T(T) \cdot f_H(H) \cdot f_E(E) \cdot f_{T,H,R}(T, H, R) \cdot f_{T,E,O}(T, E, O) \\ &= \prod_{f_\alpha \in \mathbf{F}} f_\alpha(\alpha) \end{aligned} \tag{2.3}$$

A possible ordering over these variables is $o = (T, H, R, E, O)$, in which case (for example), we see that $R >_o H$.

2.1.3 Primal Graph

A **primal graph** $\mathcal{G} = \langle \mathbf{X}, \mathbf{E} \rangle$ of a graphical model $\mathcal{M}$ is a graph consisting of set of nodes, each of which is uniquely associated with a variable of $\mathcal{M}$, along with a set of undirected edges $e \in \mathbf{E}$ that connect nodes whose variables both appear in the scope of the same local function. To simplify, we abuse notation by using the same symbols to refer to both the primal graph nodes and to their corresponding variables in $\mathcal{M}$. (For those familiar, the primal graph corresponds to the Markov Random Field graph representation of the model). The primal graph is a useful tool that enables graphical model algorithms to exploit the model's local structure.

Running Example. Our running example model has the primal graph:

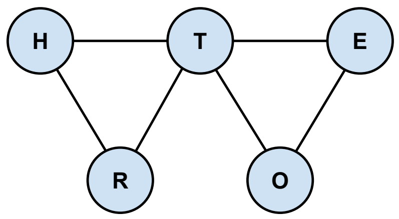


Figure 2.1: Running example primal graph.

We see that the primal graph consists of nodes corresponding to each variable, T, H, R, E and O, and has edges between each pair of $\{T, H, R\}$ since each pair appears together in at least one function $f_\alpha \in \mathbf{F}$, and similarly has edges between each pair of $\{T, E, O\}$. Variables that do not appear in the same functions (e.g., E and H) are not directly connected.

2.1.4 Pseudo Trees

We can also construct a directed tree, called a **pseudo tree** $\mathcal{T} = (\mathbf{V}_\mathcal{T}, \mathbf{E}_\mathcal{T})$, relative to our graphical model $\mathcal{M}$. Like the primal graph, the nodes of the pseudo tree correspond to each variable in $\mathcal{M}$, plus an additional "dummy root" node $\varnothing$, and we again abuse notation by using the same symbols to refer to the pseudo tree nodes as to their corresponding variables.

Definition 2.1 (pseudo tree)
A pseudo tree *of an undirected graph $G = (V, E)$ is a directed rooted tree $\mathcal{T} = (V \cup \{\varnothing\}, E_\mathcal{T})$ such that every arc of G not present in $E_\mathcal{T}$ is a "back-arc" in $\mathcal{T}$. (The edges in $E_\mathcal{T}$ are not required to be elements of E).*

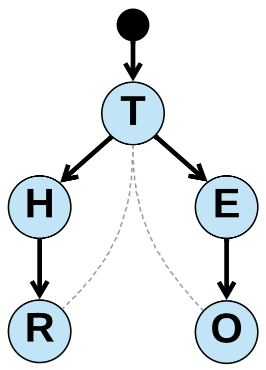


Figure 2.2: An example pseudo tree for the model described in Section **??**: **??** based on ordering T, H, R, E, O. Here the dummy root node is explicitly shown, however it is typically hidden for simplicity.

Furthermore, we say a pseudo tree is consistent with a variable order o if the following holds:

Definition 2.2 (pseudo tree variable ordering)

A pseudo tree $\mathcal{T} = (V \cup \{\varnothing\}, E_{\mathcal{T}})$ of an undirected graph $G = (V, E)$ is said to be consistent with order o if every directed edge $(i \to j) \in E_{\mathcal{T}}$ has $j >_o i$ in ordering o, and

$$j >_o i \text{ and } (i, j) \in E \setminus E_{\mathcal{T}} \quad \Rightarrow \quad i \in anc_{\mathcal{T}}(j),$$

connecting node j in $\mathcal{T}$ to one of its ancestors i in $\mathcal{T}$.

We define $X_i >_o \varnothing$ for all i, i.e., the dummy root node always comes first in a consistent ordering for a pseudo tree.

In other words, the topological structure of the pseudo tree should respect the ordering o, so that all descendants of a node in the tree should come after it in o. It should also respect the structure of G: the back-arc condition ensures that any branching in the pseudo tree indicates a conditional independence relationship visible in the original graph: any dependence (edge) (i, j) in G must connect back to an ancestor of j, and cannot connect "across" to a different branch of the tree.

2.2 Common Graphical Model Queries

There are a plethora of inference queries that can be answered using the graphical model framework. In the context of our work in computational protein *re*-design, we describe three common and related tasks in particular: finding the maximum *a posteriori* (MAP) estimate; determining the partition function, or normalizing constant of the model, denoted Z; and the marginal maximum *a posteriori* (MMAP) estimate, which can be viewed as generalizing the other tasks.

Definition 2.3 (Basic inference tasks)
Given a graphical model $\mathcal{M} = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F})$, we define the computational tasks:

$$\text{Maximum } \textit{a posteriori} \text{ value:} \qquad MAP = \max_{\boldsymbol{X}} \prod_{\boldsymbol{F}} f(\boldsymbol{x}); \tag{2.4}$$

$$\text{Partition function:} \qquad Z = \sum_{\boldsymbol{X}} \prod_{\boldsymbol{F}} f(\boldsymbol{x}); \tag{2.5}$$

$$\text{Marginal MAP value:} \qquad MMAP = \max_{\boldsymbol{Q} \subset \boldsymbol{X}} \sum_{\boldsymbol{S} = \boldsymbol{X} \setminus \boldsymbol{Q}} \prod_{\boldsymbol{F}} f(\boldsymbol{q} \cup \boldsymbol{s}) \tag{2.6}$$

MAP finds the full configuration that maximizes the value of the model and returns the computed value of that configuration. From a probabilistic model standpoint, this corresponds to finding the assignment to the variables that are most likely under the model. Given evidence (i.e., a given assignment to a subset of the variables), the MAP task corresponds to finding the assignment to the rest of the variables that makes the evidence most likely to occur. The **partition function**, **Z**, is mathematical quantity that characterizes the distribution of the model's values among a system's possible configurations $\boldsymbol{x}$. It is often used as a normalization constant for interpreting the model values as (posterior) probabilities. **MMAP** is similar to MAP with the exception that we attempt to maximize over only a subset of variables, called the "query" variables $\boldsymbol{Q}$, assigning each partial configuration a value equal to the sum over all configurations of the remaining ("sum") variables $\boldsymbol{S}$; in a probabilistic model, this value corresponds to the marginal probability of the query configuration.

2.3 Paradigms for Answering Queries

In this section, we will explore three methodologies for answering graphical model queries and discuss several schemes that utilize these methodologies.

2.3.1 Elimination Methods

In this section we will discuss inference methods that are based on systematically processing and removing (i.e., *eliminating*) variables of the model.

2.3.1.1 The Variable Elimination Framework

Many probabilistic graphical model queries can be solved by an inference framework known as **variable elimination** (**VE**). Variable elimination involves an ordered computational processing of the variables of a model, at each step removing the processed variable from further computations (thus called an *elimination* step). Each elimination step corresponds to inference, transferring the influence of information about the eliminated variables over to the remaining variables (in practice, done by creating a new function, defined over the remaining variables, to represent this information).

As an example, consider the query to find the mode of the distribution defined by our running example. Formalizing this query, we want to solve the task:

$$\max_{t,h,e,r,o} p(t,h,e,r,o) \tag{2.7}$$

which, based on Equations 2.3, in terms of our model is equivalent to

$$\max_{t,h,e,r,o} p(t,h,e,r,o) = \max_{t,h,e,r,o} (\ f_T(t) \cdot f_H(h) \cdot f_E(e) \cdot f_{T,H,R}(t,h,r) \cdot f_{T,E,O}(t,e,o)\ ) \tag{2.8}$$

To use variable elimination to solve this query, we first select an **elimination order** – a variable order in which to process and eliminate variables during variable elimination inference. Suppose we select elimination order $o_{elim} = [r, o, e, h, t]$. Then, given this ordering, we express our query as

$$\max_t(\ \max_h(\ \max_e(\ \max_o(\ \max_r(\ f_T(t) \cdot f_H(h) \cdot f_E(e) \cdot f_{T,H,R}(t,h,r) \cdot f_{T,E,O}(t,e,o)\ )\ )\ )\ )\ ) \tag{2.9}$$

where the query can then be solved inside-to-out, variable-by-variable, via computations indicated by the parenthesis. The result from each step can be interpreted as the inference performed over its corresponding variable.

One power of variable elimination is its ability to simplify computation leveraging mathematical properties of the query. Note that in our example some of the model's functions are not dependent on the variable being immediately maximized over and so can be factored out of the respective maximization. Doing so recursively, we can rewrite our query with the same ordering instead as

$$\max_t(\ f_T(t) \cdot \max_e(\ f_E(e) \cdot \max_o(\ f_{T,E,O}(t,e,o)\ )\ ) \cdot \max_h(\ f_H(h) \cdot \max_r(\ f_{T,H,R}(t,h,r)\ )\ )\ ) \tag{2.10}$$

which reduces the size of the functions being maximized over, thus reducing complexity of the computations. (More on this shortly).

▪ 2.3.1.2 Bucket Elimination

Bucket Elimination [Dechter, 1999], or **BE**, is a variable elimination scheme that can be adapted for a myriad of graphical model tasks including those described in Section 2.2: Common Graphical Model Queries.

Bucket elimination performs variable elimination according to a given elimination order by processing what are called **buckets** one-by-one, each corresponding to a variable in the provided ordering. When reaching some variable X_i in the ordering, all unprocessed functions that contain X_i in their scope are placed in bucket B_i (this includes the model's original functions as well as messages generated during the bucket elimination process). The bucket is then processed by applying an elimination operation (generically indicated by $\bigoplus$) over X_i to the combination of the bucket functions (generically indicated by $\bigotimes$), resulting in a **bucket message** denoted $\lambda_{i\to j}$, or λ_i for short:

$$\lambda_{i\to j} = \bigoplus_{X_i} \bigotimes_{f_\alpha \in B_i} f_\alpha(\alpha). \tag{2.11}$$

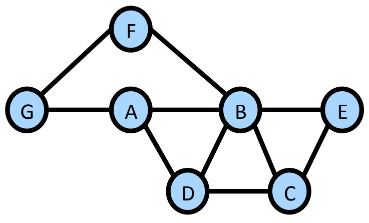
More concretely, for the inference task of computing the partition function, this corresponds to marginalizing X_i from the product of the functions,

$$\lambda_{i\to j} = \sum_{X_i} \prod_{f_\alpha \in B_i} f_\alpha(\alpha); \tag{2.12}$$

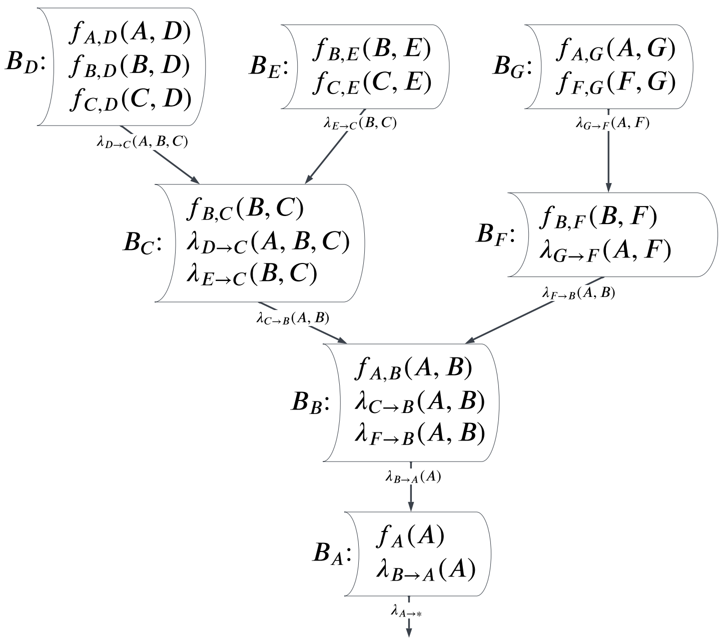
in the context of computing the MAP, we instead maximize the product of the functions over X_i:

$$\lambda_{i\to j} = \max_{X_i} \prod_{f_\alpha \in B_i} f_\alpha(\alpha). \tag{2.13}$$

The index i in $\lambda_{i\to j}$ refers to the bucket that generated the message, while j indicates the bucket this message is sent to – specifically, the next variable in the elimination ordering



(a) Example primal graph of a graphical model with 7 variables and model functions $F = \{f_A(A), f_{A,B}(A,B), f_{A,D}(A,D), f_{A,G}(A,G), f_{B,C}(B,C), f_{B,D}(B,D), f_{B,E}(B,E), f_{B,F}(B,F), f_{C,D}(C,D), f_{C,E}(C,E), f_{F,G}(F,G)\}$.


B_D: $f_{A,D}(A,D)$ $f_{B,D}(B,D)$ $f_{C,D}(C,D)$
B_E: $f_{B,E}(B,E)$ $f_{C,E}(C,E)$
B_G: $f_{A,G}(A,G)$ $f_{F,G}(F,G)$
$\lambda_{D\to C}(A,B,C)$
$\lambda_{E\to C}(B,C)$
$\lambda_{G\to F}(A,F)$
B_C: $f_{B,C}(B,C)$ $\lambda_{D\to C}(A,B,C)$ $\lambda_{E\to C}(B,C)$
B_F: $f_{B,F}(B,F)$ $\lambda_{G\to F}(A,F)$
$\lambda_{C\to B}(A,B)$
$\lambda_{F\to B}(A,B)$
B_B: $f_{A,B}(A,B)$ $\lambda_{C\to B}(A,B)$ $\lambda_{F\to B}(A,B)$
$\lambda_{B\to A}(A)$
B_A: $f_A(A)$ $\lambda_{B\to A}(A)$
$\lambda_{A\to *}$

(b) Bucket elimination schematic following an elimination order $o_{elim} = [D, E, G, C, F, B, A]$.

Figure 2.3: (a) A primal graph of a graphical model with 7 variables. (b) Illustration of BE with an ordering A B C E D F G.

that is also in the scope of the message.

The processed buckets and messages can then be used to compute the result of the corresponding query; in the case of computing the partition function or MAP, the result is simply the combination of the final messages. Figure 2.3 shows a schematic of bucket elimination on a graphical model with variables indexed from A to G and with pair-wise functions over the pairs of variables that are connected by an edge in the underlying primal graph (Figure 2.3a), namely: $F = \{f_A(A),\ f_{A,B}(A,B),\ f_{A,D}(A,D),\ f_{A,G}(A,G),\ f_{B,C}(B,C),\ f_{B,D}(B,D),\ f_{B,E}(B,E),\ f_{B,F}(B,F),\ f_{C,D}(C,D),\ f_{C,E}(C,E),\ f_{F,G}(F,G)\}$.

Bucket elimination can be viewed as a message-passing procedure defined along the structure of the **bucket tree** (Figure 2.3b). The nodes of the tree represent the different buckets. Each bucket corresponds to a single variable, and contains a subset of the model's functions, which may depend on the chosen order of processing. There is an arc from bucket B_i to a "parent bucket" B_j, if the function created at bucket B_i is placed in bucket B_j. Then, beginning at the leaves, each bucket computes its message λ_i, and passes it to bucket B_i's parent B_j; once B_j has recieved all its children's messages, it can compute its message λ_j.

In the context of optimization, bucket elimination is an example of the general framework of dynamic programming, in which we solve large problems by first solving smaller subproblems and saving their solutions, which can then be used efficiently when solving the larger problem. Each bucket corresponds to a subproblem; by solving the subproblems at the leaves, we simplify solving the optimization problem at their parents, and so on.

The complexity of bucket elimination depends on a quantity called a bucket width:

Definition 2.4 (bucket width)
The bucket width of a bucket B is equal to the number of variables involved in the functions contained within a bucket prior to processing, $|\bigcup_{f_\alpha \in B} \alpha|$.

Complexity. Both the time and space complexity of bucket elimination are exponential in the maximal bucket width, which is closely related to a graph parameter of the model called the **induced width** [Dechter, 2019]. In particular, the induced width according to a particular variable ordering is equal to the one less than the maximal bucket width of Bucket Elimination operating on that same ordering. Given that Bucket Elimination complexity is exponential in this width, Bucket Elimination becomes intractable for models with high induced width, and thus approximation schemes have been developed in response (see Section 2.3.1.4).

▨ 2.3.1.3 Induced Width

The time and space complexity of computing the message λ_i from a given bucket i depends on the scope of the message: for discrete functions, λ_i is a table over all possible configurations of its scope, and thus requires time and memory exponential in the size of the scope. In general, the largest such scope dominates the computational complexity of the procedure, which in turn depends on the elimination order being used; hence, this largest scope is called the **induced width** of the ordering [Dechter, 2019]. Identifying the minimal width over all possible elimination orderings can be framed as the graph-theoretic problem of finding the **tree-width** of a primal graph G. While finding the tree-width of G is itself a computationally difficult problem, there are many heuristic approaches for identifying good elimination orderings that correspond to low induced widths [Dechter, 2019].

▨ 2.3.1.4 Mini-Bucket and Weighted Mini-Bucket Bounding Schemes

Bucket Elimination provides an efficient dynamic programming implementation of variable elimination, but the processing of each bucket is still exponential in the number of variables it contains. This makes Bucket Elimination computationally intractable for models with high induced width.

Mini-Bucket Elimination [Dechter and Rish, 2002] addresses this via simple modification to the Bucket Elimination scheme: whenever a bucket is encountered with width larger than a provided i-bound (i.e., when the number of distinct variables in the bucket's functions is greater than the provided i-bound), the bucket is split into smaller *mini-buckets*. Consider the computation of a bucket message by Bucket Elimination from a bucket processing variable X to a next bucket Y

$$\lambda_X = \bigoplus_X \prod_{f_\alpha \in B_X} f_\alpha(\alpha).$$

Let $\{B_X^{(1)}, ..., B_X^{(J)}\}$ be a particular partitioning of bucket B_X so that no resulting B_X^j has more than i variables. Then a processing according to this partitioning would yield

$$\hat{\lambda}_X = \prod_{j \in 1,\ldots,J} \bigoplus_X \prod_{f_\alpha \in B_X^{(j)}} f_\alpha(\alpha).$$

When $\bigoplus \in \{\max, \min, \sum\}$, the approximate result $\hat{\lambda}_X$ provides a bound on the desired bucket function λ_X – an upper bound for the max and sum operations, and a lower bound for min. These bounds are called decomposition bounds since $\hat{\lambda}_X$ is computed by breaking the bucket into several overlapping components, which are then treated independently ("decomposed").

Mini-Bucket elimination follows this intuition with one refinement: in the case of summation, the algorithm uses $\bigoplus = \sum$ for only one of the resulting mini-buckets, and uses $\bigoplus = \max$ for the rest. The decomposition upper bound still holds since

Proposition 2.1 (Decomposition bounds on summation)
For non-negative functions f,

$$\sum_X f(x)g(x) \ \leq \ \sum_X f(x) \cdot \max_X g(x) \ \leq \ \sum_X f(x) \cdot \sum_X g(x) \qquad (2.14)$$

and

$$\sum_X f(x)g(x) \ \geq \ \sum_X f(x) \cdot \min_X g(x) \qquad (2.15)$$

which can yield tighter bounds. Furthermore, the above decomposition bounds allow for a lower bounding of summations by mini-buckets by summing over one mini-bucket and minimizing over the rest.

The Weighted Mini-Bucket Elimination scheme [Liu and Ihler, 2011b] improves on these bounds further by leveraging Holder's inequality [Hardy et al., 1988]:

Proposition 2.2 (Holder's inequality)

Let the power sum of $f(x)$ be defined as

$$\sum_x^w f(x) = \Big(\sum_x f(x)^{\frac{1}{w}}\Big)^w \tag{2.16}$$

Let $f_i(x)$, $i = 1...r$ be a set of functions and $w_1, ..., w_r$ be a set of positive weights, such that, $w = \sum_{i=1}^r w_i$. Then,

$$\sum_x^w \prod_{i=1}^r f_i(x) \leq \prod_{i=1}^r \sum_x^{w_i} f_i(x) \tag{2.17}$$

Using the notion from Holder's inequality, we now define a power sum over a mini bucket with the next two definitions:

Definition 2.5 (Consolidated Mini-Bucket Function)

Consider a mini-bucket $B_X^{(j)}$. We define its consolidated mini-bucket function as

$$f_{B_X^{(j)}} := \prod_{f \in B_X^{(j)}} f \tag{2.18}$$

Definition 2.6 (Mini-Bucket Power Sum)

The power sum of a mini-bucket $B_k^{(t)}$ is defined as

$$\sum_X^w f_{B_X^{(j)}} := \Big(\sum_X \big(f_{B_X^{(j)}}\big)^{\frac{1}{w}}\Big)^w. \tag{2.19}$$

In Weighted Mini-Bucket Elimination, when processing any bucket B_X with width larger than a provided i-bound (i.e., when the number of distinct variables in the bucket's functions is greater than the provided i-bound), a bounded approximation is made by partitioning the bucket functions into J mini-buckets $B_X^{(j)}$ and taking a power-sum over the bucket variable in each. Algorithm 1 shows the Weigthed Mini-Bucket Elimination algorithm.

Algorithm 1: `wMBE for Summation`

```
Input: Graphical model $\mathcal{M}$;  i-bound $i$;
elimination order $o_{elim} = [X_1, ..., X_n]$
Output: upper bound on Z: $ub_Z(\mathcal{M})$
begin
    Partition the functions $f \in \boldsymbol{F}$ into buckets $B_1, ..., B_n$ s.t. each function is placed in the bucket
      corresponding to the lowest-index variable in its scope.
    foreach $k = 1...n$ do
        Generate a mini-bucket partitioning of the bucket functions $\boldsymbol{MB_k} = \{MB_k^{(1)}, ..., MB_k^{(T)}\}$ s.t.
          $|scope(f_{MB_k^{(t)}})| \leq i$, for all $MB_k^{(t)} \in \boldsymbol{MB_k}$
        if $X_k \in \boldsymbol{MAP}$ then
            foreach $MB_k^{(t)} \in \boldsymbol{MB_k}$ do
                $\lambda_k^{(t)} \leftarrow \max_{X_k} f_{MB_k^{(t)}}$
            end
        else
            Select positive weights $\boldsymbol{w} = \{w_1, ..., w_T\}$ s.t. $\sum_{w_t \in \boldsymbol{w}} w_t = 1$
            foreach $MB_k^{(t)} \in \boldsymbol{MB_k}$ do
                $\lambda_k^{(t)} \leftarrow \sum_{X_k}^{w_t} f_{MB_k^{(t)}}$
            end
        end
        Add each $\lambda_k^t$ to the bucket of the lowest-index variable in its scope.
    end
    return $\lambda_n = ub_Z(\mathcal{M})$
end
```

In both Mini-Bucket Elimination and Weighted Mini-Bucket Elimination schemes, estimates can be improved by cost-shifting across functions of a bucket using Lagrange multipliers [Liu and Ihler, 2011b, Ihler et al., 2012].

■ 2.3.1.5 Weighted Mini-Bucket Elimination for Marginal MAP

Weighted Mini-Bucket Elimination can also be used to bound the Marginal MAP of a graphical model [Marinescu et al., 2014]. The two key changes that are needed are: (1) the elimination order needs to place variables that are to be marginalized before variables that are to be maximized (this is sometimes referred to as a *constrained elimination order*), and (2) buckets for variables that are to be maximized (these variables are called MAP variables) should be processed using a maximization operation rather than a summation operation over the bucket variable. Algorithm 2 shows the procedure for `wMBE-MMAP`, Weighted Mini-Bucket Elimination for Marginal MAP.

Algorithm 2: `wMBE-MMAP`

```
Input: Graphical model $\mathcal{M}$; i-bound $i$;
constrained elimination order $o_{elim} = [X_1, ..., X_n]$ placing $MAP$ variables last
Output: upper bound on the MMAP: $ub_{MMAP}(\mathcal{M})$
1  begin
2    Partition the functions $f \in \boldsymbol{F}$ into buckets $B_1, ..., B_n$ s.t. each function is placed in the bucket
       corresponding to the lowest-index variable in its scope.
3    foreach $k = 1...n$ do
4      Generate a mini-bucket partitioning of the bucket functions $\boldsymbol{MB_k} = \{MB_k^{(1)}, ..., MB_k^{(T)}\}$ s.t.
         $|scope(f_{MB_k^{(t)}})| \leq i$, for all $MB_k^{(t)} \in \boldsymbol{MB_k}$
5      if $X_k \in \boldsymbol{MAP}$ then
6        foreach $MB_k^{(t)} \in \boldsymbol{MB_k}$ do
7          $\lambda_k^{(t)} \leftarrow \max_{X_k} f_{MB_k^{(t)}}$
8        end
9      else
10       Select positive weights $\boldsymbol{w} = \{w_1, ..., w_T\}$ s.t. $\sum_{w_t \in \boldsymbol{w}} w_t = 1$
11       foreach $MB_k^{(t)} \in \boldsymbol{MB_k}$ do
12         $\lambda_k^{(t)} \leftarrow \sum_{X_k}^{w_t} f_{MB_k^{(t)}}$
13       end
14     end
15     Add each $\lambda_k^t$ to the bucket of the lowest-index variable in its scope.
16   end
17   return $\lambda_n = ub_{MMAP}(\mathcal{M})$
18 end
```

2.3.2 Search

In this section, we describe how a graphical model can be converted into a weighted search space upon which a variety of search schemes can be used to solve inference tasks.

2.3.2.1 OR Search Spaces

A graphical model can be cast into a search space in order to explore different configurations of the model. Figure 2.4 shows a classical search space (also known as an *OR search space* or a *State Tree*) of our running example model, corresponding to a search order that explores

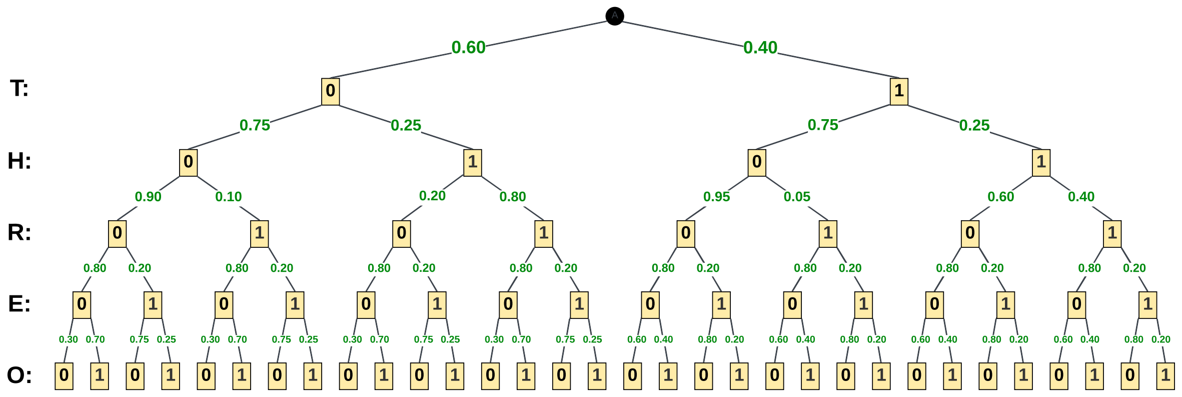


Figure 2.4: Example classical OR search space for our running example graphical model, corresponding to a search order that explores possible assignments to variable T, then H, then R, then E, then O. For simplicity, we abbreviate domain values of *low* or *no* instead with the value [0], and *high* or *yes* with [1].

possible assignments to variable T, then H, then R, then E, then O. (For compactness, we abbreviate domain values of *low* or *no* instead with the value 0, and *high* or *yes* with 1).

As we follow a path down the tree, each successive level corresponds to an assignment to the next variable in the ordering. Thus a path from the dummy root to a leaf corresponds to a **full configuration**. Given that the search tree was built to represent a model and its factorized global function (in this case a factorized probability distribution), the search tree is constructed so that the arc into a node n associated with variable X has a cost $c(n)$ equal to the product of functions $f_\alpha \in \boldsymbol{F}$ such that the path to n_X fully instantiates all $X' \in \alpha$ and such that $X \in \alpha$ [Dechter and Mateescu, 2007]. In other words, $c(n)$ equals the product of functions $f_\alpha \in \boldsymbol{F}$ such that the variable represented by n is in f's scope, and the path to n contains the assignment to every other variable in its scope. If no such functions exist, the arc is vacuously assigned the multiplicative identity, i.e., the constant value one (1.00).

More formally:

Definition 2.7 (State Space Tree for a graphical model)

Given a graphical model $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$, *where* $o = (X_1, ..., X_L)$ *is an ordering of its* *variables $\boldsymbol{X}$ having domains $D = \{D_1, ..., D_L\}$, a (weighted) state tree $T = ((V, E), c : E \to \mathbb{R}^{\geq})$ is a directed tree s.t. any node n_p in V at depth p is labeled by a value in the domain D_p. The child nodes of n in T are denoted $ch(n)$. A path π from the root to n_p corresponds to a partial assignment $\bar{x}_p = (x_1, ...x_p)$. Each arc (n_p, n_{p+1}) is associated with a positive cost $c(n_p, n_{p+1})$. The costs are extracted from $\boldsymbol{F}$ as described previously. A solution is a path of length L denoting an assignment to all the variables $\bar{x}_L = (x_1, ..., x_L)$ and its cost $C(\bar{x}_n) \propto \prod_{i=0}^{L-1} c(n_i, n_{i+1})$. Since we will mostly deal with search trees, we will associate the cost labeling each arc, denoted $c(n_i, n_{i+1})$, with its destination node, denoted $c(n_{i+1})$.*

The state tree represents a structured enumeration of all possible full configurations (the leaves of the tree) and partial configurations respecting the ordering o (internal nodes), with the model's factors distributed over the edges of the tree via the arc costs c.

Unfortunately, the OR state tree representation does not capture many of the conditional independence relationships inherent in the model's graph structure. To better represent this information, we can use a so-called AND/OR representation that can express subproblem independence.

2.3.2.2 AND/OR Search Spaces

Often, assignments to earlier variables in the search ordering results in conditional independencies between sets of variables searched at subsequent layers. For example, given our running example model, conditioning on variable T (i.e., giving an assignment to T) causes E and O to become independent of H and R. In the OR search space we can see the consequences of this independence by noticing that the edge cost into and sub-tree under nodes of E having the same assignment to T but different assignments to H and R are identical. (Figure 2.5 shows this effect more explicitly.)

We can take advantage of such conditional independencies to construct a more compact

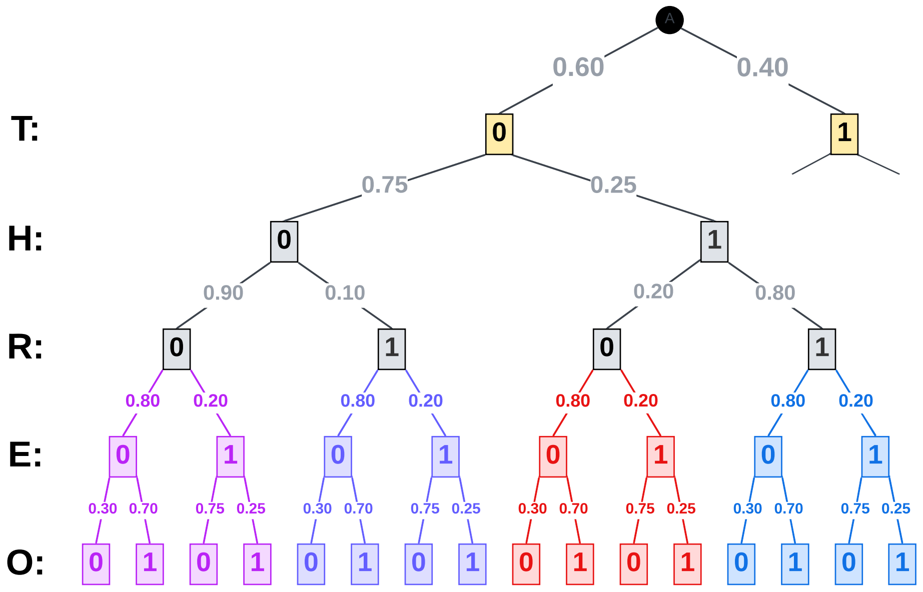


Figure 2.5: Conditional independence of E and O from H and R given assignment $T = 0$ is shown in the search space from Figure 2.4. Notice that each distinct assignment to H and R leads to equivalent sub trees of E and O (each highlighted in a different color for easy comparison).

search space known as an AND/OR search space [Dechter and Mateescu, 2007] and facilitates more effective algorithms [Marinescu and Dechter, 2009c]. Since such conditional independencies are inherently captured by pseudo trees (Section 2.1.4: Pseudo Trees), we can use a pseudo tree to guide the construction of the AND/OR search space.

Given a pseudo tree $\mathcal{T}$ of a primal graph G, the *AND/OR search tree* $T_{\mathcal{T}}$ guided by $\mathcal{T}$ has alternating levels of OR nodes corresponding to the variables, and AND nodes corresponding to an assignment from the variables' domains with edge costs extracted from the original functions $\boldsymbol{F}$ in the same way as before[Dechter and Mateescu, 2007]. Let n be an AND node in $T_{\mathcal{T}}$. So if n stands for $\bar{x}_{1..p} = (X_1 = x_1, X_2 = x_2, ..., X_P = x_p)$, then $var(n) = X_p$. Each AND node n has a cost $c(n)$ defined to be the product of all factors $f_\alpha \in \boldsymbol{F}$ that are instantiated at n but not before.

Figure 2.6 shows the AND/OR search space that results from using the pseudo tree from

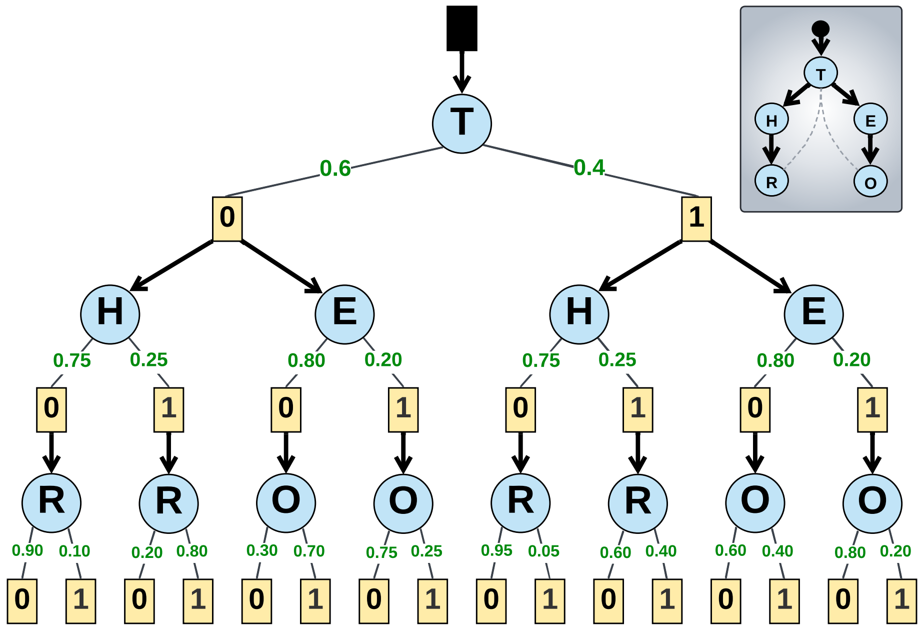
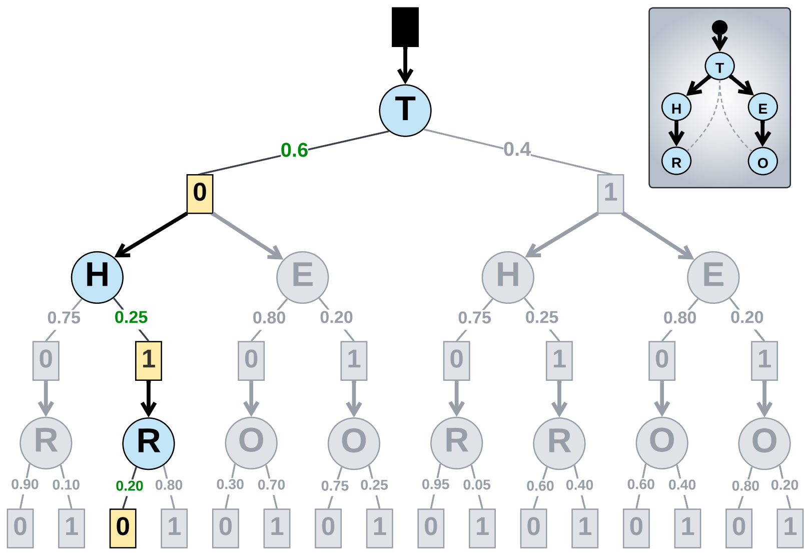


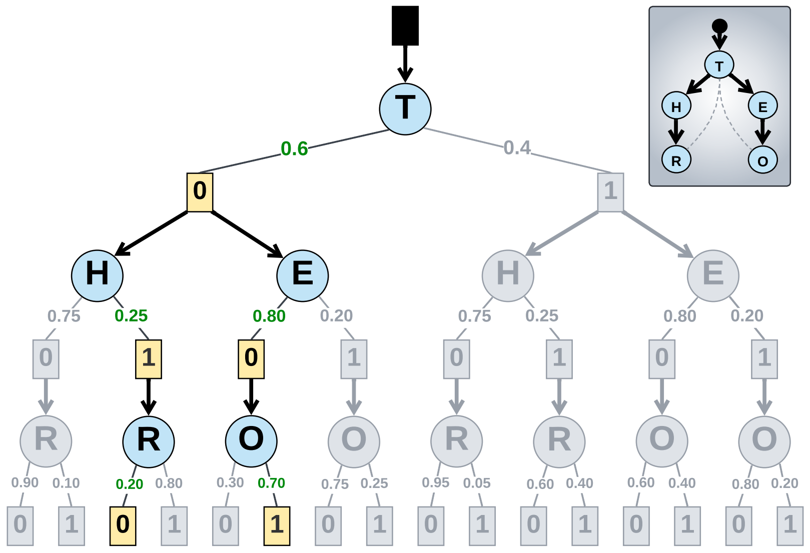
Figure 2.6: Example of an AND/OR search space for our runnin example model, guided by the pseudo tree from Section 2.1.4: Pseudo Trees. For simplicity, we abbreviate domain values of *low* or *no* instead with the value [0], and *high* or *yes* with [1]. We see that the search tree is more compact than that of the OR search tree in Figure 2.4.

Figure Figure 2.2 as a guide. In an AND/OR search space, each OR node (blue circles) is associated with a variable in the model, and each AND node (yellow rectangle) with a value (assignment) to its parent OR node's variable. Branching in the guiding pseudo tree, which represents conditional independence, is mirrored by branching in the AND/OR search space: two or more OR node children under an AND node. In the example provided, we see a branching under T in the guiding pseudo tree that captures the conditional independence of H and R from E and O given assignment to T. In the corresponding AND/OR search space, under each assignment of T (namely under each AND child node of T) we see branches leading to distinct sub-trees – one for H and R, and one for E and O. By capturing the subproblem independence, the search space is made far more compact.

Unlike in an OR search tree, in the AND/OR search tree a single path from some leaf to



(a) Paths from root to leaves in AND/OR search spaces do not necessarily correspond to full configurations. Here, the highlighted path captures partial configuration $T = 0, H = 1, R = 0$, but omits assignments to E and O.



(b) To capture a full configuration in an AND/OR search space, we must capture *all* variables that branch from paths extended leading to a sub-tree of the full search space that includes all variables of the model.

Figure 2.7: (a) An example AND/OR search space, in which a path corresponds to only a partial configuration of the variables in the model. (b) A full configuration, captured by a sub-tree over all variables.

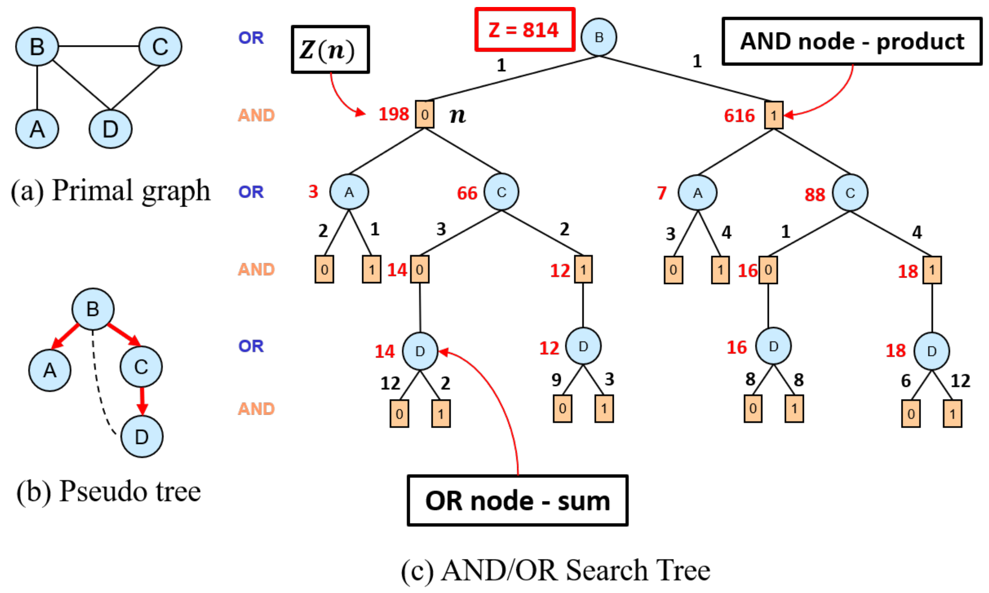


Figure 2.8: For (a) a small (four binary-valued variable) graphical model, we show (b) a pseudo-tree and (c) the AND/OR search tree associated with this pseudo-tree. Each OR node is associated with a variable, and each AND node with a value of its parent OR node's variable. Also shown are the edge costs (black) and the value function at each node (red); see main text for details.

the root in an AND/OR search space may not represent a full configuration. For example, the path from root to leaf highlighted in Figure 2.7a captures a *partial* configuration corresponding only to assignments $T = 0, H = 1, R = 0$, omitting assignments to E and O.

Definition 2.8 (Solution sub-tree)

A solution sub-tree $\tilde{S}_{\mathcal{T}}$ *(sometimes more simply called a solution tree) is a sub-tree of* $T_{\mathcal{T}}$ *satisfying: (1) it contains the root of* $T_{\mathcal{T}}$*; (2) if an OR node is in* $\tilde{S}_{\mathcal{T}}$*, exactly one of its AND child nodes is in* $\tilde{S}_{\mathcal{T}}$*; (3) if an AND node is in* $\tilde{S}_{\mathcal{T}}$ *then all its OR children are in* $\tilde{S}_{\mathcal{T}}$*. The product of the node costs on any full solution sub-tree is the solution cost. It is equal to the cost of a full configuration of the model* $\mathcal{M}$ *defining the AND/OR tree* $T_{\mathcal{T}}$ *[Dechter and Mateescu, 2007].*

Instead, a full configuration in an AND/OR tree corresponds to a *sub-tree* (for example, the sub-tree indicated in Figure 2.7b for $T = 0, H = 1, R = 0, E = 0, O = 1$) with the property that, at any OR node in the sub-tree, at least one child AND node is included in the sub-tree, and at any included AND node at which branching occurs in the search tree, *all* child OR nodes are included in the sub-tree. The cost of a full configuration in an AND/OR sub-tree can be computed by applying the model's combination operation (e.g., multiplication) to the cost of each arc traversed.

Example 2.3.2.1

Figure 2.8a is a primal graph of 4 bi-valued variables and 4 binary factors of a graphical model. Figure 2.8b shows one possible pseudo tree for the graph. Finally, Figure 2.8c displays the AND/OR search tree guided by that pseudo tree. One possible solution sub-tree is $(B = 1, A = 0, C = 1, D = 0)$*, which has a cost of* $(1 \cdot (3) \cdot (4 \cdot 6)) = 72$.

When the pseudo tree is a chain (i.e., contains no branching), the AND/OR tree reduces to the traditional OR tree representation (if we ignore the presence of the OR nodes), in which each path to a leaf corresponds to a full variable configuration.

Definition 2.9 (chain pseudo tree)

A chain pseudo tree is a pseudo tree such that all variables are connected via a single directed path, and thus the pseudo tree contains no branching.

For example, the OR search space from Figure 2.4 can be explicitly represented as the AND/OR search space shown in Figure 2.9.

Complexity. The size of the AND/OR search tree, $T_{\mathcal{T}}$, is exponential in the height of the pseudo tree $\mathcal{T}$. It is possible to merge some sub-trees using a concept known as *context* (defined in the sequel), which yields an AND/OR *graph* (as opposed to a tree) whose representation is exponential in the tree-width of the primal graph [Dechter and Mateescu, 2007]. The function $V(n)$ can be computed recursively from leaves to root by a depth-first search scheme (see Figure 2.8, in which the values are annotated by each node in red).

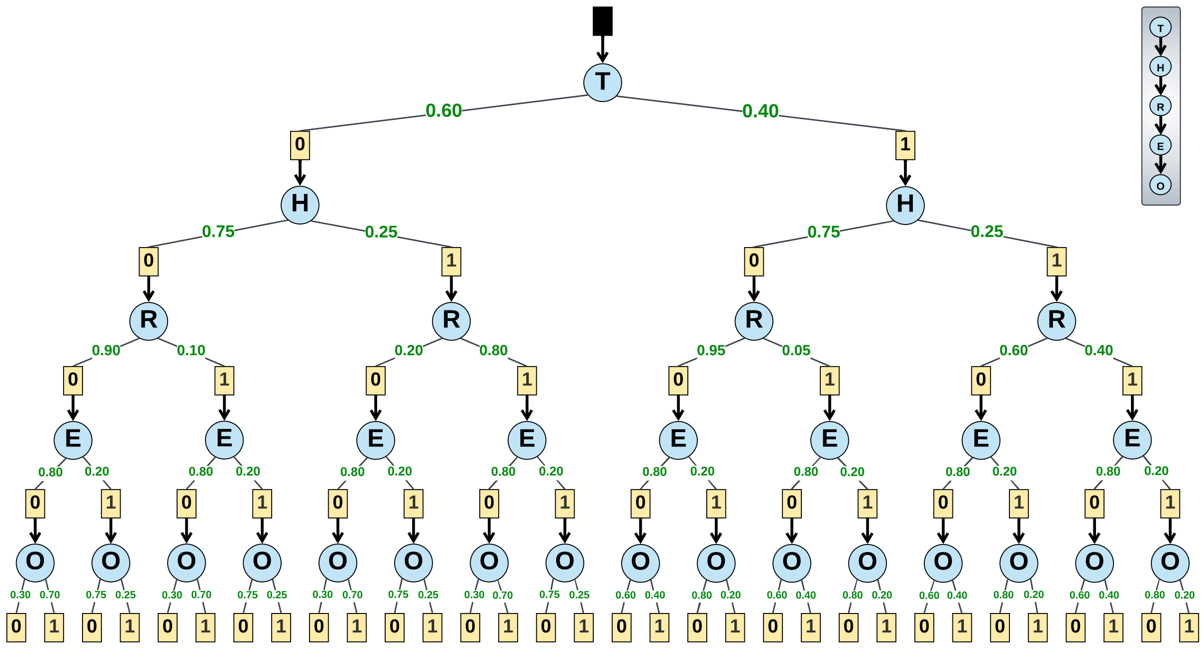


Figure 2.9: The OR search space from Figure 2.4 expressed explicitly as an AND/OR search space.

Since OR trees can be viewed as a special case of AND/OR trees, we can also use the terminology of AND/OR trees to discuss OR trees.

2.3.2.3 Search Space Notation

We use T to refer to a weighted state-space tree. More specifically, $T_{\mathcal{T}}$ refers to a state-space tree constructed based on the pseudo-tree $\mathcal{T}$. Within the context of search, we use n to indicate a (generic) node in the search tree; the set $ch(n)$ are the children of node n. To index a specific node in an AND/OR tree, we can use its corresponding partial configuration. For example, the OR node corresponding to the partial configuration $\{A\!=\!0, C\!=\!1\}$ can be identified as $n_{A=0,C=1}$. Similarly, its parent AND node (at which C has not yet been assigned) can be identified as $n_{A=0,C}$, or (abusing notation slightly) $C_{n_{A=0}}$. We use n_X to refer to a generic AND node associated with X; so, n_C might be any of the nodes $n_{A=a,C=c}$. Similarly, Y_n indicates the OR node associated with variable Y that is the child of AND node n.

For any node n, $path(n)$ is the partial configuration given by assigning each variable its corresponding value according to the assignments along the path from the root to node n. For example, the highlighted node n in Figure 2.10b has $path(n) = \{A{=}0, C{=}1\}$. The set $varpath(n)$ consists of the variables to which $path(n)$ provides a configuration. In Figure 2.10b, $varpath(n) = \{A, C\}$.

The cost of the arc to an AND node n_X is

$$c(n_X) = \prod_{f \in \{f_{\boldsymbol{\alpha}} \in \boldsymbol{F} \mid \boldsymbol{\alpha} \subseteq varpath(n_X),\, X \in \boldsymbol{\alpha}\}} f(path(n_X)). \tag{2.20}$$

or 1, vacuously. Letting $anc(n)$ be the AND node ancestors of n in the search tree, the cost of $path(n)$ is $g(n) = \prod_{n' \in anc(n)} c(n')$. In Figure 2.10b, $g(n) = 10 \cdot 5$.

2.3.2.4 Important Search Space Quantities

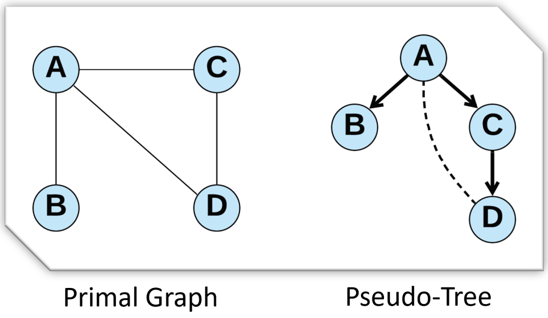
We now define some important quantities involved in evaluating search spaces.

V(n). Each node n in $T_{\mathcal{T}}$ (OR node or AND node) can be associated with a *value*, which is the sum of the cost of all solution sub-trees rooted at n. For an AND node n_X with children OR nodes $Y_{n_X} \in ch(n_X)$, $V(n_X)$ satisfies

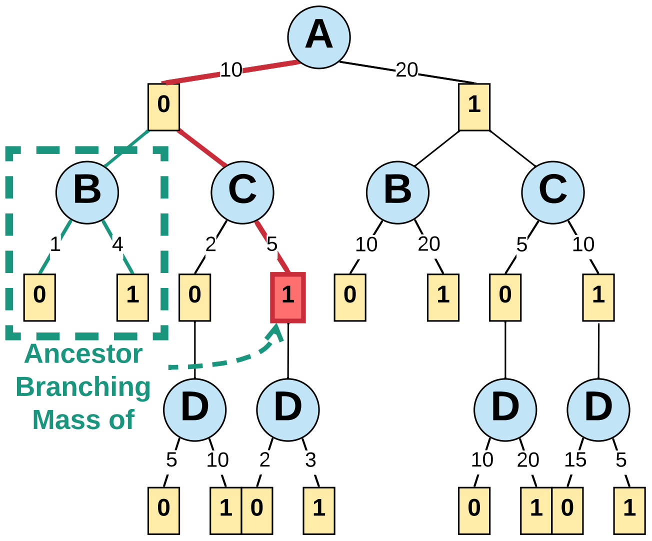
$$V(n_X) = \prod_{Y_{n_X} \in ch(n_X)} V(Y_{n_X}) \tag{2.21}$$

such that for OR nodes Y_{n_X}

$$V(Y_{n_X}) = \bigoplus_{n_Y \in ch(Y_{n_X})} c(n_Y) \cdot V(n_Y) \tag{2.22}$$



(a)



(b)

Figure 2.10: A full AND/OR tree representing 16 possible full configurations of binary variables A, B, C, and D guided by the pseudo tree shown in subfigure (a) above. The path cost for the highlighted node $n_{A=0,C=1}$ at the end of the path $\rightarrow(A{=}0)\rightarrow(C{=}1)$ is $g(n_{A=0,C=1}) = 10\cdot5$. The value of the sub-tree under $n_{A=0,C=1}$ is $V(n_{A=0,C=1}) = 2\cdot1 + 3\cdot1$. Boxed in green is the ancestor branching sub-tree for $n_{A=0,C=1}$ and it has value $R(n_{A=0,C=1}) = 1\cdot1 + 4\cdot1$. Thus, $Q(n_{A=0,C=1}) = (10\cdot5)\cdot(1\cdot1 + 4\cdot1)\cdot(2\cdot1 + 3\cdot1)$.

where $\bigoplus$ is the appropriate elimination operator given the respective query (e.g., $\sum$ when computing the partition function, max when computing a MAP task), and with $V(n_X) = 1$ in the case that n_X has no children.

In the case of OR trees the value of a node coincides with the result of performing inference on the model $\mathcal{M}$ conditioned on the configuration corresponding to $path(n)$. In the case of AND/OR trees it captures the value restricted to a subset of the variables below it in the pseudo-tree.

Note that given $n_\varnothing$ as the dummy root node of AND/OR tree T, $V(n_\varnothing)$ is equal to the target query value of the underlying model $\mathcal{M}$. We denote estimation of $V(n)$ as $\hat{V}(n)$. Heuristic estimates of $V(n)$ are more specifically denoted as $h(n)$.

R(n). On the path from the root of an AND/OR tree T to some node n_X, there may be an intermediate node n_Y associated with branching variable Y in the guiding pseudo tree $\mathcal{T}$.

Definition 2.10 (Psuedo Tree Branching Variable)
A variable in a pseudo tree $\mathcal{T}$ is branching variable if it has more than one child.

(In Figure 2.10b, on the path to the highlighted node $n_{A=0,C=1}$, node $n_{A=0}$ is traversed where A is a branching variable in $\mathcal{T}$ of Figure 2.10a). When this happens, the remaining variables of the model are split between different branches. Thus, the $V(n)$ of any node down one of the branches will necessarily omit the costs from the configurations of the variables included in the other branch(es). $R(n_X)$, or the *ancestor branching mass*, captures these omitted costs. (In Figure 2.10b, the green box shows the portion of T corresponding to $R(n_{A=0,C=1})$).

More formally, let $br(n_X)$ be the set of ancestor nodes n_{Y_i} of n_X such that each Y_i is a branching variable ancestor of X in $\mathcal{T}$. We then define $R(n_X)$ simply as:

$$R(n_X) = \prod_{n_Y \in br(n_X)} \prod_{\substack{W_{n_Y} \in ch(n_Y) \\ W_{n_Y} \notin path(n_X)}} V(W_{n_Y}), \tag{2.23}$$

(In Figure 2.10b, $br(n_{A=0,C=1}) = \{n_{A=0}\}$, A being the only branching variable ancestor of C in $\mathcal{T}$, and $B_{n_{A=0}}$ the only respective child OR node ***not*** not on the path to $n_{A=0,C=1}$. Thus, $R(n_{A=0,C=1}) = V(B_{n_{A=0}})$). We denote approximations to $R(n)$ as $r(n)$.

Q(n). We can now concisely define a quantity $Q(n)$ as the contribution to the value of the root from all full configurations consistent with $path(n)$. The quantity $Q(n)$ obeys:

$$Q(n) = g(n) \cdot R(n) \cdot V(n). \tag{2.24}$$

For the partition function task in particular, $Q(n)$ is the unnormalized probability measure of the partial configuration $path(n)$, with $P(path(n)) = \frac{Q(n)}{Z}$. $Q(n)$ can also be thought of as $Z_{|path(n)}$.

We denote approximations to $Q(n)$ as $\varrho(n)$.

Example 2.3.2.2
In the AND/OR tree in Figure 2.10b for computing the partition function, consider the path from the root to the red node $n_{A=0,C=1}$. Following $n_{A=0}$ to our node, we see OR node $B_{n_{A=0}}$ branches off of the path. So,

$$\begin{aligned} Q(n_{A=0,C=1}) &= g(n_{A=0,C=1}) \cdot R(n_{A=0,C=1}) \cdot V(n_{A=0,C=1}) \\ &= g(n_{A=0,C=1}) \cdot V(B_{n_{A=0}}) \quad \cdot V(n_{A=0,C=1}) \\ &= (10 \cdot 5) \qquad \cdot (1 \cdot 1 + 4 \cdot 1) \quad \cdot (2 \cdot 1 + 3 \cdot 1) \end{aligned}$$

■ 2.3.3 Sampling

Monte Carlo methods are a statistical technique for estimating expectations (or more generally, summation queries) using random sampling. The basic Monte Carlo estimator uses an average over a collection of i.i.d. samples to approximate the expected value of some function $u(x)$ over a distribution $p(x)$, i.e.,

$$\mathbb{E}_p\big[u(x)\big] = \sum_x p(x)u(x) \approx \frac{1}{m} \sum_j u(x^{(j)}) \quad \text{where} \quad \{x^{(j)} \sim p(x), j = 1 \dots m\}.$$

Unfortunately, in many applications, it may be difficult to sample from the distribution of interest $p(x)$; for example, $p(x)$ may be represented only implicitly using a graphical model. A technique called *importance sampling* lets us sidestep this difficulty.

2.3.3.1 Importance Sampling

Importance sampling [Rubinstein and Kroese, 2016, Liu et al., 2015, Gogate and Dechter, 2011] is a Monte Carlo sampling technique used to estimate properties of the distribution $p(x)$ using samples from a *different*, easier to sample distribution, $q(x)$. The more convenient distribution $q(x)$ is called the *proposal* distribution. The key idea is to reweight these samples to account for the difference between $p(x)$ and $q(x)$. Specifically, we can view the expectation over $p(x)$ as a related expectation over $q(x)$, and apply the Monte Carlo estimator:

$$\mathbb{E}_p\big[u(x)\big] = \sum_x p(x)u(x) = \sum_x q(x)\frac{p(x)}{q(x)}u(x) \approx \frac{1}{m}\sum_j \frac{p(x^{(j)})}{q(x^{(j)})}u(x^{(j)}) \quad \text{where} \ \ \{x^{(j)} \sim q(x)\}.$$

The right-hand estimator is thus a weighted average over samples from $q(x)$, adjusted by a weight given by the ratio $p(x)/q(x)$. We say a proposal is *valid* for a query $u(x)$ if its distribution covers all relevant states:

Definition 2.11 (Valid Importance Sampling Proposal Distribution)
Given a distribution $p(x)$, a proposal distribution $q(x)$ is a valid importance sampling proposal distribution for estimating $\mathbb{E}_p\big[u(x)\big]$ if

$$q(x)=0 \implies p(x)u(x)=0. \tag{2.25}$$

A valid propsoal $q(x)$ ensures that the importance sampling estimator is unbiased:

Definition 2.12 (Unbiased Estimator)

An estimator $\hat{\theta}$ of a quantity θ is said to be unbiased if

$$\mathbb{E}[\hat{\theta}] = \theta. \tag{2.26}$$

Namely, that the expectation of the estimate equals the true value of the quantity being predicted.

Even in settings where sampling from $p(x)$ is feasible, using importance sampling with a well-chosen proposal distribution $q(x)$ may provide better (lower variance) estimates. In particular, if $u(x) \geq 0$ is a non-negative function and we can sample from $q(x) \propto p(x)u(x)$, the resulting importance sampling estimate has zero variance (i.e., any sample from $q(x)$ gives an estimate equal to the true expected value) [Kahn and Marshall, 1953, Owen, 2013].

In addition to expectations, we can use importance sampling to estimate the partition function Z of a graphical model representing an unnormalized function $f(x)$, by noting that

$$Z = \sum_x f(x) = \sum_x \frac{f(x)}{q(x)} q(x) = \mathbb{E}_q\left[\frac{f(x)}{q(x)}\right] \approx \frac{1}{m} \sum_{j=1}^{m} \frac{f(x^{(j)})}{q(x^{(j)})}, \qquad x^{(j)} \overset{\text{iid}}{\sim} q.$$

Thus, an importance sampling estimator of Z draws independent samples from a proposal $q(x)$, then computes the ratio $f(x)/q(x)$ and averages over the samples.

Given our use of search tree representations, it is useful to envision representing the sampling process within a search tree. Since a single sample $x^{(j)}$ corresponds to a complete configuration of the model, it can be represented by a path from a leaf to root in an OR search tree, or a solution sub-tree of the AND/OR search tree. From this perspective, we can choose to sample $q(x)$ sequentially, by following the search order that defines the tree: from the (dummy) root, we include all OR children, then for each associated variable, we sample exactly one AND child (its value assignment), then add all OR children and repeat the process.

▢ 2.3.3.2 Stratified Importance Sampling

Stratified Sampling is a variance reduction technique for Monte Carlo estimators that works by first dividing the space of outcomes into disjoint strata [Rubinstein and Kroese, 2016]. In *Stratified Importance Sampling* in particular, the sample space is first divided into k strata, then representatives from each stratum are chosen and re-weighted to represent the omitted members of their respective strata, and finally the representatives are combined together.

To estimate $Z = \sum_X f(x)$ with standard importance sampling, we estimate Z by drawing individual samples from $q(\mathbf{X})$, giving estimate

$$Z^I = \frac{1}{m} \sum_j \frac{f(x^{(j)})}{q(x^{(j)})}.$$

With *stratified* importance sampling, we first partition the sample space X into k strata $X_1, \ldots, X_k$. For each $j \in \{1, \ldots, k\}$, we compute importance sampling estimators $\hat{Z}_j^I$ of $Z_j = \sum_{X_j} \frac{f(x)}{q_j(x)} q_j(x)$ from samples drawn from the conditional distributions $q_j = q \,|\, X_j$. The stratified importance sampling estimator is $\hat{Z}^{SI} = \sum_{j=1}^k \hat{Z}_j^I$. If $\boldsymbol{X}$ is partitioned so that $\sum_{X_1} q(x) = \ldots = \sum_{X_k} q(x)$, then it can be shown that

Theorem 2.3 (Rizzo [2007])
Let $\hat{Z}^{SI}$ be the stratified importance sampling estimator based on drawing m samples from each of the k strata. Let $\hat{Z}^I$ be the basic importance sampling estimator from drawing $M = mk$ samples. Lastly, letting $Z_j = \sum_{X_j} f(x)$ and Z_K be a random variable defined uniformly over $\{Z_1, \ldots, Z_k\}$. Then the variance reduction achieved by moving from $\hat{Z}^I$ to $\hat{Z}^{SI}$ is $\frac{k}{m} Var(Z_K)$, with equality if and only if $Z_1 = \ldots = Z_k$.

Proof. With $\sigma^2 = Var(\frac{f(x)}{q(x)}), x \sim q$, the variance of our importance sampling estimator can be expressed as

$$Var(\hat{Z}^I) = \frac{\sigma^2}{mk}.$$

With $\sigma_j^2 = Var(\frac{f(x)}{q_j(x)}), x \sim q_j$, the variance of our stratified importance sampling estimator can be expressed as

$$Var(\hat{Z}^{SI}) = \sum_j \frac{\sigma_j^2}{m}$$

where σ_j^2 is the variance of the importance weight in strata j. Theorem 5.3 of Rizzo [2007] (page 148) showed that:

$$\sigma^2 - k\sum_{j=1}^{k} \sigma_j^2 = k^2 \cdot Var(Z_K) \tag{2.27}$$

Substituting the definitions above into Equation 2.27 yields,

$$mkVar(\hat{Z}^I) - kmVar(\hat{Z}^{SI}) = k^2 \cdot Var(Z_K)$$

and dividing by mk gives,

$$Var(\hat{Z}^I) - Var(\hat{Z}^{SI}) = \frac{k}{m} \cdot Var(Z_K)$$

$\square$

This shows that stratified importance sampling estimates will always have equal or lower variance than basic importance sampling when its k strata are formed such that $\sum_{X_1} q(x) = \ldots = \sum_{X_k} q(x)$. Furthermore, it suggests that increased stratification (higher k) should lead to greater variance reduction, as well as selecting strata that minimize $Var(Z_K)$.

Chapter 3

Advancing Abstraction Sampling

3.1 Introduction

In this chapter, we focus specifically on computing the partition function of a graphical model $\mathcal{M}$, whose state space's AND/OR tree-width are intractably large, making exact computations such as variable elimination impossible. We thus need to resort to approximate schemes, such as Monte Carlo methods. Classical Monte Carlo methods such as forward sampling and importance sampling [Koller and Friedman, 2009] draw collections of independent samples, each of which represents a single full configuration of the model; averages over these samples are then used to estimate the quantity of interest. Abstraction sampling alters this paradigm by instead sampling a **probe**, which represents *multiple* configurations simultaneously. Each probe is represented as a sub-tree of a search tree for the model (OR tree or AND/OR tree), allowing us to apply many of the advantages of search frameworks, including heuristic functions and conditional independence information. Abstraction Sampling generates probes by using an *abstraction function* to partition the nodes in a search tree into subsets of *abstract states* under the intuition that nodes within the same abstract state are sufficiently similar to be accurately summarized by stochastic sampling. We show that, for reasonably-chosen abstractions, this procedure can give significantly more accurate estimates than standard importance sampling.

Contributions. We introduce and extend the algorithmic framework of Abstraction Sampling (first proposed in Broka et al. [2018]), that combines search with stratified importance sampling to answer summation queries (e.g., computing the partition function) over graphical models. The main contributions of this chapter include:

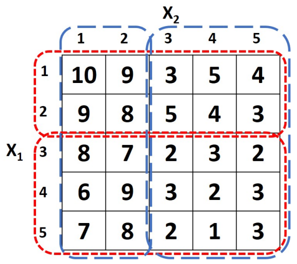
1. We introduce the algorithm `ORAS` for performing Abstraction Sampling on classical OR state space search trees.
2. We give a theoretical analysis on OR Abstraction Sampling properties, including variance reduction conditions and a proof of unbiasedness of `ORAS`.
3. We propose a new Abstraction Sampling algorithm, `AOAS`, designed for compact AND/OR search spaces. By freeing `AOAS` from upholding a restrictive property known as "properness" [Broka et al., 2018], our new scheme significantly enhances the scalability and performance of Abstraction Sampling for AND/OR spaces.
4. We give theoretical analysis on AND/OR Abstraction Sampling properties, including variance reduction conditions and a proof of unbiasedness that lifts the "properness" restriction.
5. We propose three classes of abstraction functions to guide stratification: a context-based approach from Broka et al. [2018], a method that partitions nodes by associated positive real values, and a randomized abstraction scheme. We test and compare over twenty-four distinct functions, each capable of varying abstraction granularity.
6. We perform an extensive empirical evaluation on over 480 problems from five well known summation benchmarks, comparing our Abstraction Sampling frameworks against each other and to competing schemes. Our experiments illustrate the properties of Abstraction Sampling, demonstrate its strength at estimating summation queries, and allow us to illuminate a few particularly powerful Abstraction Sampling set-ups.

▢ 3.2 Abstraction Sampling

Abstraction Sampling is inspired by the early work of Knuth [1975] and Chen [1992], who proposed a method for estimating quantities that can be expressed as aggregates (e.g., sums) of functions defined over the nodes in a graph. Our work extends this framework and applies it to graphical model queries, such as computing the partition function of weighted state-space trees (Definition 2.7).

Given an abstraction function over a weighted directed tree T, we build a sub-tree $\tilde{T}$ (called a *probe*) level-by-level. At the current level i, we first expand all surviving leaf nodes, to yield a frontier of nodes at level $i+1$. Then all current nodes at level $i+1$ are partitioned into abstract states according to the (user selected) abstraction function, and a single representative node n is stochastically selected from each abstract state. The unselected nodes are removed, and each selected node n is assigned a weight $w(n)$ to account for the removed mass of n's abstract state. The process repeats until all variables have been expanded and abstracted, after which an estimate of Z can be computed on the resulting tree $\tilde{T}$.

This probe generation process has elements of search in that the resulting probe is a sub-tree of the full search tree, and of sampling due to the stochastic selection and reweighting of representative nodes. As such, our *Abstraction Sampling* (AS) scheme aims to combine ideas from statistics and search to exploit their respective strengths. Search works systematically and deterministically to efficiently explore all configurations *exactly once* in a structured manner. "It does not leave any stone unturned and does not turn any stone more than once" [Pearl, 1984]. Sampling, on the other hand, explores a stochastic subset of the paths that are used as a stand-in for the full space. By generating and searching a sub-tree in a coordinated manner – exploring a subset of configurations – Abstraction Sampling removes stochastic redundancy to reduce the variance of the sampled estimator compared to independent sampling. By varying the size of the probe, Abstraction Sampling provides



	X_2=1	2	3	4	5
X_1=1	10	9	3	5	4
2	9	8	5	4	3
3	8	7	2	3	2
4	6	9	3	2	3
5	7	8	2	1	3

Figure 3.1: Motivating Example; Z=126.

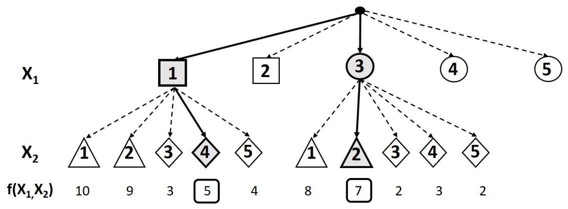


Figure 3.2: Motivating Example Tree

an interpolation between classical importance sampling and heuristic search; the abstraction function provides a notion of systematic exploration or determinism that can improve sampling's randomized exploration.

From a search perspective, abstractions can be viewed as a license to merge nodes that root similar sub-trees, sampling one of the sub-trees, thus creating a more compact graph that can be searched more efficiently. From a sampling perspective, an abstract state can be viewed as a particular "stratum" within a stratified sampling scheme [Rubinstein and Kroese, 2007, Rizzo, 2007], where the stratified sampling process is applied repeatedly, layer by layer. The goal is to obtain variance reduction of the overall estimate, based on similar principles to those of standard stratified sampling.

3.2.1 The General Scheme Through An Example

Our proposed Abstraction Sampling (AS) algorithm emulates stratified importance sampling on partial configurations (represented by nodes in the search tree), iteratively layer by layer (with each layer corresponding to a variable). To illustrate the process, consider the small two dimensional function in Figure 3.1 over variables X_1 and X_2 with domains $D_1 = D_2 = \{1, 2, 3, 4, 5\}$ and with a partition function equal to $Z = 126$ (i.e., the sum of all entries in the table). (The cell of row i and column j depicts the global function $f(X_1 = i, X_2 = j)$.) We define a search tree to explore all possible full configurations of (X_1, X_2); the root (no assignment) is extended to nodes representing each possible configuration of X_1; then each of these is extended to nodes that represent the possible assignments of X_2. For our abstraction function, we partition nodes at the first layer (corresponding to assignments of X_1) into two abstract states: one for $X_1 \in \{1, 2\}$ (drawn as squares in the search tree) and the other for $X_1 \in \{3, 4, 5\}$ (circles). Notice that this abstraction function places rows with roughly the same total mass (sum of entries) into the same abstract state. At next level (for X_2), nodes are again abstracted into two states: $X_2 \in \{1, 2\}$ and $X_2 \in \{3, 4, 5\}$, represented by triangles and diamonds, respectively.

The Abstraction Sampling process is illustrated on the partial search tree in Figure 3.2. Although any importance sampling proposal can be used, for simplicity we assume a uniform proposal distribution over the states and stochastically select representatives. At each step, we shade the nodes that are stochastically selected as that abstract state's representative. So, here we first select $X_1 = 1$ and $X_1 = 3$ as representatives of their respective abstract states; they are assigned weights of 2 and 3, respectively, in attempt to compensate for nodes not selected. (We have left out the details of the selection process and weighting for simplicity). From the two selected representatives, all possible extensions by assignment to X_2 are generated, and the newly generated nodes inherit the weights of their parents (thus accounting for the nodes that were absorbed via the previous abstraction processes). The abstraction function then splits the newly extended nodes into their abstract states, and we stochastically select a representative from each. Let's assume that we stochastically select $(X_1 = 1, X_2 = 4)$ from the diamond abstract state of six nodes, and $(X_1 = 3, X_2 = 2)$ from the triangle abstract state of four nodes. The diamond node's weight is updated to now be 15 and the triangle node's weight updated to be 10. We can now estimate the partition function using their true values along with their weights as $\hat{Z} = 5 \cdot 15 + 7 \cdot 10 = 145$.

Notice that the abstraction sampling procedure generated a subtree with multiple configurations that it used for its estimate, rather than a using only sample of a single configuration. How much does the abstraction process help our Monte Carlo estimator on this simple example? We compare the empirical variance of abstraction sampling to standard importance sampling using the same proposal distribution over 100,000 trials. Experimentally, we find that the variance is reduced from 2337 to 389, an almost 6 fold decrease, by using abstraction sampling. This illustrates the potential benefits of Abstraction Sampling schemes over traditional importance sampling schemes.

For simplicity sake we will describe Abstraction Sampling for the regular OR search trees in this section and will move to the more general AND/OR case in the following section.

▩ 3.2.2 Algorithm ORAS on OR search Trees

Abstraction Sampling algorithm is a Monte Carlo process that generates compact representatives $\tilde{T}$ of T, guided by an *abstraction function*.

Definition 3.1 (Layered Abstractions over search trees.)
Given a weighted OR tree T, an abstraction function, $a : T \rightarrow I^+$, where I^+ are integers, partitions the nodes in T, in each layer, layer by layer. Abstraction states are denoted by $\{i\}$ for an integer i.

Abstraction Sampling for OR trees (*ORAS*, Algorithm 3) builds a sub-tree $\tilde{T}$ of T, level-by-level, in a breadth-first manner. The sub-tree is also called a *probe*. Starting from the dummy root of T, at each iteration ORAS expands the nodes of the current frontier Fr, these frontier nodes denoted n_X, to their children nodes $n_Y \in ch(n_X)$ in T (lines 5-6). Note

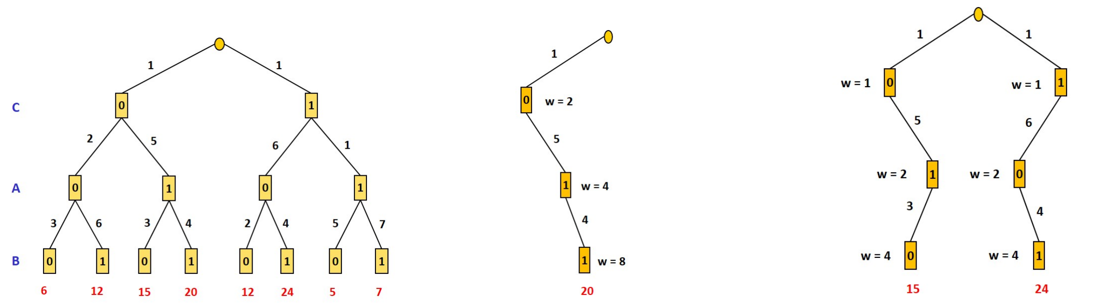
Algorithm 3: `ORAS`

Input: Graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \boldsymbol{F})$, an ordering $X_1, ..., X_N$ and a corresponding implicit OR search tree T, an abstraction function a, and a heuristic function h. For any node n, $g(n)$ = its path cost, and $w(n)$ = its importance weight. $\varnothing$ is a dummy variable whose child in $\mathcal{T}$ is X_1. $\varnothing$ is assigned a dummy value $n_\varnothing$. Fr is the frontier of the current sampled tree, $PROBE = \tilde{T}$.

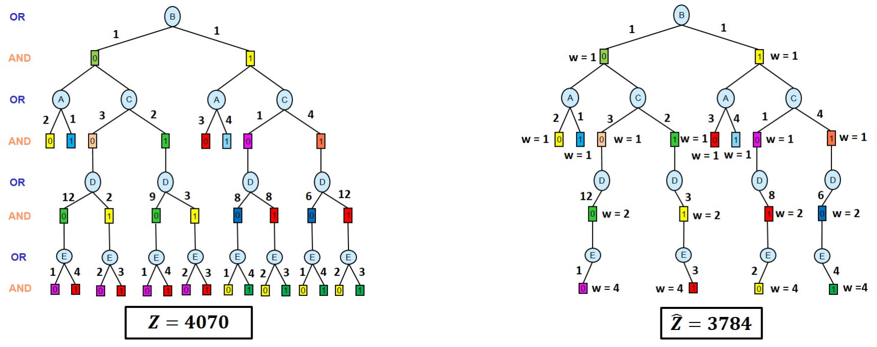
Output: $\hat{Z}$, an estimate of the partition function of $\mathcal{M}$

```
1  begin
2      $w(n_\varnothing) = 1, Fr \leftarrow \{n_\varnothing\}, X \leftarrow D$          // initialize dummy root node
3      while $X \neq X_N$ do
4          $Y \leftarrow$ child of $X$
5          foreach $n_X \in Fr$ do                          // expand frontier to next variable
6              $Fr' \leftarrow Fr$ expanded from $n_X$ to all assignments of $Y$
7              foreach $n_Y \in Fr'$ do
8                  $w(n_Y) \leftarrow w(n_X)$                         // weight inherited from parent
9                  $g(n_Y) \leftarrow g(n_X) \cdot c(n_Y)$
10             end
11         end
12         $A \leftarrow \{A_i \mid A_i = \{n_Y \in Fr' \mid a(n_Y) = i\}\}$     // abstract newly generated frontier
13         foreach $A_i \in A$ do
14             foreach $n \in A_i$ do
15                 $q(n) \leftarrow \frac{w(n) \cdot g(n) \cdot h(n)}{\sum_{m \in A_i} w(m) \cdot g(m) \cdot h(m)}$
16             end
17             $n_{Y_i} \propto_q A_i$                           // randomly select according to proposal q
18             $w(n_{Y_i}) \leftarrow w(n_{Y_i}) / q(n_{Y_i})$                  // update importance weight
19             $Fr' \leftarrow Fr' \setminus A_i \cup \{n_{Y_i}\}$               // only keep reweighted representative
20         end
21         $Fr \leftarrow Fr', X \leftarrow Y$
22     end
23     return $\hat{Z} = \sum_{n \in Fr} w(n) \cdot g(n)$
24 end
```

that Y represents the the child vriable of X in the guiding pseudo tree $\mathcal{T}$ from which T is based. Each new frontier node n_Y inherits the weight of its parent (line 8) accounting for the estimated mass of all nodes previously abstracted into its path. The new frontier nodes Fr' are partitioned into abstract states according to abstraction function a (line 12). For each abstract state, according to a proposal probability q (line 15), a representative node is selected (line 17), reweighted to account for the rest of the nodes in its abstract state (line 17), and the rest of the nodes discarded (line 20). The process repeats until all variables in $\mathcal{T}$ have been expanded, after which an estimate of the partition function can be computed using the final frontier nodes of $\tilde{T}$ (line 23).



(a) Full OR search tree **(b)** Ex. OR Probe 1 **(c)** Ex. OR Probe 2



(d) Full AND/OR search tree **(e)** Ex. AND/OR Probe

Figure 3.3: Example of Abstraction Sampling probes.

The Sampling Proposal. As with any variant of importance sampling, the proposal distribution $q(\cdot)$ plays a critical role in the performance of the algorithm. As we show in the sequel, our abstraction sampling algorithms are unbiased for any valid proposal distribution. However, in our algorithms and implementation, we specifically use $q(n) \propto w(n) \cdot g(n) \cdot h(n)$, where $g(n)$ is the path cost to n, $w(n)$ is the current abstraction sampling weight of n, and $h(n)$ is a heuristic function that provides an upper bound on the value $V(n)$ of the subproblem rooted at n. This specific proposal often works well in practice, as it provides an estimate of the contribution of this node to the overall estimate, if we were to proceed exactly (fully expand the search tree) below node n. In fact, we show in the sequel (in Theorem 3.9: exact proposal) that if our heuristic is exact, $h(n) = V(n)$ for all n, the importance sampling estimate has zero variance, and $\hat{Z} = Z$.

Example 3.2.2.1

Consider the OR search tree T in Figure 3.3a. The cost of each solution is obtained by a *product on its solution arcs. In Figure 3.3b we show a probe generated via an abstraction function that puts all nodes that represent a single variable in a single abstract state. In Figure 3.3c, we see a probe where each domain value of a variable corresponds to a different abstract state, yielding 2 states per variable, and thus 2 nodes per level of the generated tree. The estimate for Figure 3.3b is* $\hat{Z} = 20{\cdot}8 = 160$ *and for Figure 3.3c is* $\hat{Z} = 15{\cdot}4 + 24{\cdot}4 = 156$.

Of course the quality and variance of estimates produced by Abstraction Sampling depend greatly on the abstraction function, which will be discussed in Section 3.4: Candidate Abstractions.

3.2.3 Unbiasedness of ORAS

As noted earlier, a graphical model can be expressed by a directed weighted OR tree whose queries (e.g., the partition function) can be computed by evaluating a value function defined on its nodes [Dechter and Mateescu, 2007]. In the case of the partition function we can define a value function,

$$Z = V = \sum_{l \in leaves(T)} \prod_{n \in path(l)} c(n) \tag{3.1}$$

It is easy to see that V can also be expressed recursively over the tree:

Proposition 3.1 (Value of an OR node)
Given a weighted directed state-space tree, having costs, c labeling its arcs or its nodes, the value function V(n) corresponding to computation of the partition function, obeys the recursive expression

$$V(n) = \sum_{n' \in ch(n)} c(n')V(n') \tag{3.2}$$

where for leaves, $V(n) = 1$. *We assume that the root node is a dummy node* $n_\varnothing$ *with cost* $c(n_\varnothing) = 1$. *The partition function can be computed as* $Z = V(n_\varnothing)$.

Theorem 3.2 (unbiasedness)

Given a weighted search tree T of a graphical model, a value function $V(n)$ defined by Equation 3.2, and an abstraction a, the ORAS estimate $\hat{V}(n_\varnothing)$ is an unbiased estimator of Z:

$$\mathbb{E}[\hat{V}(n_\varnothing)] = V(n_\varnothing) = Z,$$

where $n_\varnothing$ the root of T as defined in Proposition 3.1: Value of an OR node.

Proof. At each step ORAS maintains a current, partially generated, sampled tree or probe $\tilde{T}^{(t)}$, where t indexes the most recent algorithm step that altered the composition of the partial probe – namely, either expansion of its leaves (Equation 5) or stochastic selection of a representative node from its representative abstract state (lines 17-20) in the currently generated partial probe. The partial probe $\tilde{T}^{(t)}$ is a stochastic sub-tree of T whose nodes are assigned weights by the algorithm. We can define an intermediate estimator $\hat{V}^{(t)}$ of V at step t of the algorithm by,

$$\hat{V}^{(t)}(n) = \begin{cases} V(n) & \text{if } n \in Fr \\ \displaystyle\sum_{n' \in ch(n)} \frac{w(n')}{w(n)} c(n') \hat{V}^{(t)}(n') & \text{if } n \in \tilde{T}^{(t)} \setminus Fr \end{cases} \tag{3.3}$$

where Fr are the frontier nodes in $\tilde{T}^{(t)}$. This is a recursive estimator combining information from the sampled nodes and estimated weights in $\tilde{T}^{(t)}$ with exact values $V(n)$ used at the nodes in the current frontier of $\tilde{T}^{(t)}$ at time t. At $t = 0$, the frontier is simply the dummy root node, r, so that $\hat{V}^{(0)}(n_\varnothing) = V(n_\varnothing) = Z$ by definition of $V(n_\varnothing)$. At the final time step, $t = L$, when the probe has been fully expanded, we have $\hat{V}^{(L)}(n_\varnothing) = \sum_{n \in leaves(\tilde{T}^{(L)})} w(n) g(n) = \hat{Z}$, the final estimate of Z of the probe produced by `ORAS` (line 23). To prove unbiasedness, we show that each step of the algorithm preserves the expectation of the intermediate estimate, $\mathbb{E}[\hat{V}^{(t+1)}(n_\varnothing) - \hat{V}^{(t)}(n_\varnothing)] = 0$, so that $\mathbb{E}[\hat{Z}] = \mathbb{E}[\hat{V}^{(L)}(n_\varnothing)] = \mathbb{E}[\hat{V}^{(0)}(n_\varnothing)] = Z$.

During abstraction sampling, there are two operations that alter the probe: expansion and abstraction. The expansion operation is deterministic, expanding all surviving frontier nodes $n \in Fr$ to yield a new temporary frontier. Consider a node n that was in the frontier at step t, with $\hat{V}^{(t)}(n) = V(n)$ by definition. After expansion, it is no longer in the frontier at step $t+1$, but its children are; so its value is given by,

$$\hat{V}^{(t+1)}(n) = \sum_{n' \in ch(n)} \frac{w(n')}{w(n)} c(n') \hat{V}^{(t)}(n') = \sum_{n' \in ch(n)} \frac{w(n')}{w(n)} c(n') V(n') = \sum_{n' \in ch(n)} c(n') V(n') = V(n),$$

since after expansion, $w(n') = w(n)$, and applying the recursive definition of the value $V(n)$, Equation 3.3. Thus, the value of each node n in the frontier at step t is preserved at step $t+1$, and so are the values of all ancestors of those nodes, including the root.

The second operation that alters the probe is a stochastic abstraction step, in which a representative node is selected and reweighted from each abstract state A_i, and all other nodes in state A_i are discarded. We write the value at the root, $\hat{V}^{(t)}(n_\varnothing)$, in terms of a sum over all nodes along the current frontier, which we partition into the contribution $C^{(t)}(A_i)$ from each abstract state A_i:

$$\hat{V}^{(t)}(n_\varnothing) = \sum_{A_i \in A} C^{(t)}(A_i), \qquad C^{(t)}(A_i) = \sum_{n \in A_i} w^{(t)}(n) \cdot g(n) \cdot V(n). \tag{3.4}$$

After a stochastic selection operation on the probe at step t, the probe is altered so that at $t+1$, only one of the nodes $n' \in A_i$ is retained from each A_i, randomly selected with probability $q(n')$ (line 15). The weight of the retained node is then updated (line 18). Therefore, at step $t+1$, only the path to node n' contributes from A_i to $\hat{V}^{(t+1)}(n_\varnothing)$; the random selection does not affect any of the other paths outside A_i. The expectation of the difference in estimates from step t to $t+1$ is therefore,

$$\begin{aligned}\mathbb{E}[\hat{V}^{(t+1)}(n_\varnothing) - \hat{V}^{(t)}(n_\varnothing)] &= \sum_{n' \in A_i} q(n')[w^{(t+1)}(n')g(n')V(n') - C^{(t)}(A_i)] \\ &= \sum_{n' \in A_i} q(n')\Big[\frac{w^{(t)}(n')}{q(n')}g(n')V(n')\Big] - \sum_{n \in A_i} w^{(t)}(n) \cdot g(n) \cdot V(n) \qquad (3.5) \\ &= 0\end{aligned}$$

□

3.3 Abstraction Sampling for AND/OR Search Trees

In search algorithms, representing a graphical model using an AND/OR search space [Dechter and Mateescu, 2007] is significantly more compact and incorporates conditional independence information, and so can facilitate more efficient algorithms [Marinescu and Dechter, 2009c]. In this section we extend abstraction sampling to AND/OR search spaces. We first extend the recursive expression of a node's value (e.g., Equation 3.2) to AND/OR trees, present an Abstraction Sampling algorithm for AND/OR trees (AOAS) and study its properties.

3.3.1 AND/OR node values and the partition function

In the OR case, the value of a node n corresponds to the partition function restricted to that node configuration, namely the sum of the cost of all full configurations consistent with the assignments of $path(n)$. In the case of AND/OR search trees the relationship is more complex. Here we extend some notation from the OR search tree case to that of AND/OR search trees.

Notation. As in Chapter 2, in an AND/OR search space, we use n_X to refer to some AND node associated with variable X, and Y_{n_X} to be the OR node associated with variable Y

that is the child of n_X. The children of a node n are denoted $ch(n)$, while $path(n)$ is the configuration of the variables defined by the assignments along the path from the root to node n. The set of variables so assigned is denoted $varpath(n)$. Each arc is associated with a cost that captures the factors in $\boldsymbol{F}$ that are now fully resolved (all arguments assigned values), so, the cost of the arc to an AND node n_X is

$$c(n_X) = \prod_{f \in \{f_{\boldsymbol{\alpha}} \in \boldsymbol{F} \mid \boldsymbol{\alpha} \subseteq varpath(n_X),\, X \in \boldsymbol{\alpha}\}} f(path(n_X)). \tag{3.6}$$

or 1, vacuously. Then, we define the *cost* $g(n)$ of the path to n as $g(n) = \prod_{n' \in anc(n)} c(n')$, where $anc(n)$ are the ancestors of n in the search tree.

Proposition 3.3 (Value of an AND/OR node)
Given an AND/OR search tree $T_{\mathcal{T}}$ guided by pseudo tree $\mathcal{T}$, the value function $V(n)$ corresponding to the computation of the partition function obeys the recursive expression:

$$V(n_X) = \prod_{Y_{n_X} \in ch(n_X)} V(Y_{n_X}) \tag{3.7}$$

where for Y_{n_X} - the OR node child of n_X corresponding to variable Y in $\mathcal{T}$- we have

$$V(Y_{n_X}) = \sum_{n_Y \in ch(Y_{n_X})} c(n_Y) \cdot V(n_Y) \tag{3.8}$$

with $V(n_X) = 1$ in the case n_X has no children.

Combined into one expression, this becomes:

$$V(n_X) = \begin{cases} 1 & \text{if } n_X \text{ a leaf in } T \\ \prod_{Y_{n_X} \in ch(n_X)} \sum_{n_Y \in ch(Y_{n_X})} c(n_Y) \cdot V(n_Y) & \text{otherwise} \end{cases} \tag{3.9}$$

With $n_\varnothing$ as the dummy root node of $T_{\mathcal{T}}$, $V(n_\varnothing) = Z$.

The value $V(n)$ of a node n in $T_{\mathcal{T}}$ is the sum cost of all solution sub-trees rooted at n. $V(n_X)$ denotes the value of an AND node n_X corresponding to variable X in $\mathcal{T}$, and $V(Y_{n_X})$ denotes the value of its OR child corresponding to variable Y in $\mathcal{T}$.

In an OR search tree, our value function $V(n)$ in Equation 3.2 represents the partition function conditioned on the partial configuration corresponding to *path*(n), i.e., the unnormalized probability mass of that partial configuration. However, in the AND/OR case, while the node root $V(n_{\varnothing})$ similarly equals Z, it is not necessarily true that for internal nodes n that $V(n)$ represents an unnormalized probability measure for the partial configuration of *path*(n). Unlike when dealing with traditional OR search trees, when dealing with AND/OR trees it is important to recognize that not all of the variables of a model can necessarily be captured through a single path from the root. Notice that in the AND/OR tree in Figure 2.10 a full configuration consisting of assignments to all of $A, B,$ and C cannot be captured through any single path from root to leaf. For AND/OR trees, it is necessary to consider variables that branch off of a path (corresponding to branchings in the guiding pseudo tree). (See Section 2.3.2.4: $\boldsymbol{R(n)}$). Thus, such branchings need to be considered when computing the unnormalized likelihood measure for any partial configuration corresponding *path*(n).

Recall *path*(n) denotes the configuration of variables on the path from the root to n in $T_{\mathcal{T}}$. We will refer to the unnormalized probability mass of the partial configuration *path*(n) as $Q(n)$. $Q(n)$ is equal to the partition function Z restricted to configurations consistent with *path*(n). Letting *Out*(*path*(n)) be the set of OR nodes (corresponding to variables) emanating from *path*(n) which are OUTside *path*(n), then Z conditioned on *path*(n) can be obtained by multiplying the cost of *path*(n), $g(n)$, by $V(n)$, and by the values of all the OR nodes branching out of *path*(n) (the latter when combined together referred to as the ancestor branching mass of n, or $R(n)$; see Figure 2.10). Formally,

Definition 3.2 (Unnormalized probability mass of AND/OR path(n))

If T is an AND/OR search tree for a graphical model $\mathcal{M}$ then,

$$Q(n) = g(n) \cdot V(n) \cdot \prod_{Y_{n'} \in Out(path(n))} V(Y_{n'}) \tag{3.10}$$

where $V(Y_{n'})$ is the value defined in Equation 3.8. It can also be written as

$$Q(n) = g(n) \cdot V(n) \cdot R(n) \tag{3.11}$$

where, $R(n)$, called ancestral branching mass*, is*

$$R(n) = \prod_{Y_{n'} \in Out(path(n))} V(Y_{n'}). \tag{3.12}$$

Letting $br(n_X)$ be the set of ancestor nodes n_{Y_i} of n_X such that each Y_i is a branching variable ancestor of X in the guiding pseudo tree, $R(n)$ can equivalently be expressed as

$$R(n) = \prod_{n_Y \in br(n_X)} \prod_{\substack{W_{n_Y} \in ch(n_Y) \\ W_{n_Y} \notin path(n_X)}} V(W_{n_Y}). \tag{3.13}$$

In the vacuous setting, we define $R(n) = 1$.

Lemma 3.4

$$Q(n) = Z_{|path(n)} = \sum_{\boldsymbol{X'}=\boldsymbol{X} \setminus varpath(n)} \prod_{f \in \boldsymbol{F}} f(\boldsymbol{x}', path(n)) \tag{3.14}$$

Proof. $Q(n)$ is $V(n_\varnothing)_{|path(n)} = Z_{|path(n)}$ when $n_\varnothing$ is the root. Using the recursive definition in Equation 3.9 yields the claim. □

Algorithm 4: `AOAS`

Input: Graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \boldsymbol{F})$, a pseudo tree $\mathcal{T}$ for $\mathcal{M}$ rooted at a dummy singleton variable D, an abstraction function a, heuristic function h. For any node n, $g(n)$ = its path cost, $w(n)$ = its importance weight, and $\hat{V}(n)$ = its estimated value (initialized to $h(n)$).

Output: $\hat{Z}$, an estimate of the partition function of $\mathcal{M}$

```
Function AOAS($\mathcal{T}$, $h$, $a$)
begin
    (Initialization)
    $PROBE \leftarrow n_\varnothing,\ g(n_\varnothing), w(n_\varnothing), r(n_\varnothing), \hat{V}(n_\varnothing) \leftarrow 1$
    $STACK \leftarrow push(\text{empty stack}, D)$
    (DFS Traversal of $\mathcal{T}$)
    while $STACK$ is not empty do
        $X \leftarrow top(STACK)$
        if $X$ has unvisited children in $\mathcal{T}$ then
            (DFS Forward Step)
            $Y \leftarrow$ the next unvisited child of $X$
            foreach $n_X \in PROBE$ do                  // expand frontier to next variable
                $PROBE \leftarrow PROBE$ expanded from $n_X$ to its $n_Y$ descendants
                $F'_Y \leftarrow$ newly generated nodes $n_Y$
                foreach $n_Y \in F'_Y$ do
                    $w(n_Y) \leftarrow w(n_X)$                  // inherit weight from parent AND node
                    $g(n_Y) \leftarrow g(n_X) \cdot c(n_Y)$
                    $r(n_Y) \leftarrow r(n_X) \cdot \prod_{\{S \neq Y \in ch_{\mathcal{T}}(X)\}} \hat{V}(S_{n_X})$    // compute estimate of $R(n_Y)$
                end
            end
            (Perform Abstractions)
            $A \leftarrow \{A_i \mid A_i = \{n_Y \in PROBE \mid a(n) = i\}\}$      // abstract newly generated nodes
            foreach $A_i \in A$ do
                foreach $n \in A_i$ do
                    $q(n) \leftarrow \frac{w(n) \cdot g(n) \cdot h(n) \cdot r(n)}{\sum_{m \in A_i} w(m) \cdot g(m) \cdot h(m) \cdot r(m)}$
                end
                $n_{Y_i} \propto_q A_i$                  // randomly select according to proposal q
                $w(n_{Y_i}) \leftarrow w(n_{Y_i}) / q(n_{Y_i})$                  // update importance weight
                $\hat{V}(n_{Y_i}) \leftarrow 1$                  // dummy initialization of node value
                $PROBE \leftarrow PROBE \setminus A_i \cup \{n_{Y_i}\}$    // only keep reweighted representative
            end
            $push(STACK, Y)$
        else
            (DFS Backtracking Step)
            $pop(STACK)$,  $W \leftarrow top(STACK)$
            $PROBE \leftarrow PROBE$ s.t. all $n_W$ without $n_X$ descendants are pruned
                                                  // prune paths abstracted into other nodes
            foreach $n_W$ in $PROBE$ do
                (update node value)
                $\hat{V}(n_W) \leftarrow \hat{V}(n_W) \cdot \sum_{n_X = child(n_W)} \hat{V}(n_X) \cdot c(n_X) \cdot \frac{w(n_X)}{w(n_W)}$
            end
        end
    end
    return $\hat{Z} = \hat{V}(n_\varnothing)$
end
```

▢ 3.3.2 Algorithm AOAS

We now present AOAS (AND/OR Abstraction Sampling), a scalable Abstraction Sampling algorithm for AND/OR search spaces. Original attempts to adapt Abstraction Sampling ran into scalability issues that will be discussed. Nevertheless these issues were overcome in AOAS which, similar to ORAS, allows for a variable-by-variable expansion of an Abstraction Sampling probe – in this case over an AND/OR space – to be used to estimate the partition function.

As input, AOAS (Algorithm 4) takes a pseudo tree $\mathcal{T}$ of a graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \boldsymbol{F})$ rooted at a dummy variable $\varnothing$, a heuristic function $h(n)$ providing upper bound on $V(n)$, and an abstraction function a, AOAS traverses the pseudo tree $\mathcal{T}$ variable-by-variable in a depth-first manner using the traversal to guide the generation of the Abstraction Sampling probe, which is a sampled partial search tree of the full search tree $T_{\mathcal{T}}$. As it progresses forward in the traversal of $\mathcal{T}$, it expands the corresponding nodes at the frontier of the partially generated probe and stochastically selects a subset to remain in the frontier using the abstraction function. Since the pseudo-tree - which may have branching variables (Definition 2.10) - is explored in a depth-first manner, the algorithm backtracks when it reaches the end of a particular branch in $\mathcal{T}$, during which the algorithm updates values of nodes returned to.

More specifically, in the forward step (lines 11-32), $PROBE$ (which refers to the probe being generated) is extended from all current leaf AND nodes n_X of variable X to AND nodes n_Y (lines 12-13), where Y is the next child of X in $\mathcal{T}$ when traversing $\mathcal{T}$ in a breadth-first manner. The newly added AND nodes n_Y become new leaves in $PROBE$. Each newly generated node n_Y inherits the weight of the AND node n_X from which it was descended (line 16) and has a path cost $g(n_Y) = g(n_X) \cdot c(n_Y)$ assigned (line 17). $r(n_Y)$, an estimate of $R(n_Y)$ in Definition 3.2: Unnormalized probability mass of AND/OR path(n), is also computed (line 18). Note that $r(n)$ is simply inherited from $r(n_X)$ if the parent variable

X is not a branching variable. Otherwise, the $r(n_X)$ from its parent is multiplied by the estimated values contributed by its sibling branches, yielding an $r(n_Y)$ which bounds the ancestor branching value for the new frontier node n_Y.

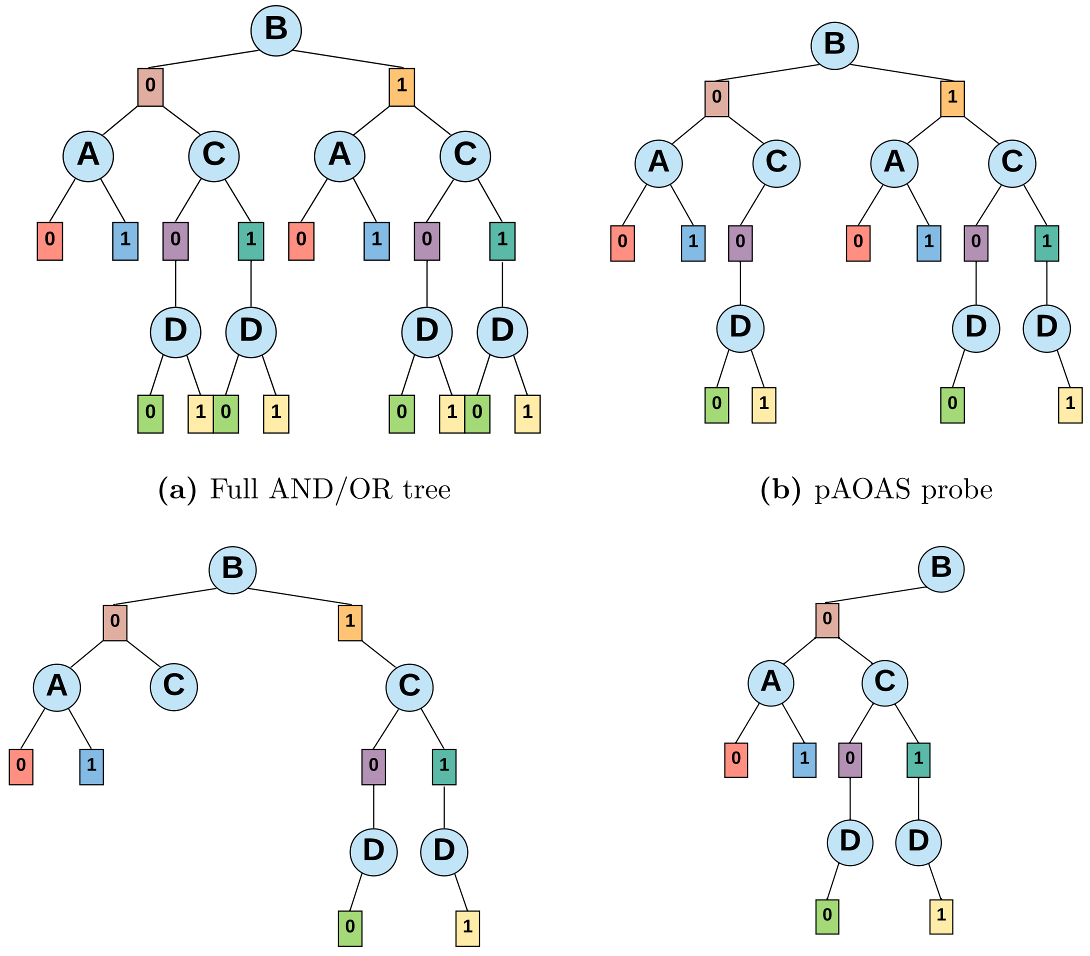
Once nodes are expanded, we perform abstractions (lines 22-31). Nodes are partitioned into abstract states according to the abstraction function a (line 22), and for each abstract state, proposals are generated (line 25) from which a representative node is stochastically selected (line 27). Although the proposal function used can be any valid distribution over the nodes, we choose a particular proposal which is proportional to the quantity $w(n) \cdot g(n) \cdot h(n) \cdot r(n)$, which is an estimate for $Q(n)$ upweighted to account for all the nodes abstracted into the path to n. The portion $g(n) \cdot h(n) \cdot r(n)$ estimates $Q(n)$ (see Definition 3.2: Unnormalized probability mass of AND/OR path(n)), where $V(n)$ is approximated by $h(n)$, and $R(n)$ is estimated by $r(n)$. We multiply by $w(n)$ to account for nodes previously abstracted into the path leading to n.

When backtracking to a variable W from variable X (lines 35-40), we prune any n_W that has no descendant n_X (line 36), thus preventing the creation of invalid probes.

Definition 3.3 (Valid and Invalid Probes)
A valid probe is a probe that contains at least one full solution tree (Definition 2.8). Probes that do not contain a full solution tree are called invalid (e.g., Figure 3.4c).

For the surviving nodes, we back up the estimated values from their children from which we have backtracked (line 39). Once the algorithm backtracks to the dummy node, the value of the dummy node, $\hat{V}(n_\varnothing)$, equals the Abstraction Sampling estimate of the model's partition function, $\hat{Z}$, which is returned.

Example 3.3.2.1 (`AOAS` **Trace)**
Figure 3.5 shows a step-by-step trace of `AOAS` *on the same model $\mathcal{M}$, pseudo-tree $\mathcal{T}$, and AND/OR search tree $T_\mathcal{T}$ from Figure 2.8. We follow a DFS traversal of $\mathcal{T}$ (order B, A, C, D),*



(a) Full AND/OR tree **(b)** pAOAS probe

(c) invalid probe **(d)** AOAS probe

Figure 3.4: An AND/OR search tree and possible probes generated by abstraction sampling. Nodes are abstracted based on having the same domain value (denoted by having the same color). (a) A full AND/OR tree, representing all 16 possible solutions, (b) A probe generated by `pAOAS` – a non-scalable original attempt at AND/OR Abstraction Sampling – containing 8 solutions, (c) An invalid probe containing 4 *partial* configurations, and no full solutions, (d) A valid probe generated by `AOAS`, containing 4 solutions.

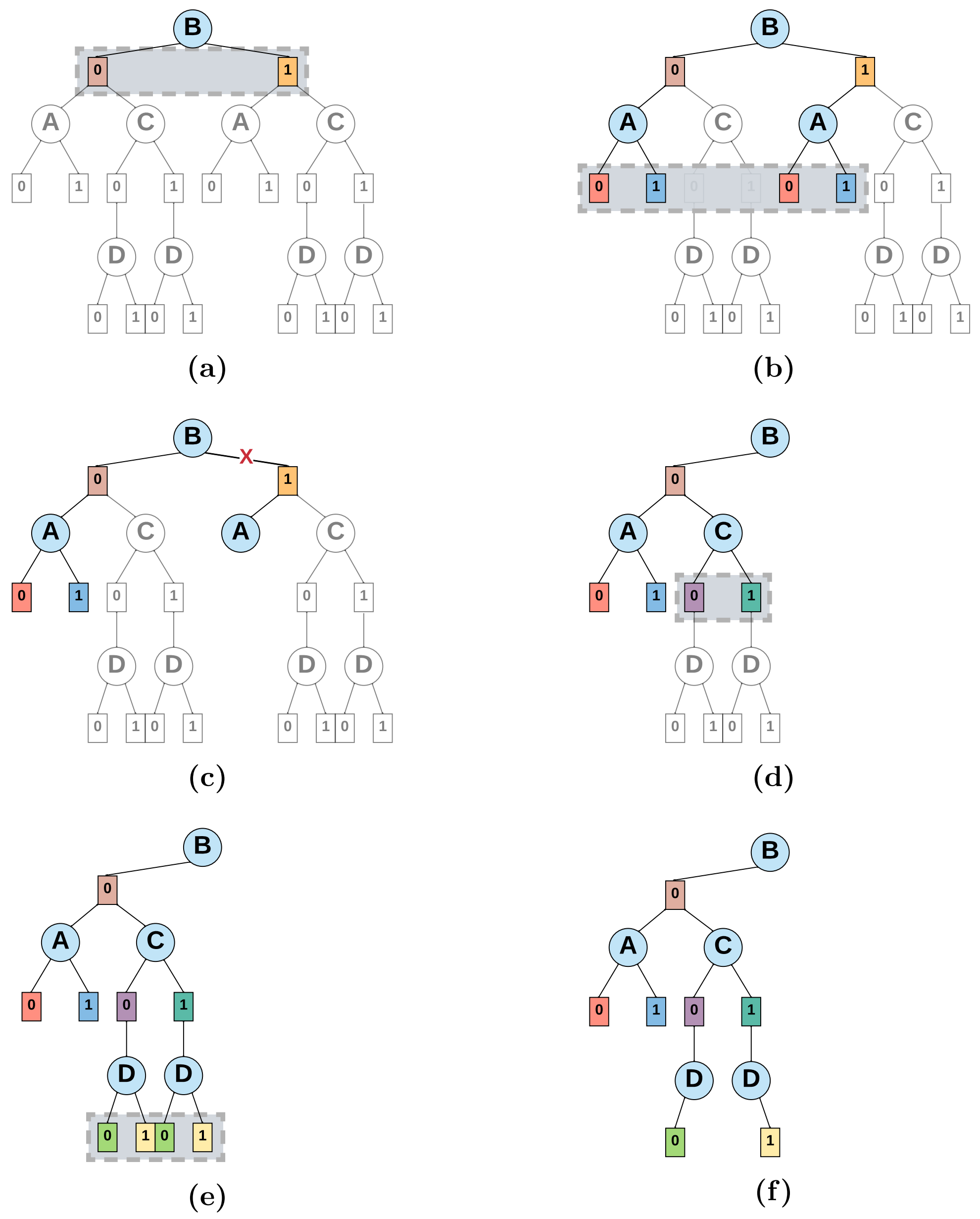
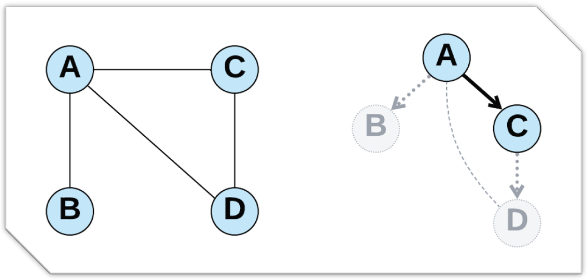


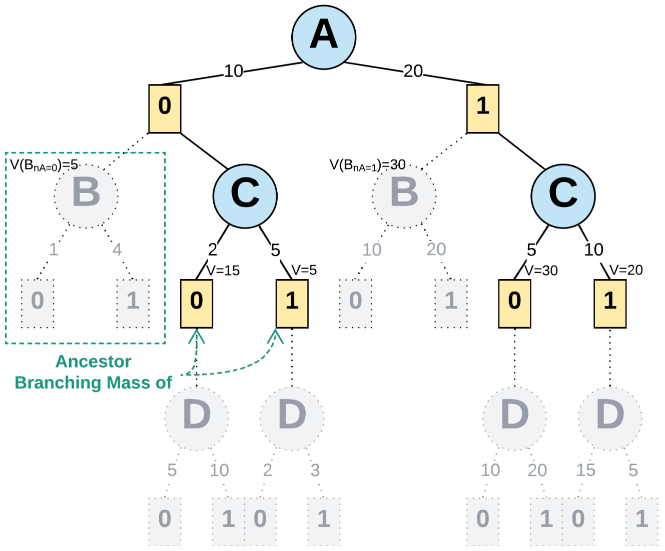
Figure 3.5: A sample trace of `AOAS` based on the AND/OR tree in Figure 3.4a. Nodes with the same domain values are abstracted (also indicated by node color). A red "X" indicates pruning. Transparent nodes indicate portions of the tree not pruned and yet to be explored.

sampling a sub-tree of $T_\mathcal{T}$ to estimate the partition function. For this example, we use an abstraction function that groups nodes corresponding to the same domain value together (also indicated by the color of the nodes). At each stage, we also group nodes being abstracted in gray. A red "X" marks a pruned sub-tree.

Starting with variable B (Figure 3.5a), each node belongs to a different abstraction and so are both retained. We next expand to A and abstract across its nodes (Figure 3.5b). Note that we partition across all *nodes of A, regardless of whether they are descended from $n_{B=0}$ or $n_{B=1}$. We find two nodes in each abstract state (colored red and blue). We calculate their respective proposals according to line 25. Notice that the proposal of each node n relies on $r(n)$ (line 18), which estimates their ancestor branching mass – i.e., the values of nodes in their $Out(path(n))$. In this case, $r(n_{B=0,A=0})$ and $r(n_{B=0,A=1})$ were estimates of $V(C_{n_{B=0}})$, and $r(n_{B=1,A=0})$ and $r(n_{B=1,A=1})$ were estimates of $V(C_{n_{B=1}})$ Since the nodes of C have not yet been expanded, we use their heuristic values as an approximation to $V(\cdot)$. We then stochastically choose a representative from each abstract state (line 27). We happen to select both red and blue representatives from under $n_{B=0}$ (Figure 3.5c). Since A has no descendant, we backtrack to B, updating its node values (line 39) and performing a pruning step (line 36). The pruning removes AND nodes of B that do not extend to AND nodes of A – in this example, $n_{B=1}$. The pruning is shown by the red "*X*" in Figure 3.5c). Pruning ensures that the resulting AND/OR probes remain valid. Finally, we expand and abstract C and D (Figures 3.5d-3.5f). Backtracking recursively from D, we arrive at the dummy root node (not shown) with no further pruning. The result is the valid probe in Figure 3.5f, which contains four full configurations: $(B=0, A=0, C=0, D=0)$, $(B=0, A=0, C=1, D=1)$, $(B=0, A=1, C=0, D=0)$, and $(B=0, A=1, C=1, D=1)$. The estimate of the partition function is $\hat{Z} = \hat{V}(B)$.*



(a) Primal graph and truncated pseudo tree.



(b) Partially expanded AND/OR tree with values.

Figure 3.6: Partially expanded AND/OR tree with values and corresponding truncated pseudo tree.

3.3.3 Unbiasedness of AOAS

To facilitate the proof of `AOAS` unbiasedness, we will use the notation $T_{\mathcal{T}'}$ to refer to a partially expanded AND/OR search tree that is being guided by the pseudo tree $\mathcal{T}$, but has only been expanded up through a subset of $\mathcal{T}$'s variables, indicated by $\mathcal{T}'$. Namely, $\mathcal{T}'$ – which we call a *truncated pseudo tree* – is the sub-tree of $\mathcal{T}$ that has been traversed so far, and through which the probe $T_{\mathcal{T}'}$ has been expanded. Figure 3.6 shows an example of a truncated pseudo tree that has been traversed only through variables A and C and a corresponding partially expanded AND/OR tree.

We first show that as we expand $T_{\mathcal{T}'}$ variable-by-variable according to a traversal of $\mathcal{T}$, we can express $Z = V(n_\varnothing)$ through just the nodes of $T_{\mathcal{T}'}$.

Then, we define an intermediate value function $\hat{V}$ for which $\hat{V}(n_\varnothing) = Z$ during probe initialization, and such that $\hat{V}(n_\varnothing) = \hat{Z}$ – the final estimate produced by by `AOAS`. We show that each expansion, abstraction, and pruning of the `AOAS` probe keeps $\mathbb{E}[\hat{V}(n_\varnothing)]$ of the probe unchanged, thus showing that when the probe is fully expanded, $\mathbb{E}[\hat{V}(n_\varnothing)] = Z$ which implies $\mathbb{E}[\hat{Z}] = Z$.

A useful property of V is that, as we expand a partial AND/OR search tree $T_{\mathcal{T}'}$, we can express $Z = V(n_\varnothing)$ through just the nodes of the truncated tree $T_{\mathcal{T}'}$. Letting the leaves of the truncated $T_{\mathcal{T}'}$ be represented by $Fr_{\mathcal{T}'}$ and abusing notation to use $\mathcal{T}'$ to also indicate the variables of the truncated tree $\mathcal{T}'$, we can write:

$$V_{\mathcal{T}'}(n_X) = \begin{cases} V(n_X), & n_X \in Fr_{\mathcal{T}'} \\ \prod_{Y' \in ch_{\mathcal{T}}(X) \setminus \mathcal{T}'} V(Y'_{n_X}) \cdot \prod_{Y' \in ch_{\mathcal{T}'}(X)} \sum_{n_Y \in ch(Y_{n_X})} c(n_Y) \cdot V_{\mathcal{T}'}(n_Y), & n_X \notin Fr_{\mathcal{T}'} \end{cases} \tag{3.15}$$

from which we can immediately see that,

Lemma 3.5
$V_{\mathcal{T}'}(n_X) = V(n_X)$ for any $n_X \in T_{\mathcal{T}'}$.

Proof. We can arrive to the claim by unrolling the expression in Equation 3.15 after substituting with Equation 3.7 and Equation 3.8. □

The most important consequence, of course, is equality at the root:

Corollary 3.6
$V_{\mathcal{T}'}(n_\varnothing) = V(n_\varnothing) = Z$.

We can alternatively express $V(n_\varnothing)$ in terms of only the frontier nodes of $T_{\mathcal{T}'}$ corresponding to a particular variable $X \in \mathcal{T}'$:

Lemma 3.7

Let frontier AND nodes n_X of $T_{\mathcal{T}'}$ corresponding to a particular leaf variable $X \in \mathcal{T}'$ be represented as Fr_X. Then,

$$Z = V(n_\varnothing) = \sum_{n_X \in Fr_X} g(n_X) \cdot R(n_X) \cdot V(n_X) \tag{3.16}$$

Proof. We can compute Z by summing over sub-spaces of configurations that are exclusive and exhaustive, such as by enumerating over all frontier nodes of a single variable. Since (from Section 2.3.2.4) we have that $Q(n) = Z_{|path(n)}$, we can write:

$$Z = \sum_{n_X \in Fr_X} Q(n_X).$$

Substituting for $Q(n)$ from Equation 3.11 we get

$$Z = \sum_{n_X \in Fr_X} g(n_X) \cdot V(n_X) \cdot R(n_X)$$

□

An example may help to clarify these relationships:

Example 3.3.3.1

Consider a truncation of the pseudo tree seen in Figure 3.6a that includes $A \rightarrow C$ but excludes B and D, and its corresponding truncated AND/OR tree Figure 3.6b. Using only *the frontier nodes $n_C \in F_C$ we can compute Z using Equation 3.7 as:*

$$
\begin{aligned}
Z &= \sum_{n_C \in F_C} g(n_C) \cdot R(n_C) \cdot V(n_C) \\
&= g(n_{A=0,C=0}) \cdot R(n_{A=0,C=0}) \cdot V(n_{A=0,C=0}) \\
&\quad + g(n_{A=0,C=1}) \cdot R(n_{A=0,C=1}) \cdot V(n_{A=0,C=1}) \\
&\qquad + g(n_{A=1,C=0}) \cdot R(n_{A=1,C=0}) \cdot V(n_{A=1,C=0}) \\
&\qquad\quad + g(n_{A=1,C=1}) \cdot R(n_{A=1,C=1}) \cdot V(n_{A=1,C=1}) \\
&= (20)\cdot(5)\cdot(15) + (50)\cdot(5)\cdot(5) + (100)\cdot(30)\cdot(30) + (200)\cdot(30)\cdot(20) \\
&= 212750
\end{aligned}
$$

We can now show our main claim.

Theorem 3.8 (unbiasedness)
Given a weighted directed AND/OR search tree T of a graphical model, a value function $V(n)$ defined by Equation 3.9, and an abstraction a, $\hat{V}(n_\varnothing)$ computed by `AOAS` *is an unbiased estimate of $V(n_\varnothing)$, so that $\mathbb{E}[\hat{V}(n_\varnothing)] = V(n_\varnothing) = Z$.*

Proof. At each step, `AOAS` maintains a current partially sampled AND/OR tree, $\tilde{T}^{(t)}$, where t indexes algorithm steps immediately after an operation altering $\tilde{T}^{(t)}$ – namely, either after an expansion step (lines 12-13), abstraction step for a single abstract state (lines 24-30), or pruning step (line 36). The partial tree $\tilde{T}^{(t)}$ corresponds to the current (stochastic) partial AND/OR sub-tree of T after step t, with node weights assigned by the algorithm, that is being generated as a probe. Note that $\tilde{T}^{(t)}$ is an AND/OR tree along a truncated pseudo-tree, $\mathcal{T}'$, for `AOAS` $\mathcal{T}$' being generated along a depth-first traversal of $\mathcal{T}$. The frontier nodes of $\tilde{T}^{(t)}$ are represented by Fr, where Fr_X are frontier nodes associated with a leaf variable X in $\mathcal{T}'$. Abusing notation, variables in $\mathcal{T}'$ are also referred to as $\mathcal{T}$'.

We define an intermediate estimator of $V(n_X)$ over $\tilde{T}^{(t)}$ as follows:

$$\hat{V}^{(t)}(n_X) = \begin{cases} V(n_X), & \text{if } n_X \in Fr \\ \prod_{Y \in ch_{\mathcal{T}}(X) \setminus \mathcal{T}'} V(Y_{n_X}) \cdot \prod_{Y' \in ch_{\mathcal{T}'}(X)} \sum_{n_{Y'} \in ch_{Y'}(n_X)} \frac{w^{(t)}(n_{Y'})}{w^{(t)}(n_X)} \, c(n_{Y'}) \, \hat{V}^{(t)}(n_{Y'}), & \text{otherwise} \end{cases} \tag{3.17}$$

This recursive estimator combines information from the sampled nodes and estimated weights in $\tilde{T}^{(t)}$ with exact values of V for the current frontier Fr and for non-expanded OR nodes that are child of AND nodes in $\tilde{T}^{(t)}$ that correspond to branching variables in $\mathcal{T}$.

The truncated AND/OR tree $\tilde{T}^{(t)}$ is well defined AND/OR tree whose node costs are $\tilde{c}(n_Y) = \frac{w^{(t)}(n_Y)}{w^{(t)}(n_X)} \, c(n_Y)$ for node $n_Y \in ch_Y(n_X)$. Expressing Equation 3.17 in terms of OR nodes we get

$$\hat{V}^{(t)}(Y_{n_X}) = \sum_{n_Y \in ch_Y(n_X)} \frac{w^{(t)}(n_Y)}{w^{(t)}(n_X)} \, c(n_Y) \, \hat{V}^{(t)}(n_Y). \tag{3.18}$$

We also define an *ancestral branching value estimator* as

$$\hat{R}^{(t)}(n_X) = \begin{cases} \prod_{Y_{n'} \in Out(n' \in path(n_X))} \hat{V}^{(t)}(Y_{n'}) \end{cases} \tag{3.19}$$

or vacuously $\hat{R}^{(t)}(n_X) = 1$.

At $t = 0$, since $Fr = \{r\}$, we have $\hat{V}^{(0)}(n_\varnothing) = V(n_\varnothing) = Z$. At the last step $t = L$, all variables have been expanded and abstracted, and all pruning completed, and $\hat{V}^{(L)}(n_\varnothing)$ is equivalent to the estimate generated by the final `AOAS` probe. This can be seen by noting that when all variables have been expanded, the estimator defined in Equation 3.17 is equivalent to the estimator used by `AOAS` (line 39).

Thus, we need to show that $\mathbb{E}[\hat{V}^{(L)}(n_\varnothing)] = Z$, which is equivalent to showing $\mathbb{E}[\hat{V}^{(L)}(n_\varnothing)] = \hat{V}^{(0)}(n_\varnothing)$. We derive this identity by showing that at each step t, the intermediate estimator on r, $\hat{V}^{(t)}(n_\varnothing)$ does not change in expectation from any step t to $t+1$. In other words, we show that $\mathbb{E}[\hat{V}^{(t+1)}(n_\varnothing) - \hat{V}^{(t)}(n_\varnothing)] = 0$

The proof is based on the simple observation implied immediately from Lemma 3.7 that $\hat{V}^{(t)}(n_\varnothing)$, for any leaf X in $\mathcal{T}'$ unfolds to the expression:

$$\hat{V}^{(t)}(n_\varnothing) = \sum_{n_X \in Fr_X} w^{(t)}(n)\, \hat{\mathcal{R}}^{(t)}(n_X)\, g(n_X)\, V(n_X). \tag{3.20}$$

In other words, since $\tilde{T}^{(t)}$ is an AND/OR tree, it obeys the conditions of Lemma 3.7 relative to its frontier Fr_X. In this case, the cost of a path to a node n is $g(n) \cdot w(n)$, yielding the expression.

There are three `AOAS` operations that change the composition of $\tilde{T}^{(t)}$. The first is deterministic, involving expanding all surviving frontier nodes to one of their child variable OR nodes and then their corresponding AND child nodes (lines 12-13), yielding a new frontier. Since each node is fully expanded with respect to the next variable, the new expression for $\hat{V}^{(t+1)}(n)$ is just the recursively expanded expression according to Equation 3.17. Namely, if n_X was a frontier node having value $\hat{V}^{(t)}(n_X)$ at step t, then after expanding $\tilde{T}^{(t)}$ from n_X towards one child variable of X in $\mathcal{T}$ its new value will remain the same since in expression Equation 3.17, all the $\hat{V}^{(t+1)}$ quantities for the new nodes are exact and inherit their weight from n_X. Thus the expansion step does not change the value of nodes in $\tilde{T}^{(t)}$, and so $\hat{V}^{(t+1)}(n_\varnothing) = \hat{V}^{(t)}(n_\varnothing) \implies \mathbb{E}[\hat{V}^{(t+1)}(n_\varnothing) - \hat{V}^{(t)}(n_\varnothing)] = 0$.

The next operation is the abstraction step for an abstract state (lines 24-30), which is stochastic. The algorithm picks a set of nodes A_i having the same abstract state i from the frontier Fr_X at time t, where here X is the variable being abstracted across. As implied by

Lemma 3.7, we can express the contribution of A_i to $\hat{V}^{(t)}(n_\varnothing)$ by

$$C^{(t)}(A_i) = \sum_{n \in A_i} w^{(t)}(n)\ \hat{\mathcal{R}}^{(t)}(n)\ g(n)\ V(n) \tag{3.21}$$

where $\hat{V}^{(t)}(n_\varnothing) = \sum_{A_i \in A} C^{(t)}(A_i)$. At step $t+1$ only one of these nodes in A_i is kept (line 30), denoted n', according to random selection with probability q (line 27) and the weight of the kept node is updated (line 28). Therefore only the path to n' and its ancestral branching value contributes to $\hat{V}^{(t+1)}(n_\varnothing)$. The random selection does not affect its ancestral branching value nor any of the other paths outside A_i. The expectation of the difference in estimates from step t to $t+1$ therefore,

$$\mathbb{E}[\hat{V}^{(t+1)}(n_\varnothing) - \hat{V}^{(t)}(n_\varnothing)] = \tag{3.22}$$

$$\sum_{n' \in A_i} q(n')[w^{(t+1)}(n')\ \hat{\mathcal{R}}^{(t)}(n')\ g(n')\ V(n') - C^{(t)}(A_i)] =$$

(since $C^{(t)}$ is not dependent on n')

$$= \sum_{n' \in A_i} q(n')[\frac{w^{(t)}(n')}{q(n')}\ \hat{\mathcal{R}}^{(t)}(n')\ g(n')\ V(n')] - \sum_{n \in A_i} w^{(t)}(n)\ \hat{\mathcal{R}}^{(t)}(n) \cdot g(n) \cdot V(n) = 0$$

Finally, the last operation in question is the deterministic pruning of nodes which will never be part of a full solution tree (line 36). This occurs when OR nodes along a path are missing all of their AND children. (An example can be found in Figure 3.5c of our sample `AOAS` trace). Based on our estimator, the value of the parent AND node of our OR-node-without-children is in part based on the product of a summation over the OR node's children (see Equation 3.17). But the OR nodes has no children and so the sum evaluates vacuously to zero. Thus the parent AND node's value will also be zero. Since the zero-value AND node does not contribute to the overall estimate, removing these nodes does not change the estimate and we get that $\hat{V}^{(t+1)}(n') = \hat{V}^{(t)}(n') \implies \mathbb{E}[\hat{V}^{(t+1)}(n_\varnothing) - \hat{V}^{(t)}(n_\varnothing)] = 0$.

This shows that for all operations of `AOAS` that change the composition of the probe, $\mathbb{E}[\hat{V}^{(t+1)}(n_\varnothing) - \hat{V}^{(t)}(n_\varnothing)] = 0$ and so the expected estimate of the final probe $\mathbb{E}[\hat{Z}] = \mathbb{E}[\hat{V}^{(L)}(n_\varnothing)] = \mathbb{E}[\hat{V}^{(0)}(n_\varnothing)] = Z$. □

3.3.4 Additional Properties

Impact of the proposal. It is easy to see that the proof of unbiasedness works for any sampling distribution q. However, in practice we also propose to select the specific proposal,

$$q(n) := \frac{w(n) \cdot g(n) \cdot h(n) \cdot r(n)}{\sum_{m \in A_i} w(m) \cdot g(m) \cdot h(m) \cdot r(m)}, \tag{3.23}$$

to reduce variance of our estimate. In particular, we can see that if the heuristic h is exact, then using this proposal, `AOAS`'s estimate of the partition function is also exact.

Theorem 3.9 (exact proposal)
If the proposal q in AS algorithm is based on an exact heuristic $h(n) = V(n)$, namely if $q(n) = \frac{w(n')g(n')V(n')r(n')}{\sum_{n \in A_i} w^{(t)}(n) \cdot g(n) \cdot V(n) r(n)}$, then $\hat{V}$ is exact after one probe for any abstraction a.

Proof. We need to show that $\hat{V}(n_\varnothing) = V(n_\varnothing) = Z$ after a single probe. We use the same strategy as in the proof of unbiasedness. The only difference is that now we show that the estimate $\hat{V}^{(t)}(n_\varnothing)$ is preserved not only in expectation but also in the stochastic step (when any n' is chosen from an abstract state A_i).

We first note that the denominator of q is just $C^{(t)}$ from Equation 3.21, namely, $q =$

$\frac{w(n')g(n')V(n')r(n')}{C^{(t)}(A_i)}$. By definition,

$$\begin{aligned}\hat{V}^{(t+1)}(n_\varnothing) - \hat{V}^{(t)}(n_\varnothing) &= w^{(t+1)}(n')\hat{R}^{(t+1)}(n')g(n')V(n') - C^{(t)}(A_i) \\ &= \frac{w^{(t)}(n')}{q(n')}\hat{R}^{(t+1)}(n')g(n')V(n') - C^{(t)}(A_i)\end{aligned}$$

Substituting for $q(n')$ we get

$$\hat{V}^{(t+1)}(n_\varnothing) - \hat{V}^{(t)}(n_\varnothing) = w^{(t)}(n')g(n')V(n')\hat{R}^{(t+1)}(n')\frac{C^{(t)}(A_i)}{w^{(t)}(n')g(n')V(n')r(n')} - C^{(t)}(A_i)$$

Since $\hat{R}(n')$ is not affected by the process, we know $\hat{R}^{(t+1)}(n') = \hat{R}^{(t)}(n')$ and, substituting, we get

$$\hat{V}^{(t+1)}(n_\varnothing) - \hat{V}^{(t)}(n_\varnothing) = w^{(t)}(n')g(n')V(n')\hat{R}^{(t)}(n')\frac{C^{(t)}(A_i)}{w^{(t)}(n')g(n')V(n')r(n')} - C^{(t)}(A_i)$$

With an exact heuristic (namely $h(n) = V(n)$), the expression for $r(n')$ in `AOAS` equals that of $\hat{R}(n')$, and so substituting we get

$$\hat{V}^{(t+1)}(n_\varnothing) - \hat{V}^{(t)}(n_\varnothing) = w^{(t)}(n')g(n')V(n')r(n')\frac{C^{(t)}(A_i)}{w^{(t)}(n')g(n')V(n')r(n')} - C^{(t)}(A_i)$$

$$= C^{(t)}(A_i) - C^{(t)}(A_i) = 0$$

□

Impact of the abstraction. Any function over the nodes in the search space, and in particular a heuristic function, can be used as an abstraction function. Abstractions can facilitate the transition between search and sampling, and so between determinism and stochasticity. On one hand, we want abstractions with low cardinality in order to bound the probe's size. On the other, we want more *accurate* estimates, namely to sample with as small a variance as possible. In the spirit of this trade-off, we can compare abstractions using a notion of refinement.

Definition 3.4 (abstraction refinement hierarchy)
We say that an abstraction a_1 *is a refinement of* a_2 *iff for any two nodes* $n_i, n_j, a_1(n_i) = a_1(n_j) \implies a_2(n_i) = a_2(n_j)$.

Definition 3.5 (abstraction refinement of functions)
We say that an abstraction a *is a refinement of a function* Γ *iff for any two nodes* $n_i, n_j, a(n_i) = a(n_j) \implies \Gamma(n_i) = \Gamma(n_j)$.

The abstraction that maps all nodes associated with a single variable to the same abstract state is referred to as the *Knuth abstraction*. The abstraction that maps every node to itself is called the *trivial abstraction*. Clearly, the Knuth abstraction is the coarsest relevant abstraction and corresponds to standard importance sampling, while the trivial abstraction is the most refined abstraction, and corresponds to exact search. We explore a variety of abstractions with tunable levels of refinement, that can range between the Knuth and the trivial abstraction.

Definition 3.6 (Value-exact abstraction)
We say that an abstraction a *is exact if* a *is a refinement of* V*, namely* $a(n) = a(n') \Rightarrow V(n) = V(n')$.

Lemma 3.10 (Conditions for value-exact abstraction leading to exact estimates)
Assuming we use the proposal in algorithm `AOAS`*, if the abstraction* a *is a refinement of both* V *and* h *on a search space guided by a chain pseudo tree (i.e., corresponding to a classical OR search space), then* $\hat{V}(n_\varnothing)$ *is exact (i.e.* $\hat{V} = V = Z$*).*

Proof. In a search space guided by a chain pseudo tree (corresponding to a classical OR search space), there are no branching variables and so $R = r = 1$.

As before, for an abstraction step at step t yielding the probe at $t+1$ we have

$$\hat{V}^{(t+1)}(n_\varnothing) - \hat{V}^{(t)}(n_\varnothing)$$
$$= w^{(t+1)}(n') \cdot \hat{R}^{(t+1)}(n') \cdot g(n') \cdot V(n') - \sum_{n \in A_i} w^{(t)}(n) \cdot \hat{R}^{(t)}(n) \cdot g(n) \cdot V(n)$$

Substituting $w^{(t+1)}(n') = \frac{w^{(t)}(n')}{q(n')}$ (`AOAS`, line 28) we get

$$= \frac{w^{(t)}(n')}{q(n')} \cdot \hat{R}^{(t+1)}(n') \cdot g(n') \cdot V(n') - \sum_{n \in A_i} w^{(t)}(n) \cdot \hat{R}^{(t)}(n) \cdot g(n) \cdot V(n)$$

Recall that $\hat{R}^{(t)}(n')$ is based on the ancestors of n' and thus not affected by the abstraction process. Therefore we know $\hat{R}^{(t)}(n') = \hat{R}^{(t+1)}(n')$, when substituting giving

$$= \frac{w^{(t)}(n')}{q(n')} \hat{R}^{(t)}(n') \cdot g(n') \cdot V(n') - \sum_{n \in A_i} w^{(t)}(n) \cdot \hat{R}^{(t)}(n) \cdot g(n) \cdot V(n)$$

Substituting for $q(n')$ based on `AOAS`, line 25 gives,

$$= w^{(t)}(n') \cdot \hat{R}^{(t)}(n') \cdot g(n') \cdot V(n') \frac{\sum_{n \in A_i} w^{(t)}(n) \cdot r(n) \cdot g(n) \cdot h(n)}{w^{(t)}(n') \cdot r(n') \cdot g(n') \cdot h(n')}$$
$$- \sum_{n \in A_i} w^{(t)}(n) \cdot \hat{R}^{(t)}(n) \cdot g(n) \cdot V(n)$$
$$= \frac{V(n') \cdot \hat{R}^{(t)}(n')}{h(n') \cdot r(n')} \sum_{n \in A_i} w^{(t)}(n) \cdot r(n) \cdot g(n) \cdot h(n) - \sum_{n \in A_i} w^{(t)}(n) \cdot \hat{R}^{(t)}(n) \cdot g(n) \cdot V(n) \quad (3.24)$$

Since the pseudo tree is a chain, we substitute $R = r = 1$.

$$= \frac{V(n')}{h(n')} \sum_{n \in A_i} w^{(t)}(n) \cdot g(n) \cdot h(n) - \sum_{n \in A_i} w^{(t)}(n) \cdot g(n) \cdot V(n) \quad (3.25)$$

Since the abstraction function is a refinement of V and h, $V(n) = V_i$, a constant over A_i and so is h, we get that h cancels out yielding

$$= V_i \sum_{n \in A_i} w^{(t)}(n) \cdot g(n) - V_i \sum_{n \in A_i} w^{(t)}(n) \cdot g(n) = 0 \tag{3.26}$$

$\square$

Note that the proof relies on removing R, $\hat{R}$, and r in the transition from step 3.24 to 3.25. For AND/OR trees guided by a branching pseudo tree, it is no longer the case that each of these values must equal 1, and the proof no longer holds.

Let $\texttt{AOAS}(a, h)$ denotes a particular execution of $\texttt{AOAS}$ with abstraction a and heuristic h. With this notation $\texttt{AOAS}(1, h)$ is identical to importance sampling using h in the proposal function. As we will show next, if a is a refinement of hr then the heuristic will have no impact on the proposal.

Theorem 3.11

If a is a refinement of hr, then $\texttt{AOAS}(a, h) = \texttt{AOAS}(a, 1)$

Proof. In line 19 of $\texttt{AOAS}$ the proposal q is defined by $q(n) \leftarrow \frac{w(n) \cdot g(n) \cdot h(n) \cdot r(n)}{\sum_{m \in A_i} w(m) \cdot g(m) \cdot h(m) \cdot r(m)}$. Since a is a refinement of hr, hr is constant within each A_i and we get $q(n) \leftarrow \frac{w(n) \cdot g(n)}{\sum_{m \in A_i} w(m) \cdot g(m)}$. $\square$

Corollary 3.12

If an abstraction a is a refinement of h, then when abstracting over a search space guided by a chain pseudo tree, $\texttt{AOAS}(a, h) = \texttt{AOAS}(a, 1)$

Finally, we hypothesize that:

Proposition 3.13

If a and b are abstractions such that a is more refined than b, then the variance of the estimate using a is smaller than the variance using b, regardless of the proposal function, or h.

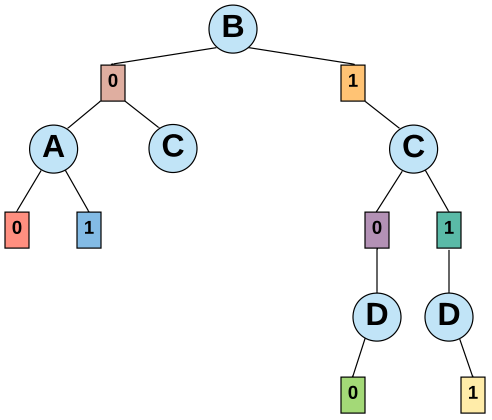


Figure 3.7: A sub-tree that could possible result from unrestricted abstraction across nodes of the variables. We consider such a tree as *invalid* as it does not capture any full configurations of the variables.

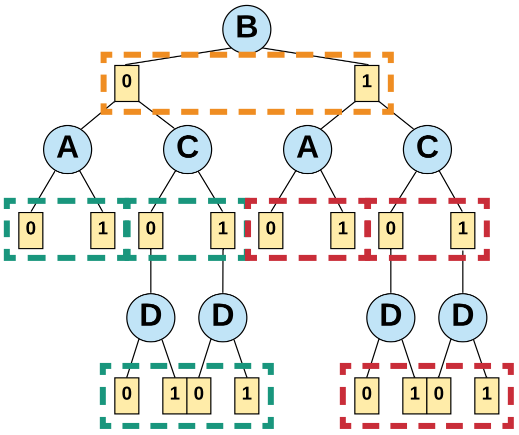


Figure 3.8: Grouping of nodes that `pAOAS` would allowed to be abstracted into the same abstract states via *proper* abstractions.

▢ 3.3.5 Complexity and scalability.

It is easy to see that `ORAS` and `AOAS` probe sizes will not be larger than $O(m \cdot n)$, where n is the number of variables and m bounds the number of abstract states per variable.

pAOAS. An earlier version of AND/OR Abstraction sampling extended *ORAS* to AND/OR search spaces in a straightforward manner by extending probes according to a traversal of the pseudo-tree $\mathcal{T}$ in a depth-first or breadth-first manner, but employed strict restrictions

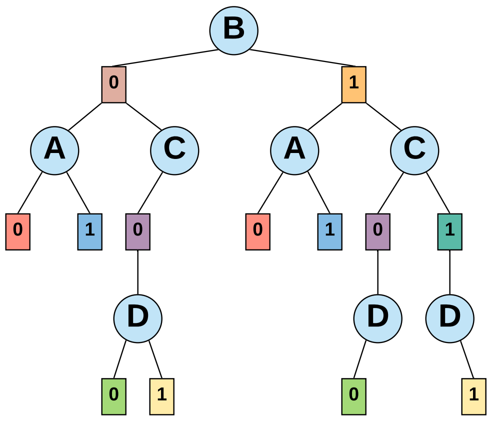


Figure 3.9: An example of a probe that could be produced by `pAOAS` employing *proper* abstractions.

regarding which nodes of a variable were allowed to be abstracted together. Since naively applying abstractions across all nodes of a variable in an AND/OR search space may leave partial (invalid) configurations (for example, Figure 3.7), such abstractions were prohibited. An algorithm called *Proper* `AOAS`, or `pAOAS` [Broka et al., 2018], avoided such probes by only allowing abstractions across nodes of a variable that were under the same AND node of the most recent branching variable ancestor in the AND/OR tree (see Figure 3.8). This ensures expansion to only valid probes. These restricted abstractions were called *proper* abstractions [Broka et al., 2018].

Definition 3.7 (Proper Abstractions)

Consider abstracting frontier nodes of variable X, F_X. Let $br^*(n_X)$ be the most recent ancestor node n_Y of n_X such that Y is a branching variable in $\mathcal{T}$. Now let

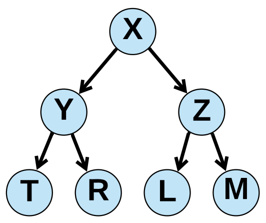
$$pA = \{pA_i \mid \bigcup_i pA_i = F_X; \bigcap_i pA_i = \emptyset; n, n' \in A_i \implies br^*(n) = br^*(n')\}.$$

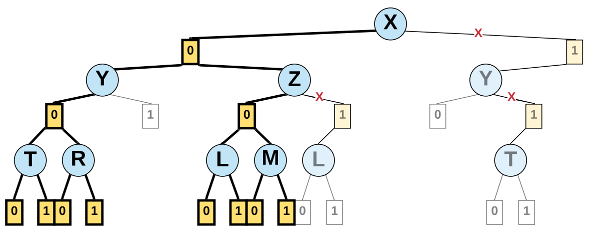
Proper abstractions over X are the set of abstractions that are equivalent to or refine pA.

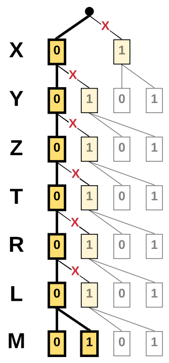
A possible probe resulting from a proper abstraction is illustrated in Figure 3.9. Since B was a branching variable and both of its AND node $n_{B=0}$ and $n_{B=1}$ were kept after abstraction, subsequent abstractions are constrained underneath their respective sub-trees resulting in a relatively large probe being generated.

Unfortunately, properness limits the diversity of generated probes and ability to control the size of the sampled AND/OR trees as it constrains the set of AND nodes that can be abstracted together into the same abstract state. In particular, instead of the probe size (number of AND nodes) being bounded by $O(nm)$, as in `ORAS`, we find that the number of AND nodes constructed via a proper abstraction can grow as quickly as $O(n \cdot m^{b+1})$, where n is the number of variables, b bounds the number of branching variables along any path, and m bounds the maximum number of abstract states per variable. This means that proper AND/OR abstraction schemes are not scalable whenever $b >> 0$, unless $m = 1$, in effect limiting us to basic Importance Sampling. Various ad hoc schemes were used in Broka et al. [2018] to control the probe sizes produced by proper abstractions. For more details of the algorithm and its analysis see Broka et al. [2018]. We illustrate the trade-off between OR abstraction and these two variants of AND/OR abstractions in an example, as well as via an analysis in the empirical evaluations (Section 5.10.

Contrasting Scalability. Figures 3.10b, 3.10c, and 3.10d show probes generated by `AOAS`, `ORAS`, and `pAOAS`, respectively, sampling over a search tree guided by the pseudo tree in Figure 3.10a. AND nodes are abstracted together according to their domain value. White nodes indicate nodes that were generated but discarded by the abstraction process. Faded yellow nodes are selected representatives from their respective abstract states that were later pruned, as indicated by a red "X". Yellow nodes constitute the final probe. In this example, `ORAS` generates 26 nodes, eight of which are retained in the final probe, which represents only two full configurations: $(X, Y, Z, T, R, L{=}0, M{=}0)$ and $(X, Y, Z, T, R, L{=}0, M{=}1)$. This is the smallest probe that can be generated by `ORAS` given the abstraction function used. In contrast, `pAOAS` has no choice but to expand and keep the entire AND/OR search tree of 42 nodes. This is a consequence of every variable being a branching variable in the pseudo-


(a) Guiding pseudo tree.


(b) `AOAS` Probe [11 nodes, 16 solutions].


(c) `ORAS` Probe [8 nodes, 2 solutions].

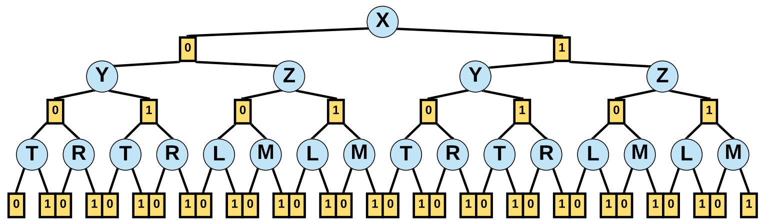

(d) `pAOAS` Probe [42 nodes, 128 solutions].

Figure 3.10: Contrasting scalability of various Abstraction Sampling schemes.

tree, and that no proper abstraction can span across these branches: for example, once our abstraction has retained $(X = 0)$ and $(X = 1)$, no subsequent abstraction step can merge or prune either of the sub-trees, causing the probe to grow uncontrollably. On the other hand, the resulting 42-node tree corresponds to the full space of 128 full configurations, thanks to the compactness of the AND/OR representation. Finally, `AOAS` gives both compactness and scale control. The illustrated probe has 11 nodes and had generated only 20 nodes during its construction, and represents 16 full configurations of the model, far more than `ORAS`. `AOAS` probes can vary in their size and representation, but always have as much or more representation as `ORAS`, while avoiding the uncontrolled growth of `pAOAS` probes.

Reducing Probe Size via Exact Heuristic Estimates. Using the Weighted Mini-Bucket heuristic, it is possible to identify when we reach a node for which the heuristic is exact (i.e. $h(n) = h^*(n) = V(n)$). When this occurs, there is no longer a need to extend the probe below the node as the value of the node - i.e., the Z value of the sub-tree it roots - is known exactly. This occurs for nodes whose variable, and all variable descendants in the guiding pseudo tree, have induced-width bounded by the i-bound i used by the weighted mini-bucket that generated the heuristic.

3.4 Candidate Abstractions

Since Abstraction Sampling is a type of stratified importance sampling, we drew inspiration from the stratified importance sampling domain, such as from Theorem 2.3: Rizzo [2007] which suggests several approaches in attempts to achieve variance reduction. One approach is to use abstractions that partition nodes into equal-mass abstract states under the proposal, mimicking the theorem's condition of equal-probability strata. An alternative approach is to try to minimize the variance of the true underlying mass *within abstract states* (thus maximizing variance of the total masses of the abstract states). The theorem's variance reduction expression is proportional to this quantity. A third approach is to employ more refined abstractions, with more abstract states per layer. At the same time, our search perspective suggests to always unify nodes that root identical sub-trees, whenever such information is available. One way to group nodes with identical subtrees is by using a graph notion called context, leading to what are called Context-based abstractions, described next.

3.4.1 Context-Based Abstractions

In Context-based abstractions, nodes are grouped into abstract states based on a graph notion of context. The **context** of a variable X in a pseudo-tree $\mathcal{T}$ identifies a subset $C(X)$ of its ancestor variables whose assignment uniquely determines the AND/OR sub-tree below the node [Dechter and Mateescu, 2007]. Therefore, if two nodes corresponding to variable X in the search tree that have the same context (namely, the same assignment to their context variables), and represent the same value $X = x$, then they root identical sub-trees. With this, we can say

Corollary 3.14

When the abstraction function is context-isomorphic, namely, $a(n) = a(n') \iff C(n) = C(n')$*, and if* h *is a mini-bucket heuristic, then the partition function estimate* $\hat{Z}$ *is exact for* `ORAS` *(or equivalently, for* `AOAS` *on a chain pseudo tree). This comes from the fact that the Mini-Bucket heuristic estimate is the same for nodes sharing the same full context (and so* a *also refines* h *in this case), and then follows directly from Lemma 3.10.*

This provides us intuition that the context is useful for abstracting nodes togeher and forms the rationale behind two possible families of abstraction functions: *relaxed context-based* and *randomized context-based* abstractions.

Definition 3.8 (Relaxed Context-based abstractions)

Let $C(X)$ *be the context of* X *according to pseudo-tree* $\mathcal{T}$*. An abstraction* a *at* X *is relaxed context-based relative to a subset* C_{Rel}*,* $C_{Rel} \subseteq C(X) \cup X$*, iff for every* n_1 *and* n_2 *having* $var(n_1) = var(n_2) = X$*, we have:* $a(n_1) = a(n_2) \iff \pi_{C_{Rel}}(n_1) = \pi_{C_{Rel}}(n_2)$*, where* $\pi_{C_{Rel}}(n)$ *is the projection of* $path(n)$ *onto* C_{Rel} *(namely, the assignments corresponding to the relaxed context,* C_{Rel}*). If* $|C_{Rel}| = j$ *we say that we use a* j*-level relaxed context-based abstraction. In particular, 0-level abstractions puts all the nodes of a variable in a single abstract state.*

When abstracting nodes of a variables according to j-level relaxed context-based abstractions, the number of abstract states is bounded by k^j, where k bounds the domain sizes of the relaxed-context variables.

The second family of context-based abstractions introduces randomness into the actual way the abstraction depends on the context. This can facilitate the generation of many different abstractions in an automated manner and potentially lead to tighter estimates. In what are called Randomized Context-Based abstraction, nodes are assined into abstract states based on the assignment to a variables full context and the value of a random hash function. The scheme ensures that nodes that share the same full context are placed into the same abstract state.

Definition 3.9 (randomized context-based abstraction)
Let $k \in \mathbb{I}^+$ and $d \in \mathbb{I}^+$ be parameters. Let N be the number of variables in the model and $K = \{1, 2, .., k\}$. We assume that all domains D_i of the variables are subsets of the positive integers. We construct an abstraction a, by first sampling a vector $\boldsymbol{\kappa} = (\kappa_1, \kappa_2, ..., \kappa_N)$ uniformly at random from K^N that will be used for hashing. Given a node n_X with partial assignment $path(n_{X_j}) = (x_1, x_2, x_3, ..., x_j)$, we compute its hash value $hash(n_{X_j}) = \sum_{i=1}^{j} \kappa_i x_i \mathbb{1}_{C(X_j)\cup X}(X_i)$, where $\mathbb{1}$ is the indicator function. We define its abstraction function value $a(n_{X_j}) = hash(n_{X_j}) \bmod nAbs$. The parameter $nAbs$ determines the number of abstract states for each layer of the tree.

■ 3.4.2 Value-Based Abstractions

We now introduce a new way to form abstractions that we call Value-Based Abstractions. They are defined by (1) a positive real-valued function $\mu : D_{\boldsymbol{X}} \to \mathbb{R}^+$, where $D_{\boldsymbol{X}}$ is a set of configurations for the variables $\boldsymbol{X}$, and by (2) a partitioning scheme ψ_μ that assigns nodes to abstract states based on their μ value and in an order-consistent manner as defined next.

Definition 3.10 (Value-Ordered Partitioning)
Given a parameter $nAbs$ bounding the number of abstract states and a function $\mu : D_{\boldsymbol{X}} \to \mathbb{R}^+$, a partitioning function $\psi_\mu : D_{\boldsymbol{X}} \to \{A_1, A_2, ...A_{nAbs}\}$, is order-consistent with μ if for any $n_1 \in \boldsymbol{A_i}$ and $n_2 \in \boldsymbol{A_j}$, $i < j \Leftrightarrow \mu(n_1) \leq \mu(n_2)$.

▩ 3.4.2.1 Value-Based Abstraction Classes

We introduce three Value-Based Abstraction classes, each characterized by a unique value function μ that signifies a notion of similarity between nodes.

Unlike partial or hashed contexts as used by the context-based abstraction schemes, heuristic estimates of $V(n)$ can often provide *quantitative* insight into potential similarities of $V(n)$ values. In particular, this intuition holds when using heuristics that provide bounds on $V(n)$ such as those via Weighted Mini-Bucket Elimination (wMBE) [Dechter and Rish, 2003, Liu and Ihler, 2011a]. Heuristic-Based (HB) Abstractions leverage this notion of similarity:

Definition 3.11 (Heuristic-Based Abstractions)
A Heuristic-Based (HB) abstraction uses $\mu(n) = h(n)$, where $h(n)$ is a heuristic estimate of $V(n)$.

However, recall that within AND/OR trees guided by a branching pseudo tree, the ancestor branching mass $R(n)$ plays an important role. Thus another intuition is to capture both $V(n)$ and $R(n)$ in our grouping of nodes. However, without access to $V(n)$ or $R(n)$ we cannot evaluate this product directly. Instead we can use the intuition that grouping based on estimated values of $h(n)r(n)$ may result in sets of nodes also with similar $V(n)R(n)$. This is captured by Heuristic and Ancestral Branching-Based (HRB) abstractions:

Definition 3.12 (Heuristic and Ancestral Branching-Based Abstractions)
Heuristic and Ancestral Branching-Based (HRB) abstractions use $\mu(n) = h(n)r(n)$, where $h(n)$ is a heuristic estimate of $V(n)$ and r(n) is an estimate of R(n).

A third intuition for generating abstractions is to group nodes by their estimated contribution to the overall partition function as estimated by $w(n)\varrho(n) = w(n)g(n)h(n)r(n)$. Q-Based abstractions aim to capture this:

Algorithm 5: $\Psi_{simpleVB}$

```
1  $baseCardinality \leftarrow \lfloor \frac{|\boldsymbol{n}|}{nAbs} \rfloor$
2  $extras \leftarrow |\boldsymbol{n}| \mod nAbs$
3  $\boldsymbol{n^*} \leftarrow SORT(\boldsymbol{n}, \mu, \text{low-to-high})$
4  $j_{begin} \leftarrow 1$
5  foreach $i \leftarrow 1, ..., nAbs$ do
6      if $extras > 0$ then
7          $j_{end} \leftarrow j_{begin} + baseCardinality$
8          $extras \leftarrow extras - 1$
9      else
10         $j_{end} \leftarrow j_{begin} + baseCardinality - 1$
11     $\boldsymbol{A_i} \leftarrow \{n^*_{j_{begin}}, ..., n^*_{j_{end}}\}$
12     $j_{begin} \leftarrow j_{end} + 1$
13 end
14 $\boldsymbol{A} \leftarrow \cup_{i=1}^{nAbs} \boldsymbol{A_i}$
15 return $\boldsymbol{A}$
```

Definition 3.13 (Q-Based Abstractions)

Q-Based (QB) abstractions use $\mu(n) = w(n)\varrho(n) = w(n)g(n)h(n)r(n)$, where $w(n)$ is the importance weight assigned to node n, $\varrho(n)$ is the estimate of $Q(n)$, $g(n)$ is the path cost from the root to n, $h(n)$ is a heuristic estimate of $V(n)$, and r(n) is an estimate of R(n).

3.4.2.2 Ordered Partitioning Schemes

Next we describe seven partitioning schemes ψ to be used with μ to partition the nodes into abstract states. Together, the value function μ and partitioning method ψ define a value-based abstraction function.

Example 3.4.2.1 (Running Node Partitioning Example)

Assume we have eight nodes with the following $\mu(n)$:

$$1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 10, 100 \tag{3.27}$$

and want to partition the nodes into $nAbs = 4$ abstract states. As we describe each partitioning scheme, we also illustrate how the scheme would partition these nodes.

For all of the partitioning schemes described below, if the number of nodes to be partitioned $|\boldsymbol{n}| \leq nAbs$, then each node $n \in \boldsymbol{n}$ is placed into its own abstract state.

We now present seven value-based node partitioning strategies.

1. simpleVB. The *simpleVB* (simple value-based) scheme groups nodes having similar $\mu(n)$ into the same state by: 1) nodes are ordered by $\mu(n)$ (low to high), and 2) nodes are partitioned into abstract states aiming towards (approximately]) equal size abstractions.

Running Example:

$1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 10, 100 \xrightarrow{simpleVB}$ {1.0, 1.1}, {1.2, 1.3}, {1.4, 1.5}, {10, 100}.

This method leverages speed while still aiming to roughly group nodes with similar $\mu(n)$ together.

2. minVarVB. *minVarVB* uses Ward's Minimum Variance Hierarchical Clustering, also known as Ward's Method [Ward, 1963] (Algorithm 6), to cluster nodes into *nAbs* abstract states. More specifically, *minVarVB* partitions $\boldsymbol{n}$ into abstract states $\boldsymbol{A} = \{\boldsymbol{A_1}, ..., \boldsymbol{A_{nAbs}}\}$ according to

$$\min_{\boldsymbol{A}}(Var_\mu(\boldsymbol{A_1}) + ... + Var_\mu(\boldsymbol{A_{nAbs}})). \tag{3.28}$$

Ward's Method can be combined with Lance-Williams linear distance updates [Lance and Williams, 1967] to increase efficiency.

Running Example:

$1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 10, 100 \xrightarrow{minVarVB}$ {1.0, 1.1, 1.2}, {1.3, 1.4, 1.5}, {10}, {100}.

In contrast to *simpleVB*, *minVarVB* takes more time due to Ward's Method's complexity, nevertheless provably forms abstractions that minimize the total within variance of $\mu(n)$

Algorithm 6: Ward's Method

1. **Initialization:** Treat each data point as an individual cluster. Assign each cluster a label.
2. **Compute Pairwise Distances:** Calculate the pairwise distances between all clusters. Various distance metrics can be used, such as Euclidean distance.
3. **Cluster Merging Iteration:**
   (a) Identify the pair of clusters $\boldsymbol{C_i}$ and $\boldsymbol{C_j}$ that, when merged into a new cluster $\boldsymbol{C_{ij}}$, results in the smallest increase in the overall within-cluster variance. This is determined using the formula:
   $\Delta Var = Var(\boldsymbol{C_{ij}}) - (Var(\boldsymbol{C_i}) + Var(\boldsymbol{C_j}))$
   where $Var(\boldsymbol{C_{ij}})$ is the variance of the merged cluster, and $Var(\boldsymbol{C_i})$ and $Var(\boldsymbol{C_j})$ are the variances of clusters $\boldsymbol{C_i}$ and $\boldsymbol{C_j}$, respectively.
   (b) Update distance measures between the newly merged cluster and all other clusters.
4. **Repeat:** Repeat steps 2-3 until the desired number of clusters is achieved.

among the abstract states.

3. equalDistVB. In attempt to combine the intuition from *minVarVB* and the speed of *simpleVB*, *equalDistVB* greedily adds nodes in order of μ (low to high) into an abstract state $\boldsymbol{A_i}$ until

$$\mu(\boldsymbol{A_{1,\dots,i}}) = \sum_{j=1}^{i} \sum_{n \in \boldsymbol{A_j}} \mu(n) \geq \mathcal{Q}_i = \frac{i \cdot \sum_{n' \in \boldsymbol{n}} \mu(n')}{nAbs}, \tag{3.29}$$

i.e., until the total sum of node values from $\boldsymbol{A_1}, \dots, \boldsymbol{A_i}$ reaches or exceeds $\frac{i}{nAbs}$ of the total across all of the nodes being partitioned. When paired with Q-based abstractions, *equalDistVB* aims to partition nodes into equal mass states under the proposal, motivated by Rizzo [2007].

Running Example:

$1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 10, 100 \xrightarrow{equalDistVB} \{1.0,\ 1.1,\ 1.2,\ 1.3,\ 1.4,\ 1.5,\ 10,\ 100\}, \{\}, \{\}, \{\}$

Although *equalDistVB* hopes to strike a balance between efficiency and low variance of $\mu(n)$ within each abstract state, from the running example we can see it may yield undesirable partitionings for skewed distributions of $\mu(\cdot)$ values. In the example, all of the nodes need to

```
Algorithm 7: $\Psi_{equalDistVB}$
1  $\boldsymbol{n^*} \leftarrow SORT(\boldsymbol{n}, \mu, \text{low-to-high})$
2  $j \leftarrow 1$
3  foreach $i \leftarrow 1, ..., nAbs$ do
4      $\boldsymbol{A_i} \leftarrow \{\}$
5      while $\mu(\boldsymbol{A_{1,...,i}}) < \mathcal{Q}_i$ do
6          $\boldsymbol{A_i} \leftarrow A_i \cup \{n^*_j\}$
7          $j \leftarrow j + 1$
8      end
9  end
10 $\boldsymbol{A} \leftarrow \cup_{i=1}^{nAbs} \boldsymbol{A_i}$
11 return $\boldsymbol{A}$
```

be placed into the first of four abstract states before the sum of their values reaches/exceeds $\frac{1}{4}$ of the total of all nodes being partitioned. Thus, the remaining abstract states end up empty.

4. equalDistVB2. A second version of the equalDist scheme, *equalDistVB2* uses a reversed sort ordering in attempt to mitigate overfilling of abstract states. Modifying the sort order from low-to-high to high-to-low in Line 1 of Algorithm 7 converts *equalDistVB* to *equalDistVB2*.

Running Example:

$$1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 10, 100 \xrightarrow{equalDistVB2} \{100\},\ \{\},\ \{\},\ \{10,\ 1.5,\ 1.4,\ 1.3,\ 1.2,\ 1.1,\ 1.0\}$$

We see that *equalDistVB2* can still over-pack abstract states. The next two variants aim to mitigate this issue further.

5. equalDistVB3. In order to further lessen over-packing and ensure abstract states are not left empty, *equalDistVB3* modifies *equalDistVB2* so that, after processing each abstract state, the next state always has a node added to it by default before checking the abstract state fill condition. In Algorithm 7, modifying the sort order from low-to-high to high-to-low in line 1 and $\boldsymbol{A_i} \leftarrow \{\}$ to $\boldsymbol{A_i} \leftarrow \{n^*_j\}; j \leftarrow j + 1;$ in line 4 converts *equalDistVB* to *equalDistVB3*.

Running Example:

$1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 10, 100 \xrightarrow{equalDistVB3} \{100\},\ \{10\},\ \{1.5\},\ \{1.4,\ 1.3,\ 1.2,\ 1.1,\ 1.0\}$.

While still very efficient, *equalDistVB3* ensures that the provided $nAbs$ granularity is honored, allowing users better control of the search vs. sampling interpolation possible with Abstraction Sampling.

6. equalDistVB4. The final equalDist variant, *equalDistVB4*, aims for more even partitioning. Before processing each abstract state $\boldsymbol{A_i}$, a new cut-off is determined based the remaining nodes $\boldsymbol{n_{rm_i}} = \boldsymbol{n} \setminus \bigcup_{j \in \{1,\ldots,i-1\}} \boldsymbol{A_j}$ and remaining abstract states $nAbs_{rm_i} = nAbs - i + 1$:

$$\begin{aligned}\widehat{\mathcal{Q}}_i &= \frac{\sum_{n \in \boldsymbol{n} \setminus \bigcup_{j \in \{1,\ldots,i-1\}} \boldsymbol{A_j}} \mu(n)}{nAbs - i + 1} \\ &= \frac{\sum_{n \in \boldsymbol{n_{rm_i}}} \mu(n)}{nAbs_{rm_i}}.\end{aligned} \tag{3.30}$$

Starting from $i = 1$, nodes are added to abstract state $\boldsymbol{A_i}$ while $\mu(\boldsymbol{A_i}) < \widehat{\mathcal{Q}}_i$. Modifying the sort order from low-to-high to high-to-low in Line 1 and $\mu(\boldsymbol{A_{1,\ldots,i}}) < \mathcal{Q}_i$ to $\mu(\boldsymbol{A_i}) < \widehat{\mathcal{Q}}_i$ in Line 5 of Algorithm 7 converts *equalDistVB* to *equalDistVB4*.

Running Example:

$1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 10, 100 \xrightarrow{equalDistVB4} \{100\},\ \{10\},\ \{1.5,\ 1.4,\ 1.3\},\ \{1.2,\ 1.1,\ 1.0\}$.

Still computationally efficient, *equalDistVB4* spreads nodes with small values more evenly across abstract states.

7. randVB. It can be beneficial to rely on randomness to ensure a diverse sampling of abstractions. *randVB* does this by sampling $nAbs - 1$ partition points uniformly at random and without replacement from between nodes sorted according to $\mu(\cdot)$, and then partitions the nodes accordingly. The resulting abstract states ensure that nodes are still grouped according to $\mu(\cdot)$, but the sizes of those groups vary.

Algorithm 8: Ψ_{randVB}

```
s ∼ $Unif(\{\boldsymbol{M} \subseteq \{1, ..., |\boldsymbol{n}| - 1\} \mid |\boldsymbol{M}| = nAbs - 1\})$
$\boldsymbol{s^*} \leftarrow SORT(\boldsymbol{s})$
$\boldsymbol{n^*} \leftarrow SORT(\boldsymbol{n}, \mu, \text{high-to-low})$
$j \leftarrow 1$
foreach $i \leftarrow 1, ..., nAbs-1$ do
    $\boldsymbol{A_i} \leftarrow \{n^*_j, ..., n^*_{s^*_i}\}$
    $j \leftarrow s^*_i + 1$
end
$\boldsymbol{A_{nAbs}} = \{n^*_j, ..., n^*_{|n^*|}\}$
$\boldsymbol{A} \leftarrow \cup_{i=1}^{nAbs} \boldsymbol{A_i}$
return $\boldsymbol{A}$
```

Running Example:

ex1: $1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 10, 100 \xrightarrow{randVB}$ {100, 10}, {1.5}, {1.4, 1.3, 1.2}, {1.1, 1.0};

ex2: $1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 10, 100 \xrightarrow{randVB}$ {100}, {10, 1.5, 1.4, 1.3}, {1.2, 1.1}, {1.0};

etc.

Complexity. Assuming $\mu(\cdot)$ is $\mathcal{O}(1)$, each of the proposed partitioning schemes have time complexity $\mathcal{O}(|\boldsymbol{n}| \log |\boldsymbol{n}|)$ and space complexity $\mathcal{O}(|\boldsymbol{n}|)$, with the exception of *minVarVB*, which requires $\mathcal{O}(|\boldsymbol{n}|^2)$ for both.

3.4.3 Random-Only Abstractions

Another approach is to use purely randomized abstraction schemes. At first glance, one may not expect such schemes to perform well, but randomization in concert with an informative heuristic and proposal may be beneficial.

Intuition. With a good proposal function, stochastic selection of a representative node *within* each abstract state favors nodes with greater mass. Random node assignments to abstract states then increase the chances of selecting nodes that might otherwise be overlooked, promoting a more diverse probe distribution.

Algorithm. More concisely referred to as RAND, the simpleRand scheme partitions nodes by: 1) first shuffling nodes to create a uniformly random permutation, and then 2) partitioning the nodes into $nAbs$ number of abstract states of (approximately) equal size.

Algorithm 9: $\Psi_{simpleRand}$

```
1  $baseCardinality \leftarrow \lfloor \frac{|\boldsymbol{n}|}{nAbs} \rfloor$
2  $extras \leftarrow |\boldsymbol{n}| \mod nAbs$
3  $\boldsymbol{n^*} \leftarrow RANDOM\_SHUFFLE(\boldsymbol{n})$
4  $j_{begin} \leftarrow 1$
5  foreach $i \leftarrow 1, ..., nAbs$ do
6      if $extras > 0$ then
7          $j_{end} \leftarrow j_{begin} + baseCardinality$
8          $extras \leftarrow extras - 1$
9      else
10         $j_{end} \leftarrow j_{begin} + baseCardinality - 1$
11     $\boldsymbol{A_i} \leftarrow \{n^*_{j_{begin}}, ..., n^*_{j_{end}}\}$
12     $j_{begin} \leftarrow j_{end} + 1$
13 end
14 $\boldsymbol{A} \leftarrow \cup_{i=1}^{nAbs} \boldsymbol{A_i}$
15 return $\boldsymbol{A}$
```

Running Example:

ex1: $1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 10, 100 \xrightarrow{simpleRand}$ {1.4, 1.1}, {1.2, 10}, {1.0, 1.3}, {100, 1.5};

ex2: $1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 10, 100 \xrightarrow{simpleRand}$ {1.0, 10}, {100, 1.5}, {1.4, 1.2}, {1.1, 1.3};

etc.

Complexity. Both time and space are $\mathcal{O}(|\boldsymbol{n}|)$.

3.5 Empirical Evaluation

We performed empirical evaluations to compare the various abstraction sampling algorithms as well as different abstraction functions. Main objectives were to observe properties of the algorithms and abstractions functions in practice, compare performance of various setups, and ultimately propose an expected best performing setup.

▢ 3.5.1 Setup

Benchmarks. High-throughput experiments were performed on over 480 problems from five well known partition function benchmarks: DBN, Grids, Linkage-Type4, Pedigree, and Promedas.

DBN. Deep Boltzmann Networks (DBNs) are a class of probabilistic graphical models composed of multiple layers of hidden units with undirected connections. They extend Restricted Boltzmann Machines (RBMs) by stacking several RBMs on top of each other, enabling the network to model complex data distributions. In DBNs, the lower layers capture low-level features, while the higher layers capture more abstract representations. DBNs are useful in unsupervised learning for tasks such as feature learning, dimensionality reduction, and generative modeling.

Grids. Grid probabilistic graphical models have a grid-like graph structure and are commonly used to represent spatial relationships. They are particularly useful in applications like image processing where pixels can be represented as nodes in a grid.

Pedigree and Linkage-Type4. Pedigree graphical models are probabilistic frameworks used to represent family relationships and inheritance patterns, particularly for genetic traits. These models consist of directed acyclic graphs where nodes denote individuals and edges represent parent-child connections. They incorporate Mendelian genetics to assess the transmission of traits and genetic disorders across generations. Pedigree models are essential in genetic counseling and medical research, as they help analyze the inheritance of traits and identify carriers of genetic conditions. The Linkage-Type4 benchmark is a particularly large version of such models, each consisting of over 5000 variables, more than triple that of many other large benchmarks used. We introduce this benchmark to challenge our algorithms and better demonstrate their performance.

Table 3.1: Exact Benchmark Statistics. Average statistics for Exact problems. **N**: number of instances, $|\mathbf{X}|$: average number of variables, **k**: average of problems' largest domain sizes, $\mathbf{w}^*$: average induced tree-width, **d**: average $\mathcal{T}$ depth.

Benchmark	N	$\|\mathbf{X}\|$	k	w*	d
DBN	66	67	2	29	30
Grids	8	250	2	22	49
Pedigree	25	690	5	25	89
Promedas	65	612	2	21	62

Promedas. Promedas, short for PRObabilistic MEdical Diagnostic Advisory System, is a probabilistic graphical model designed for medical diagnostics. It integrates a large database of medical knowledge, consisting of thousands of diagnoses and symptoms, and models the relationships between them. The system uses a Bayesian network framework, allowing it to calculate the probability of various diagnoses given a patient's symptoms. Graphical model inference algorithms can be used on Promedas instances to suggest possible medical conditions based on observed clinical data [Wemmenhove et al., 2007].

The benchmarks were split into two groups based on difficulty:

Exact: problems for which partition function solutions are tractable to compute exactly and for which exact solutions are known.

LARGE: problems for which partition function solutions are not tractable to compute exactly and for which exact solutions are not known.

Statistics for the Exact benchmarks are shown in table 3.1, and those for the LARGE benchmarks shown in table 3.2.

Implementation and Compute. All algorithms were implemented in C++. All experiments were run on a 2.66 GHz processor.

Heuristic. To inform the sampling proposal, we use Weighted Mini-Bucket Elimination (wMBE) Section 2.3.1.4 as the heuristic function. WMBE is known to pair well with both OR

Table 3.2: LARGE Benchmark Statistics. Average statistics for LARGE problems. **N**: number of instances, $|\mathbf{X}|$: average number of variables, **k**: average of problems' largest domain sizes, $\mathbf{w}^*$: average induced tree-width, **d**: average $\mathcal{T}$ depth.

Benchmark	N	$\lvert\mathbf{X}\rvert$	k	w*	d
DBN	48	216	2	78	78
Grids	19	3432	2	117	220
Linkage-Type4	82	6550	5	45	761
Promedas	173	1194	2	72	114

and AND/OR search [Mateescu and Dechter, 2005]. The i-bound (**iB**) parameter controls the strength of the wMBE heuristic; higher i-bounds generally lead to stronger heuristics, and thus better proposals, at the expense of requiring more time and memory to compute. We standardize our experiments by using the same i-bound when comparing across algorithms.

Variable Order. Every problem instance is associated with a particular variable ordering that all algorithms use. The ordering selection process attempts to minimize the resulting induced width and thus determines the structure of the pseudo-tree and the effectiveness of the heuristic. The statistics shown in table 3.1 and table 3.2 are computed based on these orderings.

Competing Schemes. The Abstraction Sampling algorithms were compared against Importance Sampling (IS) and state-of-the-art Dynamic Importance Sampling (`DIS`) [Lou et al., 2019] using an "*equal-time*" policy, which produces probabilistic bounds as well as estimates in an anytime manner. Additionally, the work of Broka et al. [2018] showed favorable performance of `ORAS` and `pAOAS` using context-based abstractions against Weighted Mini-Bucket Importance Sampling (`wMBIS`) [Liu et al., 2015] and IJGP-SampleSearch (`IJGP-ss`) [Gogate and Dechter, 2011], and thus by comparing here `AOAS` performance with `ORAS` and `pAOAS`, we also implicitly compare against these `wMBIS` and `IJGP-ss` as well.

■ 3.5.2 Abstraction Sampling Algorithm Comparisons

In order to compare properties and performance of various Abstraction Sampling algorithms, a first set of experiments were run on three classes of AS algorithms: `AOAS`, `pAOAS`, and `ORAS`, each using a depth-first ordering of the same pseudo tree per problem to guide the construction of the search space.

Abstraction Functions. To compare the various abstraction sampling schemes, we used Relaxed Context-Based (RelCB) and Randomized Context-Based (RandCB) abstraction functions with varying abstraction granularities. For RelCB, we used relaxed context sizes ranging from 0 to 8. `AOAS` and `ORAS` using RelCB with *Abs* of 0 abstract all nodes of a variable into a single abstract state and correspond to standard Importance Sampling of their respective trees. We bounded the number of abstract states RandCB will form per variable to either 16 or 128. Note that for a model containing only binary variables, RelCB with a relaxed context size of j (denoted RelCB-j) bounds the number of abstract states per level to 2^j.

Additional Setup Details. All experiments were run for one hour on each instance, using an 8 GB memory limit (increased to 24 GB for the larger Linkage-Type4 benchmark problems).

Performance Measure. To evaluate the performance of the various algorithms, we calculate error as: $error = \log_{10} \hat{Z} - \log_{10} Z^*$, where $\log_{10} \hat{Z}$ is the $\log_{10}$ of the experimentally obtained Z estimate, and $\log_{10} Z^*$ is the reference $\log_{10} Z$ value. When the exact Z value is unknown, an empirical estimate based on an average over 100×1hr of abstraction sampling is used as the reference. We verified that 98% of these estimates fell within the 95% probabilistic bounds determined by the `DIS` scheme.

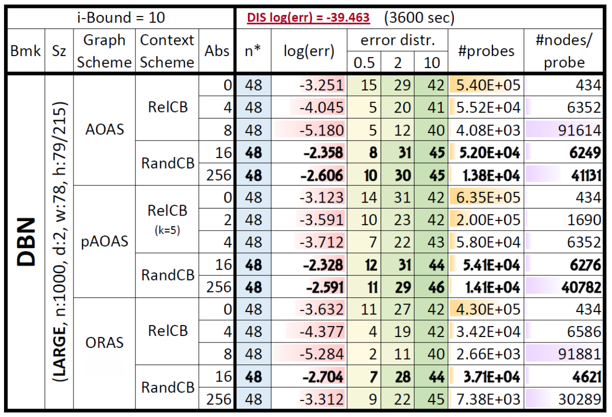
We also count the number of problems an algorithm is able to produce a non-zero estimate for. We refer to this count as the "*number of problems solved*". It's important to recognize that "solving" more complex problems can lead to less accurate estimates. However, even a less accurate estimate is preferable to not having any estimate at all. Therefore, the number of problems solved remains an important factor to consider.

Primary Questions of Interest. Our empirical evaluations aim to address the following key points/questions:

1. *Scalability*: How well can the probe size be controlled across the various algorithms? A good Abstraction Sampling algorithm will allow users the ability to control probe sizes according to their time/memory constraints.
2. *OR vs AND/OR*: What is the effect of using the more compact AND/OR search space? Can AND/OR schemes control probe sizes and scale well, and also provide better estimates?
3. *Accuracy*: How does the performances of `AOAS`, `pAOAS`, and `ORAS` compare to each other, and how do they compare to that of other state-of-the-art schemes?

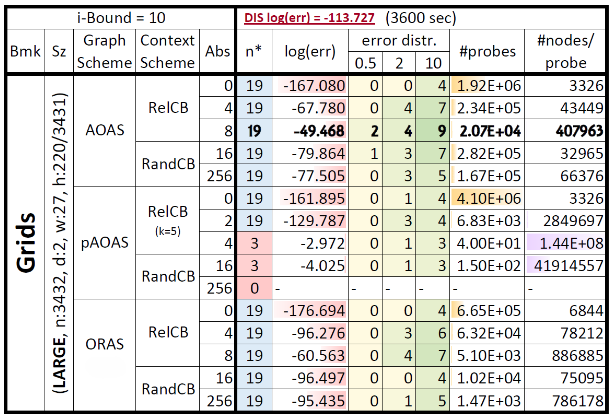
■ 3.5.2.1 Aggregated Results

Figure 3.11 shows experimental results averaged across instances of LARGE DBN, Grids, Linkage-Type4, and Promedas benchmarks separately. Experiments were performed using `AOAS`, `pAOAS`, and `ORAS`, each run with with RelCB and RandCB abstraction functions using different abstraction granularities (*Abs*). In the table, (n*) are the number of problems for which a non-zero estimate was found, (*log(err)*) is the average $\log_{10} Z$ error, (*error distr.*) counts the number of problems solved within the error threshold shown, (*#probes*) is the average number of probes generated, and (*#nodes/probe*) is the average size of a probe. Color bars act as visual aids for quickly observing relative magnitudes of values. Red n* cells indicate an algorithm's inability to solve relatively many problems. Lines in bold



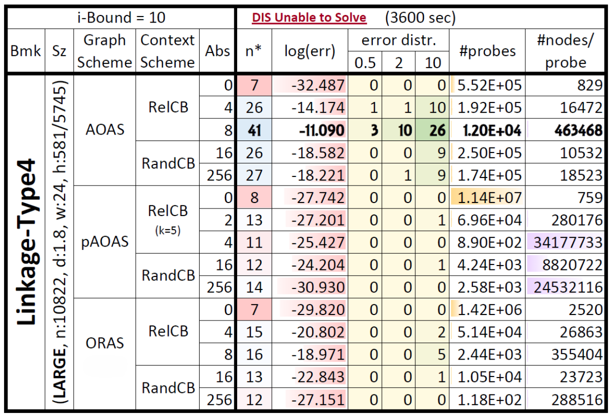
i-Bound = 10					DIS log(err) = -39.463 (3600 sec)						
Bmk	Sz	Graph Scheme	Context Scheme	Abs	n*	log(err)	error distr.			#probes	#nodes/ probe
							0.5	2	10		
DBN	(LARGE, n:1000, d:2, w:78, h:79/215)	AOAS	RelCB	0	48	-3.251	15	29	42	5.40E+05	434
				4	48	-4.045	5	20	41	5.52E+04	6352
				8	48	-5.180	5	12	40	4.08E+03	91614
			RandCB	16	**48**	**-2.358**	**8**	**31**	**45**	**5.20E+04**	**6249**
				256	**48**	**-2.606**	**10**	**30**	**45**	**1.38E+04**	**41131**
		pAOAS	RelCB (k=5)	0	48	-3.123	14	31	42	6.35E+05	434
				2	48	-3.591	10	23	42	2.00E+05	1690
				4	48	-3.712	7	22	43	5.80E+04	6352
			RandCB	16	**48**	**-2.328**	**12**	**31**	**44**	**5.41E+04**	**6276**
				256	**48**	**-2.591**	**11**	**29**	**46**	**1.41E+04**	**40782**
		ORAS	RelCB	0	48	-3.632	11	27	42	4.30E+05	434
				4	48	-4.377	4	19	42	3.42E+04	6586
				8	48	-5.284	2	11	40	2.66E+03	91881
			RandCB	16	**48**	**-2.704**	**7**	**28**	**44**	**3.71E+04**	**4621**
				256	48	-3.312	9	22	45	7.38E+03	30289

(a) DBN



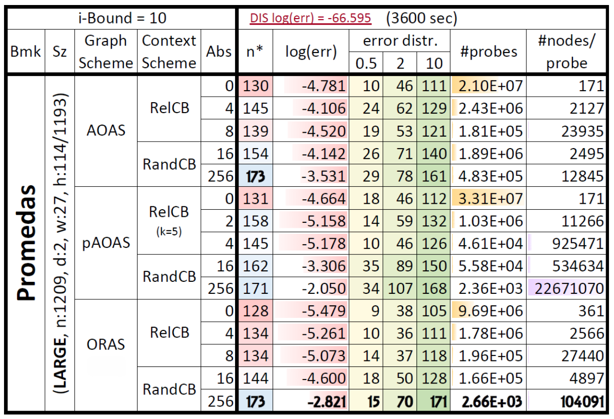
i-Bound = 10					DIS log(err) = -113.727 (3600 sec)						
Bmk	Sz	Graph Scheme	Context Scheme	Abs	n*	log(err)	error distr.			#probes	#nodes/ probe
							0.5	2	10		
Grids	(LARGE, n:3432, d:2, w:27, h:220/3431)	AOAS	RelCB	0	19	-167.080	0	0	4	1.92E+06	3326
				4	19	-67.780	0	4	7	2.34E+05	43449
				8	**19**	**-49.468**	**2**	**4**	**9**	**2.07E+04**	**407963**
			RandCB	16	19	-79.864	1	3	7	2.82E+05	32965
				256	19	-77.505	0	3	5	1.67E+05	66376
		pAOAS	RelCB (k=5)	0	19	-161.895	0	1	4	4.10E+06	3326
				2	19	-129.787	0	3	4	6.83E+03	2849697
				4	3	-2.972	0	1	3	4.00E+01	1.44E+08
			RandCB	16	3	-4.025	0	1	3	1.50E+02	41914557
				256	0	-	-	-	-	-	-
		ORAS	RelCB	0	19	-176.694	0	0	4	6.65E+05	6844
				4	19	-96.276	0	3	6	6.32E+04	78212
				8	19	-60.563	0	4	7	5.10E+03	886885
			RandCB	16	19	-96.497	0	0	4	1.02E+04	75095
				256	19	-95.435	0	1	5	1.47E+03	786178

(b) Grids



i-Bound = 10					DIS Unable to Solve (3600 sec)						
Bmk	Sz	Graph Scheme	Context Scheme	Abs	n*	log(err)	error distr.			#probes	#nodes/ probe
							0.5	2	10		
Linkage-Type4	(LARGE, n:10822, d:1.8, w:24, h:581/5745)	AOAS	RelCB	0	7	-32.487	0	0	0	5.52E+05	829
				4	26	-14.174	1	1	10	1.92E+05	16472
				8	**41**	**-11.090**	**3**	**10**	**26**	**1.20E+04**	**463468**
			RandCB	16	26	-18.582	0	0	9	2.50E+05	10532
				256	27	-18.221	0	1	9	1.74E+05	18523
		pAOAS	RelCB (k=5)	0	8	-27.742	0	0	0	1.14E+07	759
				2	13	-27.201	0	0	1	6.96E+04	280176
				4	11	-25.427	0	0	0	8.90E+02	34177733
			RandCB	16	12	-24.204	0	0	1	4.24E+03	8820722
				256	14	-30.930	0	0	0	2.58E+03	24532116
		ORAS	RelCB	0	7	-29.820	0	0	0	1.42E+06	2520
				4	15	-20.802	0	0	2	5.14E+04	26863
				8	16	-18.971	0	0	5	2.44E+03	355404
			RandCB	16	13	-22.843	0	0	1	1.05E+04	23723
				256	12	-27.151	0	0	0	1.18E+02	288516

(c) Linkage-Type4



i-Bound = 10					DIS log(err) = -66.595 (3600 sec)						
Bmk	Sz	Graph Scheme	Context Scheme	Abs	n*	log(err)	error distr.			#probes	#nodes/ probe
							0.5	2	10		
Promedas	(LARGE, n:1209, d:2, w:27, h:114/1193)	AOAS	RelCB	0	130	-4.781	10	46	111	2.10E+07	171
				4	145	-4.106	24	62	129	2.43E+06	2127
				8	139	-4.520	19	53	121	1.81E+05	23935
			RandCB	16	154	-4.142	26	71	140	1.89E+06	2495
				256	**173**	-3.531	29	78	161	4.83E+05	12845
		pAOAS	RelCB (k=5)	0	131	-4.664	18	46	112	3.31E+07	171
				2	158	-5.158	14	59	132	1.03E+06	11266
				4	145	-5.178	10	46	126	4.61E+04	925471
			RandCB	16	162	-3.306	35	89	150	5.58E+04	534634
				256	171	-2.050	34	107	168	2.36E+03	22671070
		ORAS	RelCB	0	128	-5.479	9	38	105	9.69E+06	361
				4	134	-5.261	10	36	111	1.78E+06	2566
				8	134	-5.073	14	37	118	1.96E+05	27440
			RandCB	16	144	-4.600	18	50	128	1.66E+05	4897
				256	**173**	**-2.821**	**15**	**70**	**171**	**2.66E+03**	**104091**

(d) Promedas

Figure 3.11: Aggregated statistics. Aggregated results are shown for `AOAS`, `pAOAS`, and `ORAS` (using a depth-first ordering of $\mathcal{T}$) with RelCB and RandCB on problem instances of the LARGE DBN, Grids, Linkage-Type4, and Promedas benchmarks using different abstraction granularities (*Abs*). For RelCB, the abstraction granularities correspond to the size of relaxed context to be used. For RandCB the granularity bounds the number of abstract states per level. Displayed are the number of problems for which a non-zero estimate was found ($n*$), average $\log_{10} Z$ error (*log(err)*), count of problems solved within an error threshold (*error distr.*), average number of probes (*#probes*), and average size of a probe (*#nodes/probe*). Color bars visually show the magnitude of the values, and darker colors show greater values. Red $n*$ cells indicate an algorithm's inability to solve relatively many problems. Lines in bold indicate the best performing algorithms. Each benchmark also displays the average number of nodes (n), domain size (d), tree-width (w), and AND/OR and OR search tree height (h) of its problems. Also shown in red text at the top of the table is competing state-of-the-art scheme DIS's average results on the instances of each benchmark.

indicate the best performing algorithms. Each benchmark also displays the average number of nodes (n), domain size (d), the maximum bucket width encountered by the heuristic (w), and AND/OR and OR search tree height (h) for the problems. Also shown in red text at the top of the table is competing state-of-the-art scheme DIS's average results on the instances of each benchmark. The results from the Pedigree benchmark are omitted as its problem instances were easy for all algorithms and no significant results were observed. In the following paragraph we will look more closely at how probe size scales with increased granularity of abstractions, specifically focusing on how `AOAS` and `pAOAS` can control probe sizes.

Size of Probes. One of the hallmarks of the new `AOAS` algorithm is its ability to circumvent the properness requirement of `pAOAS` while still remaining an unbiased estimator. This provides a significant advantage in graphs whose pseudo-tree branches to capture useful conditional independence relationships: by giving finer-grain control over the abstraction state sizes, we can better control the size of each probe and draw samples more quickly.

Examining the probe sizes of `AOAS` across each problem type in the aggregation tables we see that as the context size increases (the column "Abs" in each table), the size of the `AOAS` probes grow more slowly than pAOAS, allowing `AOAS` to better regulate its probe size. For example, within the Linkage-Type4 results we see the average probe size for `pAOAS` using RelCB go from 829 to 463468 when increasing granularity from 0 to 8, whereas for `pAOAS` the average size jumps from 759 to 34177733 just going from granularity 0 to 4.

Notabley, `AOAS` probes were often smaller than those of `ORAS` for the same abstraction schema with larger granularities. For example, for RandCB-256 on the Grids problems, `AOAS`'s probes averaged 66376 nodes per probe, where as `ORAS`'s probes were significantly larger averaging 786178 nodes per probe.

AOAS Performance. An important goal of our empirical analysis is to identify which scheme, if any, stands out as being the most competitive. In order to capture relative performance, we look at two statistics: (i) the number of problems solved for each benchmark, and (ii) the average error for those problems.

AOAS is always among the algorithms that solve the greatest number of problems within each benchmark - solving 48 problems for DBN, 19 for grids, and 173 for Promedas - and was the only scheme able to solve 41 problems within the Linkage-Type4 benchmark, with the runner up being ORAS which was only able to solve 16 instances.

Furthermore, AOAS's average error across the benchmarks is always among the smallest, sometimes much smaller than that of the other schemes. For example, in the Linkage-Type4 benchmark AOAS with RelCB-8 had an average error of -11.090 whereas the next smallest error from any other scheme was -18.971 by ORAS also with RelCB-8. However, AOAS did not always have the overall smallest average error. For example, for DBN ORAS with RandCB-128 had a somewhat smaller average error of -2.821 as compared to AOAS with RandCB-128 which did slightly worse with an average error of -3.531).

In comparison with pAOAS, AOAS's better overall performance is likely in part due to being able to form more flexible abstractions which can allow for more diverse probes and more advantageous grouping of nodes. In comparison with ORAS, AOAS's better performance is likely in part due to its probes being able to take advantage of the AND/OR search space which captures more configurations given comparable probe sizes. Furthermore, we explained earlier that sometimes AOAS can produce smaller probes simply because of being able to take advantage of an exact heuristic more often, which can lead to more samples being drawn (as can be seen by the *#probes* column).

Comparing with Other Non Abstraction Sampling Schemes. For all LARGE benchmarks, both ORAS and AOAS with RelCB-0 (i.e., basic Importance Sampling) were outper-

Problem	**Size**	**Total**	**∈Bnds**	**AOAS≥**	**AOAS>**
DBN	Exact	66	62	57	47
	LARGE	48	40	38	35
Grids	Exact	8	5	5	2
	LARGE	19	7	7	6
Linkage*	LARGE	82	82	82	82
Pedigree	Exact	24	24	24	19
Promedas	Exact	65	58	49	29
	LARGE	173	165	141	113

Table 3.3: How often `AOAS` estimates: fall within `DIS` probabilistic bounds ($\in Bnds$), were comparable/better than `DIS`'s ($AOAS \geq$), and were strictly better than `DIS`'s ($AOAS >$)

formed by Abstraction Sampling with positive granularities indicating Abstraction Sampling superiority over basic importance sampling. For example, for LARGE DBN, `ORAS` and `AOAS` with RelCB-0 resulted in average errors of -3.632 and -3.251, respectively, but `ORAS`, `pAOAS`, and `AOAS`, each with RandCB-16, produced average errors ranging from $-2.704\text{-}to 2.328$. At the top of each aggregation table in red we see the average error from `DIS`'s estimates. With the exception of the Grids benchmarks, the average error by the Abstraction Sampling schemes were far better than that of `DIS`. For for example, for DBN the average error by the Abstraction Sampling schemes is roughly between -5 and -2, whereas `DIS`'s average error was -39.463. Furthermore, in Broka et al. [2018] we showed that `ORAS` and `pAOAS` are highly competitive against Weighted Mini-Bucket Importance Sampling [Liu et al., 2015] and IJGP-SampleSearch [Gogate and Dechter, 2011]. By improving over the performance of `ORAS` and `pAOAS`, we also demonstrate competitiveness of `AOAS` to these other non-Abstraction Sampling state-of-the-art schemes.

We also performed a more detailed analysis with respect to `DIS`. Table 3.3 compares `AOAS` with RandCB-256 abstractions to `DIS`. For each benchmark partitioned into Exact and LARGE instances, we display the total number of instances (shown under *Total*), we count how often `AOAS`'s Z estimates: (i) fall within `DIS` 95% probabilistic bounds ($\in Bnds$), (ii) were comparable or better than DIS's estimates[†] ($AOAS \geq$), or (iii) were strictly better than DIS's estimates ($AOAS >$). Note that in order to compare Abstraction Sampling to DIS fairly, we selected only a single set of Abstraction Sampling parameters - namely AOAS RandCB-256 - that had been observed to perform generally well across all benchmarks.

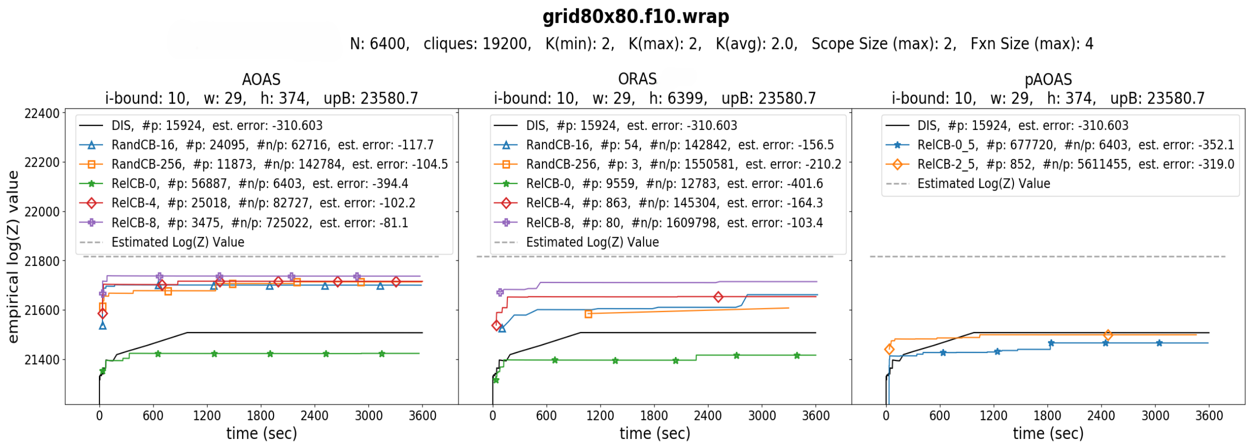
We see that in most cases AOAS estimates fall within the 95% confidence bounds produced by DIS, indicating a general quality of AOAS's estimates. For example for LARGE Promedas problems, we see AOAS estimates for 165 out of the 173 problems falling within DIS's 95% confidence bounds. AOAS estimates are also often comparable or better than that of DIS. This is particularly true of hard problems. For example, for the 173 LARGE Promedas problems, AOAS's estimates were comparable or better than that of DIS for 141 of the instances, and strictly better than DIS's estimate for 113 of the instances. We also note that unlike AS algorithms (AOAS in particular), DIS was unable to generate estimates for Linkage-Type4 problems. AOAS's performance notwithstanding, it is important to remember that DIS produces bounds while the current Abstraction Sampling schemes do not.

Differences Across Problem Types. A side observation made during our experimentation is that RelCB and RandCB seem to behave differently for different types of problems. We see that for DBN, Pedigree, and Promedas problems, RandCB abstractions generally perform better than RelCB with RandCB variants generally achieving smaller errors. In contrast, for Grids and Linkage-Type4 problems, RelCB abstractions perform better and especially for larger context sizes.

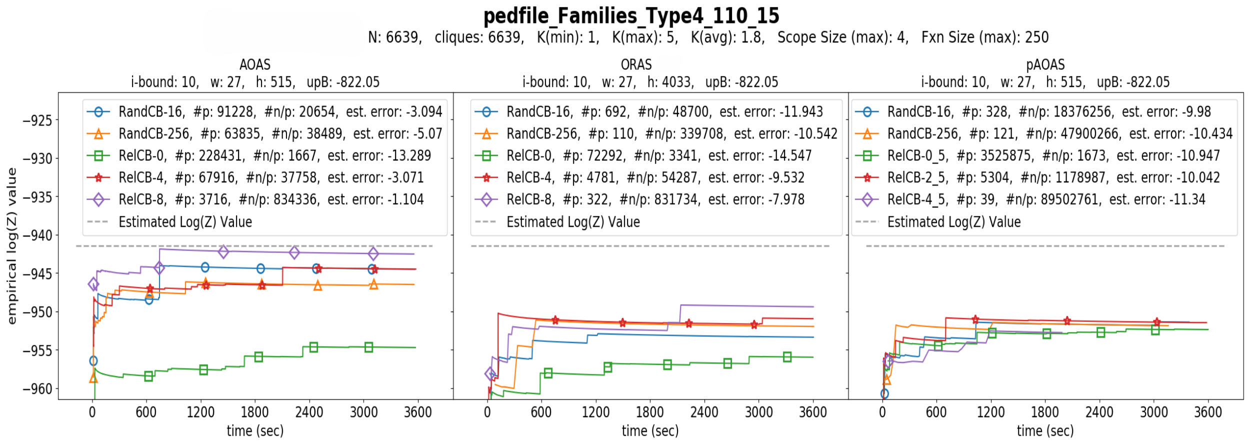
3.5.2.2 Representative Plots

Figure 3.12 shows plots of AOAS, ORAS, pAOAS, and DIS performance on a representative LARGE Grid and LARGE Linkage-Type4 problem, respectively. Under the problem name

[†]comparable means falling within ± 0.1 or ± 0.5 of the DIS $\log_{10} Z$ value, for Exact and LARGE problems respectively



(a) Plots for a representative large Grid problem.



(b) Plots for a representative Linkage-Type4 problem. Note that DIS is omitted as it was unable to produce estimates for these problems.

Figure 3.12: Plots of AOAS, ORAS, pAOAS, and DIS on representative LARGE benchmark problems. The dashed line marks the reference $\log_{10} Z$ value. The legend marks the abstraction function and granularity used, number of probes ($\#p$), average size of a probe ($\#n/p$), and $\log_{10} Z$ error (*est. error*).

we see the number of variables (N), the number of functions (cliques), the minimum domain size (K(min)), the maximum domain size (K(max)), the average domain size (K(avg)), the maximum function scope size (Scope Size (max)), and the maximum function table size (Fxn Size (max)). Underneath, each of the three subplots are labeled with the Abstraction Sampling algorithm used. Underneath each Abstraction Sampling scheme name we see the i-bound used, the maximum bucket width encountered by the heuristic (w), the AND/OR or OR search tree height (h), and the heuristic upper bound on $\log_{10} Z$ (upB). Each algorithm's progressive $\log_{10} Z$ estimates are shown on the (empirical log(Z) value) axis, with the (time (sec)) axis capturing the elapsed time. The dashed plot line marks the reference $\log_{10} Z^*$ value. The legend includes the abstraction function with the granularity used, number of probes (*#p*) generated, average size of a probe (*#n/p*), and final $\log_{10} Z$ error (*est. error*) for each algorithm. A `DIS` plot is also overlaid onto each subplot for the Grid problem for comparison. (A `DIS` overlay is omitted for Linkage-Type4 because `DIS` was unable to provide a non-zero estimate).

Although plots vary between benchmarks and problem instances, here we show plots that captures the overall trends of our data, especially highlighting that `AOAS` tends to converge towards the reference Z value faster, and has overall better performance than both `ORAS` and `pAOAS`. This was also true of `AOAS` compared to `DIS` in many cases, including for Grids as we wee in the plots with `AOAS`'s estimates growing closer to the reference Z value than `DIS`'s. We can also see the anytime nature of the algorithms in these plots as they continue to converge towards the reference Z value over time.

▩ 3.5.3 Abstraction Function Comparisons

In order to compare properties and performance of the various abstraction functions developed, a second array of experiments were run on three classes of abstraction functions: Context-Based abstractions (CTX), Value-Based Abstractions (VB), and a completely randomized abstraction scheme (RAND), each run with AOAS using a depth-first ordering of the same pseudo tree per problem to guide the construction of the search space.

Abstraction Functions. The context-based schemes (displayed as CTX) were tested with Relaxed Context-Based (rel) and Randomized Context-Based (rand) abstractions. The value-based schemes were tested heuristic-only-based abstractions (HB), heuristic-and-r-based abstractions (HRB), and with q-based abstractions (QB). Each function was tested with each of the seven partitioning schemes: simpleVB (simple), minVarVB (minVar), the four equalDistVB variants (equalDist[,2,3,4]), and randVB (rand). So, overall we experimented with twenty-one value-based abstractions. The variant of a purely randomized scheme described in Section 3.4.3 was also tested (RAND). With the exception of RelCB, each abstraction function uses a hyper parameter, $nAbs$, which bounds the number of abstract states at any level. RelCB instead uses an $nCtx$ parameter that limits the number of context variables used in assigning abstract states. To facilitate comparison, we report RelCB's $nCtx$ parameter instead as an equivalent $nAbs$ parameter assuming a domain size of 2. Ex. for $nCtx = 6$, we instead display $nAbs = 2^6$.

Additional Setup Details. When experimenting on Exact problems, algorithms use a small i-bound of 5 (weakening the heuristic estimates) and were given a limited time of 300sec to increase difficulty. For LARGE problems, an i-bound of 10 and time limit of 1200sec are used.

Primary Questions of Interest. Our empirical evaluations aim to address the following key questions:

1. *Empirical Comparison Between Context-based vs. Value-based vs. Purely Randomized Schemes*: How does `AOAS` performance using each of the three classes of abstractions compare with each other?

2. *Quality of Abstractions*: Ignoring the time it takes to process abstractions, which scheme generates the highest quality abstractions?

3. *Best Abstraction To Use in Practice*: What abstraction function and granularity is best to use overall?

iB-5, t-300sec, Exact		DBN			Grids			Pedigree			Promedas		
Class	Scheme	nAbs	Fail	Avg. Error	nAbs	Fail	Avg. Error	nAbs	Fail	Avg. Error	nAbs	Fail	Avg. Error
HB	simple	2048	0	0.440	1024	0	2.202	2048	0	0.150	1024	0	0.575
	minVar	1	0	1.361	16	0	3.251	64	0	0.422	16	2	2.509
	equalDist	1	0	1.365	2048	0	10.854	1024	0	0.303	1024	0	2.332
	equalDist2	1	0	1.570	512	0	8.050	1024	0	0.315	64	0	2.123
	equalDist3	1	0	1.489	2048	0	2.764	1024	0	0.279	256	0	2.196
	equalDist4	1024	0	2.819	64	0	6.029	512	0	0.214	2048	0	1.355
	rand	256	0	0.496	2048	0	2.248	2048	0	0.185	2048	0	0.752
HRB	simple	2048	0	0.491	4	0	9.667	256	0	0.225	2048	0	0.705
	minVar	1	0	1.500	64	0	2.319	256	0	0.309	16	1	2.801
	equalDist	1	0	1.305	256	0	10.635	1024	0	0.638	16	4	4.055
	equalDist2	1	0	1.549	2048	0	6.790	16	0	0.457	16	2	3.445
	equalDist3	1	0	1.405	1024	0	2.292	16	0	0.537	16	2	2.656
	equalDist4	1	0	1.511	512	0	1.829	64	0	0.483	2048	0	2.024
	rand	2048	0	0.451	4	0	6.122	64	0	0.666	1024	1	2.165
QB	simple	1	0	1.469	16	0	10.076	256	0	0.297	256	1	3.164
	minVar	2048	0	0.050	1024	0	1.566	64	0	0.210	64	1	1.062
	equalDist	4	0	1.174	2048	0	8.134	2048	0	0.144	2048	0	0.583
	equalDist2	2048	0	0.736	2048	0	4.405	1024	0	0.145	2048	0	0.539
	equalDist3	**2048**	**0**	**0.042**	**2048**	**0**	**1.771**	**512**	**0**	**0.148**	**2048**	**0**	**0.412**
	equalDist4	**2048**	**0**	**0.130**	**512**	**0**	**1.754**	**512**	**0**	**0.134**	**512**	**0**	**0.437**
	rand	1	0	1.295	256	0	6.048	16	0	0.740	16	2	5.988
CTX	rand	4	0	1.381	4	0	5.030	16	0	0.540	1024	1	2.442
	rel	1	0	1.472	64	0	4.021	64	0	0.424	64	6	4.349
RAND	**rand**	**2048**	**0**	**0.104**	**1024**	**0**	**1.501**	**1024**	**0**	**0.143**	**1024**	**0**	**0.513**

(a)

iB-10, t-1200sec, LARGE		DBN			Grids			Linkage-Type4			Promedas		
Class	Scheme	nAbs	Fail	Avg. Error	nAbs	Fail	Avg. Error	nAbs	Fail	Avg. Error	nAbs	Fail	Avg. Error
QB	simple	1	0	6.540	16	0	197.931	2048	13	48.681	4	34	11.919
	minVar	2048	0	1.837	1024	0	28.423	256	31	93.058	16	13	5.403
	equalDist	512	0	5.423	2048	0	118.547	2048	22	46.196	512	15	5.960
	equalDist2	2048	0	3.813	2048	0	91.994	1024	21	40.310	2048	12	4.982
	equalDist3	**2048**	**0**	**1.645**	**2048**	**0**	**19.277**	**1024**	**20**	**37.490**	**256**	**5**	**2.560**
	equalDist4	**2048**	**0**	**1.643**	**2048**	**0**	**18.866**	**2048**	**16**	**30.512**	**512**	**5**	**2.476**
	rand	4	0	6.292	16	0	163.973	256	17	156.992	4	28	11.532
CTX	rand	64	0	5.710	512	0	111.104	2048	53	194.741	256	0	3.222
	rel	1	0	6.267	1024	0	80.633	1024	37	129.189	16	34	11.247
RAND	**rand**	**2048**	**0**	**2.123**	**2048**	**0**	**19.053**	**1024**	**19**	**33.804**	**1024**	**10**	**3.936**

(b)

Figure 3.13: Summary Comparison. Each table shows the Abstraction Class (*Class*), Partitioning Scheme (*Scheme*), bound on the number of abstract states per level (*nAbs*), number of problems for which a positive solution could not be estimated (*Fail*), and average $\log_{10} Z$ error (*Avg. Error*) across Exact problems (subtable (a)) and LARGE problems (subtable (b)) in each benchmark. Color bars visualize error magnitudes. We hightliht the best performing algorithms: those for which: (1) difference in total average error (summed across the benchmarks) with respect to the best such total was less than 15% of the best, and (2) within each individual benchmark, the difference in average error with respect to the best average error was less than 35% of the best. (An exception to the latter criterion was granted to Exact DBN, on which the best average error from equalDistQB3 was unusually low).

3.5.3.1 Summary Comparison.

We tested each algorithm with a range of $nAbs \in \{1, 4, 16, 64, 256, 512, 1024, 2048\}$. For each $nAbs$ and benchmark, we calculated the average error across each benchmark and identified the *nAbs* that showed the lowest average error. Table 3.13a focuses on Exact problems and shows this lowest average error and corresponding *nAbs* for each algorithm. Table 3.13b shows the corresponding results for LARGE problems on the better performing QB and RAND classes, and the CTX class for comparison. Each table shows the Abstraction Class (*Class*), Partitioning Scheme (*Scheme*), bound on the number of abstract states per level (*nAbs*), number of problems for which a positive estimate could not be produced (*Fail*), and average $\log_{10} Z$ error (*Avg. Error*). If an algorithm was unable to produce a positive Monte Carlo Z estimate for a problem (i.e., it "Failed" on the problem), the wMBE heuristic bound was used as its Z estimate and error computed accordingly. Color bars help visualize error magnitudes and we highlight the best performing algorithms.

Comparison with Context-Based Schemes. We see that there is always a partitioning scheme for HB and HRB that can outperform the best CTX scheme on Exact problems. For example, the best performing CTX scheme for DBN was randCB (denoted CTX, *rand*) using $nAbs = 4$ resulting in an average error of 1.381, but HB and HRB using *simple* partitioning with $nAbs = 2048$ had average errors of only 0.440 and 0.491, respectively. QB with *minVar*, *equalDist3*, and *equalDist4* partitioning outperform the CTX schemes across all benchmarks always producing lower average errors. RAND also consistently outperforms the CTX schemes. Results on LARGE problems agree, with the exception of on Promedas where QB with *minVar* and RAND (having average errors of 5.403 and 3.936, respectively) have slightly greater error than randCB (with average error of 3.222) and never "Failing".

Comparison with Purely Randomized Abstractions. RAND is a particularly well performing scheme across all benchmarks, consistently having relatively low – and sometimes the lowest – average error among the schemes. (For example, its error for Exact Grids, 1.501, is the lowest of any of the other schemes). Nevertheless, the QB class using the *equalDist3* and *equalDist4* partitioning strategies is consistently comparable or better than the purely randomized scheme. No other scheme does as well.

Comparison with Non Abstraction Sampling Schemes. We saw earlier that Abstraction Sampling using CTX based abstractions was competitive against several powerful schemes such as Importance Sampling (IS), Weighted Mini-Bucket Importance Sampling (`wMBIS`) [Liu et al., 2015], IJGP-SampleSearch (`IJGP-ss`) [Gogate and Dechter, 2011], and Dynamic Importance Sampling [Lou et al., 2019]. Since the QB scheme with *equalDist3* and *equalDist4* partitioning strategies and the RAND scheme show superior performance in comparison to the CTX schemes, this also implicitly indicates competitiveness against the these other non-Abstraction Sampling methods.

3.5.3.2 Results from 100 Samples with nAbs = 256.

iB-5, m-100, Exact			DBN		Grids		Pedigree		Promedas	
Class	Scheme	nAbs	Fail	Avg. Error	Fail	Avg. Error	Fail	Avg. Error	Fail	Avg. Error
QB	simpleQB	256	0	5.350	0	17.406	0	1.059	14	9.659
	minVarQB	**256**	**0**	**0.111**	**0**	**1.911**	**0**	**0.223**	**1**	**1.634**
	equalDist	256	0	5.619	0	15.533	1	0.858	13	5.420
	equalDist2	256	0	2.319	0	11.220	0	0.563	6	3.479
	equalDist3	**256**	**0**	**0.173**	**0**	**3.615**	**0**	**0.206**	**1**	**1.473**
	equalDist4	**256**	**0**	**0.277**	**0**	**2.305**	**0**	**0.180**	**1**	**1.373**
	randQB	256	0	4.982	0	12.653	0	3.211	13	19.441
CTX	rand	256	0	3.587	0	9.568	2	4.695	3	14.386
	rel	256	0	5.265	0	8.013	0	1.097	36	10.845
RAND	**rand**	**256**	**0**	**0.288**	**0**	**2.464**	**0**	**0.325**	**3**	**2.570**

Figure 3.14: 100-Sample Comparison. For abstraction granularity of $nAbs = 256$, aggregated statistics (as described for Table 3.13) for Exact problems of each benchmark with each algorithm allotted 100 samples.

To assess the quality of abstraction functions in an implementation-agnostic manner and irrespective of resulting probe-sizes or speed of generating abstractions, we conducted experiments using a one-hundred sample limit (**m-100**) rather than using a time limit. Table 3.14 shows these results on Exact problems for the better performing QB and Rand classes using $nAbs = 256$. $nAbs = 256$ was chosen as (1) it is an intermediate granularity and (2) all schemes produced 100 samples in a reasonable time. We again highlight the best performing schemes.

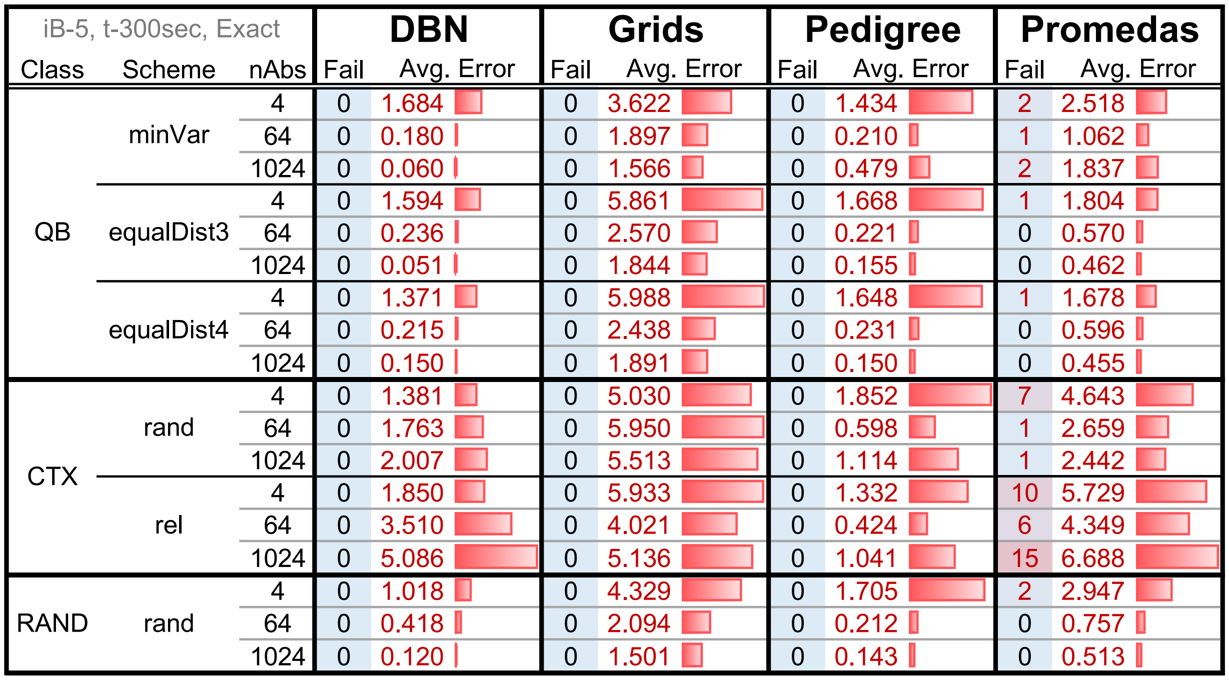
As with the previous results, we again see QB with *equalDist3* and *equalDist4*, and RAND, performing the best with few to no "Fails" and low average errors. A key difference is that, now, QB with *minVar*, which had showed slightly worse performance under a time limit, performs as well or even better. For example, for the Promedas problems QB with *minVar* only had one "Fail" where RAND had 3, and for Grids it has an average error of 1.911 where the best of the other schemes (in this case QB with *equalDist4*) had an error of 2.305. This indicates that minimizing the within variance of $q(n)$ values of nodes leads to favorable abstractions. This in part explains the success of QB *equalDist3* and *equalDist4*, which end up roughly emulating QB *minVar* while using faster greedy strategies.

minVar's worse performance under a time limit can be explained by the fact that the formation of its abstractions are computationally intensive (using Ward's Minimum Variance Hierarchical Clustering), which results in slow probe generation and thus fewer samples.

▢ 3.5.3.3 Choice of Abstraction Granularity

Table 3.15 shows average error for $nAbs \in \{4, 64, 1024\}$ on Exact problems of each benchmark. We focus on the better performing variants of QB: minVarQB, equalDistQB3, equalDistQB4; the purely randomized scheme RAND; and the context-based schemes (CTX) for comparison. In Figure 3.16 and Figure 3.17, we also show average error across a wider array of *nAbs* for minVarQB and equalDistQB4, respectively, the latter also acting as a representative for the profile of equalDistQB3 and RAND which share the same profile (omitted for brevity).

From Table 3.15 we see that for the well performing QB *equalDist3* and *equalDist4* schemes and for the RAND scheme there is a trend that greater *nAbs* improves performance. For example, for QB with *equalDist4* on Grids problems, we see that progressively increasing



iB-5, t-300sec, Exact			DBN		Grids		Pedigree		Promedas	
Class	Scheme	nAbs	Fail	Avg. Error	Fail	Avg. Error	Fail	Avg. Error	Fail	Avg. Error
QB	minVar	4	0	1.684	0	3.622	0	1.434	2	2.518
		64	0	0.180	0	1.897	0	0.210	1	1.062
		1024	0	0.060	0	1.566	0	0.479	2	1.837
	equalDist3	4	0	1.594	0	5.861	0	1.668	1	1.804
		64	0	0.236	0	2.570	0	0.221	0	0.570
		1024	0	0.051	0	1.844	0	0.155	0	0.462
	equalDist4	4	0	1.371	0	5.988	0	1.648	1	1.678
		64	0	0.215	0	2.438	0	0.231	0	0.596
		1024	0	0.150	0	1.891	0	0.150	0	0.455
CTX	rand	4	0	1.381	0	5.030	0	1.852	7	4.643
		64	0	1.763	0	5.950	0	0.598	1	2.659
		1024	0	2.007	0	5.513	0	1.114	1	2.442
	rel	4	0	1.850	0	5.933	0	1.332	10	5.729
		64	0	3.510	0	4.021	0	0.424	6	4.349
		1024	0	5.086	0	5.136	0	1.041	15	6.688
RAND	rand	4	0	1.018	0	4.329	0	1.705	2	2.947
		64	0	0.418	0	2.094	0	0.212	0	0.757
		1024	0	0.120	0	1.501	0	0.143	0	0.513

Figure 3.15: Varying nAbs. Average error when using $nAbs \in \{4, 64, 1024\}$ for minVarQB, equalDistQB3, equalDistQB4, the CTX based algorithms, and RAND, each with iB-5 and time limit of 300 sec.

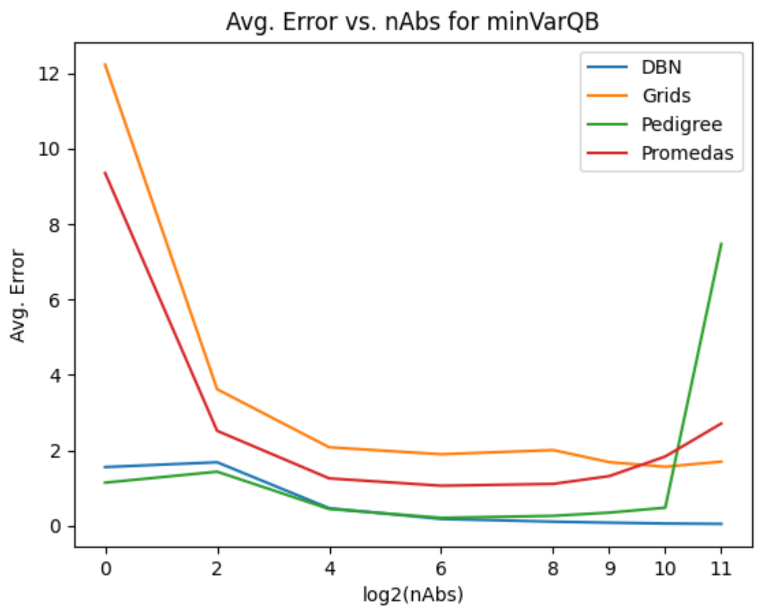


Figure 3.16: Varying *nAbs* for minVarQB. Average error on Exact problems using iB-5 and time limit 300 sec for each benchmark at various abstraction granularities (in $\log_2$).

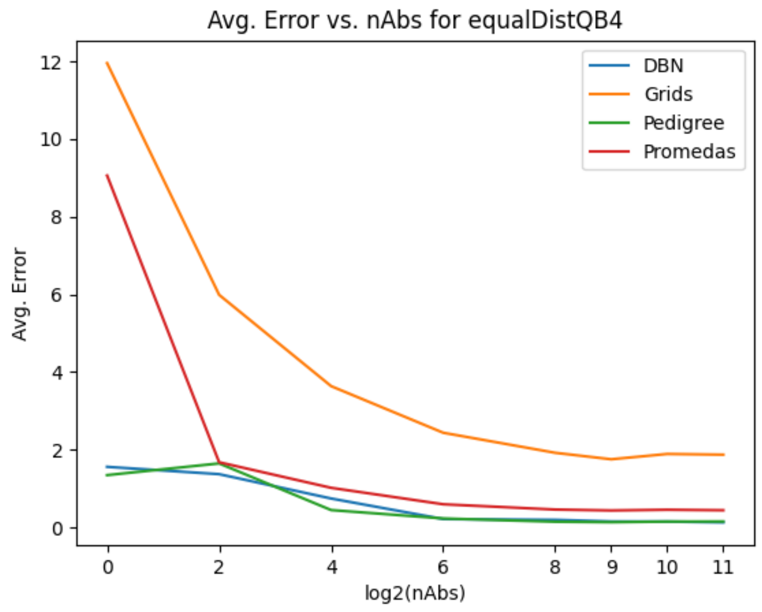


Figure 3.17: Varying *nAbs* for equalDistQB4. Average error on Exact problems using iB-5 and time limit 300 sec for each benchmark at various abstraction granularities (in $\log_2$).

nAbs from 4 to 64 to 1024 progressively reduces average error from 5.988 to 2.438 to 1.891. Figure 3.17 further supports this for QB with *equalDist4* with average error progressively decreasing with greater *nAbs* for all benchmarks. However in Figure 3.16 and Table 3.15 we see that for *minVar* error begins to increase when *nAbs* becomes too high. This can be explained by the higher computational cost of forming *minVar* abstractions (which is more time consuming), leaving less time for probe generation.

3.5.3.4 Summary of Results.

	HB	HRB	QB
simple	2.75	1.12	0.72
minVar	1.05	1.13	2.95
equalDist	0.75	0.59	1.16
equalDist2	0.84	0.75	1.82
equalDist3	1.20	1.01	4.05
equalDist4	0.87	1.14	3.90
rand	2.41	0.93	0.60

Figure 3.18: Performance Matrix. Relative average performance of value-based schemes vs. existing context-based abstractions. Values > 1.00 indicate superior performance.

Overall, our experiments show that the QB scheme with *equalDist3* or *equalDistQB4* and RAND perform the best among the newly proposed abstraction functions, and they significantly outperform the former state-of-the-art context-based schemes (Figure 3.18). These schemes tend to improve as the abstraction granularity $nAbs$ increases up to a point, past which we see little difference in performance. Thus, our study suggests that these three abstraction schemes should be the first choice when using AOAS, and be used with the largest $nAbs$ feasible.

3.6 Conclusion

We introduced Abstraction Sampling, a stratified-importance-sampling-like scheme that can be used for computing the partition function of discrete probabilistic graphical models. These schemes are based on dividing the nodes of the search tree into equivalence classes using an abstraction function, and randomly selecting one representative for each class by using probabilities derived from a heuristic evaluation function. The estimate is computed on the sampled tree generated by this procedure. Abstraction sampling is inspired by the early work of Knuth and Chen [Knuth, 1975, Chen, 1992], work that had been extended in the context of predicting the size of search trees in heuristic search [Lelis et al., 2013] and for search algorithms in graphical models [Lelis et al., 2014].

We introduced `ORAS`, an Abstraction Sampling algorithm for classic OR search spaces. We then presented Abstraction Sampling algorithm `AOAS` designed for working with compact AND/OR search spaces. In particular, we discuss that `AOAS`, unlike it's predecessor `pAOAS`,is free of handicapping restriction of *properness* and yielding a far more scalable algorithm. We illustrated that, like previous AND/OR AS schemes, `AOAS` maintains the ability to exploit the decomposition expressed in AND/OR search spaces, yet it also has far better control of probe-size thus making it uniquely scalable. This gives `AOAS` the power to more smoothly interpolate between (stochastic) sampling and (systematic) search [Broka et al., 2018]. We prove unbiasedness of the estimators and other properties of the algorithms. Our empirical evaluation shows that the abstraction function size, or refinement level, can impact the accuracy of the estimate significantly; and we saw that in most instances higher level abstractions outperforms 0-level abstraction, the latter which can be viewed as the baseline scheme equivalent to basic importance sampling.

Also presented is an exploration of an array of abstraction functions for use with AND/OR Abstraction Sampling. This exploration features comparison of abstractions based on a notion of variable context as well as those based on a value-based abstraction framework from three classes: heuristic-only based, hr-based, and and q-based, each defined by real-valued functions that aim to capture informative elements from search and sampling to guide abstractions and improve Abstraction Sampling performance. Each of the value-based classes were tested with each of seven node partitioning schemes to form twenty-one value-based abstraction functions. Additionally, a purely randomized abstraction scheme, RAND, was presented and compared.

Results from an extensive empirical evaluation on over 400 benchmark problems show that two of the QB based schemes (*equalDistQB3*, and *equalDistQB4*) and the RAND scheme having superior performance consistently and throughout all benchmarks when used with `AOAS`. In particular, performance was significantly improved relative to context-based abstractions, and thus also implicitly against Importance Sampling, Weighted Mini-Bucket Importance

Sampling, IJGP-SampleSearch, and Dynamic Importance Sampling to which the context-based abstraction functions with Abstraction Sampling were originally compared.

Based on this study, we believe that AOAS is one of the best schemes for estimating the partition function to date, in particular when used with the (*equalDistQB3*, *equalDistQB4*), or RAND abstraction schemes.

Chapter 4

UFO: Underflow-Threshold Optimization

4.1 Introduction

Search algorithms that can exploit determinism by pruning invalid partial configurations have the potential of greatly speeding up their search on problems with lots of determinism. A simple and common mechanism is to use *constraint propagation* (CP), which applies a bounded-complexity inference process to identify invalid patterns that arise from conjunctions of the problem's original constraints [Dechter, 2019, Mateescu and Dechter, 2008, Dechter, 2019]. Similar ideas have been explored in mixed integer programming [Danna et al., 2005]. In this chapter, we explore the idea of underflowing small function values – namely replacing them with 0.0 – turning them into hard constraints to empower constraint propagation in helping search algorithms to prune their search space. We present a general scheme called **underflow-threshold optimization**, or **UFO** for short, as a general method for choosing how to perform these underflows. We then provide several specific algorithms for employing UFO. Next we summarize the contributions in this chapter:

Contributions:

1. We propose a general framework called `UFO` for infusing artificial determinism into graphical models, in order to empower graphical model search algorithms to exploit

constraint processing for increased search efficiency. In particular, this facilitate early pruning of small cost branches during search.

2. We present several algorithms that employ the UFO and analyze their strengths and weaknesses.

3. We provide theoretical analysis of the scheme impact including boundedness and tractability. In particular we provide bounds for the marginal MAPp task.

4. We evaluate UFO performance with a vanilla AND/OR branch-and-bound algorithm empowered with constraint propagation on 100 problems used in the 2022 UAI Inference Competition, and compare performance against six competitive solvers used in the competition to highlight the potential of UFO.

▢ 4.2 The General UFO Scheme

We will start with a few key definitions and precisely define an **underflow-threshold**, which determine when to replace function values with 0.0. We will define what it means to apply underflow-thresholds to graphical models, and then we will present a general UFO methodology for choosing underflow-thresholds.

Definition 4.1 (τ-underflow of f, f_τ)
Let f be a non-negative function and $\tau \in \mathbb{R}^+$ be a user-selected threshold value. The τ-underflow of f is then,

$$f_\tau(x) = \begin{cases} f(x) & \textit{if } f(x) \geq \tau \\ 0 & \textit{otherwise.} \end{cases}$$

We call the action of performing this operation as applying (τ-underflow) thresholds, or more simply as thresholding.

Definition 4.2 (τ-underflow of $\mathcal{M}$, $\mathcal{M}_\tau$)

For a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$*, we define the* τ*-underflow of* $\mathcal{M}$ *to be* $\mathcal{M}_\tau = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}_\tau \rangle$*, where* $\mathbf{F}_\tau = \{f_\tau \mid f \in \boldsymbol{F}\}$*.* $\mathcal{M}_\tau$ *is referred to as a (*τ*-)underflowed or thresholded model.*

Definition 4.3 (Inconsistent Model)

A model is said to be inconsistent if $\forall \boldsymbol{x} \in D_{\boldsymbol{X}}$*,* $\prod_{f \in \boldsymbol{F}} f(x) = 0$*, i.e., there are no possible joint configurations* $\boldsymbol{x}$ *with positive value / probability.*

The idea of underflow thresholding is simple: larger thresholds τ induce more determinism into the thresholded model $\mathcal{M}_\tau$. This reduces the number of valid non-zero configurations and can constrain the portion of the search space that include valid configurations. These constraints can then be exploited by search, for example by applying constraint propagation before or during search, to make exploration of the (reduced, or "pruned") search space more efficient.

Choosing a larger underflow-threshold τ leads to more determinism, and facilitate more aggressive pruning. Clearly, if the threshold is too high, the resulting model may become too inaccurate, excluding configurations that are important to the original model's solution, and may even make the model inconsistent, pruning all its configurations. Therefore, a critical component of this scheme is to identify a threshold that provides the right balance between accuracy and efficiency. In particular, we can aim at a threshold that is as high as possible yet still results in a consistent model.

Note that inference tasks on an underflowed model can lead to can return bounds on the true solution. In particular:

Proposition 4.1 (Lower-bounds from τ-underflows)

$$\forall task \in \{Z, MAP, MMAP\}, \forall \mathcal{M}, task(\mathcal{M}_\tau) \leq task(\mathcal{M})$$

UFO algorithms. By UFO algorithms we refer to schemes that given a graphical model, $\mathcal{M}$ generate a threshold τ to be used in creating $\mathcal{M}_\tau$ for subsequent processing. Thus, our UFO (UnderFlow-threshold Optimization) scheme, provide a methodology for choosing an underflow-threshold. It typically employs a search procedure that seeks a largest threshold (or threshold$\boldsymbol{s}$) that are consistent and perhaps obey some other desirable requirements (e.g., ensuring the includion of specific configurations). Subsequently, the algorithm can decrease the threshold using a hyper-parameter δ to enable a wider array of solutions.

UFO algorithms operates under the assumption that satisfiability of a model can be determined quickly. Although this is not true in general, we find that the satisfiability sub-task underlying many optimization problems is often relatively easy. Otherwise, satisfiability can be approximated by constraint propagation schemes or other approximation methods [Dechter, 2003].

4.3 UFO Variants

In this section we will present several different algorithms that employ UFO in different ways for determining the underflow-threshold(s) to use when underflowing a graphical model.

4.3.1 UFO-GT: Global Threshold UFO

In many problem domains, function scale and semantics are uniform and thus there can exist an intuitive threshold that makes sense to underflow with across all cost functions. An example of this is problems where functions model molecular interaction energetics. In many problems of this form, energetics that correspond to very unfavorable conformations are so extreme that no amount of positive interaction can offset them. Thus, in effect, these bad conformations act as hard constraints and thus it makes sense to encode them as such by underflowing such values to zero.

Algorithm 10: `UFO-GT`

```
Input: Graphical model $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$; SAT solving algorithm, $SAT(.)$ ; time limit for binary
       search; a deflation factor $0 < \delta \leq 1$
Output: A proposed global underflow-threshold $\tau$ to use
1  begin
2      if $SAT(\mathcal{M}) = False$ then
3          return $FAILURE$
4      $\tau_{min} = 0$;   $\tau_{max} = \max_{\boldsymbol{F},\boldsymbol{X}} f(\boldsymbol{x})$
5      $\tau = \frac{\tau_{max} + \tau_{min}}{2}$
6      while time remains for $\tau$ binary search do
7          if $SAT(\mathcal{M}_\tau) = False$ then
8              $\tau_{max} = \tau$
9          else
10             $\tau_{min} = \tau$
11         $\tau = \frac{\tau_{max} + \tau_{min}}{2}$
12     end
13     $\tau = \tau_{min} \cdot \delta$
14     return $\tau$
15 end
```

Algorithm 10: `UFO-GT` (**U**nder**F**low-Threshold **O**ptimization with **G**lobal **T**hresholds) describes an iterative method for choosing a global underflow-threshold - a single threshold that is used to underflow all functions of the model. Given a model, $\mathcal{M}$, it employs binary search on numbers between 0 and the largest function value of the model to find the largest global underflow-threshold τ that still results in a satisfiable model (lines 6-12). During the binary search, in order to determine whether to raise or lower the value of the proposed threshold τ, a consistency check is done on $\mathcal{M}_\tau$ where a provided satisfiability solver $SAT(\cdot)$ checks to see if the underflowed model is consistent. If consistent, it raises the proposed threshold, if not, it lowers it. Once time runs out, the algorithm takes the largest threshold that still resulted in a satisfiable underflowed model and then decreases it using a hyper-parameter δ (line 13) to enable a wider array of solutions.

Intuition. In many problem domains, function scale and semantics are uniform and thus there can exist an intuitive threshold that makes sense to underflow with across all cost functions. An example of this is problems where functions model molecular interaction energetics. In many problems of this form, energetics that correspond to very unfavorable conformations are so extreme that no amount of positive interaction can offset them. Thus, in effect, these bad conformations act as hard constraints and thus it makes sense to encode them as such by underflowing such values to zero.

Algorithm `UFO-GT` is the simplest variant choosing only a single global threshold that is applied to all functions of a model. This makes it fast, however, when the functions of a model are on different scales and capture different semantics, applying a single threshold to all of them may not be desirable. We address this issue in our next variant which can use a different threshold for each function.

▨ 4.3.2 UFO-RT: Relative-Threshold UFO

Although in some problem domains functions share the same semantics and scale, this is not always the case. When not so, the effectiveness of `UFO-GT` can become greatly limited as it has no way of adjusting to each function's scale. `UFO-RT` mitigates this by applying thresholded-underflows relative to each function's values, thus taking into account varying scales across a problem's functions. The idea is very simple. We just want to determine a constant fraction θ which will be applied to the maximum value of each function to determine its threshold. Thus, the task now is to determine the global relative constant θ that will yield this varied function thresholds. Like earlier, we seek θ which is as high as possible to yield a consistent model, and then we iteratively reduce it in a binary search, in a manner similar to the earlier scheme.

Definition 4.4 (Relative Underflow-Threshold)
Let f_{max} be the maximum function value for a given function, and let θ be a constant between 0 and 1, Then we define the relative underflow threshold for the given function to be $\tau_f = \theta \cdot f_{max}$.

Algorithm `UFO-RT` uses binary search to find the largest θ that, when used to determine rel-

Algorithm 11: `UFO-RT`

```
Input: Graphical model $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$; SAT solving algorithm, $SAT(.)$ ; time limit for binary
       search; a deflation factor $0 < \delta \leq 1$
Output: A proposed function-relative thresholding-factor $\theta$ to use
1  begin
2      if $SAT(\mathcal{M}) = False$ then
3          return $FAILURE$
4      $\theta_{min} = 0.0$;   $\theta_{max} = 1.0$
5      $\theta = \frac{\theta_{max} + \theta_{min}}{2}$
6      $\vec{\boldsymbol{\tau}} = [\theta \cdot \max(f) \mid f \in \boldsymbol{F}]$        // a vector of underflow-thresholds for each function
7      while time remains for $\tau$ binary search do
8          if $SAT(\mathcal{M}_\tau) = False$ then
9              $\theta_{max} = \theta$
10         else
11             $\theta_{min} = \theta$
12         $\theta = \frac{\tau_{max} + \tau_{min}}{2}$
13         $\vec{\boldsymbol{\tau}} = [\theta \cdot \max(f) \mid f \in \boldsymbol{F}]$
14     end
15     $\theta = \theta_{min} \cdot \delta$
16     return $\theta$
17 end
```

ative underflow thresholds for each function (lines 6,13), still results in the resulting underflowed model being consistent satisfiable (lines 7-14). Then `UFO-RT` decreases the threshold using a hyper-parameter δ (line 15) to enable a wider array of solutions.

▨ 4.3.3 UFO-Sol: Solutions-based UFO

Algorithm 12: `UFO-Sol`

```
Input: Graphical model $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$; SAT solving algorithm, $SAT(.)$ ; time limit for binary
       search; a deflation factor $0 < \delta \leq 1$; a set of $m$ full configurations $\boldsymbol{\zeta} = \boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_m$
Output: A vector $\vec{\boldsymbol{\tau}}$ of length $|\boldsymbol{F}|$ consisting of underflow-thresholds for each of the model's
        functions
1  begin
2      $\vec{\boldsymbol{\tau}} = [\ \min_{\boldsymbol{x} \in \boldsymbol{\zeta}} f(\boldsymbol{x}) \ \mid \ f \in \boldsymbol{F}\ ]$      // a vector of underflow-thresholds for each function
3      $\vec{\boldsymbol{\tau}} = \vec{\boldsymbol{\tau}} \cdot \delta$
4      return $\vec{\boldsymbol{\tau}}$
5  end
```

In many scenarios, a set of good solutions may be known or found quickly, and the task thus becomes to find better solutions. Thus, it can be beneficial to assign thresholds such that the known good solutions remain valid, using them to ensure a minimum quality of valid solutions that are present in the underflowed model. This is what `UFO-Sol` aims to achieve.

Therefore, `UFO-Sol` Algorithm (**U**nder**F**low-threshold **O**ptimization using **Sol**utions, Algorithm 12) takes the following approach. It takes known good solutions as input (provided as a set $\boldsymbol{\zeta}$ of full configurations of the model) and use it to guide the underflow-thresholds. It does this by looking at each of the model's functions f one-by-one and, for each, chooses an underflow-threshold that satisfies $\tau = \min_{\boldsymbol{x} \in \boldsymbol{\zeta}} f(\boldsymbol{x})$ (lines 2). This ensures satisfiability of each $\boldsymbol{x} \in \boldsymbol{\zeta}$. As before, the thresholds are then decreased according to a hyper-parameter δ (line 3) to enable a wider array of solutions.

Unlinke the previous algorithms, `UFO-Sol` is non iterative and does not rely on satisfiability checks. However, it depends on having a known set of solutions. The resulting thresholds depends on the specific solutions used.

▢ 4.3.4 UFO as an Anytime Scheme

Up to now we described the UFO schemes as stand alone algorithms that determine a threshold, with the understanding that once a threshold is generated an inference scheme will take over to find an answer over the $\mathcal{M}_{\tau}$. model. In the anytime version we interleave the UFO scheme with the solving scheme in order to generate a solution that can be improved in anytime fashion.

In Algorithm 13: `UFO-GT+AI` (`UFO-GT` + **A**nytime **I**nference), a modified version of `UFO-GT` is used to determine an initial underflow-threshold during the first phase of the algorithm (called its *UFO Phase*), and then in the second phase (called the *Evaluation Phase*) an inference algorithm iteratively solves the respective underflowed problem and then decreases the threshold before iterating, iterations keeping track of the best solution found.

Specifically, after an initialization (lines 1-8), the UFO Phase begins where `UFO-GT+AI` em-

Algorithm 13: `UFO-GT+AI`

Input: Graphical model $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$; algorithm for solving the problem represented by $\mathcal{M}$, $Solve(.)$. It may use a lower bound solution value lb to enhance performance; approximate algorithm for solving the problem represented by $\mathcal{M}$, $Approx(.)$. It may use a lower bound solution value lb to enhance performance; SAT solving algorithm, $SAT(.)$; time limit for binary search; flag, $prelimSolutions$, for whether to look for preliminary solutions during τ binary search; policy to reduce τ during evaluation phase, $decreaseThreshold(.)$; time limit for evaluation phase

Output: lower bound on $Solve(\mathcal{M})$ and the corresponding variable assignments

```
1  begin
       // Check that original problem is satisfiable
2      if $SAT(\mathcal{M}) = False$ then
3          return $FAILURE$
       // initialization
4      $\tau_{max} = f_{\max}$
5      $\tau_{min} = 0$
6      $\tau = f_{\max}$ // initial underflow-threshold will underflow all function values to zero
7      $lb = -inf$
8      $bestAssignment = None$
       // UFO Phase - binary search to find a largest underflow threshold that still
          results in a satisfiable problem
9      while time remains for $\tau$ binary search do
10         if $SAT(\mathcal{M}_\tau) = False$ then
               // $\tau$ results in an unsatisfiable problem - reduce $\tau$ and continue
11             $\tau_{max} = \tau$
12         else
               // problem satisfiable given $\tau$ - solve and record solution (if asked to do
                  so), increase $\tau$, and continue
13             if $prelimSolutions = True$ then
14                 $assignment = Approx(\mathcal{M}_\tau, lb)$
15                 if $value(assignment) > lb$ then
16                     $lb = value(assignment)$
17                     $bestAssignment = assignment$
18             $\tau_{min} = \tau$
19         $\tau = \frac{\tau_{max} + \tau_{min}}{2}$
20     end
       // Evaluation phase - repeatedly perform $Solve(\mathcal{M}_\tau)$, decreasing $\tau$ after each
          iteration
21     $\tau = \tau_{min}$
22     while time remains for the evaluation phase do
23         $assignment = Solve(\mathcal{M}_\tau, lb)$
24         $lb = value(assignment)$
25         $bestAssignment = assignment$
26         $\tau = decreaseThreshold(\tau)$
27     end
28     return $lb$, $bestAssignment$
29 end
```

ploys UFO-GT with the modification that, whenever a new satisfiable underflow-threshold is found, the algorithm can be asked to use a provided approximate inference algorithm to quickly extract a sub-optimal solution from the respective underflowed model. These sub-optimal solutions can be leveraged by some inference schemes (such as branch-and-bound search algorithms) to enhance their performance. As the modified UFO-GT binary search continues, the best sub-optimal solution and its corresponding solution value are kept as *bestAssignment* and *lb*, respectively (lines 13-17). (We assume an optimization task so solutions having larger values are better, and we use *lb* (lower-bound) to store sub-optimal solution values). When the allotted time for the UFO Phase runs out, the final threshold τ found is then used in the Evaluation Phase.

In the Evaluation Phase, the algorithm uses a provided inference algorithm to solve the τ-underflowed problem, possibly using the current best-found lower bound to empower it (line 23). Lower bounds and corresponding best assignments to the variables are updated and the threshold decreased, before repeating (lines 24-26). When time runs out, the algorithm returns the best-found solution and corresponding assignment.

4.4 Empirical Evaluation of AOBB-UFO

In order to test the potential of UFO we selected a basic AND/OR branch-and-bound scheme for solving marginal MAP [Marinescu et al., 2018b] with constraint processing (via MiniSAT Eén and Sörensson [2004] and UFO-GT. We then tested this combined algorithm, referred to as AOBB-UFO, on the 2022 UAI Inference Competition final problem set, comparing results with state-of-the-art solvers that run as part of that competition.

▢ 4.4.1 Setup

Benchmarks. We chose one hundred problems that were used for the final UAI 2022 Inference Competition, which were selected from twelve different domains including computational biology, decision making, medical diagnosis, and image processing, among others.

Heuristic. To guide branch-and-bound search, we use Weighted Mini-Bucket Elimination (`wMBE`) since it is known to work well with both OR and AND/OR search. The i-bound, which controls the strength of the `wMBE` heuristic, was picked automatically according to an estimate of the largest i-bound possible while still ensuring wMBE would not exceed a given memory limits.

Variable Order. A variable ordering was automatically determined for each problem using the greedy min-fill heuristic [Dechter, 2019] which tries to minimize the resulting induced width of the graph.

UFO. UFO binary search was performed over log space for two seconds. The resulting threshold was used without being decreased (i.e., we used $\delta = 1.0$).

Competing Schemes.

BRAOBB: an anytime depth-first AND/OR search scheme that rotates through different subproblems in a round-robin manner improving anytime performance of AOBB [Marinescu et al., 2018b, Otten and Dechter, 2011].

DAOOPT-lh: depth-first AND/OR search scheme for optimization that performs look-ahead selectively intensifying search where the heuristic error is high Lam et al. [2017], Otten and Dechter [2011].

DAOOPT: depth-first AND/OR search scheme for combinatorial optimization Otten and Dechter [2011].

lbp: a residual-scheduled loopy belief propagation on the factor graph representation of the model.

toulbar2-vns: a metaheuristic called variable neighborhood search that uses (partial) tree search inside its local neighborhood exploration [Ouali et al., 2020, Givry, 2023].

toulbar2-vacint: a hybrid best-first search method utilizing an initial upper bound by the INCOP solver [Neveu and Trombettoni, 2003] and using virtual arc consistency [Givry, 2023].

toulbar2-ipr: an iterative algorithm based on toulbar2-vns with an increasing precision followed by toulbar2-vacint approach with full precision [Givry, 2023].

uai14: an amalgam belief propagation based solver created for the UAI-2014 competition by Alex Ihler including a combination loopy belief propagation and cutset conditioning schemes.

Implementation and Compute. `AOBB-UFO` was implemented in C++. All experiments were run on a 2.66 GHz processor for 1 hr using an 8 GB memory limit. All competing solvers were run via Docker'ized implementations [Merkel, 2014] on the same system.

Performance Measure. The performance measure used is $log_{10}Z(\mathcal{M}|\boldsymbol{X_{MAP}} = \boldsymbol{x_{MAP}})$ where $\boldsymbol{x_{MAP}}$ is the algorithm's best-found assignment to the Marginal MAP query variables $\boldsymbol{X_{MAP}}$. Better solutions result in greater $log_{10}Z(\mathcal{M}|\boldsymbol{X_{MAP}} = \boldsymbol{x_{MAP}})$. For `AOBB-UFO`, M_{τ} is used instead of $\mathcal{M}$, and so its solutions represent a lower-bound on the true value of its returned best-found assignment $\boldsymbol{X_{MAP}} = \boldsymbol{x_{MAP}}$

Primary Questions of Interest.

1. *Speed*: How fast does `AOBB-UFO` find solutions and terminate?
2. *Quality*: How good are the solutions returned by `AOBB-UFO`?

▢ 4.4.2 Results

Table 4.1-4.2 presents results of `AOBB-UFO` and compare against competing solvers over the UAI 2022 Inference Challenge final problem set for the Marginal MAP task.

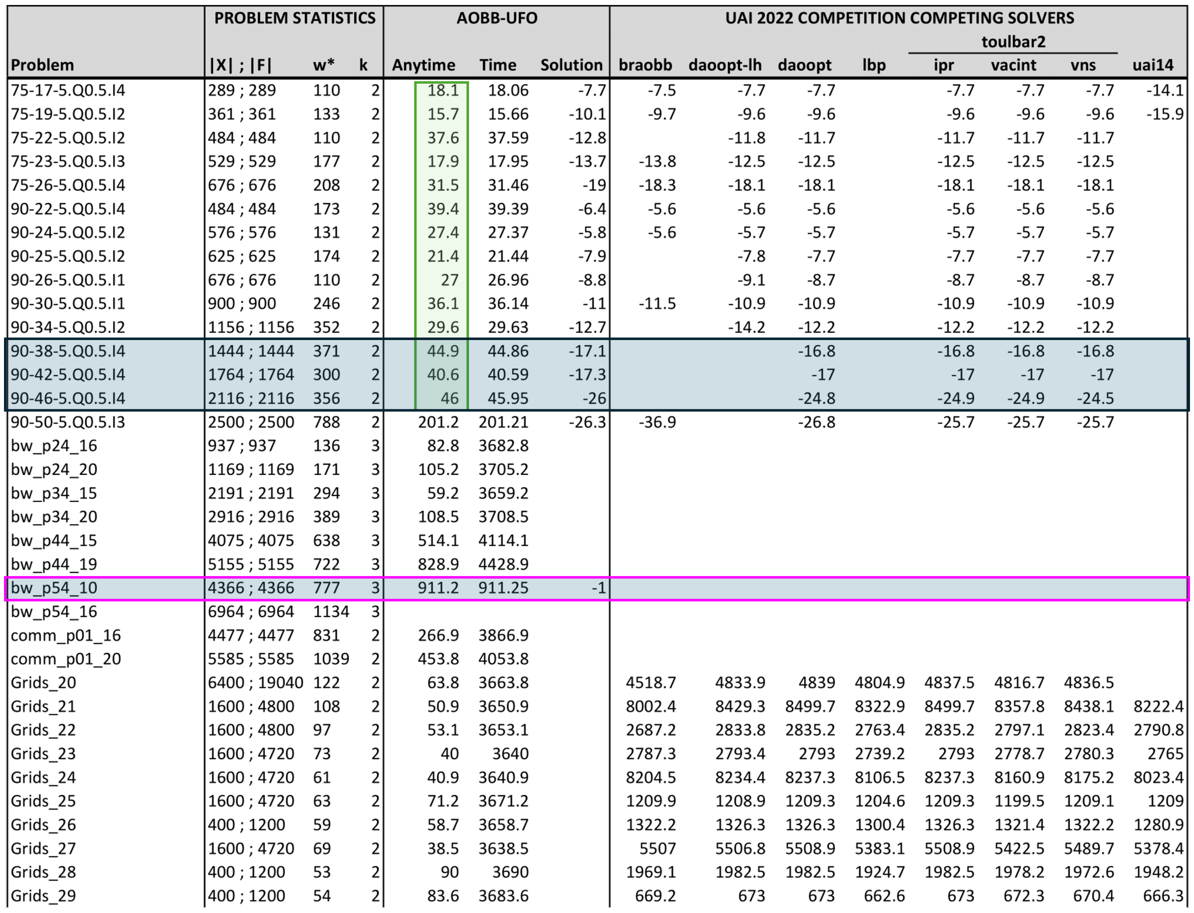
Table Keys. $|\boldsymbol{X}|$ and $|\boldsymbol{F}|$ are the number of variables and functions in the model, respectively. $\boldsymbol{w}^*$ is the induced width of the model according to the variable ordering used by `AOBB-UFO`. $\boldsymbol{k}$ is the maximum domain size of the model's variables. ***Anytime*** is the time it took `AOBB-UFO` to find its final solution. ***Time*** is the time it took `AOBB-UFO` to terminate or the actual time the algorithm was cut off. ***Solution*** is The $log_{10}Z(\mathcal{M}|\boldsymbol{X_{MAP}} = \boldsymbol{x_{MAP}})$ of `AOBB-UFO`'s best found solution $\boldsymbol{X_{MAP}} = \boldsymbol{x_{MAP}}$. The $log_{10}Z(\mathcal{M}|\boldsymbol{X_{MAP}} = \boldsymbol{x_{MAP}})$ of the best found solution for each of the competing solvers is also shown in their own columns.

The tables include the following three highlightings:

1. In blue we see problems where `AOBB-UFO`'s solution was as good or better than at least four competing solvers.
2. Bordered in magenta is when `AOBB-UFO` produced a solution ***strictly better*** than ***ALL*** other solvers.
3. In green we see when `AOBB-UFO` found its returned non-zero solution in less than 120 seconds.

While not claiming the `AOBB-UFO` algorithm is a superior all-around algorithm, we aim show that augmentation of relatively simple schemes with UFO can occasionally produce better results than top-performing solvers, and that it can be fast, making it suitable for quick approximations. Although simple UFO variants may not be reliable as a primary method, they may be valuable for initial searches to potentially uncover solutions that more complex algorithms might miss.

Specifically for `AOBB-UFO` on the 100 problems experimented on, `AOBB-UFO` produced a solution value that was equal to or better than three-to-four of the competing solvers on average. This includes instances where `AOBB-UFO` was able to find a non-zero solution and other algorithms were not. In particular, for the first fifteen problems we see `AOBB-UFO` able



	PROBLEM STATISTICS			AOBB-UFO			UAI 2022 COMPETITION COMPETING SOLVERS							
											toulbar2			
Problem	|X| ; |F|	w*	k	Anytime	Time	Solution	braobb	daoopt-lh	daoopt	lbp	ipr	vacint	vns	uai14
75-17-5.Q0.5.I4	289 ; 289	110	2	18.1	18.06	-7.7	-7.5	-7.7	-7.7		-7.7	-7.7	-7.7	-14.1
75-19-5.Q0.5.I2	361 ; 361	133	2	15.7	15.66	-10.1	-9.7	-9.6	-9.6		-9.6	-9.6	-9.6	-15.9
75-22-5.Q0.5.I2	484 ; 484	110	2	37.6	37.59	-12.8		-11.8	-11.7		-11.7	-11.7	-11.7	
75-23-5.Q0.5.I3	529 ; 529	177	2	17.9	17.95	-13.7	-13.8	-12.5	-12.5		-12.5	-12.5	-12.5	
75-26-5.Q0.5.I4	676 ; 676	208	2	31.5	31.46	-19	-18.3	-18.1	-18.1		-18.1	-18.1	-18.1	
90-22-5.Q0.5.I4	484 ; 484	173	2	39.4	39.39	-6.4	-5.6	-5.6	-5.6		-5.6	-5.6	-5.6	
90-24-5.Q0.5.I2	576 ; 576	131	2	27.4	27.37	-5.8	-5.6	-5.7	-5.7		-5.7	-5.7	-5.7	
90-25-5.Q0.5.I2	625 ; 625	174	2	21.4	21.44	-7.9		-7.8	-7.7		-7.7	-7.7	-7.7	
90-26-5.Q0.5.I1	676 ; 676	110	2	27	26.96	-8.8		-9.1	-8.7		-8.7	-8.7	-8.7	
90-30-5.Q0.5.I1	900 ; 900	246	2	36.1	36.14	-11	-11.5	-10.9	-10.9		-10.9	-10.9	-10.9	
90-34-5.Q0.5.I2	1156 ; 1156	352	2	29.6	29.63	-12.7		-14.2	-12.2		-12.2	-12.2	-12.2	
90-38-5.Q0.5.I4	1444 ; 1444	371	2	44.9	44.86	-17.1			-16.8		-16.8	-16.8	-16.8	
90-42-5.Q0.5.I4	1764 ; 1764	300	2	40.6	40.59	-17.3			-17		-17	-17	-17	
90-46-5.Q0.5.I4	2116 ; 2116	356	2	46	45.95	-26			-24.8		-24.9	-24.9	-24.5	
90-50-5.Q0.5.I3	2500 ; 2500	788	2	201.2	201.21	-26.3	-36.9		-26.8		-25.7	-25.7	-25.7	
bw_p24_16	937 ; 937	136	3	82.8	3682.8									
bw_p24_20	1169 ; 1169	171	3	105.2	3705.2									
bw_p34_15	2191 ; 2191	294	3	59.2	3659.2									
bw_p34_20	2916 ; 2916	389	3	108.5	3708.5									
bw_p44_15	4075 ; 4075	638	3	514.1	4114.1									
bw_p44_19	5155 ; 5155	722	3	828.9	4428.9									
bw_p54_10	4366 ; 4366	777	3	911.2	911.25	-1								
bw_p54_16	6964 ; 6964	1134	3											
comm_p01_16	4477 ; 4477	831	2	266.9	3866.9									
comm_p01_20	5585 ; 5585	1039	2	453.8	4053.8									
Grids_20	6400 ; 19040	122	2	63.8	3663.8		4518.7	4833.9	4839	4804.9	4837.5	4816.7	4836.5	
Grids_21	1600 ; 4800	108	2	50.9	3650.9		8002.4	8429.3	8499.7	8322.9	8499.7	8357.8	8438.1	8222.4
Grids_22	1600 ; 4800	97	2	53.1	3653.1		2687.2	2833.8	2835.2	2763.4	2835.2	2797.1	2823.4	2790.8
Grids_23	1600 ; 4720	73	2	40	3640		2787.3	2793.4	2793	2739.2	2793	2778.7	2780.3	2765
Grids_24	1600 ; 4720	61	2	40.9	3640.9		8204.5	8234.4	8237.3	8106.5	8237.3	8160.9	8175.2	8023.4
Grids_25	1600 ; 4720	63	2	71.2	3671.2		1209.9	1208.9	1209.3	1204.6	1209.3	1199.5	1209.1	1209
Grids_26	400 ; 1200	59	2	58.7	3658.7		1322.2	1326.3	1326.3	1300.4	1326.3	1321.4	1322.2	1280.9
Grids_27	1600 ; 4720	69	2	38.5	3638.5		5507	5506.8	5508.9	5383.1	5508.9	5422.5	5489.7	5378.4
Grids_28	400 ; 1200	53	2	90	3690		1969.1	1982.5	1982.5	1924.7	1982.5	1978.2	1972.6	1948.2
Grids_29	400 ; 1200	54	2	83.6	3683.6		669.2	673	673	662.6	673	672.3	670.4	666.3

Figure 4.1: AOBB-UFO on UAI 2022 Competition Final Problems (3600s)

	PROBLEM STATISTICS			AOBB-UFO			UAI 2022 COMPETITION COMPETING SOLVERS							
											toulbar2			
Problem	|X| ; |F|	w*	k	Anytime	Time	Solution	braobb	daoopt-lh	daoopt	lbp	ipr	vacint	vns	uai14
ImageAlignment_11	350 ; 3563	33	77	25.5	108.67	-1005.8	-824.2	-824.2	-824.2	-824.2	-824.2	-824.2	-824.2	-824.2
ImageAlignment_12	30 ; 465	29	58	5.3	5.28	-436.7	-436.7	-436.7	-436.7	-436.7	-436.7	-436.7	-436.7	-436.7
ImageAlignment_13	400 ; 3334	21	83	78	128.24	-3081.4	-2998.9	-2999.8	-2999.8	-2998.9	-2999.8	-2999.8	-2999.8	-2998.9
ImageAlignment_14	200 ; 2128	23	69	9.1	12.89	-1632	-1557.5	-1557.5	-1557.5	-1557.5	-1557.5	-1557.5	-1557.5	-1557.5
ImageAlignment_15	300 ; 2732	23	68	59.7	415.05	-1339.5	-1177.5	-1177.5	-1177.5	-1177.5	-1177.5	-1177.5	-1177.5	
ObjectDetection_13	60 ; 1830	59	21	6.2	3606.1	-528.6	6684.3	9967.7	9854.2	8826.5	9970.6	9970.6	9970.6	8883.1
ObjectDetection_14	60 ; 1830	59	11	31.6	3631.1	236.6	6712.5	9020.8	9020.8	8341.9	9093.7	9093.7	9093.7	8562.9
ObjectDetection_15	60 ; 1830	59	16	2155.8	3637	-404	10628	11703.8	12479	11544	12633	12633	12633	11976
ObjectDetection_16	60 ; 1830	59	21	1753.5	3606.2	-471	12023	14154.2	13536	13548	14347	14347	14347	14022
ObjectDetection_17	60 ; 1830	59	11	30.6	3630.5	-335.9	2454.9	4716.4	4816	3794.9	4887.4	4887.4	4887.4	4518.2
or_chain_11.fg.Q0.5.I3	900 ; 915	191	2	30.8	30.85	-22.9								
or_chain_16.fg.Q0.5.I3	1675 ; 1700	318	2	98.5	98.52	-38.1	-53.8	-25.2	-23.4		-23.4	-23.4	-23.4	
or_chain_22.fg.Q0.5.I3	1044 ; 1054	196	2	28.9	28.92	-15.3								
or_chain_24.fg.Q0.5.I3	1155 ; 1171	247	2	32.3	32.31	-24.4								
or_chain_25.fg.Q0.5.I3	1075 ; 1086	88	2	30.4	30.37	-16.8								
or_chain_32.fg.Q0.5.I3	1466 ; 1478	108	2	32	3632									
or_chain_36.fg.Q0.5.I3	933 ; 943	91	2	30.5	30.46	-15.3								
or_chain_39.fg.Q0.5.I3	1751 ; 1766	430	2	96.3	96.32	-22.9								
or_chain_40.fg.Q0.5.I3	988 ; 998	96	2	16.3	16.26	-15.1								
or_chain_41.fg.Q0.5.I3	1847 ; 1863	203	2	118.5	3645	-27								
or_chain_43.fg.Q0.5.I3	1692 ; 1712	216	2	44.5	44.5	-30.5								
or_chain_6.fg.Q0.5.I3	1849 ; 1876	386	2	2351.2	3710.9	-41.2								
or_chain_60.fg.Q0.5.I3	1997 ; 2023	552	2	3691.2	3736.6	-42.8								
or_chain_63.fg.Q0.5.I3	731 ; 744	97	2	26.3	26.35	-10.8								
or_chain_8.fg.Q0.5.I3	1195 ; 1203	63	2	23.6	23.58	-12.2								
pedigree1.Q0.5.I3	298 ; 334	105	4	3319.4	3641.4	-35.2		-35.9	-37.3		-35.7	-35.7	-36.1	-37.8
pedigree13.Q0.5.I1	888 ; 1077	138	3	2941.6	3623.5	-63.1		-62.8	-62.4		-62.8	-62.7	-62.5	
pedigree18.Q0.5.I1	931 ; 1184	181	5	38.8	3638.8			-112.5	-111.9		-112.5	-112.9	-113.5	-116.4
pedigree19.Q0.5.I4	693 ; 793	173	5	3280	3625.2	-97.5	-95.3	-89.5	-87.5		-87.1	-89.5	-89.6	
pedigree20.Q0.5.I2	387 ; 437	76	5	2455.1	3614.7			-46.8	-46.7		-47.9	-46.1	-46.1	-72.8
pedigree25.Q0.5.I2	993 ; 1289	178	5	3236.3	3702.4	-148.7	-154.6	-148.5	-147.8		-147.6	-147.3	-147.8	-170
pedigree30.Q0.5.I2	1015 ; 1289	109	5	30.9	3630.9			-123.6	-124.7		-123.6	-123.1	-124.3	-140.8
pedigree31.Q0.5.I2	1006 ; 1183	153	5	16	3616		-118.4	-116.2	-116		-117	-117.6	-115.6	-161.1
pedigree33.Q0.5.I2	581 ; 798	118	4	28.7	3628.7		-69.4	-70.3	-70.9		-71	-70.4	-71.1	
pedigree38.Q0.5.I2	581 ; 724	164	5	115.5	3621.9	-94.2	-91.4	-78.2	-77.6		-77.5	-77.6	-77.8	-90.5
pedigree41.Q0.5.I2	885 ; 1062	205	5	27.5	3620.7	-117		-105.6	-107.3		-106.8	-107.7	-108.2	-160.7
pedigree44.Q0.5.I4	644 ; 811	186	4	3177.3	3628.3	-89.3	-98.2	-89.1	-88.5		-89.1	-89.4	-89.7	
pedigree50.Q0.5.I1	478 ; 514	124	6	214.5	3677.4	-51.5	-55.7	-52.4	-52.8		-53.1	-53.1	-52	-69.6
pedigree7.Q0.5.I2	867 ; 1068	89	4	3579.1	3628.3	-103.8		-97.5	-97.6		-98.4	-98	-98.3	-136.4
pedigree9.Q0.5.I3	935 ; 1118	235	7	913.3	3626.6	-123.9		-113.8	-113.7		-113.4	-112.7	-114	-159
pomdp10-12_7_3_8_4	2673 ; 2701	2599	32				1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
pomdp6-12_6_2_6_3	250 ; 265	198	18	63.9	3663.9	0.9	1	1	1	1	1	1	1	1
pomdp7-20_10_2_10_3	3166 ; 3193													
pomdp8-14_9_3_12_4	2145 ; 2189	2057	48				1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4
pomdp9-14_8_3_10_4	5277 ; 5313													
ProteinFolding_11	400 ; 1160	28	2	22.4	22.44	1926.6	1962.3	1962.3	1962.3	1839.4	1962.3	1959.7	1944.8	1528.7
ProteinFolding_12	250 ; 2098	20	60	10	13.34	-1586.4	-1547	-1547	-1547	-1548	-1547	-1547	-1547	
ProteinFolding_13	100 ; 1055	21	92	24.5	24.56	-143.3	-143.3	-143.3	-143.3	-143.3	-143.3	-143.3	-143.3	-143.3
ProteinFolding_14	80 ; 847	20	49	12.6	12.61	-331.7	-331.7	-331.7	-331.7	-331.7	-331.7	-331.7	-331.7	-331.7
ProteinFolding_15	50 ; 536	22	47	10	9.97	-51.6	-51.6	-51.6	-51.6	-51.6	-51.6	-51.6	-51.6	-51.6
Segmentation_11	228 ; 852	19	21	6.4	135.02	-152.1	-132.1	-132.3	-132.3	-133.3	-132.3	-132.3	-132.3	-133.4
Segmentation_12	231 ; 856	20	2	2.9	2.88	-36.7	-21.9	-21.9	-21.9	-30.1	-21.9	-21.9	-21.9	-21.9
Segmentation_13	225 ; 832	18	2	3.8	3.76	-28.9	-21.4	-21.4	-21.4	-21.4	-21.4	-21.4	-21.4	-21.4
Segmentation_14	231 ; 863	18	2	3	2.96	-40.2	-39.3	-39.3	-39.3	-40.2	-39.3	-39.3	-39.3	-39.8
Segmentation_15	229 ; 851	18	21	6.9	336.2	-182.9	-169.6	-163.8	-163.8	-168.5	-163.8	-163.8	-163.8	-168.8
Segmentation_16	228 ; 838	18	2	2.9	2.94	-42.5	-40.4	-40.4	-40.4	-40.4	-40.4	-40.4	-40.4	-40.8
Segmentation_17	225 ; 837	20	21	10.2	23.49	-184.1	-176.2	-174.3	-174.3	-175.3	-174.3	-174.3	-174.3	-175.7
Segmentation_18	235 ; 882	20	2	3	3.04	-44.8	-34	-34	-34	-35.9	-34	-34	-34	-34
Segmentation_19	228 ; 852	20	2	3.1	3.06	-32.6	-24.1	-24.1	-24.1	-25.4	-24.1	-24.1	-24.1	-24.1
Segmentation_20	232 ; 867	21	21	8.2	3199.8	-138.1	-112	-112	-112	-112.4	-112	-112	-112	-113.5
wcsp_14	301 ; 19161	48	8	68.1	75.36	1.5	-29.2	-9.6	1.1	-402.4	1.1	1.1	1.1	-20.9
wcsp_15	125 ; 736	66	4	16.3	3610.4	-162	-163.6	-80	-74.6	-1503.1	-75.4	-78.6	-83.4	-193.8
wcsp_16	200 ; 1970	57	44	8.9	3608.9		-5.3	4.8	23.1	-251.5	22.9	22.8	23.2	-8.5
wcsp_17	340 ; 3417	95	44	12.1	3612.1		-101.8	8.3	33.6	-549.1	34.9	35	34.8	
wcsp_18	239 ; 18016	38	24	80.4	80.47	1.1	-540	-24.1	0.2	-16	0.2	0.2	0.2	-16.9

Figure 4.2: AOBB-UFO on UAI 2022 Competition Final Problems (cont'd)

to produce a solution for all of the problems whereas `braobb`, `daoopt-lh`, `lbp`, and `uai14` each sometimes fail (indicated by blanks in their respective columns). In those cases where the competing solvers do not produce a solution but `AOBB-UFO` can, we consider `AOBB-UFO`'s performance to be superior. Furthermore, for 34 of the 100 problems, the `AOBB-UFO` solution value is equal to or better than *all* of the competing solvers solution values, and it produced solution values that were strictly superior to competing solvers on 18 problems (on average doing strictly better than two-to-three competing solvers). In addition, `AOBB-UFO` generated a solution value for all but one of the Promedas problems (denoted as "or_chain_...). In contrast, all of the competing solvers failed at providing a solution for any of these problems except for one.

It is important to remember that `AOBB-UFO` solutions are evaluated on the underflowed model $\mathcal{M}_\tau$, which lead to lower bounds on the true value of the solution returned by `AOBB-UFO`.

Highlighting its speed, we observed that for 53 out of the 100 problems, `AOBB-UFO` found a non-zero solution within 120 seconds. This indicates the potential of the UFO scheme at helping find quick (albeit maybe dirty) initial solutions.

4.5 Conclusion

We introduced a new scheme called UFO for infusing artificial determinism into graphical models. Increased determinism can empower graphical model algorithms to exploit constraint processing for increased efficiency, for example by early pruning during search. We presented UFO schemes that use a binary search to find an appropriate threshold to use in underflowing function values to zero. The search for a threshold aims to ensure that we avoid thresholds that make the problem inconsistent. Once a legitimate threshold is found, it can be relaxed (i.e., its value lowered) to allow more flexible solutions.

Several variants of `UFO` were presented, each having their respective strengths and weaknesses. `UFO-GT` is the simplest variant uses a global threshold that is applied to all functions. The variant `UFO-RT` seeks a constant factor that multiply a function dependant value yielding a vector of underflow-thresholds, one for each function. This can be particularly beneficial when functions are scaled differently. `UFO-Sol` on the other hand takes into consideration known good solutions, and sets an underflow that keeps each consistent.

We ientified simple properties of UFO, being an inherently lower-bounding scheme. We derived a bound on the error and showed that it can be estimated efficiently.

Finally, we evaluated UFO by adapting a vanilla AND/OR branch-and-bound MMAP algorithm with constraint propagation and with UFO, testing the scheme on 100 MMAP problems used in the 2022 UAI Inference Competition. The scheme was able to produce an equivalent or better solution than all of the other solvers for roughly a third of the problems, and was strictly better for almost a fifth of the problems. Furthermore, the algorithm often terminated very quickly showing its potential for producing quick lower bounds.

In summary, although applying UFO forfeits anytime optimality guarantees, it provides lower bounds and given its speed and sometimes strong performance, it has the potential of empowering anytime algorithms empowered with constraint processing to more quickly find better solutions or as a way to initialize search bounds.

Chapter 5

AND/OR Search-Based Computational Protein Design

5.1 Introduction

Modern advances in molecular biology have led to heightened interest in identifying or designing proteins with desirable properties, such as high affinity antigen binding proteins (for example, to treat a disease by inhibiting its virus). However, computational protein design, even when restricted to small changes to existing analogues, is far from trivial. A protein is a sequence constructed from approximately twenty unique building blocks called *amino acids*, with some proteins reaching sizes of over 27,000 amino acids long. If we design a novel protein by replacing even just four positions in the protein's chain, considering all possible combinations at those positions would require enumerating $20^4 \approx 160000$ unique variants. If a variant needs to be created and tested *in vitro* via biochemical laboratory processes, then building, purifying, and testing each variant could take anywhere from tens to thousands of dollars (assuming availability and access to important equipment). The process is also time consuming: graduate students may spend their entire PhD cataloging just a handful of proteins. Hence, developing automated methods to help focus on the most promising variants can be invaluable.

The field of Computational Protein Design, or CPD, casts the search for potentially good protein designs as an optimization problem. CPD's task is to find a sequence of amino acids that optimizes a score corresponding to the degree to which the protein has a desired function or property. Generically, we solve:

$$\underset{\boldsymbol{r}}{\operatorname{argmax}}\, o(\boldsymbol{r}) \tag{5.1}$$

where $\boldsymbol{r}$ is a vector representing a possible amino acid combination and $o(\boldsymbol{r})$ is its score (i.e., the objective function to maximize).

Of course the choice of objective function plays a critical role in the resulting sequence produced. A good objective function should be tractable to evaluate (or approximate) and relate to the desired design goal. For example, suppose that we wish to redesign a protein to improve its binding affinity to a particular ligand. For this task, we can consider the complexed (bound) state of the protein with its ligand and search for an amino acid sequence that results in favorable interactions. In general, search for such an optimal sequence is challenging, for example because proteins may have many amino acids involved in the protein's interactions, and those interactions depend on how the protein is oriented in three-dimensional space. However, if we simplify the problem by choosing to redesign only a small portion of the protein – focusing on a few positions in the protein chain – and assume discrete orientations, the task becomes quite feasible. The flavor of such a problem is closely related to well-studied optimization tasks within the field of graphical models. As such, these problems have been cast within the framework of graphical models, and graphical model algorithms used to solve them [Ruffini et al., 2021, Hallen and Donald, 2019, Zhou et al., 2016].

Recently, a more complex objective function called K* has been proposed [Ojewole et al., 2018] for redesigning proteins to improve affinity. However, performing the optimization using A^*-like strategies in a classical search space (as in Ojewole et al. [2018]) results in a memory-intensive process with poor anytime behavior. Here, we address the same problem, but cast the K^* objective into an alternative graphical model framework, which can better leverage existing state-of-the-art graphical model algorithms [Dechter and Rish, 2002, Marinescu and Dechter, 2009a, Marinescu et al., 2014, 2018a, 2019, 2018b] and take advantage of independences that may be present by using an AND/OR search space.

Contributions. In this work we propose a framework that casts protein redesign problems as a graphical model and associated K^* inference query, and to adapt existing graphical model algorithms to solve the resulting protein redesign task. As such, this chapter provides the following contributions:

1. Two formulations of protein redesign problems as a graphical model for the task of optimizing the K^* objective, which acts as an approximation to binding affinity.
2. `wMBE-K*`, an adaptation of Weighted Mini-Bucket Elimination for use in bounding K^*. More importantly, the scheme is used to efficiently generate heuristics for search over AND/OR search spaces for the K^* objective.
3. Development of an array of `AOBB-K*` algorithms, specifically anytime depth-first branch-and-bound algorithms over AND/OR search spaces for protein redesign maximizing K^*. Variants include augmentation with weighted heuristic search, use of dynamic heuristics, and the incorporation of the `UFO` scheme.
4. Empirical analysis with real protein benchmarks comparing these schemes to state-of-the-art algorithm `BBK*` [Ojewole et al., 2018] (part of the long-standing computational protein design software OSPREY [Hallen et al., 2018]) and demonstrating our methods' strong performance.

▢ 5.2 Background

Proteins are involved in a vast number of both cellular and extracellular functions such as cell signalling, signal transduction, DNA replication, catalyzing reactions, as part of the immune response, structure and mobility, and many more. As such, designing proteins for specific structures and functions - particularly those not typically found in nature - have wide applications in chemistry, biology, pharmacology, and medicine.

Protein Structure. A protein is a complex molecule made up of a sequence of organic compounds called *amino acids*, which can be viewed as the building blocks of proteins. Each of the roughly 20 naturally occurring amino acids contains a set of identical *backbone* atoms, along with a unique *side chain* group of atoms that determines that amino acid's specific properties. The amino acids are linked together along their common backbone structure by a particular type of chemical bond called a peptide bond, to form a polypeptide chain. This polypeptide chain folds into a three-dimensional structure, which can interact with other such folded chains. As such, protein structure is organized into four levels:

Primary Structure: The linear sequence of amino acids in the polypeptide chain, determined by the genetic code. The specific sequence dictates the protein's overall shape and function.

Secondary Structure: Local folding patterns within the polypeptide chain, such as alpha-helices and beta-sheets, stabilized by hydrogen bonds between backbone atoms.

Tertiary Structure: The overall three-dimensional shape of a single polypeptide chain, formed by interactions between the side chains of the amino acids, including hydrogen bonds, ionic bonds, hydrophobic interactions, and disulfide bridges.

Quaternary Structure: Some proteins consist of multiple polypeptide chains, or subunits, that come together to form a functional protein complex. Quaternary structure refers to the arrangement and interaction of these subunits.

The term *conformation* refers to the three-dimensional shape taken on by the protein and its constituent atoms (so, its secondary and tertiary structure), which may be affected by the environment or other molecules (quaternary structure).

The Dynamic Nature of Proteins. However, proteins are not static structures; they are dynamic and flexible molecules that continuously undergo movements and conformational changes essential for their biological functions. This dynamic behavior allows proteins to: interact with other molecules by adjusting their shape to bind specifically and effectively with substrates, inhibitors, and other proteins; perform chemical reactions by allowing conformational changes during catalysis to facilitate the conversion of substrates into products; and respond to environmental changes by altering their structure in response to changes in acidity, temperature, and the presence of signaling molecules, enabling appropriate cellular responses. The dynamic nature of the protein structure is critical for processes such as signal transduction, molecular recognition, and cellular regulation. Mutations that lead to misfolding or impaired dynamics can interfere with a protein's function, and have been associated with a number of diseases.

Protein Design. Protein design can be broadly partitioned into two sub-fields: *de novo* protein design and protein redesign. The task of *de novo* protein design is to design completely new proteins that perform a target function or have a desired structure, designing them without a starting blueprint such as an existing protein structure. Example uses of *de novo* protein design include constructing a protein that has a binding interaction with a given target (such as a small binding protein to a viral antigen), that folds into a particular topology (such membrane protein channels), or that is able to catalyze a desired reaction (such as enzymes) [Cao et al., 2020, Korendovych and DeGrado, 2020, Watson et al., 2023]. In contrast, protein re-design is the task of altering an existing protein's sequence to achieve a desired change in behavior or structure but without altering other aspects of the protein's structure and function. Some common tasks for redesign include modifying a protein subunit to enhance its binding to other subunits (improving the energetics of the complexed form), ligands (such as for an enzyme substrate interaction), or antigens (such as with antibody bonding to antigens), or creating or removing allosteric binding sites. All of these involve optimizing a portion of the protein's amino acid sequence to either maximize or minimize binding to a target. *Computational* Protein Design (CPD), then, is to use computational methods to achieve the goals of protein design.

Computational Protein Design. In this chapter, we exclusively focus on the task of automated protein redesign to increase affinity between known protein subunits. For simplicity, in the sequel we use "computational protein design" (or "CPD") to refer to this specific task.

Within this setting, for a given protein of interest, we designate certain amino acid positions (or *residues*) as *mutable* – these are the positions in the sequence at which alternate amino acids will be considered. Then, through a computational process, we try to identify a preferred residue sequence. Given a particular sequence (or in some methods, a partial sequence), we use a *score function* to estimate the resulting protein's suitability, or "goodness", for the target application, for example its ability to bind to a target site, form a particular geometry, or catalyze a reaction. Although we are searching over the protein sequence (primary structure), our score function should take into account the possible conformations of the resulting protein (e.g., its secondary and tertiary structures).

Simplifying Assumptions. The state space for these conformations is continuous – or, if discretized, still extremely large – making the inference problem intractable. As such, many simplifications are commonly assumed to speed up the redesign process:

Select of Mutable Residues: We typically consider of only a subset of the residues –

often those involved in the interactions – as mutable, keeping the rest fixed.

Predetermined Side-Chain Rotamers: We assume that the continuous-valued side-chain conformations can be well-approximated using a discrete collection of the most common positions, called *rotamers.*

Fixed Backbone Structure: We assume that the conformation of the backbone of the protein is held fixed.

With these simplifying assumptions, many algorithms have been designed to find mutations with the potential to provide improved protein functionality Hallen and Donald [2019], Zhou et al. [2016].

5.2.1 Suitable Objective Functions

In order to facilitate automated design, a quantitative notion of a protein's "goodness" must be defined for the optimization. Here we describe two such objective functions in the context of protein redesign for maximizing protein-ligand affinity: the GMEC and K*.

5.2.1.1 The GMEC Objective

In general, proteins are dynamic structures that can assume a large number conformations [Frauenfelder et al., 1991]. Each particular conformation c of a protein corresponds to a specific free energy value $E(c)$, where lower free energy correspond to more stable conformations. The probability of a protein adopting any one conformation can be related to its energy by the Boltzmann distribution:

$$P(c) = \frac{1}{Z^{\int}}\, e^{-\frac{E(c)}{\mathscr{R}T}}, \quad Z^{\int} = \int_{\boldsymbol{c}} e^{-\frac{E_\gamma(c)}{\mathscr{R}T}} dc \tag{5.2}$$

where $\mathscr{R}$ is the so-called Boltzmann constant, and the partition function $Z^{\int}$ is the normalizing constant integrating the contribution of all possible conformations. In summary, lower energy conformations are both more stable and more probable.

The Global Minimum Energy Conformation, or GMEC, of a dynamic molecular system is the conformational state that results in the lowest overall free energy. For many years it was believed that the GMEC determined the native structure of a protein [Anfinsen, 1973], and the energy corresponding to the GMEC is still often used as an approximation of overall protein stability [Hallen et al., 2018]. Letting $\mathcal{C}(\boldsymbol{r})$ be the possible conformational states of the amino acid sequence $\boldsymbol{R} = \boldsymbol{r}$, a protein redesign problem using the GMEC objective would then be:

$$\operatorname*{argmin}_{\boldsymbol{R}} GMEC(\boldsymbol{r}), \quad GMEC(\boldsymbol{r}) = \min_{c \in \mathcal{C}(\boldsymbol{r})} E(c) \tag{5.3}$$

Although the GMEC is widely accepted as a reasonable, if crude, approximate score function, there are a number of issues with its use as an objective. Perhaps most obvious is that the GMEC only considers the single best conformational state of each sequence in its comparisons; however, the dynamic nature of proteins suggests it can be important to consider the contributions from other conformations as well. Furthermore, and especially in the context of affinity, it is not enough to understand the goodness of a complexed state alone. Instead, it should be judged in comparison to the goodness of the dissociated (un-bound) state – however low the energy of the bound state, if the unbound state has even lower energy, the protein will not bind effectively. Thus, another objective function known as K* was developed.

▨ 5.2.1.2 The K* Objective

The affinity between two interacting protein subunits P and L is correlated to an equilibrium of the chemical reaction forming their complexed state PL:

$$P + L \rightleftharpoons PL. \tag{5.4}$$

The symbol "$\rightleftharpoons$" indicates that the conversion can go in both directions – namely P and L can come together to form the complex PL, and that the complex PL can dissociate into separated P and L. In fact, both directions happen simultaneously. Initially, one direction is more favored based on the concentrations of P, L, and PL present. The reaction proceeds in the more favorable direction more quickly until an *equilibrium* is reached, after which the reaction continues in both directions at equal rates

This equilibrium point is associated with a constant, K_a, that can be determined *in vivo* or *in vitro* by observing the persisting concentrations of each species, and is defined by,

$$K_a = \frac{[PL]}{[P][L]}. \tag{5.5}$$

where brackets "[ ]" indicate the concentration of the enclosed substance. However, in order to compare the equilibrium constant values K_a of various designs *in vitro*, it is necessary to synthesize the interacting subunits through molecular processes that are both time consuming and costly.

The constant K_a can also be approximated by,

$$K^{\int} = \frac{Z^{\int}_{PL}}{Z^{\int}_{P} Z^{\int}_{L}}, \quad Z^{\int}_{\gamma} = \int_{\boldsymbol{C}} e^{-\frac{E_\gamma(c)}{\mathscr{R}T}} dc \tag{5.6}$$

where $Z^{\int}_{PL}$, $Z^{\int}_{P}$, and $Z^{\int}_{L}$ are partition functions of the bound and unbound states that capture the entropic contributions of their various conformations $\boldsymbol{\mathcal{C}}$. ($E_\gamma(c)$ represents the energy of a particular conformation c of state $\gamma \in \varphi$ where $\varphi = \{P, L, PL\}$, $\mathscr{R}$ is the universal gas constant, and T is temperature (in Kelvin). We can further use a model that discretizes the conformation space. This computed estimate is denoted as K* Ojewole et al. [2018]:

$$K^* = \frac{Z_{PL}}{Z_P Z_L}, \quad Z_\gamma = \sum_{\boldsymbol{c} \in \boldsymbol{D(C)}} e^{-\frac{E_\gamma(c)}{\mathscr{R}T}} \tag{5.7}$$

Simplifying the expressing, we will use:

$$K^* = \frac{Z_B}{Z_U}, \tag{5.8}$$

where B respresents the bound (complexed) state(s) and U represents the unbound (dissociate) states. (For the two-subunit system in our example, $B = \{PL\}$ and $U = \{P\} \cup \{L\}$). This more generalized representation can also be used directly for K* computations involving more than two subunits.

Where the GMEC, being a pure minimum over the energies of the complex's conformations, ignores the realization that protein structures are dynamic, the K* captures this through the computation of the partition function for the different subunits. Furthermore, by the GMEC focusing only on the protein's complexed state, it ignores the dynamicity of the subunit interactions. The K* captures this through taking a ratio of the bound and unbound states. However, since minimizing the GMEC results in a pure optimization task – a task much easier than that of mixed inference – many solvers adopt this objective. On the other hand, the K* objective captures both the dynamicity of protein conformations and subunit interactions, and so can be considered as a better objective. K*MAP is the formalization of computational protein design as a task to maximize K*,

$$K^*\text{MAP} = \underset{\boldsymbol{R}}{\operatorname{argmax}}\, K^*(r), \tag{5.9}$$

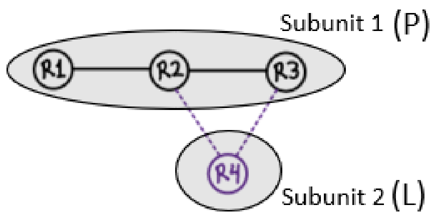
where we look for amino acid assignments $\boldsymbol{R}=r$ that maximize K*. Thus, the goal this line of work is to develop efficient algorithms for computing K*MAP, from which one can identify a small set of promising sequences to experimentally test *in vitro* and *in vivo*, saving much time and cost. Our work taps into recent algorithms developed for the marginal MAP task on graphical models.

5.3 Graphical Model for K*MAP

As the first main contribution of this chapter, we describe two formulations of the CPD problem as a graphical model for computing K*MAP. These build upon previous work from MMAP (see Marinescu et al. [2018b]) and formulations for optimizing the weaker GMEC objective [Zhou et al., 2016]. Put briefly, these models contain two types of variables: a set $\boldsymbol{R}$ of variables representing the various residue positions being considered, and a set $\boldsymbol{C}$ representing the various conformational states the amino acids can take for each residue. Domains of the $\boldsymbol{R}$ variables are the amino acids being considered for each residue. Domains of the $\boldsymbol{C}$ variables are the various conformational states being considered for the amino acid selected for that residue. The functions of the model can be partitioned into two groups: a set of constraints denoted $\mathscr{C}$ that enforce consistency between residue and conformation assignment pairs, and a set of energy functions, denoted $\boldsymbol{E}$, that estimate the contribution to overall free energy from interactions between the amino acid side chains and those of other residues or the protein backbone.

In order to illustrate the graph structure of our two formulations, we construct a toy example protein whose schematic is shown in Figure 5.1. Our toy protein has two subunits, labeled **P** and **L**. Subunit **P** has three residues, with each residue interacting only with its direct neighbor (indicated by a black line connecting them). Subunit **L** is composed of a single residue; when the protein complexes – namely **P** and **L** come together – the residue in **L** interacts with each of the last two residues in **P** (indicated by dotted lines).

Figure 5.1: Example protein subunit interaction schematic.



5.3.1 Formulation 1 (F1)

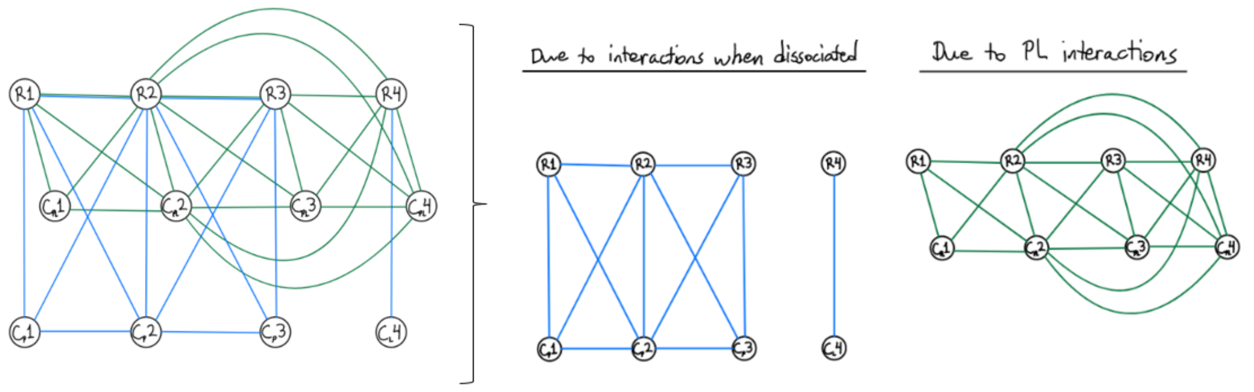


Figure 5.2: Illustration of CPD Formulation F1.

Variables and Domains: Let $\boldsymbol{R} = \{R_i \,|\, i \in \{1, 2, ..., N\}\}$ be the set of **residue variables** representing N different residues (i.e., positions) of the protein. Each R_i has a corresponding domain $D_{R_i} = \{aa | aa$ is a possible amino acid assignment to residue $i\}$. For residues that are being considered for mutation (**mutable residues**), each R_i considers one of ~20 possible amino acid assignments in its domain of values. These are the MAP variables maximized over in the K*MAP task.

$\boldsymbol{C}_\gamma = \{C_{\gamma(i)} \mid i \in \{1, 2, ..., N\}\}$ are **conformation variables**, each indexing a discrete spacial conformation (rotamer) of the amino acid at residue R_i when its subunit is in form $\gamma \in \{B, U\}$, where B represents the subunit when it is in the bound or complexed state, and U when the subunit is dissociated and the protein is in the unbound state. Each $C_{\gamma(i)}$

has corresponding domain $D_{C_{\gamma(i)}} = \{1, 2, ..., M_{\gamma(i)}\}$, where $M_{\gamma(i)}$ is the maximum number of rotamers for any amino acid assignment to R_i in form γ. The assignment to $C_{\gamma(i)}$ acts as an index to the possible side chain conformations of the amino acid assigned to R_i. If assignment $C_{\gamma(i)} = c_{\gamma(i)}$ is greater than the maximum number of rotamers for amino acid $R_i = r_i$, then the joint assignment $(R_i = r_i, C_{\gamma(i)=c_{\gamma(i)}})$ is said to be *inconsistent*. During inference, the $\boldsymbol{C_\gamma}$ are the SUM variables to be marginalized over.

Functions: We consider two sets of energy functions: $E^{sb}_\gamma = \{E^{sb}_{\gamma(i)}(R_i, C_{\gamma(i)}) | i \in \{1, 2, ..., N\}\}$ that capture the energies of interaction of the amino acid at each residue i with itself and the surrounding backbone, and $E^{pw}_\gamma = \{E^{pw}_{\gamma(ij)}(R_i, C_{\gamma(i)}, R_j, C_{\gamma(j)}) \,|\, \text{for } i, j \text{ s.t. } R_i \text{ and } R_j \text{ interact}\}$ that capture pair-wise energies of interaction between amino acids in close proximity. Functions with R_i and $C_{\gamma(i)}$ in their scope are assigned infinite energy values for inconsistent assignments to R_i and $C_{\gamma(i)}$. As a result, inconsistent assignments have a zero probability (see Equation 5.2) and thus these infinite energies act as implicit constraints.

Figure 5.2 shows the graph structure resulting from the toy protein in Figure 5.1 cast into the F1 graphical model formulation. On the left we see the overall graphical model, and on the right the portions of the model that correspond to the local interactions of the unbound and bound states have been extracted. Both the bound and unbound states share the same residue variables. We also see that, in the unbound states, the local interactions of the different subunits are independent of each other.

Objective: The K* objective can be expressed as $K^*(R_1...R_N) = \frac{Z_B(R_1...R_N)}{Z_U(R_1...R_N)}$, where we assume temperature T in Kelvin and Boltzmann constant $\mathscr{R}$ and where:

$$Z_\gamma(R_1...R_N) = \sum_{C_{\gamma(1)},...,C_{\gamma(N)}} \prod_{E^{sb}_{\gamma(i)} \in \boldsymbol{E^{sb}_\gamma}} e^{-\frac{E^{sb}_{\gamma(i)}(R_i, C_{\gamma(i)})}{\mathscr{R}T}} \cdot \prod_{E^{pw}_{\gamma(ij)} \in \boldsymbol{E^{pw}_\gamma}} e^{-\frac{E^{pw}_{\gamma(ij)}(R_i, C_{\gamma(i)}, R_j, C_{\gamma(j)})}{\mathscr{R}T}} \tag{5.10}$$

In summary, Formulation 1 provides us with a way to express the protein and its various conformations by using an indexing scheme into the amino acid rotamers. However, we see that its graph structure is densely connected. Formulation 2, described next, takes a different approach which helps lead to a more sparse graph structure.

5.3.2 Formulation 2 (F2)

Formulation 2, inspired by the works of Viricel et al. [2018] and Vucinic et al. [2019], uses a different approach to capture amino acid - rotamer pairs that involves explicit constraints to restrict invalid combinations. This approach can lead to more sparse graph structures that can be leveraged by algorithms.

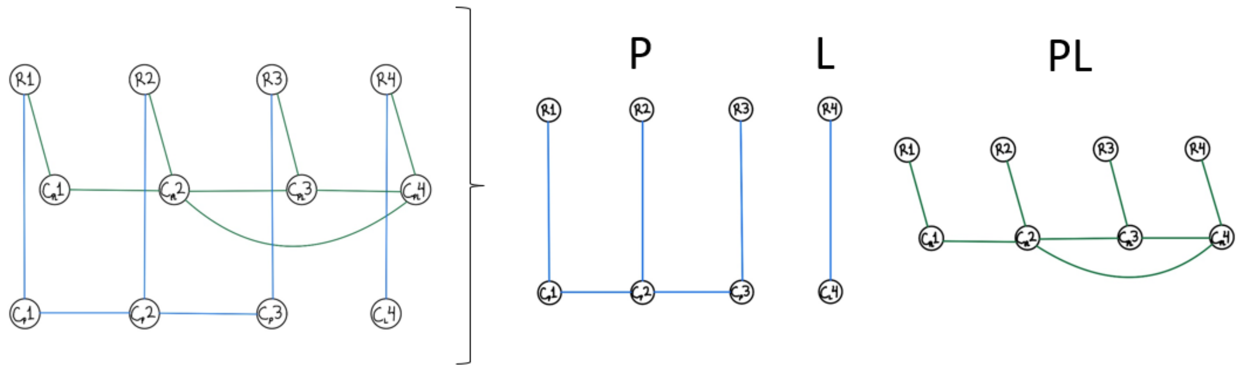


Figure 5.3: Illustration of CPD Formulation F2.

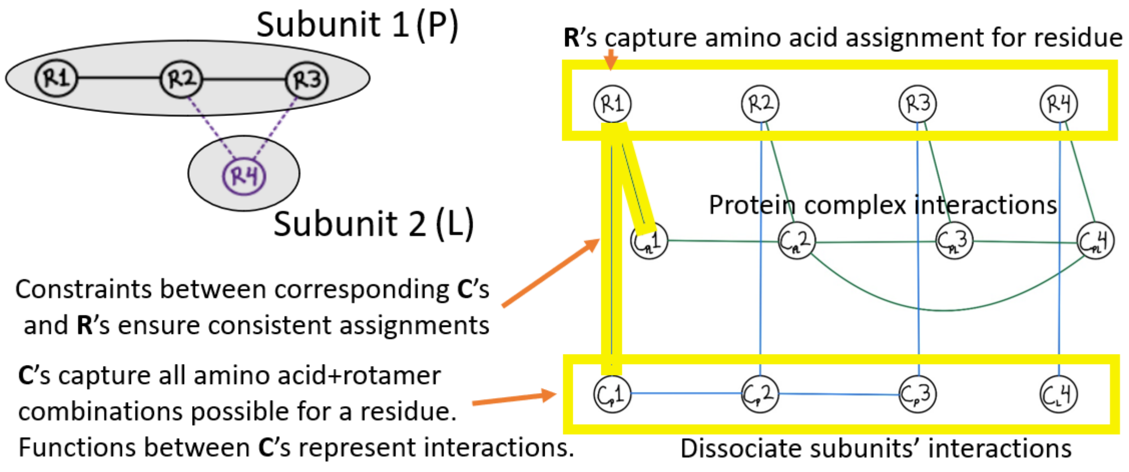
Variables and Domains: As in F1, we define both a set of residue variables $\boldsymbol{R} = \{R_i \,|\, i \in \{1, 2, ..., N\}\}$ and conformation variables $\boldsymbol{C}_\gamma = \{C_{\gamma(i)} \mid i \in \{1, 2, ..., N\}\}$. However, in this formulation, each $C_{\gamma(i)}$ represents a specific amino acid *and* conformation of the residue. In other words, each $C_{\gamma(i)}$ has a domain $D_{C_{\gamma(i)}}$ that includes all possible rotamers of any possible amino acid value R_i. Then, the value of R_i acts as a "selector" into the possible assignments to $C_{\gamma(i)}$, disallowing the rotamers of other possible values of R_i. For example, suppose that at residue i, we consider only two possible amino acids: $R_i \in \{r'_i, r''_i\}$. When the protein is in state γ, for value r'_i only a single rotamer is possible, $c^{(1)}_{r'_i}$; for value r''_i there are two possible rotamers, $c^{(1)}_{r''_i}$ and $c^{(2)}_{r''_i}$. Then, $D_{C_{\gamma(i)}} = \{c^{(1)}_{r'_i}, c^{(1)}_{r'_i}, c^{(2)}_{r''_i}\}$. However, attempting to assign $R_i = r'_i$ but $C_{\gamma(i)} = c^{(2)}_{r''_i}$ would be invalid, since amino acid r'_i does not take on rotamer $c^{(2)}_{r''_i}$. As in formulation F1, we call these mismatched joint assignments *inconsistent*.

If we compare formulations F1 and F2, we see that F1's conformation variables $\boldsymbol{C}$ have much smaller domain sizes, but that this comes at the cost of more interactions between the $\boldsymbol{R}$ and $\boldsymbol{C}$ variables. Conversely, F2's conformation variables have large domain sizes, but its interaction energies are more easily expressed:

Functions: As in F1, we have energy functions to capture singleton and pairwise interactions. However, unlike F1, energy functions for F2 can capture these interactions using only the conformation variables, since their assignments capture both amino acid and rotamer identities. Our energy functions consist of: $E^{sb}_{\gamma} = \{E^{sb}_{\gamma(i)}(C_{\gamma(i)}) \mid i \in \{1, 2, ..., N\}\}$ which captures the energies of interaction of the amino acid at each residue i with itself and the surrounding backbone, and $E^{pw}_{\gamma} = \{E^{pw}_{\gamma(ij)}(C_{\gamma(i)}, C_{\gamma(j)}) \mid \text{for } i, j \text{ s.t. } R_i \text{ and } R_j \text{ interact}\}$, capturing the pair-wise energies of interaction between amino acids in close proximity. In addition, $\mathscr{C} = \{\mathscr{C}_{\gamma(i)}(R_i, C_{\gamma(i)}) \mid i \in \{1, 2, ..., N\}, \gamma \in \varphi\}$ is a set of constraints that ensure the assigned rotamer to $C_{\gamma(i)}$ belongs to the amino acid assigned to R_i. These constraints prevent inconsistent $(R_i, C_{\gamma(i)})$ assignments.

Figure 5.3 shows the graph structure resulting from the toy protein in Figure 5.1 cast into the F2 graphical model formulation. On the left we see the overall graphical model, and on the right the portions of the model that correspond to the local interactions of the unbound and bound states have been extracted. As before, the bound and unbound states share the same residue variables, and in the unbound states the local interactions of the different subunits are independent of each other. Compared to F1, F2's graph is much sparser, with any densely connected regions corresponding to only conformation variables. Figure 5.4 highlights this point by labeling different portions in the graph of our toy protein's model.

Figure 5.4: F2 graph with arcs and nodes labeled.


Subunit 1 (P)
R1
R2
R3
R4
Subunit 2 (L)
R's capture amino acid assignment for residue
R1
R2
R3
R4
Protein complex interactions
Cp1
Cp2
Cp3
Cp4
Constraints between corresponding C's and R's ensure consistent assignments
C's capture all amino acid+rotamer combinations possible for a residue. Functions between C's represent interactions.
C1
C2
C3
C4
Dissociate subunits' interactions

Objective: Again, the K* objective can be expressed as $K^*(R_1...R_N) = \frac{Z_B(R_1...R_N)}{Z_U(R_1...R_N)}$, where

$$Z_\gamma(\boldsymbol{r}) = \sum_{\boldsymbol{C}_\gamma} \prod_{\mathscr{C}_{\boldsymbol{\gamma}}} \mathscr{C}_{\gamma(i)}(r_i, c_{\gamma(i)}) \cdot \prod_{\boldsymbol{E^{sb}_\gamma}} e^{-\frac{E^{sb}_{\gamma(i)}(c_{\gamma(i)})}{\mathscr{R}T}} \cdot \prod_{\boldsymbol{E^{pw}_\gamma}} e^{-\frac{E^{pw}_{\gamma(ij)}(c_{\gamma(i)}, c_{\gamma(j)})}{\mathscr{R}T}} \tag{5.11}$$

5.3.3 Resulting Pseudo Tree

We saw in Figures 5.2 and 5.3 that for both F1 and F2, the extracted local interactions for the protein in its unbound state are independent of those when the protein is in its bound state, with the exception of the shared residue variables. Thus, given a variable ordering that places residue variables at the beginning of the ordering, the resulting pseudo tree structure for both can be depicted by the general schematic shown in Figure 5.5.

The decomposition captured by this schematic shows that the AND/OR search space will result in independent sub-trees, creating a more compact search space and enabling more efficient search.

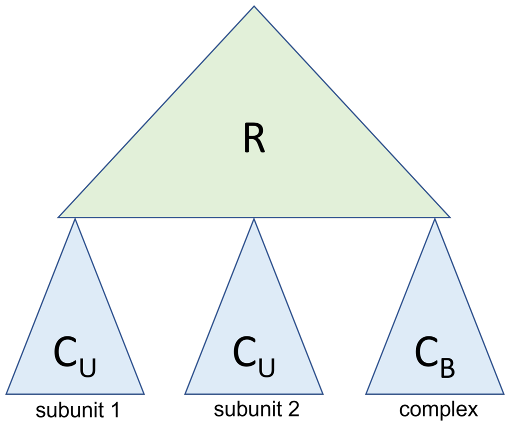


Figure 5.5: Schematic of resulting pseudo tree for CPD formulated as F1 or F2.

5.3.4 Subunit-Stability Thresholds

It is important to note that, semantically, low values for $Z_\gamma(\boldsymbol{r})$ imply estimated low stability with amino acids $\boldsymbol{R} = \boldsymbol{r}$, and thus it is unlikely that subunit γ will retain the desired structure. Thus, even if very low $Z_\gamma(\boldsymbol{r})$ can yield high K*, such solutions are not biologically relevant and we prefer to omit $\boldsymbol{R} = \boldsymbol{r}$ as a possible solution.

We define subunit-stability thresholds S_γ for which we restrict valid solutions $\boldsymbol{R} = \boldsymbol{r}$ to satisfy $Z_\gamma(\boldsymbol{r}) > S_\gamma$ for all subunits γ, as in Ojewole et al. [2018]:

Definition 5.1 (subunit-stability threshold)
For each subunit γ, its corresponding subunit-stability threshold is

$$S_\gamma = Z_\gamma(\boldsymbol{r}_{wt}) \cdot e^{\frac{-5}{\mathscr{R}T}} \tag{5.12}$$

where $\boldsymbol{r}_{wt}$ is the wild-type (i.e., naturally occurring) amino acid sequence, $\mathscr{R}$ is the universal gas constant, and T is the temperature in Kelvin.

As such, we can add corresponding global constraints to the model to enforce biologically relevant solutions according to S_γ.

Definition 5.2 (CPD Graphical Model)
Let $\mathcal{M}_{cpd} = \langle \boldsymbol{X} = \boldsymbol{R} \cup \boldsymbol{C},\ \boldsymbol{D},\ \boldsymbol{F} = \mathscr{C} \cup \boldsymbol{E},\ \boldsymbol{S} \rangle$ *be a CPD graphical model for* K^**MAP optimization.*

With a graphical model framework in place, next we describe a heuristic that can provide a bound on the K*MAP value and that can be used to guide search.

5.4 wMBE-K*

Next we present a weighted mini-bucket based approximation scheme for K^*, which adapts weighted mini-bucket for MMAP [Dechter and Rish, 2002, Marinescu et al., 2014] to the K^* objective. Since mini-bucket bounds are compatible with AND/OR search, we leverage these bounds as heuristics to guide branch-and-bound schemes.

Algorithm 14 describes `wMBE-K*`, which operates similarly to wMBE-MMAP (Algorithm 2). Two key similarities are that (1) it uses an elimination order that constrains buckets of MAP variables, over which maximization occurs, to be processed last (line 3); and (2) for any bucket that has a width larger than the provided i-bound, a bounded approximation is made by partitioning the bucket functions into T mini-buckets (line 4) and taking the product of their power-sums over the bucket variable (lines 10-14, 15-22).

For K*MAP, two key innovations are required: (1) buckets corresponding to variables in $\boldsymbol{C_U}$, whose marginal belongs to the denominator of the K* expression, must be lower-bounded (to provide an upper bound on K*) using a modification of Holder's inequality that incorporates negative weights [Liu and Ihler, 2011b] (lines 15-22); and (2) when messages are passed from buckets corresponding to variables in $\boldsymbol{C_U}$ to that of $\boldsymbol{R}$, the messages are inverted to capture their role in the denominator (line 20). Although we omit the details here, `wMBE-K*` can also employ cost shifting to tighten its bounds (see Flerova et al. [2011], Liu and Ihler [2011b]).

Algorithm 14: `wMBE-K*`

```
Input: CPD Graphical model $\mathcal{M}_{cpd}$ (Def 5.2);  i-bound $i$;
constrained elimination order $o_{elim} = [X_1, ..., X_n]$
Output: upper bound on the K$^*$MAP: $ub_{K^*MAP}(\mathcal{M}_{cpd})$
begin
    Partition the functions $f \in \boldsymbol{F}$ into buckets $B_1, ..., B_n$ s.t. each function is placed in the bucket
      corresponding to the lowest-index variable in its scope.
    foreach $k = 1...n$ do
        Generate a mini-bucket partitioning of the bucket functions $\boldsymbol{MB_k} = \{MB_k^{(1)}, ..., MB_k^{(T)}\}$ s.t.
          $|scope(f_{MB_k^{(t)}})| \leq i$, for all $MB_k^{(t)} \in \boldsymbol{MB_k}$
        if $X_k \in \boldsymbol{MAP}$ then
            foreach $MB_k^{(t)} \in \boldsymbol{MB_k}$ do
                $\lambda_k^{(t)} \leftarrow \max_{X_k} f_{MB_k^{(t)}}$
            end
        else
            if $X_k \in \boldsymbol{C_B}$ then                                  // upper-bound for numerator
                Select positive weights $\boldsymbol{w} = \{w_1, ..., w_T\}$ s.t. $\sum_{w_t \in \boldsymbol{w}} w_t = 1$
                foreach $MB_k^{(t)} \in \boldsymbol{MB_k}$ do
                    $\lambda_k^{(t)} \leftarrow \sum_{X_k}^{w_t} f_{MB_k^{(t)}}$
                end
            else if $X_k \in \boldsymbol{C_U}$ then                             // lower-bound for denominator
                Select a negative weight $w_1$ and positive weights $\boldsymbol{w} = \{w_2, ..., w_T\}$ s.t. $\sum_{w_t \in \boldsymbol{w}} w_t = 1$
                foreach $MB_k^{(t)} \in \boldsymbol{MB_k}$ do
                    $\lambda_k^{(t)} \leftarrow \sum_{X_k}^{w_t} f_{MB_k^{(t)}}$
                    if $scope(\lambda_k^{(t)}) \subseteq \boldsymbol{R}$ then
                        $\lambda_k^{(t)} \leftarrow 1/\lambda_k^{(t)}$
                    end
                end
            end
        end
        Add each $\lambda_k^t$ to the bucket of the lowest-index variable in its scope.
    end
    return $\lambda_n = ub_{K^*MAP}(\mathcal{M}_{cpd})$
end
```

In our empirical evaluation cost-shifting is implemented as well. Finally, the complexity of the algorithm, which is parameterized by the i-bound i, is time and space exponential in i.

As can be expected, bounding a ratio of functions (as in K*) is particularly challenging, requiring both upper and lower bounds. Lower bounding of functions, particularly functions with constraints, can be especially challenging. For larger problems and low i-bounds, we find this often yields relatively weak bounds, and propose an improvement to partially remedy this behavior.

5.4.1 Domain-Partitioned MBE

The preceding discussion suggested applying mini-bucket without regard to the presence of hard constraints and consistency requirements. But these constraints, which are represented as zeros in functions, can lead to poor lower bound generation. For example, in standard mini-bucket elimination lower bounds [Dechter and Rish, 2003], the bound involves a minimum over function values – so, zero values can cause the resulting lower bound to become zero as well. However, in the CPD domain, the functions represent protein energetics, and satisfiable configurations correspond to positive function values. We can thus guarantee a positive lower bound by using a simple remedy:

Theorem 5.1 (Bounded Domain-Partitioning)
Consider three variables X, Y, and Z and objective

$$obj = \sum_{X} f(x,y) \cdot g(x,z) \tag{5.13}$$

Let $X' = \{x \in X | g(x,z) \neq 0\}$, and take $\epsilon_{X'} = \min_{x \in X'} g(x,z)$. Since by definition $\epsilon_{X'} > 0$, we can derive:

$$\begin{aligned} obj &= \sum_{x \in X'} f(x,y) \cdot g(x,z) + \sum_{x \in X \setminus X'} f(x,y) \cdot g(x,z) && (5.14)\\ &= \sum_{x \in X'} f(x,y) \cdot g(x,z) \quad \geq \quad \epsilon_{X'} \cdot \sum_{x \in X'} f(x,y) && (5.15)\\ &> 0 && (5.16) \end{aligned}$$

when $f(x,z)$ is not identically zero over X'.

Standard Mini-Bucket Elimination, even under the conditions above, produces its lower

Algorithm 15: `AOBB-K*`

```
Input: CPD graphical model $\mathcal{M}$; pseudo-tree $\mathcal{T}$; $K^*$ upper-bounding heuristic function $h_{K^*}(.)$; $Z_\gamma$
       upper-bounding heuristic function $h_{Z_\gamma}(.)$; and subunit stability threshold $S_\gamma$ for each
       subunit $\gamma$
Output: $K^*MAP(\mathcal{M})$
begin
    Encode deterministic relations in $\mathcal{M}$ into CNF
    $\pi \leftarrow$ root OR node $s$
    $ub_{K^*}(s) \leftarrow h_{K^*}(s)$
    $lb_{K^*}(s) \leftarrow -inf$
    $g(s) \leftarrow 1$
    foreach $\gamma \in \varphi$ do
        $UB_{Z_\gamma}(s) \leftarrow \prod_{m \in ch_{T_\gamma}(s)} h_{Z_\gamma}(m)$
    end
    while $n_X \leftarrow EXPAND(\pi)$ do
        if $ConstraintPropagation(\pi) = false$ then
            $PRUNE(\pi)$
        else if $\exists \gamma \in \varphi$ s.t. $UB_{Z_\gamma}(n_X) < S_\gamma$ then
            $PRUNE(\pi)$
        else if $X \in \boldsymbol{R}$ then
            if $\exists a \in anc^{OR}(n)$ s.t. $ub_{K^*}(a, \pi) < lb_{K^*}(a)$ then
                $PRUNE(\pi)$
            end
        else if $ch_T^{unexp}(n) = \emptyset$ then
            $BACKTRACK(\pi)$
    end
    return $ub_{K^*}(s) = lb_{K^*}(s) = K^*MAP(\mathcal{M})$
end
```

bound as $\min_X g(x,z) \cdot \sum_{x \in X'} f(x,y)$; while min value can be zero, we see that $\epsilon_{X'} \cdot \sum_{x \in X'} f(x,y)$ provides a tighter, and strictly positive, lower bound.

5.5 AOBB-K*

State-of-the-art K* optimizers, such as `BBK*`, utilize memory-intensive best-first A*-like search algorithms [Ojewole et al., 2018, Hallen et al., 2018]. In contrast, depth-first algorithms offer a linear space complexity, enabling them to solve problems that best-first methods may not be be able to due to memory limitations [Zhou et al., 2016]. As a key algorithmic contribution of this work, we present `AOBB-K*`, a depth-first AND/OR branch-and-bound scheme for solving K*MAP.

`AOBB-K*` (Algorithm 15) traverses the underlying AND/OR search of pseudo tree $\mathcal{T}$, expanding nodes in a depth-first manner (line 10), and pruning whenever any of three conditions are triggered: (1) the resulting variable assignments violate constraints encoded as infinite values in $\mathcal{M}$ (line 11); (2) a subunit-stability constraint (SSC) – a constraint which enforces the partition function of each protein subunit, Z_γ, to be greater than a biologically-relevant threshold S_γ [Ojewole et al., 2018] – is violated (line 13); or (3) when it can be determined that the current partial assignment cannot produce a K* value greater than that previously found during the search (line 16). Backtracking occurs when all the node's children have been explored and returned from (line 19), at which point the K* value of the subproblem rooted at that node is computed exactly, and bounds of its parents are tightened accordingly. Infinite energy tuples in $\mathcal{M}$'s functions are used to forbid inconsistent amino acid - rotamer pairs. During search, unit-propagation (e.g., Eén and Sörensson [2004]) propagates values through these constraints and detects infeasible configurations (line 11).

The algorithm progresses until it backtracks to, and updates, the root of the search tree with the maximal K* value of an amino acid sequence that also satisfies the subunit-stability thresholds.

Throughout search, each node n maintains a progressively tightened upper bound $ub_{K^*}(n)$ on the K*MAP of the sub problem rooted at n. When a node is expanded, this bound is initialized based on the upper-bounding heuristic function $h^{ub}_{K^*}(.)$ (line 4). As search progresses, $ub_{K^*}(n)$ decreases, converging towards the K*MAP of the sub problem rooted at n. Furthermore, each node n also maintains a progressively improved upper bound on the partition function of each subunit γ consistent with the path to n, $UB_{Z_\gamma}(n)$ (line 8). At each step $UB_{Z_\gamma}(n)$ is recomputed to ensure that it is greater than the provided subunit-stability threshold S_γ, thus satisfying the subunit-stability constraints and ensuring that only biologically relevant solutions are considered [Ojewole et al., 2018]. Note that the subunit stability constraints are not encoded into the problem, and thus add another layer of complexity not present in classical inference tasks such as MMAP.

`AOBB-K*` is a systematic search algorithm that only prunes its search space when it can prove sub-optimal or invalid (under the subunit-stability constraints) solutions. Therefore we have,

Theorem 5.2 (`AOBB-K*` correctness, completeness)
`AOBB-K*` *is sound and complete for optimal K^* under the subunit-stability constraints.*

From Marinescu and Dechter [2009c,b] it follows that,

Theorem 5.3 (`AOBB-K*` complexity)
`AOBB-K*` *is time $O(n \cdot k^d)$ and space $O(n)$, where n is the number of variables, k is the maximum domain size, and d is the depth of the guiding pseudo tree. When* `AOBB-K*` *is modified to search the context minimal AND/OR graph it is both time and space $O(n \cdot k^{w^*})$, where w^* is the induced width of the pseudo tree.*

5.5.1 Weighted Search for K*

Weighted best-first search (e.g., WA* [Pohl, 1970], WAO* [Desarkar et al., 1987]) is a well known principle for converting best-first search into an anytime scheme by multiplying the heuristic function $h(n)$ of a node n in the search space by a weight $\omega > 1$. The solution is guaranteed to be ω-optimal (i.e., within a factor ω of the optimal solution).

Therefore, `AOBB-K*` can easily be relaxed to an ω-approximation scheme (for $\omega \in [0, 1)$) by multiplying $h_{K^*}(n_R)$ at each node n_R in the AND/OR search tree by a factor of ω. The resulting solution will be at worst $\omega \cdot K^*MAP$ [Pohl, 1970, Flerova et al., 2014]. We explore the performance of applying weighted search methods to a class of difficult CPD problem instances in Section 5.10: Empirical Evaluation.

5.6 Boosting AOBB-K*

In order to further enhance `AOBB-K*`, we developed `AOBB-K*-b` (boosted) as an advancement of `AOBB-K*` with modifications that improve its scalability. These enhancements, outlined below, are a mix of CPD domain-specific enhancements as well as principled enhancements that can be generalized to other graphical models tasks and problem domains.

5.6.1 Boosted wMBE-K*

A main cause of the scalability limitations of the `AOBB-K*` is a sometimes weak or unbounded heuristic estimate by `wMBE-K*` (Algorithm 14). This occurs primarily because of difficulties in the lower-bounding computations corresponding to the denominator of K* and lead to loose upper bounds on the K*MAP. Such loose bounds (or lack of bounds) do not allow pruning during search, and can lead to traversal of a much larger search space than if more pruning can occur.

To improve on `wMBE-K*`'s estimates, we introduce `wMBE-K*-b` **(boosted)** which we developed by augmenting `wMBE-K*` with three sequential improvements: (1) adjustment of the power-sum mechanism (lines 13,18) to produce non-zero lower-estimates at the cost of losing bound guarantees, (2) adjustment to the cost shifting mechanism to prevent cost-shifts with zeros, and (3) maximization with finite values over infinite ones (line 7). The specific adjustments and justifications for these modifications are explained next.

1) Enforcing non-zero lower estimates. `wMBE-K*` uses uses a power-sum computation leveraging Holder's inequality to compute bounds on the K*MAP objective, with a version using negative weights for lower bounding the denominator portion of the K* expression (lines 13,18). When doing the lower bounding of the denominator using negative weights, if the consolidated mini-bucket function $f_{B_X^{(j)}}$ contains zeros, then $\left(f_{B_X^{(j)}}\right)^{\frac{1}{w}}$ is undefined –

or infinity according to the continuous limit (Equation 2.5, Equation 2.6). This results in the overall power-sum evaluating to zero (and so the denominator of the K* is set to zero) leading to an unbounded K* upper bound.

Thus, deriving inspiration from Section 5.4.1: Domain-Partitioned MBE we adjust the computation to omit zeros in the lower-bounding power-sum (line 18) thus forcing non-zero estimates for consistent sub problems. This is also in line with the fact that the partition functions of biologically relevant protein subunits are in fact non-zero.

Definition 5.3 (Zero-Omitted Weighted Function)
The zero-omitted w-weighted function, denoted $f^{\triangleleft w}$, of a function f having scope $\boldsymbol{Y}$ is defined by: $f^{\triangleleft w}(y) := f(y)^w$ for $f(y) \neq 0$ and 0 otherwise.

Definition 5.4 (Zero-Omitted Power Sum)
The zero-omitted power-sum of a function f that includes X in its scope is defined by: $\sum_X^{\triangleleft w} f := (\sum_X f(x)^{\triangleleft \frac{1}{w}})^w$ where $\frac{0}{0} := 0$.

Specifically, line 18 is changed to: $\lambda_k^t \leftarrow \sum_{X_k}^{\triangleleft w} f_{MB_k^{(t)}}$.

This modified version can no longer guarantee a lower bound for the denominator of the K* expression, however boundedness can be retained if the omitted zeros correspond to conditions for bounded domain-partitioning (Theorem 5.1).

2) Cost shifting with non-zero values. We similarly adjust the cost-shifting mechanism (described in Flerova et al. [2011], Ihler et al. [2012]) to only consider non-zero values when choosing a Lagrange multiplier. This in turn restricts cost-shifts to be only with non-zero values, helping to prevent numerical instabilities and ensures a positive lower bound for consistent sub problems. This change does not disrupt bound guarantees.

3) Maximizing over finite values. With adjustments 1) and 2) above, the resulting K* approximation for any (partial) configuration of the residues that are consistent will necessarily have a finite positive value (the upper bound estimates on the $K^* = \frac{Z_B}{Z_U}$ numerator are inherently finite and, now, the lower estimate of the denominator is also forced to be positive and finite for consistent sub problems). Thus, during the maximization step in `wMBE-K*` (line 7), we now instead maximize only over the available finite values.

5.6.2 Tuning search

Two key enhancements were also made to `AOBB-K*` search.

Prioritizing the wild-type assignment. Unlike many other problem domains, for protein re-design a good initial assignment to the variables is known ahead of time: the amino acid sequence that corresponds to the wild-type (naturally occurring) protein [Kuhlman and Baker, 2000]. We force the wild-type sequence to be explored first, ensuring that we begin search with a strong initial lower bound.

Prioritizing nodes with a finite heuristic value. Since `wMBE-K*-b` produces infinite K^* estimates only for invalid configurations, we adjust node ordering during search to first explore nodes that have a finite heuristic value. This ensures that consistent configurations are traversed first.

In the Empirical Evaluation section we evaluate the performance impact of these changes.

5.7 Weighted Search

Weighted best-first search (e.g., WA* [Pohl, 1970], WAO* [Desarkar et al., 1987]) is a well known principle for converting best-first search into an anytime scheme by multiplying the heuristic function $h(n)$ of a node n in the search space by a weight $\omega > 1$. The solution is guaranteed to be ω-optimal (i.e., within a factor ω from the optimal one).

Therefore, AOBB-K* can easily be relaxed to an ω-approximation scheme (for $\omega \in [0, 1)$) by multiplying $h_{K^*}(n)$ at each node n in the AND/OR search tree by a factor of ω. It can be shown that the resulting solution will be at worst $\omega \cdot K^*MAP$ [Pohl, 1970, Flerova et al., 2014]. We explore the performance of applying such approximations to a class of difficult CPD problem instances in Section 5.10.

5.8 Dynamic Heuristics

So far, we have used a pre-computed and fixed heuristic function to guide the search process. However, computing bounds dynamically during the search, after some variables have been assigned values, can produce much tighter bounds but at the cost of requiring potentially many bound recomputations. (e.g., Lam et al. [2014]). AOBB-K*-DH (Algorithm 16) provides a general framework for using dynamic heuristics within AOBB-K*.

AOBB-K*-DH performs search similarly to AOBB-K* with the exception that, at each node expansion not resulting immediately in pruning, it makes a decision whether or not to dynamically recompute a new K* upper-bounding heuristic conditioned on the current search path (line 12 This decision is based on two hyper-parameters: *maxDepth*, a maximum depth at which to consider recomputations, and *dhThreshold*, a numerical bound on existing heuristic estimates over which re-computations occur. These hyper-parameters serve to regulate the frequency of dynamic heuristic re-computations since they can be costly both in time and memory. (In particular, wMBE-K* is exponential in its *i-bound* hyper-parameter). When pruning or backtracking past the point of the most recent heuristic re-computation, the K* heuristic tables H_{K^*} are rolled back to cached tables from previous computations (not shown explicitly).

Algorithm 16: AOBB-K*-DH

```
Input: CPD graphical model $\mathcal{M}_{cpd}$ (Def 5.2);
pseudo tree $\mathcal{T}$ guiding node expansions;
$K^*$ upper-bounding heuristic function $h_{K^*}(.)$;
$Z_\gamma$ upper-bounding heuristic function $h_{Z_\gamma}(.)$
Output: $K^*MAP(\mathcal{M}_{cpd})$
begin
    Encode deterministic relations in $\mathcal{M}_{cpd}$ into CNF
    $\pi \leftarrow$ search path initialized with a dummy root node $r$
    $H_{K^*} \leftarrow$ tables precomputed by $h_{K^*}(r)$
    $H_{Z_\gamma} \leftarrow$ tables precomputed by $h_{Z_\gamma}(r)$ for each $\gamma$
    while $EXPAND(\pi, \mathcal{T})$ do
        if $ConstraintPropagation(\pi) = false$ then
            $PRUNE(\pi)$
        else if $\exists \gamma \in \varphi$ $s.t.$ $ub_{Z_\gamma}(\pi, H_{Z_\gamma}) < S_\gamma$ then
            $PRUNE(\pi)$
        else
            if $depth(\pi) \leq maxDepth$ and $H_{K^*}(\pi) > dhThreshold$ then
                $H_{K^*} \leftarrow$ tables recomputed by $h_{K^*}(\pi)$
            if $X \in \boldsymbol{R}$ then
                if $ub_{K^*}(\pi, H_{K^*}) < lb_{K^*}$ then
                    $PRUNE(\pi)$
                end
        while $\pi$ has no unexpanded children do
            $BACKTRACK(\pi)$
        end
    end
    return $lb_{K^*} = K^*MAP(\mathcal{M}_{cpd})$
end
```

Parameters regulating dynamic heuristic re-computation. Dynamic heuristic re-computation can be expensive, and as the search progresses deeper, many heuristics (including the weighted Mini-Bucket Heuristic) becomes more accurate. Thus, a simple way to bound the number of times the heuristic is recomputed is to limit the depth at which re-computation of the heuristic can occur. The provided *maxDepth* parameter does just this, indicating the maximum depth .

Dynamic heuristic re-computations aim to improve bounds and enhance pruning. However, if the existing heuristic value is already tight, the cost of re-computation may outweigh traversing the search space with the current heuristic. Determining what is considered "already tight" can be difficult for general search tasks. However, in the case of protein re-design, valid solutions almost always have a cost similar in magnitude to the score of the wild-type. Therefore, we can initially select a value for *dhThreshold* relative to the native wild-type K^* value and change them towards more speed (increasing the threshold) or accuracy (decreasing it) as desired.

▢ 5.9 Incorporating UFO

In Chapter 4 we discussed the potential of infusing determinism to accelerate search empowered by constraint propagation. In particular, UFO schemes such as `UFO-GT` (Algorithm 10) were developed to replace very small function values (i.e., "near constraints") with 0.0 turning them into hard constraints. Protein design is one example domain in which this approach is potentially quite powerful. Very poor side chain interactions within proteins – namely, interactions that contribute a very high free energy – typically cause instability in the protein structure. Thus, it is uncommon to find stable functional proteins containing such high energy interactions. By applying the UFO methodology, we explicitly forbid these very unfavorable interactions. As our last set of algorithmic improvements, we present `AOBB-K*-UFO`, `AOBB-K*` augmented with a CPD-specific UFO scheme.

In general, larger values of the UFO underflow threshold lead to more determinism, and consequently more aggressive constraint propagation pruning. However, if the threshold is set too high, the resulting model becomes inaccurate and may even become inconsistent, leaving no configuration capable of producing a non-zero value. To maximize our pruning without ruining the model quality, we seek a threshold that is as high as possible yet still ensures a consistent model. To identify the threshold value, `UFO-GT` (Algorithm 10) employs binary search to find the largest threshold that still results in a satisfiable model (lines 6-12).

In CPD, we actually know more information than that the model should remain consis-

Algorithm 17: `AOBB-K*-UFO`

```
Input: $\mathcal{M}_{cpd}$ (Def 5.2); $x_{wt}$, wild-type assignment to $X$; SAT solving algorithm,
       $SAT(.)$ ; time limit for binary search; a deflation factor $0 < \delta \leq 1$; pseudo
       tree $\mathcal{T}$ guiding search; $K^*$ upper-bounding heuristic function $h_{K^*}(.)$; $Z_\gamma$
       upper-bounding heuristic function $h_{Z_\gamma}(.)$;
Output: approximation to the true $K^*MAP(\mathcal{M})$
1 begin
2     $\tau \leftarrow UFO_{cpd}(\mathcal{M}_{cpd}, x_{wt}, SAT(.), time\text{-}limit, \delta)$
3     $K^{*\prime} \leftarrow AOBB\text{-}K^*(\mathcal{M}_{cpd_\tau}, \mathcal{T}, h_{K^*}(.), h_{Z_\gamma}(.))$
4     return $K^{*\prime}$
5 end
```

tent – we know that the wild-type configuration is both valid and reasonable, and we can additionally require that the wild-type sequence should not be made inconsistent by UFO.

To this end, we modify UFO so that as it performs underflows on the Boltzmann transformed E^{sb} and E^{pw} functions (see Equation 5.11), it not only enforces general satisfiability but also enforces satisfiability of the wild-type sequence. This modified UFO scheme, called $\texttt{UFO}_{\texttt{cpd}}$, alters the vanilla `UFO-GT`'s satisfiability check (Algorithm 10, line 7) with one that enforces consistency of the wild-type sequence. Once an appropriate initial threshold is selected, the algorithm decreases the threshold using a hyper-parameter δ (Algorithm 10, line 13) to enable a wider array of solutions in the model.

AOBB-K*-UFO . `AOBB-K*-UFO` (Algorithm 17) empowers `AOBB-K*` by generating an underflowed model $\mathcal{M}_{cpd_\tau}$ with τ determined by UFO_{cpd}.

■ 5.10 Empirical Evaluation

■ 5.10.1 Experimental methodology

Benchmarks. We performed empirical evaluation on benchmarks derived from re-design problems for real proteins provided by the Bruce Donald Lab at Duke University. To gradually increase difficulty, small problems with two mutable residues (with five to ten total residues) were incrementally enlarged by making more of the residues mutable. Experiments were performed on an initially created "Expanded" problem set consisting of 12 problems with 3 mutable residues, then also additionally set of 32 problems expanded to have 4 mutable residues, and a set of 18 problems expanded to have 5 mutable residues. The names of the problems from the "Expanded" set correspond to the design number of the instance obtained from the Donald Lab. The names of problems from the the latter two sets of created benchmarks are shown with three parts: d[g]-[M]-[p] (e.g., d27-4-1), where [g] represents the problem design number as obtained from the Donald Lab, [M] indicates the number of mutable residues after enlarging, and [p] is an integer representing an index into the possible $\binom{n}{M}$ combinations of M mutable residues chosen from n residues available for redesign. The resulting conformation spaces for these problems ranged from on the order of 10^6 for 3 mutable residues to 10^{11} for 5 mutable residues.

Algorithms. We experimented with 6 algorithms: `AOBB-K*`; `AOBB-K*-`ω using weighted search (Section 5.7); `AOBB-K*-b` (boosted) with an improved `wMBE-K*-b` heuristic and search enhancements (Section 5.6); `AOBB-K*-b-DH`, which is `AOBB-K*-b` with dynamic heuristics (Section 5.8); `AOBB-K*-b-UFO`, which is `AOBB-K*-b` empowered with a CPD-specific UFO scheme (Section 5.9); and `BBK*`, state-of-the-art best-first search algorithm in comprehensive CPD software OSPREY 3.0 [Ojewole et al., 2018, Hallen et al., 2018].

Each `AOBB-K*` algorithm was implemented in C++. `AOBB-K*-b-DH` dynamic heuristic re-computations were regulated with $maxDepth = 2$ and $dhThreshold = 10^{20} \cdot K^*_{wt}$, where K^*_{wt} is the wild-type K* value. The UFO scheme used by `AOBB-K*-b-UFO` performed binary search in log-space and decreased the resulting threshold with $\delta = 0.2$ (Algorithm 10: UFO-GT, line 13). Because the `AOBB-K*-b` algorithms use the `wMBE-K*-b` heuristic which does not guarantee bounds, they do not guarantee discovery of the optimal K* (i.e., they are not complete). Similarly, schemes empowered with UFO lose optimality guarantees.

`BBK*` is implemented in Java, was set to use rigid rotamers, and given a bound-tightness of 1×10^{-200}[*]. Despite the extremely small bound tightness parameter, `BBK*` still failed to yield an optimal solution in various instances and therefore cannot be considered as sound and complete.

Experiments were run on a 2.66 GHz processor, and given 4 GB of memory and a time limit of 1hr for each problem. As `BBK*` can take advantage of parallelism, it was given access to 4 CPU cores.

Domain Sizes. In all problems, each mutable residues considers ~21 different amino acid assignments. Conformation variables of non-mutable residues have a domain size of ~2-14 rotamers (most having domain sizes ~4-9). Conformation variables of mutable residues have a domain size of ~34-35 when formulated as F1 and ~203-205 when formulated as F2.

5.10.2 Results

Below are common keys to the tables presented in the results:

Table Keys. **F**: formulation type, $\boldsymbol{\omega}$: weight applied to the heuristic for weighted search, **iB** or **i**: the i-bound used for the heuristic generation, **w***: induced width, **d**: pseudo tree depth, $|\mathbf{R}|$: number of mutable residues (i.e., variables to be maximized over), $|\mathbf{X}|$: total number

[*]`BBK*`'s bound tightness parameter does not correlate directly with an ω-approximation. See Ojewole et al. [2018].

of variables, $\mathbf{D}^{\mathbf{max}}$: maximum domain size, **UB**: heuristic upper bound at the root (empty cells representing no finite bound), **pre-t**: pre-processing time (ex. compiling heuristics), **search**: search time, **time**: total time, $\mathbf{K}^{*}$ or **Soln**: the returned K^{*} solution in log_{10} (for `AOBB-K*-b-UFO`, the solution was recomputed using the native, not underflowed, model), **wt** $\mathbf{K}^{*}$: the wild-type sequence K^{*} value (in log_{10}), and **Anytime**: the time it took to find the best solution found. The favorable values are printed in green or blue. Unfavorable values are printed in red.

Table 5.1: F1 vs F2 on original and Expanded(*) problems. Times for F1 are in black, and times for F2 are in green.

AOBB-K*										
benchmark	F	iB	|R|	|X|	D^{max}	w*	d	UB	pre-t	time
1a0r_00031	F1	3	2	16	34	8	8		0.9	226.9
	F2	3	2	16	203	6	8		0.7	**93.0**
1gwc_00021	F1	6	2	12	34	6	6	10.28	69.2	101.2
	F2	4	2	12	203	4	6	10.29	6.8	**15.7**
1gwc_00033	F1	3	2	18	35	9	9		1.0	30.2
	F2	3	2	18	203	7	9		1.3	**12.3**
2hnu_00026	F1	6	2	14	34	7	7	15.18	14.6	25.0
	F2	4	2	14	203	5	7	15.08	1.7	**7.3**
2rfe_00012*	F1	3	3	15	34	8	8		2.9	24.5
	F2	4	3	15	205	5	8	14.80	58.3	**60.9**
2xgy_00020*	F1	6	3	15	35	8	8	12.28	122.0	775.2
	F2	5	3	15	203	5	8	11.39	81.8	**360.7**
3u7y_00011*	F1	5	3	13	34	7	7		12.9	45.1
	F2	4	3	13	203	4	7	12.29	74.0	**76.1**
4wwi_00019*	F1	6	3	15	34	8	8	16.93	119.6	1494.9
	F2	5	3	15	203	5	8	16.05	169.2	**181.3**

Formulation 1 vs. Formulation 2. Table 5.1 shows results from both formulations on problems that were able to be solved exactly. The first four benchmarks shown are original small problems with 2 mutable residues obtained form the Donald Lab. The latter four problems are "Extended" problems with 3 mutable residues. Since the results are exact, we compare performance by the computation time. We see that F2 is generally superior to F1. Recall that F1 uses an indexing scheme between amino-acids and their rotamers, which implies that all functions need to include *both* residue and conformation variables thus leading to densely connected graphs. That being said, a strength of F1 vs. F2 is its

Table 5.2: `AOBB-K*` ($\omega = 1$) vs. Weighted `AOBB-K*`-ω ($\omega = 0.001$) on Expanded F2 problems. Times for $\omega = 1$ are in black, and times for $\omega = 0.001$ are in green. Red K* indicates a suboptimal solution.

						(203 ≤ Dmax ≤ 206)			ω-AOBB-K*	
benchmark	ω	iB	w*	d	|X|	UB	pre-t	search	time	K*
1gwc_00021*	1	4	4	7	13	28.80	123.8	81.3	205.1	11.92
	0.001	4	4	7	13	28.80	124.3	12.1	136.4	
2hnv_00025*	1	4	6	9	17	42.32	109.8	44.0	153.8	16.18
	0.001	4	6	9	17	42.32	109.3	12.2	121.5	
2rf9_00013*	1	4	6	9	17	37.68	83.0	17.8	100.8	15.03
	0.001	4	6	9	17	37.68	82.8	1.6	84.4	
2rfe_00012*	1	4	5	8	15	34.07	58.3	2.6	60.9	13.93
	0.001	4	5	8	15	34.07	58.6	0.3	58.9	
2rfe_00014*	1	4	5	8	15	35.07	58.2	2.4	60.6	14.36
	0.001	4	5	8	15	35.07	58.2	1.3	59.4	
2rfe_00017*	1	5	5	8	15	26.39	166.8	167.8	334.6	10.86
	0.001	4	5	8	15	27.39	89.1	5.0	94.1	
2rfe_00030*	1	4	5	8	15	31.34	115.2	161.5	276.6	11.1
	0.001	4	5	8	15	31.34	101.4	0.5	101.9	10.9
2xgy_00020*	1	5	5	8	15	26.24	81.8	278.9	360.7	10.96
	0.001	4	5	8	15	27.24	60.4	8.2	68.6	
3u7y_00009*	1	4	4	7	13	11.41	62.6	36.8	99.5	4.51
	0.001	4	4	7	13	11.41	62.5	2.1	64.7	
3u7y_00011*	1	4	4	7	13	28.29	74.0	2.1	76.1	11.85
	0.001	4	4	7	13	28.29	83.4	0.1	83.5	
4wwi_00019*	1	5	5	8	15	36.96	169.2	12.1	181.3	14.99
	0.001	4	5	8	15	37.96	62.0	7.2	69.2	

smaller domain sizes as seen via column "D$^{\max}$" (35 for F1 vs. 205 for F2). The smaller domain sizes facilitates the use of higher i-bounds as compared with F2

Weighted Search. Each pair of rows in Table 5.2 compares `AOBB-K*` (denoted $\omega = 1$) and `AOBB-K*`-ω ($\omega = 0.001$) on all Expanded problems. The listed i-bound is the value leading to the best result for each algorithm. Using approximate search reduces the MAP search space and improves search time (see values in the column "search"), sometimes by more than a factor of ten (e.g., for 3u7y_prepped_00009* `AOBB-K*`-ω took 2.1 seconds during search compared to 36.8 seconds with `AOBB-K*`). For 2rfe_prepped_00017*, 2xgy_prepped_00020*, and 4wwi_prepped_00019*, we see the optimal i-bound is reduced from 5 to 4 when using weighted search, highlighting the increased pruning of the search space enabled by the weighted heuristic even when using a weaker (but more quickly computed) heuristic. (Even when using the same i-bound, weighted search still showed significant speed-ups solving these problems). For the majority of these problems `AOBB-K*`-ω still found the optimal K*– in the table, the

Table 5.3: Performance of `AOBB-K*-b` vs `AOBB-K*` on the "Expanded" benchmarks with 3 mutable residues. Displayed are the i-bound ("i") used by each, their respective best-found K^* value ("Soln"), their completion time ("Time"), and, as reference, the wild-type K^* value ("wt K^* ").

3 Mut	AOBB-K*-b				AOBB-K*		
Problem	i	Soln	Time	wt K*	i	Soln	Time
00007	4	**14.73**	**269.3**	14.08	-	-inf	t/o
00009	4	4.51	**79.9**	4.09	4	4.51	99.5
00011	4	11.85	102.2	11.75	4	11.85	**76.1**
00012	4	13.93	69.1	13.93	4	13.93	60.9
00013	4	15.03	101.9	13.25	4	15.03	100.8
00014	4	14.36	70.9	13.96	4	14.36	60.6
00017	4	10.86	**118.0**	10.52	5	10.86	334.6
00019	4	14.99	**77.6**	14.99	5	14.99	181.3
00020	4	10.96	**101.5**	10.60	5	10.96	360.7
00021	4	11.92	200.4	9.37	4	11.92	205.1
00025	4	16.18	168.6	10.74	4	16.18	153.8
00030	4	11.12	**154.3**	10.35	4	11.12	276.6

column "K^*" reports the output of the algorithms, and only for 2rfe_prepped_00030* does `AOBB-K*-`ω produce a slightly suboptimal solution).

Comparing `AOBB-K*-b` vs `AOBB-K*`. In Table 5.3 we examine performance of `AOBB-K*-b` (with tightened `wMBE-K*-b`) vs. `AOBB-K*` on the problems with three mutable residues. We compare solution quality and speed of the two algorithms. The i-bound of `AOBB-K*-b` was set to $i = 4$. For `AOBB-K*` we use the best performing i-bound as reported previously. We highlight in blue any better K^* solutions and any significantly faster completion times (20% or more improvement over the competing algorithm's time). The wild-type K^* value ("wt K^*") is also shown.

The highlighted blue times show `AOBB-K*-b` finishing significantly faster for half of the problems, as well as solving a problem that `AOBB-K*` could not solve. Finally, `AOBB-K*-b` was able to find optimal solutions for each of these problems, although it does not prove optimality.

Evaluating Dynamic Heuristics. Table 5.4 compares `AOBB-K*-b` with and without the dynamic heuristic scheme described in Section 5.8 on problems with 3 or 4 mutable residues for which both algorithms found optimal solutions within an hour. We compare the size of the explored search space between the two algorithms (counting the number of OR and AND nodes of the residue variables traversed) and highlight when there are differences.

We see that dynamic heuristic re-computation reduces the size of the traversed search space in the majority of problems. For example, for problem d18-4-2 AOBB-K*-DH searched 40 OR nodes and 56 AND nodes whereas AOBB-K*-b traversed 279 OR nodes 1214 AND nodes. In two cases (highlighted in red) dynamic heuristics cause an increase in the search space. This may occur when dynamic heuristic re-computation causes the K^* upper-estimate for a node to *increase*, resulting in less pruning. This can occur because the lower-estimate of the K^* denominator from wMBE-K*-b does not guarantee a lower-bound. Thus, tightening of the heuristic may cause this lower-estimate to decrease in comparison to the originally computed heuristic. With an even lower value in the denominator, the K^* upper-estimate now increases, and thus less pruning may occur.

We also see that in the majority of the problems AOBB-K*-DH had better time performance despite the naive implementation of dynamic heuristic re-computations. For example, AOBB-K*-DH took 598.38 seconds to solve problem d18-4-2 whereas AOBB-K*-b took 3488.56 seconds.

UFO Impact and Cross Comparisons. The culmination of our results, Table 5.5 compares the performance of AOBB-K*-b-UFO, AOBB-K*-b-DH, AOBB-K*-b, and BBK* on problems with three, four, and five mutable residues. The AOBB-K*-based algorithms are displayed in a top-down ranking per problem, with the best ranking algorithm placed at the top. Ranking is based first on the quality of K^* found and then by the speed at which their respective solution was first discovered (measured in seconds and denoted "Anytime", highlighting the anytime nature of AOBB-K* search). Large text highlights the value responsible for the algorithm's higher ranking, and blue color indicates that BBK* was outperformed.

From the rank-based ordering of the algorithms, the competitiveness of the UFO scheme is apparent, with AOBB-K*-b-UFO ranked higher than the other two AOBB-K* schemes for all

Table 5.4: Comparison of the explored search space by `AOBB-K*-b` with and without use of a dynamic heuristic. Displayed are the respective i-bounds ("i") used, best-found K* solutions ("Soln"), completion times ("Time"), and the size of the traversed AND/OR search space (number of residue OR and AND nodes).

M	Problem	Algorithm	i	Soln	Time	OR	AND
3	d11-3-1	AOBB-K*-b-DH	3	11.85	24.64	**3**	**15**
		AOBB-K*-b	3	11.85	60.99	58	197
		AOBB-K*-b	4	11.85	102.18	3	5
	d12-3-1	AOBB-K*-b-DH	3	13.93	22.06	**3**	**13**
		AOBB-K*-b	3	13.93	20.72	21	122
		AOBB-K*-b	4	13.93	69.05	3	4
	d14-3-1	AOBB-K*-b-DH	3	14.36	26.95	**3**	**16**
		AOBB-K*-b	3	14.36	21.92	25	132
		AOBB-K*-b	4	14.36	70.88	3	5
	d30-3-1	AOBB-K*-b-DH	3	11.12	54.02	**70**	**141**
		AOBB-K*-b	3	11.12	2019.77	254	3666
		AOBB-K*-b	4	11.12	154.28	22	25
4	d18-4-2	AOBB-K*-b-DH	3	16.58	598.38	**40**	**56**
		AOBB-K*-b	3	16.58	3488.56	279	1214
	d24-4-1	AOBB-K*-b-DH	3	12.96	407.78	**92**	**251**
		AOBB-K*-b	3	12.96	487.66	94	437
	d27-4-1	AOBB-K*-b-DH	3	15.55	405.89	57	137
		AOBB-K*-b	3	15.55	254.67	57	137
	d28-4-1	AOBB-K*-b-DH	3	15.27	37.78	9	**12**
		AOBB-K*-b	3	15.27	21.62	9	19
	d28-4-2	AOBB-K*-b-DH	3	15.27	576.98	93	230
		AOBB-K*-b	3	15.27	323.45	59	166
	d42-4-1	AOBB-K*-b-DH	3	22.65	2897.35	**18**	**61**
		AOBB-K*-b	3	22.65	3025.80	24	114
	d43-4-2	AOBB-K*-b-DH	3	18.04	483.55	346	476
		AOBB-K*-b	3	18.04	112.50	56	75

problems except d27-5-1 (for which it found the same K* value as the other algorithms and was slower than `AOBB-K*-b` by just 2.61 seconds). The frequency of blue coloring shows the algorithms' competitiveness against `BBK*` on problems having three and four mutable residues. For problems d7-4-2, d47-4-2, and d47-5-2 `AOBB-K*-b-UFO` found a better solution than all other algorithms, `BBK*` included.

On problems having 5 mutable residues the `AOBB-K*-b` schemes begin to struggle. This is likely due to the loss of bounds from the *boosted* modifications of `wMBE-K*-b` in conjunction with a low i-bound, heavy underflows, and longer message passing for these larger problems. Nevertheless, `AOBB-K*-UFO` is still able to find good solutions, more often better than `BBK*`

Table 5.5: Comparison of the `AOBB-K*-b-[UFO/DH]` schemes and `BBK*` on problems ranging from 3 to 5 mutable residues. Shown is the i-bound used, best-found K* solution (recomputed without underflow-thresholding), the time at which the best-found solution was first discovered ("Anytime"), and the completion time ("Time"). The wild-type K* solution is also shown.

		AOBB-K*-b-[DH/UFO]						BBK*	
M	Problem	Algorithm	i	Soln	Anytime	Time	wt K*	Soln	Time
3	d19-3-1	AOBB-K*-b-UFO	3	14.99	**6.15**	621.83	14.99	14.99	34.00
		AOBB-K*-b-DH	3	14.99	**11.31**	56.05	14.99	14.99	34.00
		AOBB-K*-b	4	14.99	75.99	76.00	14.99	14.99	34.00
	d20-3-1	AOBB-K*-b-UFO	3	10.96	**13.70**	480.77	10.60	10.96	1388.13
		AOBB-K*-b-DH	3	10.96	**39.67**	339.91	10.60	10.96	1388.13
		AOBB-K*-b	4	10.96	100.02	100.03	10.60	10.96	1388.13
	d21-3-1	AOBB-K*-b-UFO	3	**11.92**	**89.03**	628.59	9.37	11.72	551.27
		AOBB-K*-b-DH	3	**11.92**	**136.44**	1307.45	9.37	11.72	551.27
		AOBB-K*-b	4	**11.92**	**193.83**	196.52	9.37	11.72	551.27
	d25-3-1	AOBB-K*-b-UFO	3	**16.18**	**14.02**	64.82	10.74	13.65	880.46
		AOBB-K*-b-DH	3	**16.18**	**51.92**	80.22	10.74	13.65	880.46
		AOBB-K*-b	4	**16.18**	**166.74**	166.75	10.74	13.65	880.46
4	d7-4-2	AOBB-K*-b-UFO	3	**14.89**	3391.78	timeout	14.08	14.54	278.08
		AOBB-K*-b-DH	3	**14.49**	3543.27	timeout	14.08	14.54	278.08
		AOBB-K*-b	3	14.49	3293.62	timeout	14.08	14.54	278.08
	d13-4-1	AOBB-K*-b-UFO	3	15.03	**12.69**	1974.43	13.25	15.03	46.46
		AOBB-K*-b-DH	3	15.03	**22.05**	79.88	13.25	15.03	46.46
		AOBB-K*-b	4	15.03	165.48	timeout	13.25	15.03	46.46
	d17-4-1	AOBB-K*-b-UFO	3	**10.86**	**29.39**	timeout	10.52	10.80	89.94
		AOBB-K*-b	4	**10.86**	**657.54**	timeout	10.52	10.80	89.94
		AOBB-K*-b-DH	3	**10.86**	660.16	timeout	10.52	10.80	89.94
	d21-4-1	AOBB-K*-b-UFO	3	**11.92**	**196.30**	timeout	9.37	11.72	687.66
		AOBB-K*-b-DH	3	**11.92**	**614.88**	timeout	9.37	11.72	687.66
		AOBB-K*-b	4	11.72	264.92	timeout	9.37	11.72	687.66
	d43-4-1	AOBB-K*-b-UFO	3	**18.19**	**76.49**	484.69	18.04	18.18	119.88
		AOBB-K*-b-DH	3	**18.19**	**386.49**	timeout	18.04	18.18	119.88
		AOBB-K*-b	3	**18.19**	896.67	timeout	18.04	18.18	119.88
	d47-4-2	AOBB-K*-b-UFO	3	**22.87**	**72.53**	239.88	22.70	22.83	1339.15
		AOBB-K*-b	3	22.74	**130.95**	timeout	22.70	22.83	1339.15
		AOBB-K*-b-DH	3	22.74	140.66	timeout	22.70	22.83	1339.15
5	d7-5-1	AOBB-K*-b-UFO	3	**15.17**	1570.30	timeout	14.08	14.73	401.09
		AOBB-K*-b-DH	3	14.73	**57.91**	timeout	14.08	14.73	401.09
		AOBB-K*-b	3	14.73	62.53	timeout	14.08	14.73	401.09
	d7-5-3	AOBB-K*-b-UFO	3	**14.84**	891.90	timeout	14.08	15.60	205.56
		AOBB-K*-b	3	14.73	**67.53**	timeout	14.08	15.60	205.56
		AOBB-K*-b-DH	3	14.73	156.68	timeout	14.08	15.60	205.56
	d27-5-1	AOBB-K*-b	3	15.55	**274.30**	timeout	15.48	15.55	1270.65
		AOBB-K*-b-UFO	3	15.55	**276.91**	timeout	15.48	15.55	1270.65
		AOBB-K*-b-DH	3	15.55	321.02	timeout	15.48	15.55	1270.65
	d31-5-1	AOBB-K*-b-UFO	3	7.88	**22.35**	128.75	7.63	7.88	130.04
		AOBB-K*-b	3	7.88	**129.43**	timeout	7.63	7.88	130.04
		AOBB-K*-b-DH	3	7.88	145.63	timeout	7.63	7.88	130.04
	d47-5-1	AOBB-K*-b-UFO	3	**23.08**	2068.22	timeout	22.70	23.05	timeout
		AOBB-K*-b	3	22.74	**222.66**	timeout	22.70	23.05	timeout
		AOBB-K*-b-DH	3	22.74	241.88	timeout	22.70	23.05	timeout

(e.g., problems d7-5-1 and d47-5-1) than worse (e.g., only problem d7-5-3).

We also see the potential of the `AOBB-K*-DH` scheme: on many problems shown, `AOBB-K*-b-DH` performs better than `AOBB-K*-b`– sometimes providing a better solution (e.g., for problem d21-4-1 where it found a K^* of 11.92 as opposed to the K^* of 11.72 found by `AOBB-K*-b`), in other cases finding good solutions faster (e.g., for problem d13-4-1 where it solved the problem in 22.05 seconds as opposed to `AOBB-K*-b` which took 165.48 seconds).

Finally, although `AOBB-K*-b` generally ranked lower than other `AOBB-K*-b` variants, it keeps up with `BBK*` on problems with 4 mutable residues (previously out of range for `AOBB-K*`), and even finds acceptable solutions for some 5-mutable-residue problems.

▨ 5.10.3 Summary of Results

Casting the protein redesign problem of optimizing K^* to a graphical model allowed us to adapt an existing AND/OR branch-and-bound algorithm for MMAP to solve the K^*MAP task. Testing this new algorithm, `AOBB-K*`, we saw that our formulation called F2 tends to lend itself to better performance. Nevertheless, `AOBB-K*` did not scale well. The innovations of `AOBB-K*-`ω, `AOBB-K*-b`, `AOBB-K*-b-DH`, and `AOBB-K*-b-UFO` enabled search to solve harder problems, including up to five mutable residues. Our UFO scheme demonstrated particularly strong performance, and was competitive with `BBK*` on these problems. Analysis of `AOBB-K*-b-DH`'s explored search space shows its promise, but the current naive implementation showed limited gains on larger problems.

Determinism. A major factor that can lead to decreased performance on large problem for `AOBB-K*` schemes (particularly when using the F2 formulation) is that domain sizes increase significantly with an increasing number of mutable residues. This is because each conformation variable associated with a mutable residue needs to encode the rotamers for *all* of the possible amino acids being considered for its corresponding residue. For the problems that we experimented on, this changed the domain size of the conformation variables from tens to hundreds. This increase in domain size in turn restricts `wMBE-K*` to using lower i-bounds and correspondingly degrades its heuristic accuracy. To explore the potential of moving to more compact representations that could enable higher i-bounds, we evaluated determinism in `wMBE-K*-b`'s computed messages for `AOBB-K*-b-UFO`. For problems with 5 mutable residues, the largest tables generated by `wMBE-K*-b` often had a determinism ratio > 0.95 – namely 95% of the entries were zero. This insight adds motivation to shifting to representations that can take advantage of sparsity or other repeated patterns, such as relational representations.

5.11 Conclusion

In this chapter, we present a graphical model framework for formulating and solving the protein redesign task of optimizing an objective function called K^*. This new framework allows for the leveraging of recent powerful advancements in graphical model algorithms towards protein redesign.

We presented two distinct graphical model formulations for this task, F1 and F2, and after determining F2 to be more effective, we employed a wide range of ideas within the AND/OR search space framework for graphical models. Focusing specifically on recent advances in marginal MAP algorithms, we adapted an AND/OR branch-and-bound scheme for marginal MAP to address the K^* optimization task. The resulting algorithm, `AOBB-K*`, was then extended to several approximate variants to improve scalability.

The `AOBB-K*-`ω variant uses weighted heuristic search, allowing for more aggressive pruning of the search space and improving `AOBB-K*` performance. `AOBB-K*-b` (boosted) made several modifications to its search routine to find better solutions early and speed up search. A dynamic heuristic scheme, `AOBB-K*-DH`, was tested and showed promise. Lastly, a specialized version of UFO for use with CPD was incorporated into `AOBB-K*`-b as `AOBB-K*`-b-UFO and showed competitive performance against state-of-the-art `BBK*` on problems with up to 5 mutable residues.

These algorithms relied on a new heuristic for bounding K^*MAP, `wMBE-K*`, which is based on Weighted Mini-Bucket Elimination for bounding MMAP. Evaluation of Mini-Bucket Elimination properties inspired an additional variant, `wMBE-K*-b`, which sacrificed bound guarantees for tighter estimates and was used by the `AOBB-K*-b` variants.

With these new frameworks in place, we can now turn our directions towards further advancements. Some possible directions include: adapting richer implementations of dynamic heuristics [Lam et al., 2014], possibly incorporating look-ahead schemes [Lam et al., 2017]; extending K^* optimization to finding the n-best K^*'s such as approaches by Flerova et al. [2016], Ruffini et al. [2021]; to explore more compact representations of `wMBE-K*` that can take advantage of the high levels of determinism [Mateescu and Dechter, 2008, Larkin and Dechter, 2003] or other scalable heuristics [Lee et al., 2016]; to extend these algorithms by incorporating other state-of-the-art inference schemes, especially approximate schemes [Yanover and Weiss, 2002, Hurley et al., 2016, Lou et al., 2018a,b, Marinescu et al., 2019, 2018a,b]; and to apply these schemes to develop mutation targets for real-world biological problems.

Chapter 6

Conclusion

6.1 Summary

We conclude this dissertation having focused on advancing the field of graphical models from three directions.

First we presented several advancements to a recently developed Monte Carlo sampling method called Abstraction Sampling, which is an unbiased stratified importance sampling-like scheme that leverages abstractions (similar to stratification) to solve summation queries for graphical models. We introduced `ORAS` for performing Abstraction Sampling on classical OR trees, and then presented a significant advancement in `AOAS` that is designed for compact AND/OR search spaces. We gave theoretical analysis on both OR and AND/OR Abstraction Sampling properties, including variance reduction conditions and a proof of unbiasedness. We provided three classes of abstraction functions that guide Abstraction Sampling's stratification process, one based on a graph notion of "context"; a second that partitions nodes based on positive real number values associated with them; and lastly a purely randomized abstraction scheme. From within these classes, over twenty-four distinct abstraction functions were designed and tested, each with the ability to vary granularity of their abstractions. An extensive empirical evaluation on over 400 problems from five well known summation benchmarks demonstrated the properties of Abstraction Sampling and also showed its superiority in estimating summation queries compared to well-known methods such as Importance Sampling, Weighted Mini-Bucket Importance Sampling, IJGP-SampleSearch, and Dynamic Importance Sampling. Based on this study, we believe that AOAS is one of the best schemes for estimating the partition function to date, in particular when used with the (*equalDistQB3*, *equalDistQB4*), or RAND abstraction schemes.

We also presented a novel scheme called UFO for infusing artificial determinism into graphical models, empowering algorithms that leverage constraint processing. An example of such an algorithm is `AOBB` [Marinescu et al., 2018b], a branch-and-bound algorithm that can augmented to use constraint propagation to prune regions of the search space that are provably inconsistent. We derived theoretical properties of applying such artificial determinism, including showing lower-boundedness for common graphical model tasks. We evaluated `AOBB`-with-UFO performance problems used in the 2022 UAI Inference Competition which demonstrated that, at times, even simple schemes infused with UFO can outperform powerful solvers. And that UFO can sometimes lead to very fast solutions. In summary, UFO has the potential to be a good augmentation for some algorithms, such as to produce quick lower bounds or to speed up approximate search.

And finally in more applied research, we presented work adapting graphical model frameworks for Computational Protein Design. We focus on the automated redesign of proteins to maximize K*, an approximation of binding affinity. We provided two graphical model formulations for this redesign task and provided algorithms that operate over them. In particular, we provided a heuristic scheme called `wMBE-K*`, based on Weighted Mini-Bucket Elimination [Liu and Ihler, 2011b, Ihler et al., 2012], for the K* maximization task, and provide an array of anytime AND/OR depth-first branch-and-bound algorithms, the parent version of which is called `AOBB-K*`. Variants include augmentation with weighted heuristic search, the use of dynamic heuristics, and incorporating the `UFO` technique. Empirical analysis on real proteins show strong performance from these schemes compared to the state-of-the-art algorithm `BBK*` [Ojewole et al., 2018], which is used as part of a long-standing computational protein design software called OSPREY [Hallen et al., 2018].

■ 6.2 Future Directions

There are still a myriad of directions the work presented in this dissertation can be extended towards. We outline a few below.

Abstraction Sampling. An observation about Abstraction Sampling is that it shows varied performance on different benchmarks. This begs the question of how Abstraction Sampling can be tuned for specific benchmarks. In similar spirit, for some practical domain applications, there may be expert domain knowledge that can be used to guide abstractions. Current abstraction frameworks do not allow for this, and so such extensions could be useful. Furthermore, the current Abstraction Sampling framework is solely designed for discrete graphical models. The Abstraction Sampling framework can [in theory] be adapted for the continuous domain as well, and such an extension would be invaluable. Algorithmically, `AOAS` showed powerful performance. Nevertheless, its performance may be able to be further improved by combining the scheme with other well known schemes. For example, many discrete graphical model instances are such that a few configurations capture most of the model's mass. For such models, enumerating those configurations exactly and then sampling over the rest could prove to be a valuable strategy. Such amalgams may prove to be fruitful in enhancing the quality of estimates. Furthermore, it may be valuable to choose abstractions also based on the quality of estimates for node values, such as by incorporating notions of error in the heuristic estimates as in [Lam et al., 2017]. To speed up and drive towards a lower estimate, UFO can be infused with Abstraction Sampling as well. Finally, a user friendly API for wider use of Abstraction Sampling can provide access across the community.

UFO. Non-zero lower-bounds are often difficult to obtain when problems contain determinism. However, UFO provides a new method to achieve these bounds by adding more determinism through underflow-thresholding and solving these problems exactly. It could be valuable to empirically evaluate the speed and quality of such lower bounds compared to other schemes. Furthermore, an analysis on which domains UFO can be most fruitful, and why, can be instrumental in guiding its use. (For example, it already showed promise within the CPD domain). There can also be more principled and better thresholding schemes which could also be an interesting direction for future research.

AND/OR Computational Protein Design. One very natural direction that would be helpful to see this work extended to is the application of the provided framework for real biology tasks that include *in vivo* or *in vitro* analysis. However, in practice, it is rare that the optimal computationally derived biological designs are necessarily optimal (or even good) *in vitro* and especially rare for it being so *in vivo*. Thus, extending our framework to produce k-best solutions (such as in approaches by Flerova et al. [2016], Ruffini et al. [2021]) would allow for a greater chance of finding a good candidate. We can adapt richer implementations of dynamic heuristics [Lam et al., 2014] and consider look-ahead schemes [Lam et al., 2017] to help decide when to recompute them. In terms of problem representation, it could be beneficial to explore more compact representations of `wMBE-K*` that can take advantage of the high levels of determinism [Mateescu and Dechter, 2008, Larkin and Dechter, 2003] or other scalable heuristics [Lee et al., 2016], particularly given the framework's success with UFO. And finally, it makes sense to next extend this framework to incorporate other state-of-the-art inference schemes, especially approximate schemes [Yanover and Weiss, 2002, Hurley et al., 2016, Lou et al., 2018a,b, Marinescu et al., 2019, 2018a,b], and including Abstraction Sampling.

Bibliography

C. B. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181(4096):223–230, 1973. doi: 10.1126/science.181.4096.223. URL `https://www.science.org/doi/abs/10.1126/science.181.4096.223`.

F. Broka, R. Dechter, A. Ihler, and K. Kask. Abstraction sampling in graphical models. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 632–641, 2018. URL `http://auai.org/uai2018/proceedings/papers/234.pdf`.

L. Cao, I. Goreshnik, B. Coventry, J. B. Case, L. Miller, L. Kozodoy, R. E. Chen, L. Carter, A. C. Walls, Y.-J. Park, E.-M. Strauch, L. Stewart, M. S. Diamond, D. Veesler, and D. Baker. De novo design of picomolar sars-cov-2 miniprotein inhibitors. *Science*, 370 (6515):426–431, 2020. doi: 10.1126/science.abd9909. URL `https://www.science.org/doi/abs/10.1126/science.abd9909`.

P.-C. Chen. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM Journal on Computing*, 21:295–315, 1992.

E. Danna, E. Rothberg, and C. L. Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102:71–90, 2005.

A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

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

R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

R. Dechter. *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013. doi: 10.2200/S00529ED1V01Y201308AIM023. URL `http://dx.doi.org/10.2200/S00529ED1V01Y201308AIM023`.

R. Dechter. Reasoning with probabilistic and deterministic graphical models: Exact algorithms, second edition. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 13:1–199, 02 2019.

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

R. Dechter and I. Rish. Mini-buckets: A general scheme for approximating inference. *Journal of the ACM*, pages 107–153, 2002.

R. Dechter and I. Rish. Mini-buckets: A general scheme for bounded inference. *J. ACM*, 50(2):107–153, 2003. doi: 10.1145/636865.636866. URL `http://doi.acm.org/10.1145/636865.636866`.

S. Desarkar, P. Chakrabarti, and S. Ghose. Admissibility of AO* when heuristics overestimate. *Artificial Intelligence*, 34(1):97–113, 1987.

N. Eén and N. Sörensson. An extensible sat-solver. In E. Giunchiglia and A. Tacchella, editors, *Theory and Applications of Satisfiability Testing*, pages 502–518, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

N. Flerova, A. Ihler, R. Dechter, and L. Otten. Mini-bucket elimination with moment matching. In *Workshop on Discrete Optimization in Machine Learning (DISCML) at NIPS*, 2011.

N. Flerova, R. Marinescu, and R. Dechter. Evaluating weighted DFS branch and bound over graphical models. In S. Edelkamp and R. Barták, editors, *Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014*. AAAI Press, 2014. URL `http://www.aaai.org/ocs/index.php/SOCS/SOCS14/paper/view/8937`.

N. Flerova, R. Marinescu, and R. Dechter. Searching for the M best solutions in graphical models. *J. Artif. Intell. Res.*, 55:889–952, 2016.

H. Frauenfelder, S. G. Sligar, and P. G. Wolynes. The energy landscapes and motions of proteins. *Science*, 254(5038):1598–1603, 1991.

S. D. Givry. toulbar2, an exact cost function network solver. In *24ème édition du congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision ROADEF 2023*, Rennes, France, Feb. 2023. URL `https://hal.science/hal-04021879`.

V. Gogate and R. Dechter. Samplesearch: Importance sampling in presence of determinism. *Artif. Intell.*, 175(2):694–729, 2011. doi: 10.1016/j.artint.2010.10.009. URL `https://doi.org/10.1016/j.artint.2010.10.009`.

M. Hallen, J. Martin, A. Ojewole, J. Jou, A. Lowegard, M. Frenkel, P. Gainza, H. Nisonoff, A. Mukund, S. Wang, G. Holt, D. Zhou, E. Dowd, and B. Donald. Osprey 3.0: Open-source protein redesign for you, with powerful new features. *Journal of Computational Chemistry*, 39, 10 2018.

M. A. Hallen and B. R. Donald. Protein design by provable algorithms. *Commun. ACM*, 62 (10):76–84, sep 2019.

G. Hardy, J. Littlewood, and G. Pólya. *Inequalities.* Cambridge Mathematical Library. Cambridge University Press, 1988.

B. Hurley, B. O'sullivan, D. Allouche, G. Katsirelos, T. Schiex, M. Zytnicki, and S. d. Givry. Multi-language evaluation of exact solvers in graphical model discrete optimization. *Constraints*, 21:413–434, 2016.

A. Ihler, N. Flerova, R. Dechter, and L. Otten. Join-graph based cost-shifting schemes. In *UAI*, pages 397–406, 2012.

H. Kahn and A. W. Marshall. Methods of reducing sample size in monte carlo computations. *Journal of the Operations Research Society of America*, 1(5):263–278, 1953. ISSN 00963984. URL `http://www.jstor.org/stable/166789`.

D. Knuth. Estimating the efficiency of backtracking algorithms. *Math. Comput.*, 29:1121–136, 1975.

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

I. V. Korendovych and W. F. DeGrado. De novo protein design, a retrospective. *Quarterly reviews of biophysics*, 53:e3, 2020.

B. Kuhlman and D. Baker. Native protein sequences are close to optimal for their structures. *Proceedings of the National Academy of Sciences*, 97(19):10383–10388, 2000. doi: 10.1073/pnas.97.19.10383. URL `https://www.pnas.org/doi/abs/10.1073/pnas.97.19.10383`.

W. Lam, K. Kask, R. Dechter, and A. Ihler. Beyond static mini-bucket: Towards integrating with iterative cost-shifting based dynamic heuristics. In *Seventh Annual Symposium on Combinatorial Search*, 2014.

W. Lam, K. Kask, J. Larrosa, and R. Dechter. Residual-guided look-ahead in AND/OR search for graphical models. *J. Artif. Intell. Res.*, 60:287–346, 2017. doi: 10.1613/JAIR.5475. URL `https://doi.org/10.1613/jair.5475`.

G. N. Lance and W. T. Williams. A General Theory of Classificatory Sorting Strategies: 1. Hierarchical Systems. *The Computer Journal*, 9(4):373–380, 02 1967. ISSN 0010-4620. doi: 10.1093/comjnl/9.4.373. URL `https://doi.org/10.1093/comjnl/9.4.373`.

D. Larkin and R. Dechter. Bayesian inference in the presence of determinism. *AI and Statistics(AISTAT03)*, 2003.

J. Lee, R. Marinescu, R. Dechter, and A. Ihler. From exact to anytime solutions for marginal map. AAAI'16, page 3255–3262. AAAI Press, 2016.

L. H. S. Lelis, S. Zilles, and R. C. Holte. Predicting the size of ida*'s search tree. *Artif. Intell.*, 196:53–76, 2013. doi: 10.1016/j.artint.2013.01.001. URL `http://dx.doi.org/10.1016/j.artint.2013.01.001`.

L. H. S. Lelis, L. Otten, and R. Dechter. Memory-efficient tree size prediction for depth-first search in graphical models. In B. O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 481–496. Springer, 2014. ISBN 978-3-319-10427-0. doi: 10.1007/978-3-319-10428-7_36. URL `http://dx.doi.org/10.1007/978-3-319-10428-7`.

Q. Liu and A. Ihler. Bounding the partition function using holder's inequality. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 849–856, 2011a.

Q. Liu and A. Ihler. Bounding the partition function using Hölder's inequality. In *International Conference on Machine Learning (ICML)*, pages 849–856. ACM, June 2011b.

Q. Liu, J. W. Fisher III, and A. Ihler. Probabilistic variational bounds for graphical models. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1432–1440. Curran Associates, Inc., 2015.

Q. Lou, R. Dechter, and A. Ihler. Anytime anyspace and/or best-first search for bounding marginal map. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018a.

Q. Lou, R. Dechter, and A. Ihler. Finite-sample bounds for marginal MAP. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 725–734. AUAI Press, 2018b.

Q. Lou, R. Dechter, and A. Ihler. Interleave variational optimization with monte carlo sampling: A tale of two approximate inference paradigms. 2019.

R. Marinescu and R. Dechter. And/or branch-and-bound search for combinatorial optimization in graphical models. *Artif. Intell.*, 173(16-17):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 and R. Dechter. AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16-17):1457–1491, 2009c.

R. Marinescu, R. Dechter, and A. Ihler. And/or search for marginal map. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, UAI'14, page 563–572. AUAI Press, 2014.

R. Marinescu, R. Dechter, and A. Ihler. Stochastic anytime search for bounding marginal map. In *IJCAI*, pages 5074–5081, 2018a.

R. Marinescu, J. Lee, R. Dechter, and A. Ihler. And/or search for marginal map. *J. Artif. Int. Res.*, 63(1):875–921, sep 2018b.

R. Marinescu, A. Kishimoto, A. Botea, R. Dechter, and A. Ihler. Anytime recursive best-first search for bounding marginal map. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):7924–7932, Jul. 2019.

R. Mateescu and R. Dechter. The relationship between and/or search and variable elimination. pages 380–387, 01 2005.

R. Mateescu and R. Dechter. Mixed deterministic and probabilistic networks. *Annals of mathematics and artificial intelligence*, 54(1):3–51, 2008.

D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

B. Neveu and G. Trombettoni. Incop: An open library for incomplete combinatorial optimization. In F. Rossi, editor, *Principles and Practice of Constraint Programming – CP 2003*, pages 909–913, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-45193-8.

A. Ojewole, J. D. Jou, V. G. Fowler, and B. R. Donald. *BBK* (Branch and Bound Over K*):* A provable and efficient ensemble-based protein design algorithm to optimize stability and binding affinity over large sequence spaces. *J. Comput. Biol.*, 25(7):726–739, 2018.

L. Otten and R. Dechter. Anytime and/or depth-first search for combinatorial optimization. In *SOCS*, 2011.

A. Ouali, D. Allouche, S. de Givry, S. Loudni, Y. Lebbah, L. Loukil, and P. Boizumault. Variable neighborhood search for graphical model energy minimization. *Artificial Intelligence*, 278:103194, 2020. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2019.103194. URL `https://www.sciencedirect.com/science/article/pii/S0004370218305927`.

A. B. Owen. *Monte Carlo theory, methods and examples.* `https://artowen.su.domains/mc/`, 2013.

J. Pearl. *Heuristics: Intelligent Search Strategies.* Addison-Wesley, 1984.

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

I. Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3-4): 193–204, 1970.

M. L. Rizzo. *Statistical computing with R.* Chapman & Hall/CRC, 2007.

R. Y. Rubinstein and D. P. Kroese. *Simulation and the Monte Carlo Method (Wiley Series in Probability and Statistics).* 2 edition, 2007. ISBN 0470177942, 9780470177945.

R. Y. Rubinstein and D. P. Kroese. *Simulation and the Monte Carlo Method.* Wiley Publishing, 3rd edition, 2016. ISBN 1118632168.

M. Ruffini, J. Vucinic, S. de Givry, G. Katsirelos, S. Barbe, and T. Schiex. Guaranteed diversity and optimality in cost function network based computational protein design methods. *Algorithms*, 14(6), 2021.

C. Viricel, S. de Givry, T. Schiex, and S. Barbe. Cost function network-based design of protein-protein interactions: predicting changes in binding affinity. *Bioinformatics (Oxford, England)*, 34, 02 2018.

J. Vucinic, D. Simoncini, M. Ruffini, S. Barbe, and T. Schiex. Positive multistate protein design. *Bioinformatics (Oxford, England)*, 36, 06 2019.

J. H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963. ISSN 01621459. URL `http://www.jstor.org/stable/2282967`.

J. L. Watson, D. Juergens, N. R. Bennett, B. L. Trippe, J. Yim, H. E. Eisenach, W. Ahern, A. J. Borst, R. J. Ragotte, L. F. Milles, et al. De novo design of protein structure and function with rfdiffusion. *Nature*, 620(7976):1089–1100, 2023.

B. Wemmenhove, J. M. Mooij, W. Wiegerinck, M. Leisink, H. J. Kappen, and J. P. Neijt. Inference in the promedas medical expert system. In R. Bellazzi, A. Abu-Hanna, and J. Hunter, editors, *Artificial Intelligence in Medicine*, pages 456–460, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

C. Yanover and Y. Weiss. Approximate inference and protein-folding. *Advances in neural information processing systems*, 15, 2002.

Y. Zhou, Y. Wu, and J. Zeng. Computational protein design using and/or branch-and-bound search. *Journal of computational biology : a journal of computational molecular cell biology*, 23, 05 2016.