UNIVERSITY OF CALIFORNIA,

IRVINE

AND/OR Search Spaces for Graphical Models

DISSERTATION

submitted in partial satisfaction of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Science

by

Robert Eugeniu Mateescu

Dissertation Committee:

Professor Rina Dechter, Chair

Professor Padhraic Smyth

Professor Sandra Irani

2007

The dissertation of Robert Eugeniu Mateescu

is approved and is acceptable in quality and form for

publication on microfilm and in digital formats:

Committee Chair

University of California, Irvine

2007

DEDICATION

To my parents, Margareta and Victor

To my wife, Cipriana

To my daughter, Andreea

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am indebted to all those who have helped me finish this dissertation: my advisor, committee members, colleagues, friends and family.

My gratitude goes towards my advisor, Professor Rina Dechter, to whom I owe the greatest part of what I am as a researcher. I consider myself very fortunate to have had the chance of such a close collaboration with her. She guided my first steps in research, introduced me to the realm of sophisticated algorithms for automated reasoning, but at the same time taught me to never lose sight of the bigger picture, and always build upon first principles. I thank her both for the freedom to pursue my own ideas, that I very much appreciated, and for her keen criticism that constantly pushed me to raise my own standards. I have benefited enormously from the innumerable hours of discussions and revisions to our papers, and am very grateful for her patience in times when I needed to find my path. Thank you Rina, for everything.

My warmest thanks to my committee members, Professor Padhraic Smyth and Professor Sandy Irani, for their comments on my research, on preliminary drafts of the dissertation, and for their help and encouragement.

My graduate school work was generously supported by the NSF grants IIS-0086529 and IIS-0412854, by the MURI ONR award N00014-00-1-0617 and by the Donald Bren School of Information and Computer Science at University of California, Irvine.

It has been a joy to be a member of our research group. I want to thank my colleagues Bozhena Bidyuk, Radu Marinescu, Vibhav Gogate, Kalev Kask and Lars Otten for the uncountable hours of research discussions. I am grateful to Radu and Kalev in particular, for contributing to the experimental evaluation of some of the algorithms that I have designed and studied.

For having embraced me and my family with so much love from the moment we came to California, I would like to thank Rev. Fr. Cornel Avramescu and Mrs. Eugenia Avramescu. They made our adjustment to a new home much easier, and provided us the much needed friendship and support during the more difficult times.

My close family is most important to me, and they have influenced and contributed to my achievements immensely. My parents, Margareta and Victor, were always in my heart, and gave me the courage and confidence to take greater challenges. They taught me the simple and important things in life, that I will carry with me wherever I go, and most importantly believed in me, and in my power to follow my dreams. Their emotional and financial support were invaluable, and they were always ready to mitigate our worries. Words are not enough to thank them, I only wish I could be a parent like them.

I thank my wonderful wife, Cipriana, my love and my best friend ever since we met. We shared our years in graduate school, yet she always found unbelievable resources and energy to support, encourage and give me the power to continue. Thank you Cipi, for giving so much more meaning to my life and work. I am very grateful to my wife's parents, Rodica and Constantin, for their love and for caring so tenderly for our daughter. Their invaluable help has given me the precious time to work on my research and finish this dissertation.

I want to thank my brother, Eduard, for being my inspiration to come to graduate school in the US. He generously shared his knowledge with me, and made my transition to California much easier.

Above all, I cannot overestimate the influence my daughter, Andreea, has on all my life. Her smile and laughter were enough to lift me up even during the toughest times. I promise to make up for the story time that we missed in the late nights when I was wrapping up this dissertation.

CURRICULUM VITAE

Robert Eugeniu Mateescu

EDUCATION

Ph.D. Information and Computer Science, 2007
Donald Bren School of Information and Computer Science
University of California Irvine
Dissertation: AND/OR Search Spaces for Graphical Models
Advisor: Rina Dechter.

M.S. Information and Computer Science, 2003
Donald Bren School of Information and Computer Science
University of California Irvine

B.S. Computer Science and Mathematics, 1997
University of Bucharest, Romania

PUBLICATIONS

[1] Rina Dechter and Robert Mateescu. AND/OR Search Spaces for Graphical Models. *Artificial Intelligence* 171(2-3):73-106, 2007.

[2] Robert Mateescu and Rina Dechter. AND/OR Multi-Valued Decision Diagrams (AOMDDs) for Weighted Graphical Models. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence (UAI'07)*, 2007.

[3] Robert Mateescu and Rina Dechter. A Comparison of Time-Space Schemes for Graphical Models. In *Proceedings of the Twentieth International Joint Conferences on Artificial Intelligence (IJCAI'07)*, pages 2346-2352, 2007.

[4] Robert Mateescu and Rina Dechter. Compiling Constraint Networks into AND/OR Multi-Valued Decision Diagrams (AOMDDs). In *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP'06)*, pages 731–736, 2006.

[5] Robert Mateescu and Rina Dechter. The Relationship Between AND/OR Search and Variable Elimination. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI'05)*, pages 380–387, 2005.

[6] Robert Mateescu and Rina Dechter. AND/OR Cutset Conditioning. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 230–235, 2005.

[7] Rina Dechter and Robert Mateescu. The Impact of AND/OR Search Spaces on Constraint Satisfaction and Counting. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP'04)*, pages 731–736, 2004.

[8] Rina Dechter and Robert Mateescu. Mixtures of Deterministic-Probabilistic Networks and their AND/OR Search Space. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI'04)*, pages 120–129, 2004.

[9] Rina Dechter and Robert Mateescu. A Simple Insight into Iterative Belief Propagation's Success. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI'03)*, pages 175–183, 2003.

[10] Rina Dechter, Robert Mateescu and Kalev Kask. Iterative Join-Graph Propagation. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI'02)*, pages 128–136, 2002.

[11] Robert Mateescu, Rina Dechter and Kalev Kask. Tree Approximation for Belief Updating. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, pages 553–559, 2002.

ABSTRACT OF THE DISSERTATION

AND/OR Search Spaces for Graphical Models

By

Robert Eugeniu Mateescu

Doctor of Philosophy in Information and Computer Science

University of California, Irvine, 2007

Professor Rina Dechter, Chair

Graphical models are a widely used knowledge representation framework that captures independencies in the data and allows for a concise representation. Well known examples of graphical models include Bayesian networks, constraint networks, Markov random fields and influence diagrams. Graphical models are applicable to diverse domains such as planning, scheduling, design, diagnosis and decision making.

This dissertation is concerned with graphical model algorithms that leverage the structure of the problem. We investigate techniques that capitalize on the independencies expressed by the model's graph by decomposing the problem into independent components, resulting in often exponentially reduced computational costs.

The algorithms that we develop can be characterized along three main dimensions: (1) search vs. dynamic programming methods; (2) deterministic vs. probabilistic information; (3) approximate vs. exact algorithms.

We introduce the AND/OR search space perspective for graphical models. In contrast to the traditional OR search, the new AND/OR search is sensitive to problem decomposition. The AND/OR search tree search is in most cases exponentially smaller (and never larger) than the OR search tree. The AND/OR search graph is exponential in the treewidth of the graph, while the OR search graph is exponential in the pathwidth.

We introduce *mixed networks*, a new graphical model framework that combines belief and constraint networks. By keeping the two types of information separate we are able to more efficiently exploit them by specific methods. We describe the primary algorithms for processing such networks, based on inference and on AND/OR search.

In terms of approximate algorithms, we investigate message-passing schemes based on join tree clustering and belief propagation. We introduce Mini-Clustering (MC), which performs bounded inference on a tree decomposition. We then combine MC with the iterative version of Pearl's belief propagation (IBP), creating Iterative Join-Graph Propagation (IJGP). We show empirically that IJGP is one of the most powerful approximate schemes for belief networks. Through analogy with arc consistency algorithms from constraint networks, we show that IBP and IJGP infer zero-beliefs correctly, and empirically show that this also extends to extreme beliefs.

We apply the AND/OR paradigm to cutset conditioning and show that the new method is a strict improvement, often yielding exponential savings. The AND/OR cutset is the inspiration of a new caching scheme for AND/OR search, which led to the design of our most powerful and flexible algorithm *AND/OR Adaptive Caching*.

Furthermore we make a comparison of AND/OR search and inference methods. We analyze them side by side by describing the context minimal graph that they traverse. We also investigate three hybrid schemes, based on search and inference and show that Adaptive Caching is never worse than the other two.

Finally, we apply the AND/OR perspective to decision diagrams. We extend them with AND nodes capturing function structure decomposition, resulting in AND/OR Multi-Valued Decision Diagrams (AOMDDs). The AOMDD is a canonical form that compiles a graphical model and has size bounded exponentially by the treewidth, rather than pathwidth (as is the case for OR decision diagrams). We present two compilation algorithms, one based on AND/OR search, the other based on a Variable Elimination schedule.

Chapter 1

Introduction

Graphs are one of the fundamental concepts in mathematics, computer and information sciences. They exist in different flavors, but the basic definition includes a set of vertices and a set of edges (directed or undirected) between pairs of vertices. Due to this simple definition, graphs are a convenient and natural way of representing the relationships (as edges) between objects (vertices), and they can express a wide range of processes and systems. For example, individuals in a community may be represented by vertices and some relationships between them by edges. A complex product could have its modules represented by vertices and the interactions between them by edges. We may be interested in modeling events, and we could represent them by vertices, and the causal links between them by (directed) edges. In medicine, diseases and symptoms can be modeled by vertices, and edges would express the appropriate connections between them. The World Wide Web is another example of a graph, where pages are vertices and a link from one page to another is a directed edge. The examples could continue, but the previous should give at least a glimpse of the power of abstraction of graphs.

Modeling real-life decision problems requires the specification and reasoning with probabilistic and deterministic information. Graphical models are a widely used knowledge representation framework that captures independencies in the data and allows for a concise representation. Essential to a graphical model is the underlying graph that captures the problem structure. The vertices are the variables of interest, and the edges represent the interactions between them. Some well known examples include Bayesian (or belief) networks, constraint networks, Markov random fields, influence diagrams etc. There are numerous examples of problems defined as graphical models, including design, scheduling, planning, diagnosis, decision making or genetic linkage analysis.

Graphical models are the representation of choice for many problems because they are a simple abstract mathematical model, and there exist many algorithms for solving different tasks on them. Given a decision (or reasoning) problem, one can: (1) model it as a graphical model; (2) apply some specific algorithm to solve it and then (3) interpret the results in terms of the original problem. Modeling a problem (part 1), is an interesting and important process in itself, but in most cases it is based on expert knowledge of the problem at hand, or learned from data. Our work and the research presented in this dissertation is concerned with the algorithms (part 2) that can be applied to a graphical model. Owing to the power of abstraction, an efficient algorithm for graphical models is immediately applicable to many different types of problems (e.g., constraint satisfaction, belief updating, optimization).

The research presented in this dissertation is concerned with the study of graphical model algorithms along three different dimensions:

1. *search* vs. *dynamic programming* methods;

2. *deterministic* vs. *probabilistic* information;

3. *approximate* vs. *exact* algorithms.

The following section will describe the outline of the dissertation and the contributions. The rest of this chapter contains preliminary definitions and gives examples of graphical models.

1.1 Dissertation Outline and Contributions

We describe here the structure of the dissertation, and the following sections will describe in more detail the main results and contributions that appear in each chapter.

1. Chapter 2 introduces the AND/OR search space perspective for graphical models. In contrast to the traditional (OR) search space view, the AND/OR search tree explicitly displays some of the independencies that are present in the graphical model and may sometimes reduce the search space exponentially. For memory intensive search, familiar parameters such as the depth of a spanning tree, treewidth and pathwidth are shown to play a key role in characterizing the effect of AND/OR search graphs vs. the traditional OR search graphs.

2. Chapter 3 introduces *mixed networks*, a new framework for expressing and reasoning with *probabilistic* and *deterministic* information. The framework combines belief networks and constraint networks, and we define its semantics and its graphical representation and outline the primary algorithms for processing such networks.

3. Chapter 4 investigates approximate message-passing algorithms for mixed networks. We study the advantages of bounded inference provided by anytime schemes such as Mini-Clustering (MC), and combine them with the virtues of iterative algorithms such as Iterative Belief Propagation (IBP). Our resulting hybrid algorithm Iterative Join-Graph Propagation (IJGP) is shown empirically to surpass the performance of both MC and IBP on several classes of networks. Although there is still little understanding of why or when IBP works well, it exhibits tremendous performance on different classes of problems, most notably coding and satisfiability problems. We investigate the iterative algorithms for Bayesian networks by making connections with well known constraint processing algorithms, and this helps explain when IBP infers extreme beliefs correctly.

4. Chapter 5 describes the *AND/OR cutset conditioning*, which is an application of the AND/OR paradigm to the method of cutset conditioning. Cutset conditioning is one of the methods of solving reasoning tasks for graphical models, especially when space restrictions make inference (e.g., jointree-clustering) algorithms infeasible. The *w-cutset* is a natural extension of the method to a hybrid algorithm that performs search on the conditioning variables and inference on the remaining problems of induced width bounded by w. We take a fresh look at these methods through the spectrum of AND/OR search spaces for graphical models. The resulting *AND/OR cutset method* is a strict improvement over the traditional one, often by exponential amounts.

5. Chapter 6 compares search and inference in graphical models through the new framework of AND/OR search. Specifically, we compare Variable Elimination (VE) and memory-intensive AND/OR Search (AO) and place algorithms such as graph-based backjumping and no-good and good learning within the AND/OR search framework. We also compare three parameterized algorithmic schemes for graphical models that can accommodate trade-offs between time and space: 1) AND/OR Cutset Conditioning (**AOC(i)**); 2) Variable Elimination with Conditioning (**VEC(i)**); and 3) Tree Decomposition with Conditioning (**TDC(i)**). We show that **AOC(i)** can simulate any execution of the other two schemes, and thus is at least as good as them.

6. Chapter 7 describes how to augment Multi-Valued Decision Diagrams with AND nodes, in order to capture function decomposition structure and to extend these compiled data structures to general weighted graphical models (e.g., probabilistic models). We present the *AND/OR multi-valued decision diagram* (AOMDD) which compiles a graphical model into a canonical form that supports polynomial (e.g., solution counting, belief updating) or constant time (e.g. equivalence of graphical models) queries. We provide two compilation algorithms for AOMDDs. The first is

based on AND/OR search and the subsequent reduction of the traversed context minimal graph. The second is based on a Bucket Elimination schedule to combine the AOMDDs of the original functions. The algorithm uses the APPLY operator which combines two AOMDDs by a given operation. For both approaches, the complexity of the compilation time and the size of the AOMDD are bounded exponentially by the *treewidth* of the graphical model, rather than the *pathwidth,* as is known for ordered binary decision diagrams (OBDDs). We also introduce the concept of *semantic treewidth,* which helps explain why the size of a decision diagram is often much smaller than the worst case bound.

7. Chapter 8 briefly presents the software implementation of the algorithms described in the dissertation, and possible future directions. Chapter 9 concludes.

1.1.1 AND/OR Search Spaces for Graphical Models (Chapter 2)

Search-based algorithms (*e.g.,* depth-first branch-and-bound, best-first search) traverse the search space of the model, where each path represents a partial or full solution. The linear structure of search spaces does not retain the independencies represented in the underlying graphical models and, therefore, search-based algorithms may not be nearly as effective as inference-based algorithms in using this information. On the other hand, the space requirements of search-based algorithms may be much less severe than those of inference-based algorithms and they can accommodate a wide spectrum of space-bounded algorithms, from linear space to treewidth bounded space. In addition, search methods require only an implicit, generative, specification of the functional relationship (given in a procedural or functional form) while inference schemes often rely on an explicit tabular representation over the (discrete) variables. For these reasons, search-based algorithms are the only choice available for models with large treewidth and with implicit representation.

Contributions

The primary contribution of this chapter is in viewing search for graphical models in the context of AND/OR search spaces rather than OR spaces. We introduce the AND/OR search tree, and show that its size can be bounded exponentially by the depth of its pseudo tree over the graphical model, and is never larger than the size of the OR search tree. This implies exponential savings for any linear space algorithm traversing the AND/OR search tree vs. the OR search tree. Specifically, if the graphical model has treewidth w^*, there exists a pseudo tree of depth $O(w^* \cdot \log n)$. Therefore, the size of the AND/OR search tree is $O(n \cdot \exp(w^* \cdot \log n))$, as opposed to the size of the OR search tree which is $O(\exp n)$.

The AND/OR search tree can be transformed into a graph by merging identical subtrees. We show that the size of the minimal AND/OR search graph is exponential in the treewidth while the size of the minimal OR search graph is exponential in the pathwidth. Since for some graphs the difference between treewidth and pathwidth is substantial (*e.g.,* balanced pseudo trees) the AND/OR representation implies substantial time and space savings for memory intensive algorithms traversing the AND/OR graph. Searching the AND/OR search *graph* can be implemented by goods caching during search, while no-good recording is interpreted as pruning portions of the search space independent of it being a tree or a graph, an OR or an AND/OR. For finding a single solution, pruning the search space is the most significant action. For counting and probabilistic inference, using AND/OR graphs can be of much help even on top of no-good recording.

We also discuss the unifying power of the AND/OR search framework, and its relationship with other existing algorithms, such as Variable Elimination, Recursive Conditioning [23], Backtracking with Tree-Decomposition [99], Value Elimination [5], Case-Factor Diagrams [80] and compilation schemes.

1.1.2 Mixed Networks (Chapter 3)

The communities of probabilistic networks and constraint networks matured in parallel with only minor interaction. Nevertheless some of the algorithms and reasoning principles that emerged within both frameworks, especially those that are graph-based, are quite related. Both frameworks can be viewed as graphical models, a popular paradigm for knowledge representation in general.

Researchers within the logic-based and constraint communities have recognized for some time the need for augmenting deterministic languages with uncertainty information, leading to a variety of concepts and approaches such as non-monotonic reasoning, probabilistic constraint networks and fuzzy constraint networks. The belief networks community started only recently to look into mixed representation [87, 84, 62, 35] perhaps because it is possible, in principle, to capture constraint information within belief networks [86].

In principle, constraints can be embedded within belief networks by modeling each constraint as a Conditional Probability Table (CPT). One approach is to add a new variable for each constraint that is perceived as its *effect* (child node) in the corresponding causal relationship and then to clamp its value to *true* [86, 21]. While this approach is semantically coherent and complies with the acyclic graph restriction of belief networks, it adds a substantial number of new variables, thus cluttering the problem's structure. An alternative approach is to designate one of the arguments of the constraint as a child node (namely, as its effect). This approach, although natural for functions (the arguments are the causes or parents and the function variable is the child node), is quite contrived for general relations (e.g., $x + 6 \neq y$). Such constraints may lead to cycles, which are disallowed in belief networks. Furthermore, if a variable is a child node of two different CPTs (one may be deterministic and one probabilistic) the belief network definition requires that they be combined into a single CPT.

The main shortcoming, however, of any of the above integrations is computational. Constraints have special properties that render them attractive computationally. When constraints are disguised as probabilistic relationships, their computational benefits may be hard to exploit. In particular, the power of constraint inference and constraint propagation may not be brought to bear.

Contributions

We propose a simple framework that combines deterministic and probabilistic networks, called *mixed network*. In the mixed network framework the identity of the respective relationships, as constraints or probabilities, will be maintained explicitly, so that their respective computational power and semantic differences can be vivid and easy to exploit. The mixed network approach allows two distinct representations: causal relationships that are directional and normally (but not necessarily) quantified by CPTs and symmetrical deterministic constraints.

The proposed scheme's value is in providing: (1) semantic coherence; (2) user-interface convenience (the user can relate better to these two pieces of information if they are distinct); and most importantly, (3) computational efficiency.

We outline the main types of algorithms for mixed networks: inference-based and search-based. In particular, we discuss the application of AND/OR search to mixed networks, and the exploitation of constraint propagation algorithms.

1.1.3 Iterative Algorithms for Mixed Networks (Chapter 4)

Probabilistic inference is the principal task in Bayesian networks, and it is known to be an NP-hard problem [21]. Most of the commonly used exact algorithms such as join-tree clustering [66, 57] or Variable Elimination [28, 103], exploit the network structure. Yet, they are time and space exponential in a graph parameter called *induced width* (or *tree-width*), rendering them essentially intractable even for moderate size problems. Approximate algorithms are therefore necessary for most of the practical problems, although approximation within given error bounds is also NP-hard [22, 92].

In this chapter we present iterative inference-based algorithms for graphical models, focusing primarily on the task of belief updating. They are inspired by Pearl's belief propagation algorithm [86], which is known to be exact for poly-trees, and by the Mini-Buckets algorithm [43], which is a bounded inference scheme, an anytime version of Variable Elimination. As a distributed algorithm, belief propagation is also well defined for networks that contain cycles, and it can be applied iteratively in the form of Iterative Belief Propagation (IBP), also known as loopy belief propagation. When the networks contain cycles, IBP is no longer guaranteed to be exact, but in many cases it provides very good approximations upon convergence, most notably in the case of coding networks [89] and some classes of satisfiability [69].

Contributions

In this chapter we investigate: (1) The quality of bounded inference in anytime schemes such as Mini-Clustering, which is a generalization of Mini-Buckets to arbitrary tree-decompositions. (2) The virtues of iterative message-passing algorithms, combined with bounded inference, result in our new Iterative Join-Graph Propagation (IJGP). (3) We also make connections with well known and understood consistency enforcing algorithms for constraint satisfaction, giving strong support for iterating messages, and helping identify cases of strong and weak inference power for IBP and IJGP. More specifically, the contributions are as follows.

Specifically, we present the Mini-Clustering (MC) algorithm, which is inspired by the Mini-Buckets algorithm [43]. MC is a message-passing algorithm guided by a user adjustable parameter called *i-bound*, offering a flexible tradeoff between accuracy and efficiency in anytime style (in general the higher the i-bound, the better the accuracy). MC operates on a tree-decomposition, and similar to Pearl's belief propagation algorithm [86] it converges in two passes, up and down the tree. Our contribution beyond other work in this area [43, 34] consists in:

1. Extending the partition-based approximation for belief updating from mini-buckets to general tree-decompositions, thus allowing the computation of the updated beliefs for all the variables at once. This extension is similar to the one proposed in [34] but replaces optimization with probabilistic inference.

2. Providing for the first time empirical evaluation demonstrating the effectiveness of the partition-based idea for belief updating.

We were motivated by the success of Iterative Belief Propagation (IBP) in trying to make MC benefit from the apparent virtues of iterating. The resulting algorithm, Iterative Join-Graph Propagation (IJGP) is still a message-passing algorithm, but it operates on a general join-graph decomposition which may contain cycles. It also provides a user adjustable *i-bound* that defines the maximum cluster size of the graph (and hence the complexity), so it is both anytime and iterative. IJGP can be viewed as a generalized belief propagation algorithm [102], with user adjustable cluster size. Since both MC and IJGP are approximate schemes, empirical results on various classes of problems are included, shedding light on their average case performance.

The work presented in the last part of the chapter is based on some simple observations that may shed light on IBP's behavior, and on the more general class of IJGP algorithms. Zero-beliefs are variable-value pairs that have zero conditional probability given the evidence. We show that: (1) if a value of a variable is assessed as having zero-belief in any iteration of IBP, it remains a zero-belief in all subsequent iterations; (2) that IBP converges in a finite number of iterations relative to its set of zero-beliefs; and, most importantly (3) that the set of zero-beliefs decided by any of the iterative belief propagation methods is sound. Namely any zero-belief determined by IBP corresponds to a true zero conditional probability relative to the given probability distribution expressed by the Bayesian network.

While each of these claims can be proved directly, our approach is to associate a belief network with a constraint network and show a correspondence between IBP applied to the belief network and an arc-consistency algorithm applied to the corresponding constraint network. Since arc-consistency algorithms are well understood this correspondence not only immediately proves the targeted claims, but may provide additional insight into the behavior of IBP and IJGP. In particular, it not only immediately justifies the iterative application of belief propagation algorithms on the one hand, but it also illuminates its "distance" from being complete, on the other.

1.1.4 AND/OR Cutset Conditioning (Chapter 5)

The complexity of a reasoning task over a graphical model depends on the induced width of the graph. For inference-type algorithms, the space complexity is exponential in the induced width in the worst case, which often makes them infeasible for large and densely connected problems. In such cases, space can be traded at the expense of time by conditioning (assigning values to variables). Search algorithms perform conditioning on all the variables. Cycle-cutset schemes [86, 26] only condition on a subset of variables such that the remaining network is singly connected and can be solved by inference tree algorithms. The more recent hybrid *w-cutset* scheme [90, 10] conditions on a subset of variables such that, when removed, the remaining network has induced width w or less, and can be solved by a variable elimination [29] type algorithm.

Contributions

We apply the AND/OR paradigm to the cycle cutset method. We show that the *AND/OR cycle cutset* is a strict improvement of the traditional cycle cutset method (and the same holds for the extended w-cutset version). The result goes beyond the simple organization of the traditional cutset in an AND/OR pseudo tree and its exploration by AND/OR search, which would be just the straightforward improvement.

The complexity of exploring the traditional cutset is time-exponential in the number of nodes in the cutset, and therefore it calls for finding a minimal cardinality cutset $\mathcal{C}$. The complexity of exploring the AND/OR cutset is time-exponential in its depth, and therefore it calls for finding a minimal depth *AND/OR cutset* $AO\text{-}\mathcal{C}$. That is, a set of nodes that can be organized in a start pseudo tree of minimal depth. Therefore, while the cardinality of the optimal AND/OR cutset, $|AO\text{-}\mathcal{C}|$, may be far larger than that of the optimal traditional cutset, $|\mathcal{C}|$, the depth of $AO\text{-}\mathcal{C}$ is always smaller than or equal to $|\mathcal{C}|$.

1.1.5 AND/OR Search and Inference Algorithms (Chapter 6)

It is convenient to classify algorithms that solve reasoning problems of graphical models as either search (*e.g.*, depth first, branch and bound) or inference (*e.g.*, variable elimination, join-tree clustering). Search is time-exponential in the number of variables, yet it can be accomplished in linear memory. Inference exploits the graph structure of the model and can be accomplished in time and space exponential in the *treewidth* of the problem. When the treewidth is big, inference must be augmented with search to reduce the memory requirements. In the past three decades search methods were enhanced with structure exploiting techniques. These improvements often require substantial memory, making the distinction between search and inference fuzzy. Recently, claims regarding the superiority of memory-intensive search over inference or vice-versa are were [5]. Our aim is to clarify this relationship and to create cross-fertilization using the strengths of both schemes.

We also address some long-standing questions regarding the computational merits of several time-space sensitive algorithms for graphical models. In the past ten years, four types of algorithms have emerged, based on: (1) cycle-cutset and w-cutset [86, 26]; (2) alternating conditioning and elimination controlled by induced width w [90, 63, 47]; (3) Recursive Conditioning [23], which was recently recast as context-based AND/OR search [45]; (4) varied separator-sets for tree decompositions [32]. The question is how do all these methods compare and, in particular, is there one that is superior? A brute-force analysis of time and space complexities of the respective schemes does not settle the question.

Contributions

First, we compare pure search with pure inference algorithms in graphical models through the new framework of AND/OR search. Specifically, we compare Variable Elimination (VE) against memory-intensive AND/OR Search (AO), and place algorithms such as graph-based backjumping, no-good and good learning, and look-ahead schemes [31] within the AND/OR search framework. We show that there is no principled difference between memory-intensive search restricted to fixed variable ordering and inference beyond: (1) different direction of exploring a common search space (top down for search vs. bottom-up for inference); (2) different assumption of control strategy (depth-first for search and breadth-first for inference). We also show that those differences have no practical effect, except under the presence of determinism. Our analysis assumes a fixed variable ordering. Some of the conclusions may not hold for dynamic variable ordering.

Second, we compare three hybrid schemes, that can trade time for space, governed by a parameter bounding the memory limit. They have all emerged from seemingly different principles: Adaptive Caching AND/OR Search (**AOC(i)**) is search based, Tree Decomposition with Conditioning (**TDC(i)**) is inference based and Variable Elimination and Conditioning (**VEC**) combines search and inference. We show that if the graphical models contain no determinism, **AOC(i)** can have a smaller time complexity than the vanilla versions of both **VEC(i)** and **TDC(i)**. This is due to a more efficient exploitation of the graphical structure of the problem through AND/OR search, and the adaptive caching scheme that benefits from the cutset principle. These ideas can be used to enhance **VEC(i)** and **TDC(i)**. We show that if **VEC(i)** uses AND/OR search over the conditioning set and is guided by the pseudo tree data structure, then there exists an execution of **AOC(i)** that is identical to it. We also show that if **TDC(i)** processes clusters by AND/OR search with adaptive caching, then there exists an execution of **AOC(i)** identical to it. AND/OR search with adaptive caching (**AOC(i)**) emerges therefore as a unifying scheme, never worse than the other two. All the analysis was done by using the context minimal data structure, which provides a powerful methodology for comparing the algorithms. When the graphical model contains determinism, all the above schemes become incomparable. This is due to the fact that they process variables in reverse orderings, and will encounter and exploit deterministic information differently.

1.1.6 AND/OR Multi-Valued Decision Diagrams (Chapter 7)

Decision diagrams are widely used in many areas of research, especially in software and hardware verification [18, 81]. A binary decision diagram (BDD) represents a Boolean function by a directed acyclic graph with two sink nodes (labeled 0 and 1), and every internal node is labeled with a variable and has exactly two children: *low* corresponding to the 0 value and *high* corresponding to the 1 value. If isomorphic nodes were not merged, on one extreme we would have the full search *tree*, also called a Shannon tree, which is the usual full tree explored by the backtracking algorithm. The tree can be ordered if we impose that variables be encountered in the same order along every branch. It can then be compressed by merging isomorphic nodes (i.e., with the same label and identical children), and by eliminating redundant nodes (i.e., whose *low* and *high* children are identical). The result is the celebrated *reduced ordered binary decision diagram*, or OBDD for short, introduced by Bryant [16]. However, the underlying structure is OR, because the initial Shannon tree is an OR tree. If AND/OR search trees are reduced by node merging and redundant nodes elimination we get a compact search graph that can be viewed as a BDD representation augmented with AND nodes.

Contributions

In this chapter we apply the AND/OR decomposition to decision diagrams. As a detail, the number of values is also increased from two to any constant, but this is less significant for the algorithms. The benefit of moving from OR structure to AND/OR is in a lower complexity of the algorithms and size of the compiled structure. It typically moves from being bounded exponentially by the *pathwidth* pw^*, which is characteristic of chain decompositions or linear structures, to being exponentially bounded by the *treewidth* w^*, which is characteristic of tree structures (it always holds that $w^* \leq pw^*$ and $pw^* \leq w^* \cdot \log n$). In both cases, the compactness achieved in practice is often far smaller than what the bounds suggest.

Our contributions are as follows: (1) We formally describe the AND/OR Multi-Valued Decision Diagram (AOMDD) and prove that it is a canonical representation for constraint networks. (2) We extend the AOMDD to general weighted graphical models. (3) We give a compilation algorithm based on AND/OR search, that saves the trace of the memory intensive search (which is a subset of the context minimal graph), and then reduces it in one bottom up pass. (4) We describe the APPLY operator that combines two AOMDDs by an operation, and show that its complexity is at most quadratic in the input. (5) We give a scheduling of building the AOMDD of a graphical model starting with the AOMDDs of its functions. It is based on an ordering of variables, which gives rise to a pseudo tree according to the execution of Variable Elimination algorithm. This guarantees that the complexity is at most exponential in the *induced width* along the ordering (equal to the treewidth of the corresponding decomposition). (6) We show how AOMDDs relate to various earlier and recent compilation frameworks, providing a unifying perspective for all these methods. (7) We also introduce the concept of *semantic treewidth*, which helps explain why the size of a decision diagram is often much smaller than the worst case bound.

1.2 Preliminaries

The remaining of the chapter contains preliminary notation and definitions and gives examples of graphical models.

Notations A reasoning problem is defined in terms of a set of variables taking values on finite domains and a set of functions defined over these variables. We denote variables or subsets of variables by uppercase letters (*e.g.*, $X, Y, \ldots$) and values of variables by lower case letters (*e.g.*, $x, y, \ldots$). Sets are usually denoted by bold letters, for example $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a set of variables. An assignment $(X_1 = x_1, \ldots, X_n = x_n)$ can be abbreviated as $x = (\langle X_1, x_1 \rangle, \ldots, \langle X_n, x_n \rangle)$ or $x = (x_1, \ldots, x_n)$. For a subset of variables $\mathbf{Y}$, $D_{\mathbf{Y}}$ denotes the Cartesian product of the domains of variables in $\mathbf{Y}$. The projection of an assignment $x = (x_1, \ldots, x_n)$ over a subset $\mathbf{Y}$ is denoted by $x_{\mathbf{Y}}$ or $x[\mathbf{Y}]$. We will also denote by $Y = y$ (or y for short) the assignment of values to variables in $\mathbf{Y}$ from their respective domains. We denote functions by letters f, g, h etc., and the scope (set of arguments) of the function f by $scope(f)$.

1.2.1 Graph Concepts

A *directed graph* is a pair $G = \{V, E\}$, where $V = \{X_1, \ldots, X_n\}$ is a set of vertices, and $E = \{(X_i, X_j) | X_i, X_j \in V\}$ is the set of edges (arcs). If $(X_i, X_j) \in E$, we say that X_i *points to* X_j. The degree of a variable is the number of arcs incident to it. For each variable X_i, $pa(X_i)$ or pa_i, is the set of variables pointing to X_i in G, while the set of child vertices of X_i, denoted $ch(X_i)$, comprises the variables that X_i points to. The family of X_i, F_i, includes X_i and its parent variables. A directed graph is acyclic if it has no directed cycles. An *undirected graph* is defined similarly to a directed graph, but there is no directionality associated with the edges.

DEFINITION **1.2.1 (induced width)** *An* ordered graph *is a pair* (G, d) *where* G *is an undirected graph, and* $d = X_1, \ldots, X_n$ *is an ordering of the nodes. The* width of a node *is the number of the node's neighbors that precede it in the ordering. The* width of an ordering d, *is the maximum width over all nodes. The* induced width of an ordered graph, $w^*(d)$, *is the width of the induced ordered graph obtained as follows: nodes are processed from last to first; when node* X *is processed, all its preceding neighbors are connected. The* **induced width** *of a graph, denoted by* w^*, *is the minimal induced width over all its orderings.*

DEFINITION **1.2.2 (hypergraph)** *A* hypergraph *is a pair* $H = (X, S)$*, where* $S = \{S_1, \ldots, S_t\}$ *is a set of subsets of* V *called* hyperedges.

DEFINITION **1.2.3 (tree decomposition)** *A* tree decomposition *of a hypergraph* $H = (X, S)$ *is a tree* $T = (V, E)$ *(*V *is the set of nodes, also called "clusters", and* E *is the set of edges) together with a labeling function* χ *that associates with each vertex* $v \in V$ *a set* $\chi(v) \subseteq X$ *satisfying:*

1. *For each* $S_i \in S$ *there exists a vertex* $v \in V$ *such that* $S_i \subseteq \chi(v)$*;*
2. (running intersection property) *For each* $X_i \in X$*, the set* $\{v \in V \mid X_i \in \chi(v)\}$ *induces a connected subtree of* T*.*

DEFINITION **1.2.4 (treewidth, pathwidth)** *The* width *of a tree decomposition of a hypergraph is the size of its largest cluster minus 1 (*$\max_v |\chi(v)| - 1$*). The* **treewidth** *of a hypergraph is the minimum width along all possible tree decompositions. The* **pathwidth** *is the treewidth over the restricted class of chain decompositions.*

It is easy to see that given an induced graph, the set of maximal cliques (also called clusters) provide a tree decomposition of the graph, namely the clusters can be connected in a tree structure that satisfies the running intersection property. It is well known that the induced width of a graph is identical to its treewidth [41]. For various relationships between these and other graph parameters see [3, 14, 13].

1.2.2 AND/OR Search Graphs

AND/OR search spaces An AND/OR state space representation of a problem is defined by a 4-tuple $\langle S, O, S_g, s_0 \rangle$. S is a set of states which can be either OR or AND states (the OR states represent alternative ways for solving the problem while the AND states often represent problem decomposition into subproblems, all of which need to be solved). O

is a set of operators. An OR operator transforms an OR state into another state, and an AND operator transforms an AND state into a set of states. There is a set of goal states $S_g \subseteq S$ and a start node $s_0 \in S$. Example problem domains modeled by AND/OR graphs are two-player games, parsing sentences and Tower of Hanoi [85].

The AND/OR state space model induces an explicit AND/OR search *graph*. Each state is a node and its child nodes are those obtained by applicable AND or OR operators. The search graph includes a *start* node. The terminal nodes (having no child nodes) are marked as Solved (S), or Unsolved (U).

Solution subtree A *solution subtree* of an AND/OR search graph G is a subtree which: (1) contains the start node s_0; (2) if n in the subtree is an OR node then it contains one of its child nodes in G and if n is an AND node it contains all its children in G; 3. all its terminal nodes are "Solved" (S). AND/OR graphs can have a cost associated with each arc, and the cost of a solution subtree is a function (*e.g.*, sum-cost) of the arcs included in the solution subtree. In this case we may seek a solution subtree with optimal (maximum or minimum) cost. Other tasks that enumerate all solution subtrees (*e.g.*, counting solutions) can also be defined.

1.2.3 Graphical Models

Graphical models include constraint networks [31] defined by relations of allowed tuples, (directed or undirected) probabilistic networks [86], defined by conditional probability tables over subsets of variables, cost networks defined by costs functions and influence diagrams [54] which include both probabilistic functions and cost functions (*i.e.*, utilities) [30]. Each graphical model comes with its typical queries, such as finding a solution, or an optimal one (over constraint networks), finding the most probable assignment or updating the posterior probabilities given evidence, posed over probabilistic networks, or finding optimal solutions for cost networks. The task for influence diagrams is to choose a sequence of actions that maximizes the expected utility. Markov random fields are the undirected counterparts of probabilistic networks. They are defined by a collection of probabilistic functions called potentials, over arbitrary subsets of variables. The framework presented in this dissertation is applicable across all graphical models that have discrete variables, however we will draw most of our examples from constraint networks and directed probabilistic networks.

In general, a graphical model is defined by a collection of functions F, over a set of variables X, conveying probabilistic, deterministic or preferential information, whose structure is captured by a graph.

DEFINITION **1.2.5 (graphical model)** *A* graphical model $\mathcal{M}$ *is a 4-tuple,* $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \otimes \rangle$*, where:*

1. $\mathbf{X} = \{X_1, \ldots, X_n\}$ *is a finite set of variables;*
2. $\mathbf{D} = \{D_1, \ldots, D_n\}$ *is the set of their respective finite domains of values;*
3. $\mathbf{F} = \{f_1, \ldots, f_r\}$ *is a set of positive real-valued discrete functions, each defined over a subset of variables* $\mathbf{S}_i \subseteq \mathbf{X}$*, called its scope, and denoted by* $scope(f_i)$*.*
4. $\otimes$ *is a combination operator*[1] *(e.g.,* $\otimes \in \{\prod, \sum, \bowtie\}$ *(product, sum, join)).*

The graphical model represents the combination of all its functions: $\otimes_{i=1}^{r} f_i$*.*

Next, we introduce the notion of *universal* graphical model that is defined by a single function.

DEFINITION **1.2.6 (universal equivalent graphical model)** *Given a graphical model* $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}_1, \otimes \rangle$ *the universal equivalent model of* $\mathcal{M}$ *is* $u(\mathcal{M}) = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}_2 = \{\otimes_{f_i \in \mathbf{F}_1} f_i\}, \otimes \rangle$*.*

[1]The combination operator can also be defined axiomatically [95].

Two graphical models are **equivalent** if they represent the same set of solutions. Namely, if they have the same universal model.

DEFINITION **1.2.7 (weight (or cost) of a full and a partial assignment)** *Given a graphical model* $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$*, the weight of a full assignment* $x = (x_1, \ldots, x_n)$ *is defined by* $w(x) = \otimes_{f \in \mathbf{F}} f(x[scope(f)])$*. Given a subset of variables* $\mathbf{Y} \subseteq \mathbf{X}$*, the* weight of a partial assignment y *is the combination of all the functions whose scopes are included in* $\mathbf{Y}$ *(denoted by* $\mathbf{F_Y}$*) evaluated at the assigned values. Namely,* $w(y) = \otimes_{f \in \mathbf{F_Y}} f(y[scope(f)])$.

We can restrict a graphical model by conditioning on a partial assignment.

DEFINITION **1.2.8 (conditioned graphical model)** *Given a graphical model* $\mathcal{R} = \langle X, D, F, \bigotimes \rangle$ *and given a partial assignment* $Y = y$*,* $Y \subset X$*, the conditional graphical model is* $\mathcal{R}|_y = \langle X, D|_y, F|_y, \bigotimes \rangle$*, where* $D|_y = \{D_i \in D, X_i \notin Y\}$ *and* $F|_y = \{f|_{Y=y}, f \in F$*, and* $scope(f) \not\subseteq Y\}$.

Consistency For most graphical models, the range of the functions has a special zero value "0" that is *absorbing* relative to the combination operator (*e.g.,* multiplication). Combining anything with "0" yields a "0". The "0" value expresses the notion of inconsistent assignments. It is a primary concept in constraint networks but can also be defined relative to other graphical models that have a "0" element.

DEFINITION **1.2.9 (consistent partial assignment, solution)** *Given a graphical model having a "0" element, a partial assignment is consistent if its cost is non-zero. A solution is a consistent assignment to all the variables.*

Throughout the dissertation, we will use two examples of graphical models: constraint networks and belief networks. In the case of constraint networks, the functions can be understood as relations. In other words, the functions (also called constraints) can take only two values, $\{0, 1\}$ (or $\{true, false\}$). A 0 value indicates that the corresponding assignment to the variables is inconsistent (not allowed), and a 1 value indicates consistency.

Belief networks are an example of the more general case of graphical models (also called *weighted* graphical models). The functions in this case are conditional probability tables, so the values of a function are any real number in the interval $[0, 1]$.

Flat functions Each function in a graphical model having a "0" element expresses implicitly a constraint. The *flat* constraint of function f_i is a constraint R_i over its scope that includes all and only the consistent tuples. In the following chapters, when we talk about a constraint network, we refer also to the flat constraint network that can be extracted from the general graphical model. When all the full assignments are consistent we say that the graphical model is *strictly positive*.

DEFINITION **1.2.10 (primal graph)** *The* primal graph *of a graphical model is an undirected graph that has variables as its vertices and an edge connects any two variables that appear in the scope of the same function.*

The primal graph captures the structure of the knowledge expressed by the graphical model. In particular, graph separation indicates independency of sets of variables given some assignments to other variables. All of the advanced algorithms for graphical models exploit the graphical structure, by using a heuristically good elimination order, or a tree decomposition or some similar method. We will use the concept of pseudo tree, which resembles the tree rearrangements introduced in [48]:

DEFINITION **1.2.11 (pseudo tree)** *A* pseudo tree *of a graph $G = (\mathbf{X}, E)$ is a rooted tree $\mathcal{T}$ having the same set of nodes $\mathbf{X}$, such that every arc in E is a back-arc in $\mathcal{T}$ (i.e., it connects nodes on the same path from root).*

DEFINITION **1.2.12 (reasoning problem)** *A* reasoning problem *over a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \otimes \rangle$ is defined by a marginalization operator and a set of subsets of $\mathbf{X}$ that are of interest. It is therefore a triplet, $\mathcal{P} = \langle \mathcal{M}, \Downarrow_{\mathbf{Y}}, \{\mathbf{Z}_1, \ldots, \mathbf{Z}_t\} \rangle$, where $\mathbf{Z} = \{\mathbf{Z}_1, \ldots, \mathbf{Z}_t\}$ is a set of subsets of variables of $\mathbf{X}$. If $\mathbf{S}$ is the scope of function f*

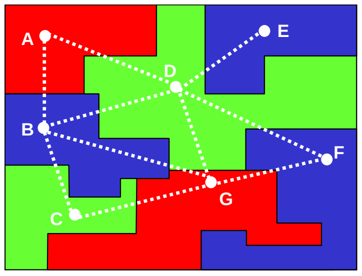
and $\mathbf{Y} \subseteq \mathbf{X}$*, then* $\Downarrow_{\mathbf{Y}} f \in \{\overset{max}{\mathbf{S}-\mathbf{Y}} f, \overset{min}{\mathbf{S}-\mathbf{Y}} f, \prod_{\mathbf{Y}} f, \sum_{\mathbf{S}-\mathbf{Y}} f\}$ *is a marginalization operator.* $\mathcal{P}$ *can be viewed as a vector function over the scopes* $\mathbf{Z}_1, \ldots, \mathbf{Z}_t$*. The reasoning problem is to compute* $\mathcal{P}_{\mathbf{Z}_1,\ldots,\mathbf{Z}_t}(\mathcal{M}) = (\Downarrow_{\mathbf{Z}_1} \otimes_{i=1}^{r} f_i, \ldots, \Downarrow_{\mathbf{Z}_t} \otimes_{i=1}^{r} f_i)$ *.*

We will focus primarily on reasoning problems defined by $\mathbf{Z} = \emptyset$. The marginalization operator is sometimes called *elimination* operator because it removes some arguments from the scope of the input function. Specifically, $\Downarrow_{\mathbf{Y}} f$ is defined on $\mathbf{Y}$. It therefore removes variables $\mathbf{S} - \mathbf{Y}$ from $\mathbf{S} = scope(f)$. Note that here $\prod_{\mathbf{Y}} f$ is the relational projection operator and unlike the rest of the marginalization operators the convention is that is defined by the scope of variables that are *not* eliminated.

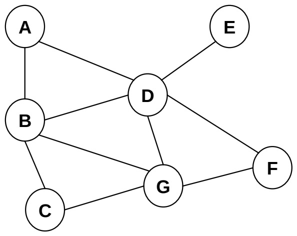
1.2.4 Constraint Networks

Constraint networks provide a framework for formulating real world problems, such as scheduling and design, planning and diagnosis, and many more as a set of constraints between variables. For example, one approach to formulating a scheduling problem as a constraint satisfaction problem (CSP) is to create a variable for each resource and time slice. Values of variables would be the tasks that need to be scheduled. Assigning a task to a particular variable (corresponding to a resource at some time slice) means that this resource starts executing the given task at the specified time. Various physical constraints (such as that a given job takes a certain amount of time to execute, or that a task can be executed at most once) can be modeled as constraints between variables. The *constraint satisfaction task* is to find an assignment of values to all the variables that does not violate any constraints, or else to conclude that the problem is inconsistent. Other tasks are finding all solutions and counting the solutions.

DEFINITION **1.2.13 (constraint network, constraint satisfaction problem)** *A* constraint network (CN) *is a 4-tuple,* $\langle \mathbf{X}, \mathbf{D}, \mathbf{C}, \bowtie \rangle$*, where* $\mathbf{X}$ *is a set of variables* $\mathbf{X} = \{X_1, \ldots, X_n\}$*, associated with a set of discrete-valued domains,* $\mathbf{D} = \{D_1, \ldots, D_n\}$*, and a set of con-*



(a) Graph coloring problem



(b) Constraint graph

Figure 1.1: Constraint network

straints $\mathbf{C} = \{C_1, \ldots, C_r\}$. *Each constraint* C_i *is a pair* $(\mathbf{S}_i, R_i)$, *where* R_i *is a relation* $R_i \subseteq D_{\mathbf{S}_i}$ *defined on a subset of variables* $\mathbf{S}_i \subseteq \mathbf{X}$. *The relation denotes all compatible tuples of* $D_{\mathbf{S}_i}$ *allowed by the constraint. The combination operator,* $\bowtie$, *is join. The primal graph of a constraint network is called* constraint graph. *A* solution *is an assignment of values to all the variables* $x = (x_1, \ldots, x_n)$, $x_i \in D_i$, *such that* $\forall\, C_i \in \mathbf{C}$, $x_{\mathbf{S}_i} \in R_i$. *The constraint network represents its set of solutions,* $\bowtie_i C_i$.

Constraint satisfaction is a reasoning problem $\mathcal{P} = \langle \mathcal{R}, \Pi, \mathbf{Z} \rangle$, where $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{C}, \bowtie \rangle$ is a constraint network, and the marginalization operator is the projection operator Π. Namely, for constraint satisfaction $\mathbf{Z} = \{\emptyset\}$, and $\Downarrow_{\mathbf{Y}}$ is $\Pi_{\mathbf{Y}}$. So the task is to find $\Downarrow_{\emptyset} \otimes_i f_i = \Pi_{\emptyset}(\bowtie_i f_i)$ which corresponds to enumerating all solutions. When the combination operator is a product over the cost-based representation of the relations, and the marginalization operator is logical summation we get "1" if the constraint problem has a solution and "0" otherwise. For *counting*, the marginalization operator is summation and $\mathbf{Z} = \{\emptyset\}$ too.

Example 1.2.1 *Figure 1.1(a) shows a graph coloring problem that can be modeled by a constraint network. Given a map of regions, the problem is to color each region by one of the given colors* {*red, green, blue*}, *such that neighboring regions have different colors. The variables of the problems are the regions, and each one has the domain* {*red, green, blue*}. *The constraints are the relation* "different" *between neighboring regions. Figure 1.1(b) shows the constraint graph, and a solution (A=red, B=blue, C=green, D=green, E=blue, F=blue, G=red) is given in Figure 1.1(a).*

Cost Networks An immediate extension of constraint networks are *cost networks* where the set of functions are real-valued cost functions, and the primary task is optimization.

DEFINITION **1.2.14 (cost network, combinatorial optimization)** *A* cost network *is a 4-tuple,* $\langle \mathbf{X}, \mathbf{D}, \mathbf{C}, \sum \rangle$*, where* $\mathbf{X}$ *is a set of variables* $\mathbf{X} = \{X_1, \ldots, X_n\}$*, associated with a set of discrete-valued domains,* $\mathbf{D} = \{D_1, \ldots, D_n\}$*, and a set of cost functions* $\mathbf{C} = \{C_1, \ldots, C_r\}$*. Each* C_i *is a real-valued function defined on a subset of variables* $\mathbf{S}_i \subseteq \mathbf{X}$*. The combination operator, is* $\sum$*. The reasoning problem is to find a minimum cost solution which is expressed via the marginalization operator of minimization, and* $Z = \{\emptyset\}$*.*

The task of MAX-CSP, namely finding a solution that satisfies the maximum number of constraints (when the problem is inconsistent), can be defined by treating each relation as a cost function that assigns "0" to consistent tuples and "1" otherwise. The combination operator is summation and the marginalization operator is minimization. Namely, the task is to find $\Downarrow_{\emptyset} \otimes_i f_i = \min_{\mathbf{X}}(\sum_i f_i)$.

Propositional Satisfiability A special case of a CSP is *propositional satisfiability* (SAT). A formula φ in *conjunctive normal form* (CNF) is a conjunction of *clauses* $\alpha_1, \ldots, \alpha_t$, where a clause is a disjunction of *literals* (propositions or their negations). For example, $\alpha = (P \vee \neg Q \vee \neg R)$ is a clause, where P, Q and R are propositions, and P, $\neg Q$ and $\neg R$ are literals. The SAT problem is to decide whether a given CNF theory has a *model*, *i.e.*, a truth-assignment to its propositions that does not violate any clause. Propositional satisfiability (SAT) can be defined as a CSP, where propositions correspond to variables, domains are {0, 1}, and constraints are represented by clauses, for example the clause $(\neg A \vee B)$ is a relation over its propositional variables that allows all tuples over (A, B) except $(A = 1, B = 0)$.

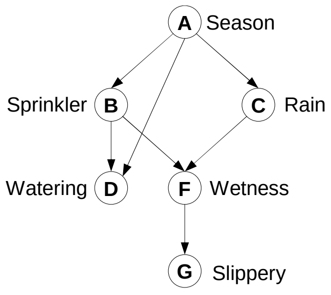
1.2.5 Belief Networks

Belief networks [86], also known as Bayesian networks, provide a formalism for reasoning about partial beliefs under conditions of uncertainty. They are defined by a directed acyclic graph over vertices representing random variables of interest (*e.g.*, the temperature of a device, the gender of a patient, a feature of an object, the occurrence of an event). The arcs can signify the existence of direct causal influences between linked variables quantified by conditional probabilities that are attached to each cluster of parents-child vertices in the network. But these relationships need not necessarily be causal and we can still have a perfectly well defined belief network.

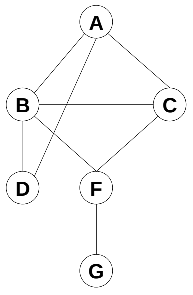
DEFINITION **1.2.15 (belief networks)** *A* belief network (BN) *is a graphical model* $\mathcal{P} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}_G, \prod \rangle$*, where* $\mathbf{X} = \{X_1, \ldots, X_n\}$ *is a set of variables over multi-valued domains* $\mathbf{D} = \{D_1, \ldots, D_n\}$*. Given a directed acyclic graph* G *over* $\mathbf{X}$ *as nodes,* $P_G = \{P_1, \ldots, P_n\}$*, where* $P_i = \{P(X_i \,|\, pa\,(X_i)\,)\}$ *are conditional probability tables (CPTs for short) associated with each* X_i*, where* $pa(X_i)$ *are the parents of* X_i *in the acyclic graph* G*. A belief network represents a probability distribution over* $\mathbf{X}$*,* $P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i|x_{pa(X_i)})$*. An evidence set* e *is an instantiated subset of variables.*

When formulated as a graphical model, functions in F *denote conditional probability tables and the scopes of these functions are determined by the directed acyclic graph* G*: each function* f_i *ranges over variable* X_i *and its parents in* G*. The combination operator is product,* $\otimes = \prod$*. The primal graph of a belief network is called a moral graph. It connects any two variables appearing in the same CPT.*

Example 1.2.2 *Figure 1.2(a) gives an example of a belief network over 6 variables, and Figure 1.2(b) shows its moral graph . The example expresses the causal relationship between variables "Season" (*A*), "The configuration of an automatic sprinkler system" (*B*), "The amount of rain expected" (*C*), "The amount of manual watering necessary" (*D*), "The wetness of the pavement" (*F*) and "Whether or not the pavement is slippery"*



(a) Directed acyclic graph



(b) Moral graph

Figure 1.2: Belief network

(G). The belief network expresses the probability distribution $P(A, B, C, D, F, G) = P(A) \cdot P(B|A) \cdot P(C|A) \cdot P(D|B, A) \cdot P(F|C, B) \cdot P(G|F)$*.*

Two of the most popular tasks for belief networks are defined below:

DEFINITION **1.2.16 (belief updating)** *Given a belief network and evidence* e*, the* belief updating *task is to compute the posterior marginal probability of variable* X_i*, conditioned on the evidence. Namely,*

$$Bel(X_i = x_i) = P(X_i = x_i \mid e) = \alpha \sum_{\{(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n)|E=e,X_i=x_i\}} \prod_{k=1}^{n} P(x_k, e|x_{pa_k}),$$

where α *is a normalization constant. In this case, the marginalization operator is* $\Downarrow_{\mathbf{Y}} = \sum_{\mathbf{X}-\mathbf{Y}}$*, and* $\mathbf{Z}_i = \{X_i\}$*. Namely,* $\forall X_i, \Downarrow_{X_i} \otimes_k f_k = \sum_{\{X-X_i|X_i=x_i\}} \prod_k P_k$*. The query of finding the probability of the evidence is defined by* $Z = \emptyset$*.*

DEFINITION **1.2.17 (most probable explanation)** *The* most probable explanation (MPE) *task is to find a complete assignment which agrees with the evidence, and which has the highest probability among all such assignments. Namely, to find an assignment* $(x_1^o, \ldots, x_n^o)$ *such that*

$$P(x_1^o, \ldots, x_n^o) = max_{x_1,\ldots,x_n} \prod_{k=1}^{n} P(x_k, e|x_{pa_k}).$$

As a reasoning problem, an MPE task is to find $\Downarrow_{\emptyset} \otimes_i f_i = \max_X \prod_i P_i$. Namely, the marginalization operator is $\max$ and $\mathbf{Z} = \{\emptyset\}$.

Chapter 2

AND/OR Search Spaces for Graphical Models

2.1 Introduction

Bayesian networks, constraint networks, Markov random fields and influence diagrams, commonly referred to as graphical models, are all languages for knowledge representation that use graphs to capture conditional independencies between variables. These independencies allow both the concise representation of knowledge and the use of efficient graph-based algorithms for query processing. Algorithms for processing graphical models fall into two general types: inference-based and search-based. Inference-based algorithms (*e.g.*, Variable Elimination, Tree Clustering) are better at exploiting the independencies captured by the underlying graphical model. They provide a superior worst case time guarantee, as they are time exponential in the treewidth of the graph. Unfortunately, any method that is time-exponential in the treewidth is also space exponential in the treewidth or separator width and, therefore, not practical for models with large treewidth.

Search-based algorithms (*e.g.*, depth-first branch-and-bound, best-first search) traverse the model's search space where each path represents a partial or full solution. The linear structure of search spaces does not retain the independencies represented in the underlying graphical models and, therefore, search-based algorithms may not be nearly as effective as inference-based algorithms in using this information. On the other hand, the space requirements of search-based algorithms may be much less severe than those of inference-based algorithms and they can accommodate a wide spectrum of space-bounded algorithms, from linear space to treewidth bounded space. In addition, search methods require only an implicit, generative, specification of the functional relationship (given in a procedural or functional form) while inference schemes often rely on an explicit tabular representation over the (discrete) variables. For these reasons, search-based algorithms are the only choice available for models with large treewidth and with implicit representation.

2.1.1 Contributions

In this chapter we propose to use the well-known idea of an AND/OR search space, originally developed for heuristic search [85], to generate search procedures that take advantage of information encoded in the graphical model. We demonstrate how the independencies captured by the graphical model may be used to yield AND/OR search trees that are exponentially smaller than the standard search tree (that can be thought of as an OR tree). Specifically, we show that the size of the AND/OR search tree is bounded exponentially by the depth of a spanning pseudo tree over the graphical model. Subsequently, we move from AND/OR search trees to AND/OR search graphs. Algorithms that explore the search graph involve controlled memory management that allows improving their time-performance by increasing their use of memory. The transition from a search tree to a search graph in AND/OR representations also yields significant savings compared to the same transition in the original OR space. In particular, we show that the size of the minimal AND/OR graph is bounded exponentially by the treewidth, while for OR graphs it is bounded exponentially by the pathwidth.

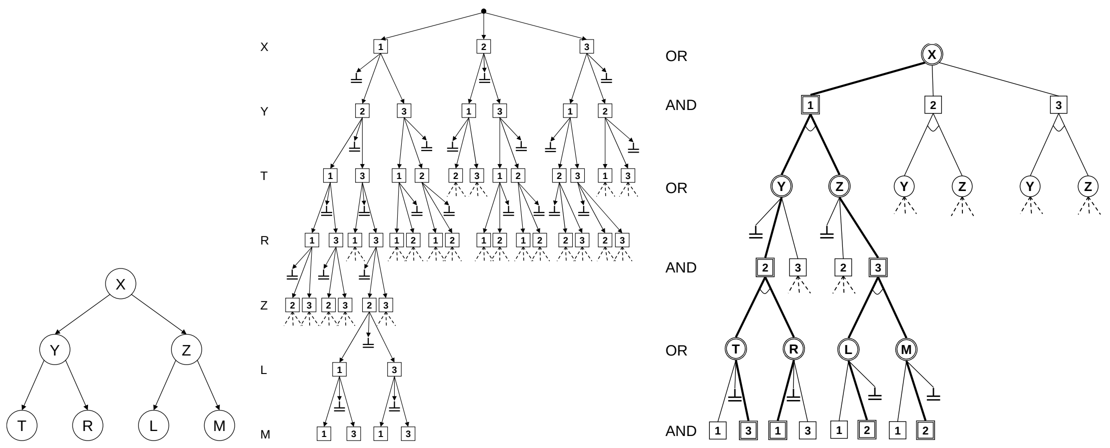
Our idea of the AND/OR search space is inspired by search advances introduced sporadically in the past three decades for constraint satisfaction and more recently for probabilistic inference and for optimization tasks. Specifically, it resembles pseudo tree rearrangement [48, 49], briefly introduced two decades ago, which was adapted subsequently for distributed constraint satisfaction [19, 20] and more recently in [83], and was also shown to be related to graph-based backjumping [27]. This work was extended in [6] and more recently applied to optimization tasks [65]. Another version that can be viewed as exploring the AND/OR graphs was presented recently for constraint satisfaction [99] and for optimization [98]. Similar principles were introduced recently for probabilistic inference (in algorithm Recursive Conditioning [23] as well as in Value Elimination [5, 4]) and currently provide the backbones of the most advanced SAT solvers [93].

The research presented in this chapter is based in part on [45, 44, 38].

2.2 AND/OR Search Trees

We will present the AND/OR search space for a general *graphical model* starting with an example of a constraint network.

Example 2.2.1 *Consider the simple tree graphical model (*i.e.*, the primal graph is a tree) in Figure 2.1(a), over domains* $\{1, 2, 3\}$*, which represents a graph-coloring problem. Namely, each node should be assigned a value such that adjacent nodes have different values. Once variable* X *is assigned the value 1, the search space it roots can be decomposed into two independent subproblems, one that is rooted at* Y *and one that is rooted at Z, both of which need to be solved independently. Indeed, given* $X = 1$*, the two search subspaces do not interact. The same decomposition can be associated with the other assignments to* X*,* $\langle X, 2\rangle$ *and* $\langle X, 3\rangle$*. Applying the decomposition along the tree (in Figure 2.1(a) yields the AND/OR search tree in Figure 2.1(c). In the AND/OR space a full assignment to all the variables is not a path but a subtree. For comparison, the traditional* OR *search tree is depicted in Figure 2.1(b). Clearly, the size of the AND/OR search space is smaller than*



(a) A constraint tree (b) OR search tree (c) AND/OR search tree with one of its solution subtrees

Figure 2.1: OR vs. AND/OR search trees; note the connector for AND arcs

that of the regular OR space. The OR search space has $3 \cdot 2^7$ *nodes while the AND/OR has* $3 \cdot 2^5$ *(compare 2.1(b) with 2.1(c)). If* k *is the domain size, a balanced binary tree with* n *nodes has an OR search tree of size* $O(k^n)$*. The AND/OR search tree, whose pseudo tree has depth* $O(\log_2 n)$*, has size* $O((2k)^{\log_2 n}) = O(n \cdot k^{\log_2 n}) = O(n^{1+\log_2 k})$*. When* $k = 2$*, this becomes* $O(n^2)$.

The AND/OR space is not restricted to tree graphical models. It only has to be guided by a *backbone* tree which spans the original primal graph of the graphical model in a particular way. We will define the AND/OR search space relative to a depth-first search tree (DFS tree) of the primal graph first, and will generalize to a broader class of backbone spanning trees subsequently. For completeness sake we define *DFS spanning tree,* next.

DEFINITION **2.2.1 (DFS spanning tree)** *Given a DFS traversal ordering* $d = (X_1, \ldots, X_n)$*, of an undirected graph* $G = (V, E)$*, the* DFS spanning tree $\mathcal{T}$ *of* G *is defined as the tree rooted at the first node,* X_1*, which includes only the traversed arcs of* G*. Namely,* $\mathcal{T} = (V, E')$*, where* $E' = \{(X_i, X_j) \mid X_j \ traversed \ from \ X_i\}$.

We are now ready to define the notion of AND/OR search tree for a graphical model.

DEFINITION **2.2.2 (AND/OR search tree)** *Given a graphical model $\mathcal{R} = \langle X, D, F, \bigotimes \rangle$, its primal graph G and a backbone DFS tree $\mathcal{T}$ of G, the associated AND/OR search tree, denoted $S_{\mathcal{T}}(R)$, has alternating levels of AND and OR nodes. The OR nodes are labeled X_i and correspond to the variables. The AND nodes are labeled $\langle X_i, x_i \rangle$ (or simply x_i) and correspond to the value assignments of the variables. The structure of the AND/OR search tree is based on the underlying backbone tree $\mathcal{T}$. The root of the AND/OR search tree is an OR node labeled by the root of $\mathcal{T}$. A path from the root of the search tree $S_{\mathcal{T}}(\mathcal{R})$ to a node n is denoted by π_n. If n is labeled X_i or x_i the path will be denoted $\pi_n(X_i)$ or $\pi_n(x_i)$, respectively. The assignment sequence along path π_n, denoted $asgn(\pi_n)$ is the set of value assignments associated with the sequence of AND nodes along π_n:*

$$\begin{aligned} asgn(\pi_n(X_i)) &= \{\langle X_1, x_1 \rangle, \langle X_2, x_2 \rangle, \ldots, \langle X_{i-1}, x_{i-1} \rangle\}, \\ asgn(\pi_n(x_i)) &= \{\langle X_1, x_1 \rangle, \langle X_2, x_2 \rangle, \ldots, \langle X_i, x_i \rangle\}. \end{aligned}$$

The set of variables associated with OR nodes along path π_n is denoted by $var(\pi_n)$: $var(\pi_n(X_i)) = \{X_1, \ldots, X_{i-1}\}$, $var(\pi_n(x_i)) = \{X_1, \ldots, X_i\}$. The exact parent-child relationship between nodes in the search space are defined as follows:

1. *An OR node, n, labeled by X_i has a child AND node, m, labeled $\langle X_i, x_i \rangle$ iff $\langle X_i, x_i \rangle$ is consistent with the assignment $asgn(\pi_n)$. Consistency is defined relative to the flat constraints.*

2. *An AND node m, labeled $\langle X_i, x_i \rangle$ has a child OR node r labeled Y, iff Y is child of X in the backbone tree $\mathcal{T}$. Each OR arc, emanating from an OR to an AND node is associated with a weight to be defined shortly (see Definition 2.2.6).*

Clearly, if a node n is labeled X_i (OR node) or x_i (AND node), $var(\pi_n)$ is the set of variables mentioned on the path from the root to X_i in the backbone tree, denoted by $path_{\mathcal{T}}(X_i)$[1].

[1]When the AND/OR tree is extended to dynamic variable orderings the set of variables along different paths may vary.

A solution subtree is defined in the usual way:

DEFINITION **2.2.3 (solution subtree)** *A* solution subtree *of an AND/OR search tree contains the root node. For every OR nodes it contains one of its child nodes and for each of its AND nodes it contains all its child nodes, and all its leaf nodes are consistent.*

Example 2.2.2 *In the example of Figure 2.1(a), $\mathcal{T}$ is the DFS tree which is the tree rooted at X, and accordingly the root OR node of the AND/OR tree in 2.1(c) is X. Its child nodes are labeled $\langle X, 1\rangle, \langle X, 2\rangle, \langle X, 3\rangle$ (only the values are noted in the Figure), which are AND nodes. From each of these AND nodes emanate two OR nodes, Y and Z, since these are the child nodes of X in the DFS tree of (2.1(a)). The descendants of Y along the path from the root, $(\langle X, 1\rangle)$, are $\langle Y, 2\rangle$ and $\langle Y, 3\rangle$ only, since $\langle Y, 1\rangle$ is inconsistent with $\langle X, 1\rangle$. In the next level, from each node $\langle Y, y\rangle$ emanate OR nodes labeled T and R and from $\langle Z, z\rangle$ emanate nodes labeled L and M as dictated by the DFS tree. In 2.1(c) a solution tree is highlighted.*

2.2.1 Weights of OR-AND Arcs

The arcs in AND/OR trees are associated with weights w that are defined based on the graphical model's functions and combination operator. The simplest case is that of constraint networks.

DEFINITION **2.2.4 (arc weight for constraint networks)** *Given an AND/OR tree $S_{\mathcal{T}}(\mathcal{R})$ of a constraint network $\mathcal{R}$, each terminal node is assumed to have a single, dummy, outgoing arc. The outgoing arc of a terminal AND node always has the weight "1" (namely it is consistent and thus solved). An outgoing arc of a terminal OR node has weight "0", (there is no consistent value assignments). The weight of any internal OR to AND arc is "1". The arcs from AND to OR nodes have no weight.*

We next define arc weights for any graphical model using the notion of buckets of functions.

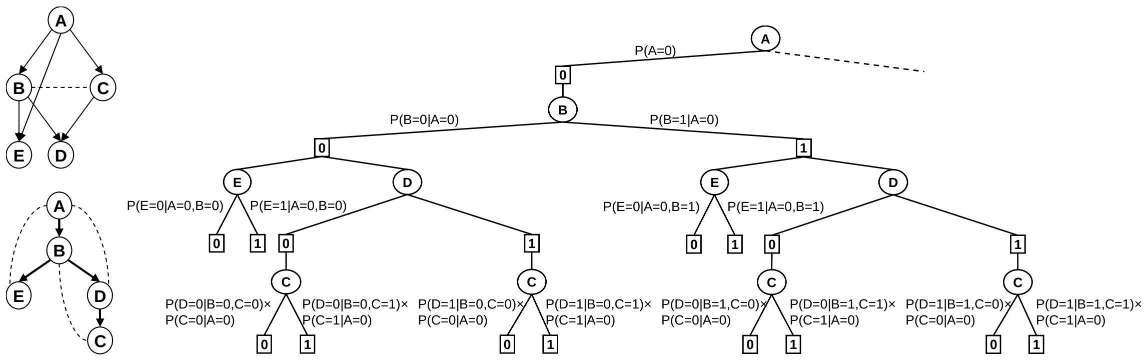

A
B
C
E
D
P(A=0)
P(B=0|A=0)
P(B=1|A=0)
P(E=0|A=0,B=0)
P(E=1|A=0,B=0)
P(E=0|A=0,B=1)
P(E=1|A=0,B=1)
P(D=0|B=0,C=0)× P(C=0|A=0)
P(D=0|B=0,C=1)× P(C=1|A=0)
P(D=1|B=0,C=0)× P(C=0|A=0)
P(D=1|B=0,C=1)× P(C=1|A=0)
P(D=0|B=1,C=0)× P(C=0|A=0)
P(D=0|B=1,C=1)× P(C=1|A=0)
P(D=1|B=1,C=0)× P(C=0|A=0)
P(D=1|B=1,C=1)× P(C=1|A=0)
0
1

Figure 2.2: Arc weights for probabilistic networks

DEFINITION **2.2.5 (buckets relative to a backbone tree)** *Given a graphical model $\mathcal{R} = \langle X, D, F, \bigotimes \rangle$ and a backbone tree $\mathcal{T}$, the* bucket *of X_i relative to $\mathcal{T}$, denoted $B_{\mathcal{T}}(X_i)$, is the set of functions whose scopes contain X_i and are included in $path_{\mathcal{T}}(X_i)$, which is the set of variables from the root to X_i in $\mathcal{T}$. Namely,*

$$B_{\mathcal{T}}(X_i) = \{f \in F | X_i \in scope(f), scope(f) \subseteq path_{\mathcal{T}}(X_i)\}.$$

DEFINITION **2.2.6 (OR-to-AND weights)** *Given an AND/OR tree $S_{\mathcal{T}}(\mathcal{R})$, of a graphical model $\mathcal{R}$, the weight $w_{(n,m)}(X_i, x_i)$ of arc (n, m) where X_i labels n and x_i labels m, is the* combination *of all the functions in $B_{\mathcal{T}}(X_i)$ assigned by values along π_m. Formally, $w_{(n,m)}(X_i, x_i) = \bigotimes_{f \in B_{\mathcal{T}}(X_i)} f(asgn(\pi_m)[scope(f)])$.*

DEFINITION **2.2.7 (weight of a solution subtree)** *Given a weighted AND/OR tree $S_{\mathcal{T}}(\mathcal{R})$, of a graphical model $\mathcal{R}$, and given a solution subtree t having OR-to-AND set of arcs $arcs(t)$, the weight of t is defined by $w(t) = \bigotimes_{e \in arcs(t)} w(e)$.*

Example 2.2.3 *Figure 2.2 shows a belief network, a DFS tree that drives its weighted AND/OR search tree, and a portion of the AND/OR search tree with the appropriate weights on the arcs expressed symbolically. In this case the bucket of E contains the function $P(E|A, B)$, and the bucket of C contains two functions, $P(C|A)$ and $P(D|B, C)$. Note*

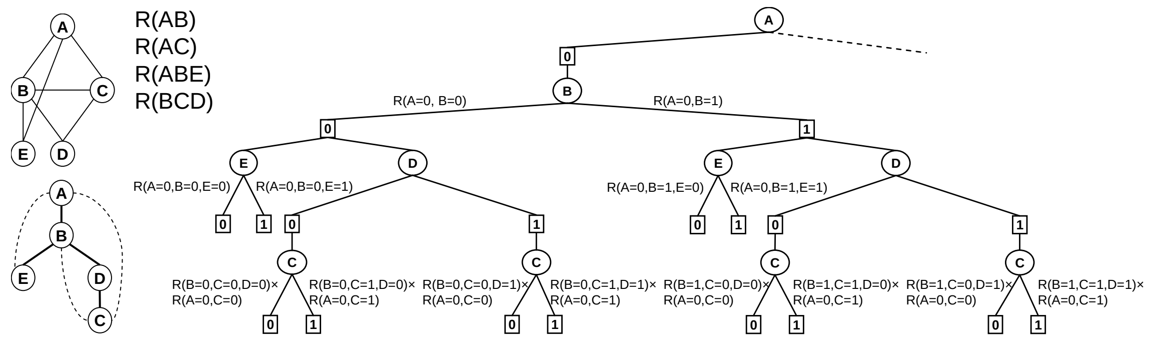

R(AB)
R(AC)
R(ABE)
R(BCD)
R(A=0, B=0)
R(A=0,B=1)
R(A=0,B=0,E=0)
R(A=0,B=0,E=1)
R(A=0,B=1,E=0)
R(A=0,B=1,E=1)
R(B=0,C=0,D=0)× R(A=0,C=0)
R(B=0,C=1,D=0)× R(A=0,C=1)
R(B=0,C=0,D=1)× R(A=0,C=0)
R(B=0,C=1,D=1)× R(A=0,C=1)
R(B=1,C=0,D=0)× R(A=0,C=0)
R(B=1,C=1,D=0)× R(A=0,C=1)
R(B=1,C=0,D=1)× R(A=0,C=0)
R(B=1,C=1,D=1)× R(A=0,C=1)

Figure 2.3: Arc weights for constraint networks

that $P(D|B,C)$ belongs neither to the bucket of B nor to the bucket of D, but it is contained in the bucket of C, which is the last variable in its scope to be instantiated in a path from the root of the tree. We see indeed that the weights on the arcs from the OR node E and any of its AND value assignments include only the instantiated function $P(E|A,B)$, while the weights on the arcs connecting C to its AND child nodes are the products of the two functions in its bucket instantiated appropriately. Figure 2.3 shows a constraint network with four relations, a backbone DFS tree and a portion of the AND/OR search tree with weights on the arcs. Note that the complex weights would reduce to "0"s and "1"s in this case. However, since we use the convention that arcs appear in the search tree only if they represent a consistent extension of a partial solution, we will not see arcs having zero weights.

2.2.2 Properties of AND/OR Search Tree

Any DFS tree $\mathcal{T}$ of a graph G has the property that the arcs of G which are not in $\mathcal{T}$ are backarcs. Namely, they connect a node and one of its ancestors in the backbone tree. This ensures that each scope of F will be fully assigned on some path in $\mathcal{T}$, a property that is essential for the validity of the AND/OR search tree.

THEOREM **2.2.4 (correctness)** *Given a graphical model $\mathcal{R}$ having a primal graph G and a DFS spanning tree $\mathcal{T}$ of G, its weighted AND/OR search tree $S_{\mathcal{T}}(\mathcal{R})$ is sound and complete,* *namely: 1) there is a one-to-one correspondence between solution subtrees of $S_{\mathcal{T}}(\mathcal{R})$ and solutions of $\mathcal{R}$; 2) the weight of any solution tree equals the cost of the full solution it denotes; namely, if t is a solution tree of $S_{\mathcal{T}}(\mathcal{R})$ which denotes a solution $x = (x_1, ...x_n)$ then $c(x) = w(t)$.*

Proof. 1) By definition, all the arcs of $S_{\mathcal{T}}(\mathcal{R})$ are consistent. Therefore, any solution tree of $S_{\mathcal{T}}(\mathcal{R})$ denotes a solution for $\mathcal{R}$ whose assignments are all the labels of the AND nodes in the solution tree. Also, by definition of the AND/OR tree, every solution of $\mathcal{R}$ must corresponds to a solution subtree in $S_{\mathcal{T}}(\mathcal{R})$. 2) By construction, the set of arcs in every solution tree have weights such that each function of F contribute to one and only one weight via the combination operator. Since the total weight of the tree is derived by combination, it yields the cost of a solution. $\square$

The virtue of an AND/OR search tree representation is that its size may be far smaller than the traditional OR search tree. The size of an AND/OR search tree depends on the depth of its backbone DFS tree $\mathcal{T}$. Therefore, DFS trees of smaller depth should be preferred to drive the AND/OR search *tree*. An AND/OR search tree becomes an OR search tree when its DFS tree is a chain.

THEOREM **2.2.5 (size bounds of AND/OR search tree)** *Given a graphical model $\mathcal{R}$, with domains size bounded by k, and a DFS spanning tree $\mathcal{T}$ having depth m and l leaves, the size of its AND/OR search tree $S_{\mathcal{T}}(\mathcal{R})$ is $O(l \cdot k^m)$ (and therefore also $O(nk^m)$ and $O((bk)^m)$ when b bounds the branching degree of $\mathcal{T}$ and n bounds the number of nodes). In contrast the size of its OR search tree along any ordering is $O(k^n)$. The above bounds are tight and realizable for fully consistent graphical models. Namely, one whose all full assignments are consistent.*

Proof. Let p be an arbitrary directed path in the DFS tree $\mathcal{T}$ that starts with the root and ends with a leaf. This path induces an OR search subtree which is included in the AND/OR search tree $S_{\mathcal{T}}$, and its size is $O(k^m)$ when m bounds the path length. The DFS tree $\mathcal{T}$

Table 2.1: OR vs. AND/OR search size, 20 nodes

		OR space		AND/OR space		
treewidth	height	time (sec.)	nodes	time (sec.)	AND nodes	OR nodes
5	10	3.154	2,097,151	0.03	10,494	5,247
4	9	3.135	2,097,151	0.01	5,102	2,551
5	10	3.124	2,097,151	0.03	8,926	4,463
5	10	3.125	2,097,151	0.02	7,806	3,903
6	9	3.124	2,097,151	0.02	6,318	3,159

is covered by l such directed paths, whose lengths are bounded by m. The union of their individual search trees covers the whole AND/OR search tree $S_{\mathcal{T}}$, where every distinct full path in the AND/OR tree appears exactly once, and therefore, the size of the AND/OR search tree is bounded by $O(l \cdot k^m)$. Since $l \leq n$ and $l \leq b^m$, it concludes the proof. □

Table 2.1 demonstrates the size saving of AND/OR vs. OR search spaces for 5 random networks having 20 bivalued variables, 18 CPTs with 2 parents per child and 2 root nodes, when all the assignments are consistent (remember that this is the case when the probability distribution is strictly positive). The size of the OR space is the full binary tree of depth 20. The size of the full AND/OR space varies based on the backbone DFS tree. We can give a better analytic bound on the search space size by spelling out the depth m_i of each leaf node L_i in $\mathcal{T}$.

Proposition 1 *Given a graphical model $\mathcal{R}$, with domains size bounded by k, and a backbone spanning tree $\mathcal{T}$ having $L = \{L_1, \ldots, L_l\}$ leaves, where depth of leaf L_i is m_i, then the size of its full AND/OR search tree $S_{\mathcal{T}}(\mathcal{R})$ is $O(\sum_{k=1}^{l} k^{m_i})$. Alternatively, we can use the exact domain sizes for each variable yielding an even more accurate expression $O(\sum_{L_k \in L} \Pi_{\{X_j | X_j \in path_{\mathcal{T}}(L_k)\}} |D(X_j)|)$.*

Proof. The proof is similar to that of Theorem 2.2.5, only each nodes contributes with its actual domain size rather than the maximal one, and each path to a leaf in $\mathcal{T}$ contributes with its actual depth, rather than the maximal one. □

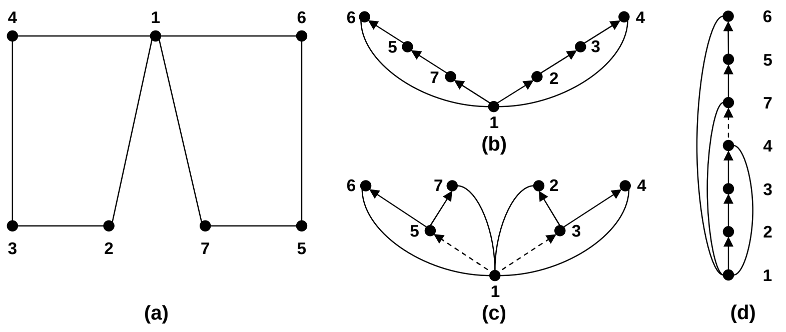


Figure 2.4: (a) A graph; (b) a DFS tree $\mathcal{T}_1$; (c) a pseudo tree $\mathcal{T}_2$; (d) a chain pseudo tree $\mathcal{T}_3$

2.2.3 From DFS Trees to Pseudo Trees

There is a larger class of trees that can be used as backbones for AND/OR search trees, called *pseudo trees* [48]. They have the above mentioned back-arc property.

DEFINITION **2.2.8 (pseudo tree, extended graph)** *Given an undirected graph $G = (V, E)$, a directed rooted tree $\mathcal{T} = (V, E')$ defined on all its nodes is a* pseudo tree *if any arc of G which is not included in E' is a back-arc in $\mathcal{T}$, namely it connects a node in $\mathcal{T}$ to an ancestor in $\mathcal{T}$. The arcs in E' may not all be included in E. Given a pseudo tree $\mathcal{T}$ of G, the* extended graph *of G relative to $\mathcal{T}$ is defined as $G^{\mathcal{T}} = (V, E \cup E')$.*

Clearly, any DFS tree and any chain of a graph are pseudo trees.

Example 2.2.6 *Consider the graph G displayed in Figure 2.4(a). Ordering $d_1 = (1, 2, 3, 4, 7, 5, 6)$ is a DFS ordering of a DFS tree $\mathcal{T}_1$ having the smallest DFS tree depth of 3 (Figure 2.4(b)). The tree $\mathcal{T}_2$ in Figure 2.4(c) is a pseudo tree and has a tree depth of 2 only. The two tree-arcs (1,3) and (1,5) are not in G. Tree $\mathcal{T}_3$ in Figure 2.4(d), is a chain. The extended graphs $G^{\mathcal{T}_1}$, $G^{\mathcal{T}_2}$ and $G^{\mathcal{T}_3}$ are presented in Figure 2.4(b),(c),(d) when we ignore directionality and include the dotted arcs.*

It is easy to see that the weighted AND/OR search tree is well defined when the backbone trees is a pseudo tree. Namely, the properties of soundness and completeness hold and the size bounds are extendible.

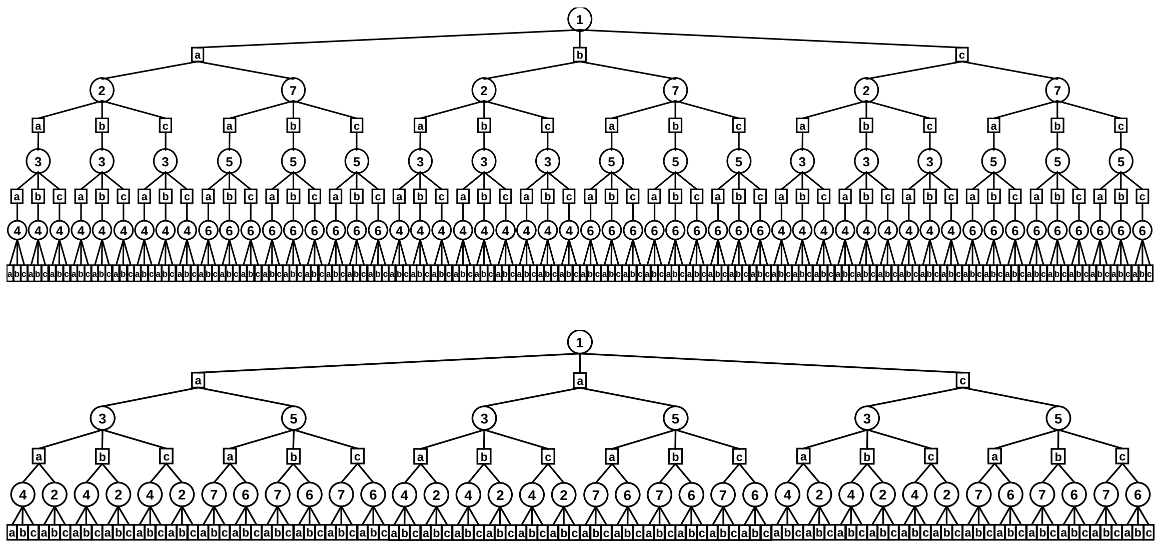


Figure 2.5: AND/OR search tree along pseudo trees $\mathcal{T}_1$ and $\mathcal{T}_2$

THEOREM **2.2.7 (properties of AND/OR search trees)** *Given a graphical model $\mathcal{R}$ and a backbone pseudo tree $\mathcal{T}$, its weighted AND/OR search tree $S_{\mathcal{T}}(\mathcal{R})$ is sound and complete, and its size is $O(l \cdot k^m)$ where m is the depth of the pseudo tree, l bounds its number of leaves, and k bounds the domain size.*

Proof. All the arguments in the proof for Theorem 2.2.4 carry immediately to AND/OR search spaces that are defined relative to a pseudo tree. Likewise, the bound size argument in the proof of Theorem 2.2.5 holds relative to the depth of the more general pseudo tree. $\square$

Example 2.2.8 *Figure 2.5 shows the AND/OR search trees along the pseudo trees $\mathcal{T}_1$ and $\mathcal{T}_2$ from Figure 2.4. Here the domains of the variables are $\{a, b, c\}$ and the constraints are universal. The AND/OR search tree based on $\mathcal{T}_2$ is smaller, because $\mathcal{T}_2$ has a smaller depth than $\mathcal{T}_1$. The weights are not specified here.*

Finding good pseudo trees. Finding a pseudo tree or a DFS tree of minimal depth is known to be NP-complete. However various greedy heuristics are available. For example, pseudo trees can be obtained by generating a heuristically good induced graph along an ordering d and then traversing the induced graph depth-first, breaking ties in favor of earlier variables [6]. For more information see [71, 2].

The definition of buckets relative to a backbone tree extends to pseudo trees as well, and this allows the definitions of weights for an AND/OR tree based on pseudo tree. Next we define the notion of a *bucket tree* and show that it corresponds a pseudo tree. This relationship will be used to make additional connections between various graph parameters.

DEFINITION **2.2.9 (bucket tree [59])** *Given a graphical model, its primal graph G and an ordering d, the* bucket tree *of G along d is defined as follows. Let G_d^* be the induced graph of G along d. Each variable X has an associated* bucket*, denoted by B_X, that contains X and its earlier neighbors in the induced graph G_d^* (similar to Definition 2.2.5). The nodes of the bucket tree are the n buckets. Each node B_X points to B_Y (B_Y is the parent of B_X) if Y is the latest earlier neighbor of X in G_d^*.*

The following relationship between the treewidth and the depth of pseudo trees is known [6, 14]. Given a *tree decomposition* of a primal graph G having n nodes, whose treewidth is w^*, there exists a pseudo tree $\mathcal{T}$ of G whose depth, m, satisfies: $m \leq w^* \cdot \log n$. It can also be shown that any bucket tree [59] yields a pseudo tree and that a min-depth bucket tree yields min-depth pseudo trees. The depth of a bucket tree was also called *elimination depth* in [14].

In summary,

Proposition 2 *[6, 14] The minimal depth m over all pseudo trees satisfies $m \leq w^* \cdot \log n$, where w^* is the treewidth of the primal graph of the graphical model.*

Therefore,

THEOREM **2.2.9** *A graphical model that has a treewidth w^* has an AND/OR search tree whose size is $O(n \cdot k^{(w^* \cdot \log n)})$, where k bounds the domain size and n is the number of variables.*

Table 2.2: Average depth of pseudo trees vs. DFS trees; 100 instances of each random model

Model (DAG)	width	Pseudo tree depth	DFS tree depth
(N=50, P=2, C=48)	9.5	16.82	36.03
(N=50, P=3, C=47)	16.1	23.34	40.60
(N=50, P=4, C=46)	20.9	28.31	43.19
(N=100, P=2, C=98)	18.3	27.59	72.36
(N=100, P=3, C=97)	31.0	41.12	80.47
(N=100, P=4, C=96)	40.3	50.53	86.54

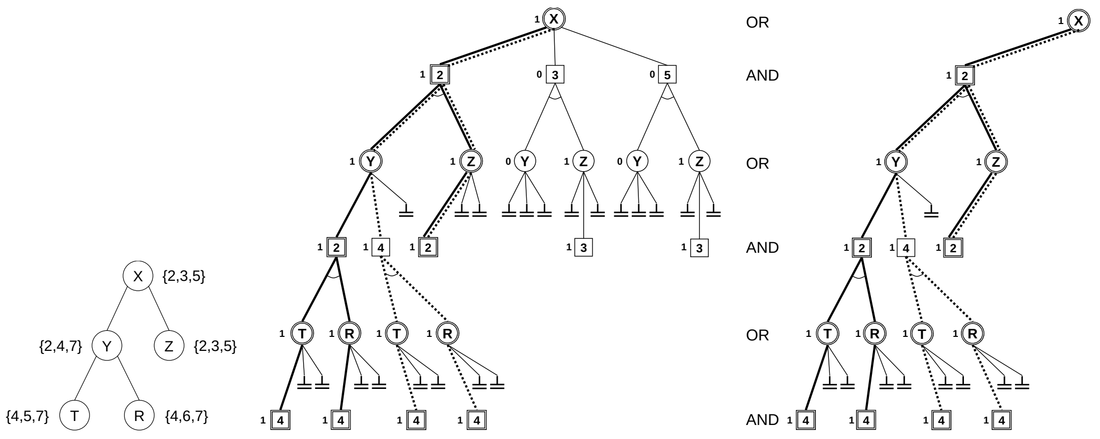
For illustration, Table 2.2 shows the effect of DFS spanning trees against pseudo trees, both generated using brute-force heuristics over randomly generated graphs, where N is the number of variables, P is the number of variables in the scope of a function and C is the number of functions.

2.2.4 Pruning Inconsistent Subtrees for the Flat Constraint Network

Most advanced constraint processing algorithms incorporate no-good learning, and constraint propagation during search, or use variable elimination algorithms such as *adaptive-consistency* and *directional resolution* [31], generating all relevant no-goods, prior to search. Such schemes can be viewed as compiling a representation that would yield a *pruned* search tree. We next define the *backtrack-free* AND/OR search tree.

DEFINITION **2.2.10 (backtrack-free AND/OR search tree)** *Given an AND/OR search tree $S_{\mathcal{T}}(\mathcal{R})$, the* backtrack-free AND/OR search tree *of $\mathcal{R}$ based on $\mathcal{T}$, denoted $BF_{\mathcal{T}}(\mathcal{R})$, is obtained by pruning from $S_{\mathcal{T}}(\mathcal{R})$ all inconsistent subtrees, namely all nodes that root no consistent partial solution.*

Example 2.2.10 *Consider 5 variables X, Y, Z, T, R over domains $\{2, 3, 5\}$, where the constraints are: X divides Y and Z, and Y divides T and R. The constraint graph and the AND/OR search tree relative to the DFS tree rooted at X, are given in Figure 2.6(a). In 2.6(b) we present the $S_{\mathcal{T}}(\mathcal{R})$ search space whose nodes' consistency status (which will latter will be referred to as* values *) are already evaluated having value "1" is consistent*



(a) A constraint tree (b) Search tree (c) Backtrack-free search tree

Figure 2.6: AND/OR search tree and backtrack-free tree

and "0" otherwise. We also highlight two solutions subtrees; one depicted by solid lines and one by dotted lines. Part (c) presents $BF_{\mathcal{T}}(\mathcal{R})$*, where all nodes that do not root a consistent solution are pruned.*

If we traverse the backtrack-free AND/OR search tree we can find a solution subtree without encountering any dead-ends. Some constraint networks specifications yield a backtrack-free search space. Others can be made backtrack-free by massaging their representation using *constraint propagation* algorithms before or during search. In particular, it is well known that variable-elimination algorithms such as *adaptive-consistency* [40] and directional resolution [90], applied in a reversed order of d (where d is the DFS order of the pseudo tree) compile a constraint specification (resp., a Boolean CNF formula) that has a backtrack-free search space. Assuming that the reader is familiar with variable elimination algorithms [29] we define:

DEFINITION **2.2.11 (directional extension [40, 90])** *Let* $\mathcal{R}$ *be a constraint problem and let* d *be a DFS traversal ordering of a backbone pseudo tree of its primal graph, then we denote by* $E_d(\mathcal{R})$ *the constraint network (resp., the CNF formula) compiled by Adaptive-consistency (resp., directional resolution) in reversed order of* d*.*

Proposition 3 *Given a Constraint network* $\mathcal{R}$*, the AND/OR search tree of the directional* *extension $E_d(\mathcal{R})$ when d is a DFS ordering of $\mathcal{T}$, is identical to the backtrack-free AND/OR search tree of $\mathcal{R}$ based on $\mathcal{T}$. Namely $S_{\mathcal{T}}(E_d(\mathcal{R})) = BF_{\mathcal{T}}(\mathcal{R})$.*

Proof. First, we should note that if $\mathcal{T}$ is a pseudo tree of $\mathcal{R}$ and if d is a DFS ordering of $\mathcal{T}$, then $\mathcal{T}$ is also a pseudo tree of $E_d(\mathcal{R})$ and therefore $S_{\mathcal{T}}(E_d(\mathcal{R}))$ is a faithful representation of $E_d(\mathcal{R})$. $E_d(\mathcal{R})$ is equivalent to $\mathcal{R}$, therefore $S_{\mathcal{T}}(E_d(\mathcal{R}))$ is a supergraph of $BF_{\mathcal{T}}(\mathcal{R})$. We only need to show that $S_{\mathcal{T}}(E_d(\mathcal{R}))$ does not contain any dead-ends, in other words any consistent partial assignment must be extendable to a solution of $\mathcal{R}$. Adaptive consistency makes $E_d(\mathcal{R})$ strongly directional $w^*(d)$ consistent, where $w^*(d)$ is the induced width of R along ordering d [40]. It follows from this that either $\mathcal{R}$ is inconsistent, in which case the proposition is trivially satisfied, both trees being empty, or else any consistent partial assignment in $S_{\mathcal{T}}(E_d(\mathcal{R}))$ can be extended to the next variable in d, and therefore no dead-end is encountered. $\square$

Example 2.2.11 *In Example 2.2.10, if we apply adaptive-consistency in reverse order of X, Y, T, R, Z, the algorithm will remove the values $3, 5$ from the domains of both X and Z yielding a tighter constraint network $\mathcal{R}'$. The AND/OR search tree in Figure 2.2.10(c) is both $S_{\mathcal{T}}(\mathcal{R}')$ and $BF_{\mathcal{T}}(\mathcal{R})$.*

Proposition 3 emphasizes the significance of no-good learning [26] for deciding inconsistency or for finding a single solution. These techniques are known as clause learning in SAT solvers, first introduced by [7] and are currently used in most advanced solvers [72]. Namely, when we apply no-good learning we explore the search space whose many inconsistent subtrees are pruned. For counting however, and for other relevant tasks, pruning inconsistent subtrees and searching the backtrack-free search tree yields a partial help only, as we elaborate later.

2.3 AND/OR Search Graphs

It is often the case that a search space that is a tree can become a graph if identical nodes are merged, because identical nodes root identical search subspaces, and correspond to identical reasoning subproblems. Any two nodes that root identical weighted subtrees can be *merged*, reducing the size search graph. For example, in Figure 2.1(c), the search trees below any appearance of $\langle Y, 2\rangle$ are all identical, and therefore can be merged.

Sometimes, two nodes may not root identical subtrees, but they could still root search subspaces that correspond to equivalent subproblems. Nodes that root equivalent subproblems having the same universal model (see Definition 2.3.1) even though the weighted subtrees may not be identical, can be *unified*, yielding an even smaller search graph, as we will show.

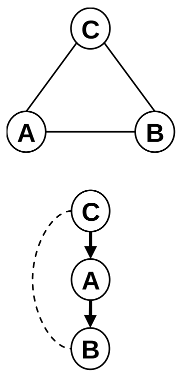
We next formalize the notions of *merging* and *unifying* nodes and define the minimal AND/OR search graph.

2.3.1 Minimal AND/OR Search Graphs

An AND/OR search tree can also be viewed as a data structure that defines a *universal* graphical model (see Definition 1.2.6), defined by the weights of its set of solution subtrees (see Definition 2.2.3).

DEFINITION **2.3.1 (universal graphical model of AND/OR search trees)** *Given a weighted AND/OR search tree $\mathcal{G}$ over a set of variables X and domains D, its* universal graphical model, *denoted by $U(\mathcal{G})$, is defined by its set of solutions as follows: if t is a solution subtree and $x = asgn(t)$ is the set of assignments associated with t then $u(x) = w(t)$; otherwise $u(x) = 0$.*

A graphical model $\mathcal{R}$ is equivalent to its AND/OR search tree, $S_{\mathcal{T}}(\mathcal{R})$, which means that $u(\mathcal{R})$ is identical to $U(S_{\mathcal{T}}(\mathcal{R}))$. We will next define sound merge operations that transform AND/OR search trees into graphs that preserve equivalence.

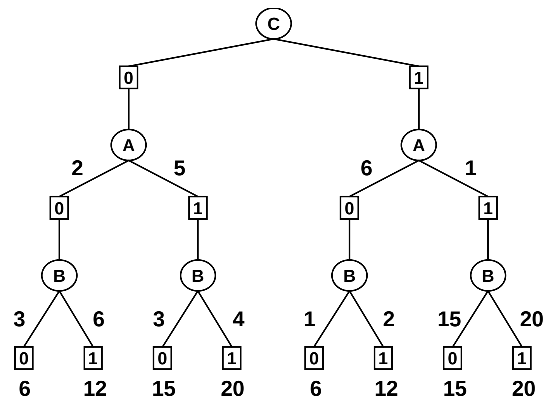


(a)

C	A	B	f(C,A,B)
0	0	0	**3**
0	0	1	**6**
0	1	0	**3**
0	1	1	**4**
1	0	0	**1**
1	0	1	**2**
1	1	0	**15**
1	1	1	**20**

C	A	f(C,A)
0	0	**2**
0	1	**5**
1	0	**6**
1	1	**1**

(b)



(c)

Figure 2.7: Merge vs. unify operators

DEFINITION **2.3.2 (merge)** *Assume a given weighted AND/OR search graph $S'_{\mathcal{T}}(\mathcal{R})$ ($S'_{\mathcal{T}}(\mathcal{R})$ can be the AND/OR search tree $S_{\mathcal{T}}(\mathcal{R})$), and assume two paths $\pi_1 = \pi_{n_1}(x_i)$ and $\pi_2 = \pi_{n_2}(x_i)$ ending by AND nodes at level i having the same label x_i. Nodes n_1 and n_2 can be* merged *iff the weighted search subgraphs rooted at n_1 and n_2 are identical. The* merge *operator, $merge(n_1, n_2)$, redirects all the arcs going into n_2 into n_1 and removes n_2 and its subgraph. It thus transforms $S'_{\mathcal{T}}$ into a smaller graph. When we merge AND nodes only we call the operation AND-merge. The same reasoning can be applied to OR nodes, and we call the operation OR-merge.*

We next define the semantic notion of *unifiable* nodes, as opposed to the syntactic definition of *merge*.

DEFINITION **2.3.3 (unify)** *Given a weighted AND/OR search graph $\mathcal{G}$ for a graphical model $\mathcal{R}$ and given two paths π_{n_1} and π_{n_2} having the same label on nodes n_1 and n_2, then n_1 and n_2 are* unifiable, *iff they root equivalent conditioned subproblems (Definition 1.2.8). Namely, if $\mathcal{R}|_{asgn(\pi_1)} = \mathcal{R}|_{asgn(\pi_2)}$.*

Example 2.3.1 *Let's follow the example in Figure 2.7 to clarify the difference between* merge *and* unify. *We have a graphical model defined by two functions (*e.g. *cost functions) over three variables. The search tree given in Figure 2.7(c) cannot be reduced to a graph* *by* merge*, because of the different arc weights. However, the two OR nodes labeled A root equivalent conditioned subproblems (the cost of each individual solution is given at the leaves). Therefore, the nodes labeled A can be* unified*, but they cannot be recognized as identical by the* merge *operator.*

Proposition 4 (minimal graph) *Given a weighted AND/OR search graph $\mathcal{G}$ based on pseudo tree $\mathcal{T}$:*

1. *The* merge *operator has a unique fix point, called the* **merge-minimal** *AND/OR search graph and denoted by $M_{\mathcal{T}}^{merge}(\mathcal{G})$.*

2. *The* unify *operator has a unique fix point, called the* **unify-minimal** *AND/OR search graph and denoted by $M_{\mathcal{T}}^{unify}(\mathcal{G})$.*

3. *Any two nodes n_1 and n_2 of $\mathcal{G}$ that can be merged can also be unified.*

Proof. (1) All we need to show is that the *merge* operator is not dependant on the order of applying the operator. Mergeable nodes can only appear at the same level in the AND/OR graph. Looking at the initial AND/OR graph, before the merge operator is applied, we can identify all the mergeable nodes per level. We prove the proposition by showing that if two nodes are initially mergeable, then they must end up merged after the operator is applied exhaustively to the graph. This can be shown by induction over the level where the nodes appear.

Base case: If the two nodes appear at the leaf level (level 0), then it is obvious that the exhaustive merge has to merge them at some point.

Inductive step: Suppose our claim is true for nodes up to level k and two nodes n_1 and n_2 at level $k+1$ are initially identified as mergeable. This implies that, initially, their corresponding children are identified as mergeable. These children are at level k, so it follows from the inductive hypothesis that the exhaustive merge has to merge the corresponding children. This in fact implies that nodes n_1 and n_2 will root the same subgraph when the exhaustive merge ends, so they have to end up merged. Since the graph only becomes smaller by merging, based on the above the process of merging has to stop at a fix point. (2) Analogous to (1). (3) If the nodes can be merged, it follows that the subgraphs are identical, which implies that they define the same conditioned subproblems, and therefore the nodes can also be unified. □

DEFINITION **2.3.4 (minimal AND/OR search graph)** *The unify-minimal AND/OR search graph of $\mathcal{R}$ relative to $\mathcal{T}$ will also be simply called the **minimal AND/OR search graph** and be denoted by $M_{\mathcal{T}}(\mathcal{R})$.*

When $\mathcal{T}$ is a chain pseudo tree, the above definitions are applicable to the traditional OR search tree as well. However, we may not be able to reach the same compression as in some AND/OR cases, because of the linear structure imposed by the OR search tree.

Example 2.3.2 *The smallest OR search graph of the graph-coloring problem in Figure 2.1(a) is given in Figure 2.9 along the DFS order X, Y, T, R, Z, L, M. The smallest AND/OR graph of the same problem along the DFS tree is given in Figure 2.11. We see that some variable-value pairs (AND nodes) must be repeated in Figure 2.9 while in the AND/OR case they appear just once. In particular, the subgraph below the paths $(\langle X, 1\rangle, \langle Y, 2\rangle)$ and $(\langle X, 3\rangle, \langle Y, 2\rangle)$ in the OR tree cannot be merged at $\langle Y, 2\rangle$. You can now compare all the four search space representations side by side in Figures 2.8-2.11.*

Note that in the case of constraint networks we can accommodate an even more general definition of merging of two AND nodes that are assigned *different* values from their domain, or two OR nodes labeled by different variables, as long as they root identical subgraphs. In that case the merged node should be labeled by the disjunction of the two assignments (this is similar to interchangeable values [101]).

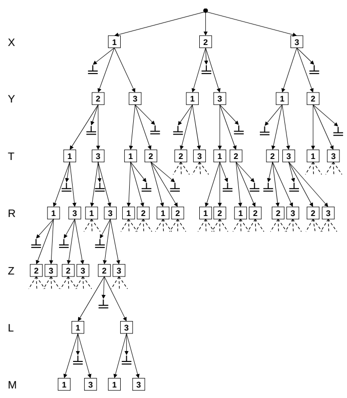


Figure 2.8: OR search tree for the tree problem in Figure 2.1(a)

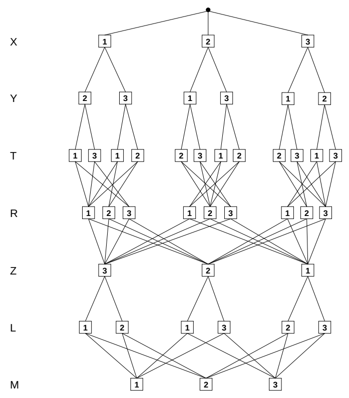


Figure 2.9: The minimal OR search graph of the tree graphical model in Figure 2.1(a)

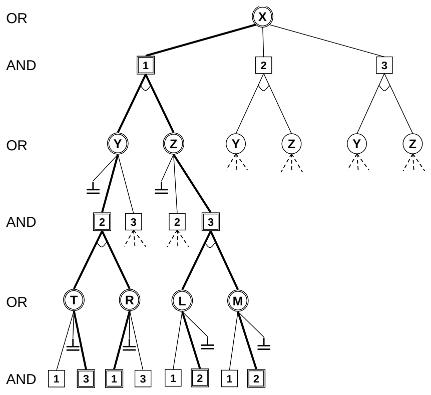


Figure 2.10: AND/OR search tree for the tree problem in Figure 2.1(a)

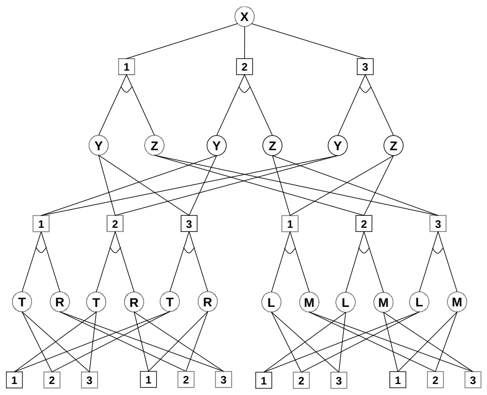


Figure 2.11: The minimal AND/OR search graph of the tree graphical model in Figure 2.1(a)

2.3.2 Building AND/OR Search Graphs

In this subsection we will discuss practical algorithms for generating compact AND/OR search graphs of a given graphical model. In particular we will identify effective rules for recognizing unifiable nodes, aiming towards the minimal AND/OR search graph as much as computational resources allow. The rules allow generating a small AND/OR graph called *the context minimal graph* without creating the whole search tree $S_{\mathcal{T}}$ first. We focus first on AND/OR search graphs of graphical models having no cycles, called *tree models* (*i.e.*, the primal graph is a tree).

Building AND/OR search graphs for Tree Models and Tree Decompositions

Consider again the graph in Figure 2.1(a) and its AND/OR search tree in Figure 2.1(c) representing a constraint network. Observe that at level 3, node $\langle Y, 1\rangle$ appears twice, (and so are $\langle Y, 2\rangle$ and $\langle Y, 3\rangle$). Clearly however, the subtrees rooted at each of these two AND nodes are identical and we can reason that they can be merged because any specific assignment to Y uniquely determines its rooted subtree. Indeed, the AND/OR search graph in Figure 2.11 is equivalent to the AND/OR search tree in Figure 2.8 (same as Figure 2.1(c)).

DEFINITION **2.3.5 (explicit AND/OR graphs for constraints tree models)** *Given a tree model constraint network and the pseudo tree $\mathcal{T}$ identical to its primal graph, the* explicit AND/OR search graph *of the tree model relative to $\mathcal{T}$ is obtained from $S_{\mathcal{T}}$ by merging all AND nodes having the same label $\langle X, x\rangle$.*

Proposition 5 *Given a rooted tree model $\mathcal{T}$: (1) Its explicit AND/OR search* graph *is equivalent to $S_{\mathcal{T}}$. (2) The size of the explicit AND/OR search graph is $O(nk)$. (3) For some tree models the explicit AND/OR search graph is minimal.*

Proof. Parts 1 and 2 follow from definitions. Regarding claim 3, for the graph coloring problem in Figure 2.1(a), the minimal AND-OR search graph is identical to its explicit AND/OR search graph, $G_{\mathcal{T}}$. (See Figure 2.11). $\square$

The notion of explicit AND/OR search graph for a tree model is extendable to any general graphical models that are trees. The only difference is that the arcs have weights. Thus, we need to show that merged nodes via the rule in definition 2.3.5 root identical weighted AND/OR trees.

Proposition 6 *Given a general graphical model whose graph is a tree $\mathcal{T}$, its explicit AND/OR search* graph *is equivalent to $S_{\mathcal{T}}$, and its size is $O(nk)$;*

Proof. In tree models, the functions are only over two variables. Therefore, after an assignment $\langle X, x\rangle$ is made and the appropriate weight is given to the arc from X to $\langle X, x\rangle$, the variable X and all its ancestors in the pseudo tree do not contribute to any arc weight below in the AND/OR search tree. Therefore, the conditioned subproblems rooted at any AND node labeled by $\langle X, x\rangle$ depend only on the assignment of X to x (and do not depend on any other assignment on the current path), so it follows that all the AND nodes labeled by $\langle X, x\rangle$ can be merged. Since the equivalence of AND/OR search spaces is preserved by merge, the explicit AND/OR search graph is equivalent to $S_{\mathcal{T}}$. At each AND level in the explicit graph there are at most k values, and therefore its size is $O(nk)$. $\square$

Next, the question is how to identify *efficiently* mergeable nodes for *general* non-tree graphical models. A guiding idea is to transform a graphical model into a tree decomposition first, and then apply the explicit AND/OR graph construction to the resulting tree decomposition. The next paragraph sketches this intuition.

A *tree decomposition* [59] (see Definition 1.2.3) of a graphical model partitions the functions into clusters. Each cluster corresponds to a subproblem that has a set of solutions and the clusters interact in a tree-like manner. Once we have a tree decomposition of a graphical model, it can be viewed as a regular (meta) tree model where each cluster is a node and its domain is the cross product of the domains of variables in the cluster. The constraint between two adjacent nodes in the tree decomposition is equality over the common variables. For more details about tree decompositions see [59]. For the meta-tree model the explicit AND/OR search graph is well defined: the OR nodes are the scopes of clusters in the tree decomposition and the AND nodes, are their possible value assignments. Since the graphical model is converted into a tree, its explicit AND/OR search graph is well defined and we can bound its size.

THEOREM **2.3.3** *Given a* tree decomposition *of a graphical model, whose domain sizes are bounded by k, the* explicit AND/OR search graph *implied by the tree decomposition has a size of $O(rk^{w^*})$, where r is the number of clusters in the tree decomposition and w^* is the size of the largest cluster.*

Proof. The size of an explicit AND/OR *graph* of a tree model was shown to be $O(n \cdot k)$ (Proposition 5), yielding, $O(r \cdot k^{w^*})$ size for the explicit AND/OR graph, because k is replaced by k^{w^*}, the number of possible assignments to a cluster of scope size w^*, and r replaces n. $\square$

The tree decomposition can guide an algorithm for generating an AND/OR search graph whose size is bounded exponentially by the induced width, which we will refer to in the next section as the *context minimal graph*.

While the idea of explicit AND/OR graph based on a tree decomposition can be extended to any graphical model it is somewhat cumbersome. Instead, in the next section we propose a more direct approach for generating the context minimal graph.

The Context Based AND/OR Graph

We will now present a generative rule for merging nodes in the AND/OR search graph that yields the size bound suggested above. We will need the notion of *induced width of a pseudo tree of G* for bounding the size of the AND/OR search *graphs*. We denote by $d_{DFS}(\mathcal{T})$ a linear DFS ordering of a tree $\mathcal{T}$.

DEFINITION **2.3.6 (induced width of a pseudo tree)** *The induced width of G relative to the pseudo tree $\mathcal{T}$, $w_{\mathcal{T}}(G)$, is the induced width along the $d_{DFS}(\mathcal{T})$ ordering of the extended graph of G relative to $\mathcal{T}$, denoted $G^{\mathcal{T}}$.*

Proposition 7 *(1) The minimal induced width of G over all pseudo trees is identical to the induced width (treewidth), w^*, of G. (2) The minimal induced width restricted to chain pseudo trees is identical to its pathwidth, pw^*.*

Proof. (1) The induced width of G relative to a given pseudo tree is always greater than w^*, by definition of w^*. It remains to show that there exists a pseudo tree $\mathcal{T}$ such that $w_{\mathcal{T}}(G) = w^*$. Consider an ordering d that gives the induced width w^*. The ordering d

defines a bucket tree BT (see Definition 2.2.9), which can also be viewed as a pseudo tree for the AND/OR search, therefore $w_{BT}(G) = w^*$. (2) Analogous to (1). $\square$

Example 2.3.4 *In Figure 2.4(b), the induced graph of G relative to $\mathcal{T}_1$ contains also the induced arcs (1,3) and (1,5) and its induced width is 2. $G^{\mathcal{T}_2}$ is already triangulated (no need to add induced arcs) and its induced width is 2 as well. $G^{\mathcal{T}_3}$ has the added arc (4,7) and when ordered it will have the additional induced arcs (1,5) and (1,3) edges, yielding induced width 2 as well.*

We will now provide definitions that will allow us to identify nodes that can be merged in an AND/OR graph. The idea is to find a minimal set of variable assignments from the current path that will always generate the same conditioned subproblem, regardless of the assignments that are not included in this minimal set. Since the current path for an OR node X_i and an AND node $\langle X_i, x_i \rangle$ differ by the assignment of X_i to x_i (Definition 2.2.2), the minimal set of assignments that we want to identify will be different for X_i and for $\langle X_i, x_i \rangle$. In the following two definitions ancestors and descendants are with respect to the pseudo tree $\mathcal{T}$, while connection is with respect to the primal graph G.

DEFINITION **2.3.7 (parents)** *Given a primal graph G and a pseudo tree $\mathcal{T}$ of a reasoning problem $\mathcal{P}$, the* parents *of an OR node X_i, denoted by pa_i or pa_{X_i}, are the ancestors of X_i that have connections in G to X_i or to descendants of X_i.*

DEFINITION **2.3.8 (parent-separators)** *Given a primal graph G and a pseudo tree $\mathcal{T}$ of a reasoning problem $\mathcal{P}$, the* parent-separators *of X_i (or of $\langle X_i, x_i \rangle$), denoted by pas_i or pas_{X_i}, are formed by X_i and its ancestors that have connections in G to descendants of X_i.*

It follows from these definitions that the parents of X_i, pa_i, separate in the primal graph G (and also in the extended graph $G^{\mathcal{T}}$ and in the induced extended graph $G^{\mathcal{T}^*}$) the ancestors (in $\mathcal{T}$) of X_i, from X_i and its descendants (in $\mathcal{T}$). Similarly, the parents separators of X_i,

pas_i, separate the ancestors of X_i from its descendants. It is also easy to see that each variable X_i and its parents pa_i form a clique in the induced graph $G^{\mathcal{T}^*}$. The following proposition establishes the relationship between pa_i and pas_i.

Proposition 8 *1. If Y is the single child of X in $\mathcal{T}$, then $pas_X = pa_Y$.*

2. If X has children $Y_1, \ldots, Y_k$ in $\mathcal{T}$, then $pas_X = \cup_{i=1}^k pa_{Y_i}$.

Proof. Both claims follow directly from Definitions 2.3.7 and 2.3.8. □

THEOREM **2.3.5 (context based merge)** *Given $G^{\mathcal{T}^*}$, let π_{n_1} and π_{n_2} be any two partial paths in an AND/OR search graph, ending with two nodes, n_1 and n_2.*

1. *If n_1 and n_2 are AND nodes annotated by $\langle X_i, x_i \rangle$ and*

$$asgn(\pi_{n_1})[pas_{X_i}] = asgn(\pi_{n_2})[pas_{X_i}] \tag{2.1}$$

   *then the AND/OR search subtrees rooted by n_1 and n_2 are identical and n_1 and n_2 can be merged. $asgn(\pi_{n_i})[pas_{X_i}]$ is called the **AND context** of n_i.*

2. *If n_1 and n_2 are OR nodes annotated by X_i and*

$$asgn(\pi_{n_1})[pa_{X_i}] = asgn(\pi_{n_2})[pa_{X_i}] \tag{2.2}$$

   *then the AND/OR search subtrees rooted by n_1 and n_2 are identical and n_1 and n_2 can be merged. $asgn(\pi_{n_i})[pa_{X_i}]$ is called the **OR context** of n_i.*

Proof. (1) The conditioned graphical models (Definition 1.2.8) at n_1 and n_2 are defined by the functions whose scopes are not fully assigned by π_{n_1} and π_{n_2}. Since n_1 and n_2 have the same labeling $\langle X_i, x_i \rangle$, it follows that $var(\pi_{n_1}) = var(\pi_{n_2})$, and therefore the two conditioned subproblems are based on the same set of functions, let's call it $F|_{var(\pi_{n_1})}$.

The scopes of functions in $F|_{var(\pi_{n_1})}$ determine connections in the primal graph between ancestors of X_i and its descendants. Therefore, the only relevant variables that define the restricted subproblems are those in pas_i, and equation 2.1 ensures that they have identical assignments. It follows that the conditioned subproblems are identical, and n_1 and n_2 can be merged.
(2) Analogous to (1). $\square$

Example 2.3.6 *For the balanced tree in Figure 2.1 consider the chain pseudo tree* $d = (X, Y, T, R, Z, L, M)$. *Namely the chain has arcs* $\{(X, Y), (Y, T), (T, R), (R, Z), (Z, L), (L, M)\}$ *and the extended graph includes also the arcs* $(Z, X), (M, Z)$. *The parent-separator of* T *along* d *is* XYT *(since the induced graph has the arc* (T, X)*), of* R *it is* XR*, for* Z *it is* Z *and for* M *it is* M. *Indeed in the first 3 levels of the OR search graph in Figure 2.9 there are no merged nodes. In contrast, if we consider the AND/OR ordering along the DFS tree, the parent-separator of every node is itself yielding a single appearance of each AND node having the same assignment annotation in the minimal AND/OR graph.*

DEFINITION **2.3.9 (context minimal AND/OR search graph)** *The AND/OR search graph of* $\mathcal{R}$ *based on the backbone tree* $\mathcal{T}$ *that is closed under context-based merge operator is called* context minimal *AND/OR search graph and is denoted* $C_{\mathcal{T}}(\mathcal{R})$.

We should note that we can in general merge nodes based both on AND and OR contexts. However, Proposition 8 shows that doing just one of them renders the other unnecessary (up to some small constant factor). In practice, we would recommend just the OR context based merging, because it has a slight (albeit by a small constant factor) space advantage. In the examples that we give in this chapter, $C_{\mathcal{T}}(\mathcal{R})$ refers to an AND/OR search graph for which either the AND context based or OR context based merging was performed exhaustively.

Example 2.3.7 *Consider the example given in Figure 2.12(a). The OR context of each node in the pseudo tree is given in square brackets. The context minimal AND/OR search graph (based on OR merging) is given in Figure 2.12(b).*

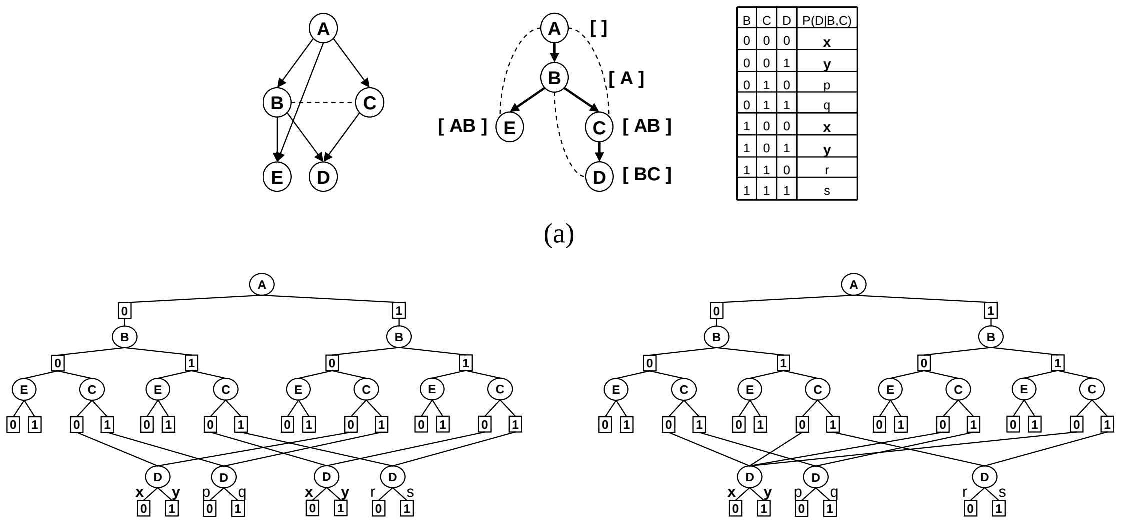
Since the number of nodes in the context minimal AND/OR search graph cannot exceed the number of different contexts, we can bound the size of the context minimal graph.

THEOREM **2.3.8** *Given a graphical model $\mathcal{R}$, its primal graph G, and a pseudo tree $\mathcal{T}$ having induced width $w = w_{\mathcal{T}}(G)$, the size of the context minimal AND/OR search graph based on $\mathcal{T}$, $C_{\mathcal{T}}(\mathcal{R})$, is $O(n \cdot k^w)$, when k bounds the domain size.*

Proof. The number of different nodes in the context minimal AND/OR search graph, $C_{\mathcal{T}}$, does not exceed the number of contexts. From equations 2.1 and 2.2 we see that, for any variable, the number of contexts is bounded by the number of possible instantiations of the largest context in $G^{\mathcal{T}^*}$, which is bounded by $O(k^w)$. For all the n variables, the bound $O(n \cdot k^w)$ follows. $\square$

Note that the criterion in equations 2.1 and 2.2 is cautious. First, the real number of assignments over context variables includes only consistent assignments. Second, we have already seen (Example 2.3.1) that there exist nodes that can be *unified* but not *merged*. Here we give an example that shows that contexts can not identify all the nodes that can be *merged*. There could be paths whose contexts are not identical, yet they might root identical subgraphs.

Example 2.3.9 *Let's return to the example of the Bayesian network given in Figure 2.12(a), where $P(D|B,C)$ is given in the table, and the OR-context of each node in the pseudo tree is given in square brackets. Figure 2.12(b) shows the context minimal graph. However, we can see that $P(D=0|B=0,C=0) = P(D=0|B=1,C=0) = x$ and $P(D=1|B=0,C=0) = P(D=1|B=1,C=0) = y$. This allows the* unification *of the corresponding OR nodes labeled with D, and Figure 2.12(c) shows the (unify) minimal graph.*



B	C	D	P(D|B,C)
0	0	0	**x**
0	0	1	**y**
0	1	0	p
0	1	1	q
1	0	0	**x**
1	0	1	**y**
1	1	0	r
1	1	1	s

(a)

(b) (c)

Figure 2.12: Context minimal vs. minimal AND/OR graphs

The context based merge offers a powerful way of bounding the search complexity:

THEOREM **2.3.10** *The context minimal AND/OR search graph $C_{\mathcal{T}}$ of a graphical model having a backbone tree with bounded treewidth w can be generated in time and space $O(nk^w)$.*

Proof. We can generate $C_{\mathcal{T}}$ using depth-first or breadth first search which caches all nodes via their contexts and avoids generating duplicate searches for the same contexts. Therefore, the generation of the search graph is linear in its size, which is exponential in w and linear in n. $\square$

Since the unify minimal AND/OR graph $M_{\mathcal{T}}^{unify}$ and the merge minimal AND/OR graph $M_{\mathcal{T}}^{merge}$ are subsets of $C_{\mathcal{T}}$, both are bounded by $O(n \cdot k^w)$, where $w = w_{\mathcal{T}}(G)$. Since $\min_{\mathcal{T}}\{w_{\mathcal{T}}(G)\}$ is equal to the treewidth w^* and since $\min_{\mathcal{T} \in chains}\{w_{\mathcal{T}}(G)\}$ is equal to the pathwidth pw^*, we get:

Corollary 1 *Given a graphical model $\mathcal{R}$, there exists a backbone tree $\mathcal{T}$ such that the unify minimal, merge minimal and context minimal AND/OR search graphs of $\mathcal{R}$ are bounded exponentially by the treewidth of the primal graph. The unify, merge and context minimal OR search graphs can be bounded exponentially by the pathwidth only.*

More on OR vs. AND/OR

It is well known [14] that for any graph $w^* \leq pw^* \leq w^* \cdot \log n$. It is easy to place m^* (the minimal depth over pseudo trees) in that relation yielding $w^* \leq pw^* \leq m^* \leq w^* \cdot \log n$. It is also possible to show that there exist primal graphs for which the upper bound on pathwidth is attained, that is $pw^* = O(w^* \cdot \log n)$.

Consider a complete binary tree of depth m. In this case, $w^* = 1$, $m^* = m$, and it is also known [91, 12]) that:

THEOREM **2.3.11 ([12])** *If $\mathcal{T}$ is a binary tree of depth m then $pw^*(\mathcal{T}) \geq \frac{m}{2}$.*

Theorem 2.3.11 shows that for graphical models having a bounded tree width w, the minimal AND/OR graph is bounded by $O(nk^w)$ while the minimal OR graph is bounded by $O(nk^{w \cdot \log n})$. Therefore, even when caching, the use of an AND/OR vs. an OR search space can yield a substantial saving.

Remark. We have seen that AND/OR *trees* are characterized by the *depth* of the pseudo trees while minimal AND/OR *graphs* are characterized by their *induced width*. It turns out however that sometimes a pseudo tree that is optimal relative to w is far from optimal for m and vice versa. For example a primal graph model that is a chain has a pseudo tree having $m_1 = n$ and $w_1 = 1$ on one hand, and another pseudo tree that is balanced having $m_2 = \log n$ and $w_2 = \log n$. There is no single pseudo tree having both $w = 1$ and $m = \log n$ for a chain. Thus, if we plan to have linear space search we should pick one kind of a backbone pseudo tree, while if we plan to search a graph, and therefore cache some nodes, another pseudo tree should be used.

2.3.3 On the Canonicity and Generation of the Minimal AND/OR Graph

We showed that the merge minimal AND/OR graph is unique for a given graphical model, given a backbone pseudo tree (Proposition 4). In general, it subsumes the minimal

AND/OR graph, and sometimes can be identical to it. For constraint networks we will now prove a more significant property of uniqueness relative to all equivalent graphical models given a backbone tree. We will prove this notion relative to *backtrack-free* search graphs which are captured by the notion of strongly minimal AND/OR graph. Remember that any graphical model can have an associated flat constraint network.

DEFINITION **2.3.10 (strongly minimal AND/OR graph)** [2] *A* strongly minimal *AND/OR graph of $\mathcal{R}$ relative to a pseudo tree $\mathcal{T}$ is the minimal AND/OR graph, $M_{\mathcal{T}}(\mathcal{R})$, that is backtrack-free (*i.e. *any partial assignment in the graph leads to a solution), denoted by $M^*_{\mathcal{T}}(\mathcal{R})$. The strongly context minimal graph is denoted $C^*_{\mathcal{T}}(\mathcal{R})$.*

Canonicity of Strongly Minimal AND/OR Search Graphs

We briefly discuss here the canonicity of the strongly minimal graph, focusing on constraint networks. Given two equivalent constraint networks representing the same set of solutions, where each may have a different constraint graph, are their strongly minimal AND/OR search graphs identical?

The above question is not well defined however, because an AND/OR graph for $\mathcal{R}$ is defined only with respect to a backbone pseudo tree. We can have two equivalent constraint networks having two different graphs where a pseudo tree for one graph may not be a pseudo tree for the other. Consider, for example a constraint network having three variables: X, Y and Z and equality constraints. The following networks, $\mathcal{R}_1 = \{R_{XY} = (X = Y),\ R_{YZ} = (Y = Z)\}$ and $\mathcal{R}_2 = \{R_{XZ} = (X = Z),\ R_{YZ} = (Y = Z)\}$ and $\mathcal{R}_3 = \{R_{XY} = (X = Y),\ R_{YZ} = (Y = Z),\ R_{XZ} = (X = Z)\}$ are equivalent. However, $\mathcal{T}_1 = (X \leftarrow Y \rightarrow Z)$ is a pseudo tree for $\mathcal{R}_1$, but not for $\mathcal{R}_2$ neither for $\mathcal{R}_3$. We ask therefore a different question: given two equivalent constraint networks and given a backbone tree that is a pseudo tree for both, is the strongly minimal AND/OR graph relative to $\mathcal{T}$ unique?

[2]The minimal graph is built by lumping together "unifiable" nodes, which are those that root equivalent subproblems. Therefore, at each level (corresponding to one variable), all the nodes that root inconsistent subproblems will be unified. If we eliminate the redundant nodes, the minimal graph is already backtrack free.

We will answer this question positively quite straightforwardly. We first show that equivalent networks that share a backbone tree have identical backtrack-free AND/OR search trees. Since the backtrack-free search trees uniquely determine their strongly minimal graph the claim follows.

DEFINITION **2.3.11 (shared pseudo trees)** *Given a collection of graphs on the same set of nodes, we say that the graphs share a tree $\mathcal{T}$, if $\mathcal{T}$ is a pseudo tree of each of these graphs. A set of graphical models defined over the same set of variables share a tree $\mathcal{T}$, iff their respective primal graphs share $\mathcal{T}$.*

Proposition 9 *1. If $\mathcal{R}_1$ and $\mathcal{R}_2$ are two equivalent constraint networks that share $\mathcal{T}$, then $BF_{\mathcal{T}}(\mathcal{R}_1) = BF_{\mathcal{T}}(\mathcal{R}_2)$ (see Definition 2.2.10). 2. If $\mathcal{R}_1$ and $\mathcal{R}_2$ are two equivalent graphical models (not necessarily constraint networks) that share $\mathcal{T}$, then $BF_{\mathcal{T}}(\mathcal{R}_1) = BF_{\mathcal{T}}(\mathcal{R}_2)$ as AND/OR search trees although their arcs may not have identical weights.*

Proof. Let $B_1 = BF_{\mathcal{T}}(\mathcal{R}_1)$ and $B_2 = BF_{\mathcal{T}}(\mathcal{R}_2)$ be the corresponding backtrack-free AND/OR search trees of $\mathcal{R}_1$ and $\mathcal{R}_2$, respectively. Namely, $BF_{\mathcal{T}}(\mathcal{R}_1) \subseteq S_{\mathcal{T}}(\mathcal{R}_1)$, $BF_{\mathcal{T}}(\mathcal{R}_2) \subseteq S_{\mathcal{T}}(\mathcal{R}_2)$. Clearly they are subtrees of the same full AND/OR tree. We claim that a path appears in B_1 iff it appears in B_2. If not, assume without loss of generality that there exists a path in B_1, π, which does not exists in B_2. Since this is a backtrack-free search tree, every path appears in some solution and therefore there is a solution subtree in B_1 that includes π which does not exist in B_2, contradicting the assumption that $\mathcal{R}_1$ and $\mathcal{R}_2$ have the same set of solutions. The second part has an identical proof based on flat functions (namely positive values of a function are associated with 1 and indicate allowed tuples, and zero values remain 0). $\square$

THEOREM **2.3.12** *If $\mathcal{R}_1$ and $\mathcal{R}_2$ are two equivalent constraint networks that share $\mathcal{T}$, then $M^*_{\mathcal{T}}(\mathcal{R}_1) = M^*_{\mathcal{T}}(\mathcal{R}_2)$.*

Proof. From Proposition 9 we know that $\mathcal{R}_1$ and $\mathcal{R}_2$ have the same backtrack-free AND/OR tree. Since the backtrack-free AND/OR search tree for a backbone tree $\mathcal{T}$ uniquely determines the strongly minimal AND/OR graph, the theorem follows. □

Theorem 2.3.12 implies that $M^*_{\mathcal{T}}$ is a canonical representation of a constraint network $\mathcal{R}$ relative to $\mathcal{T}$.

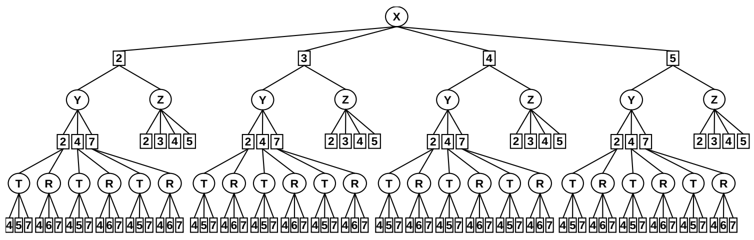
Generating the strongly minimal AND/OR graphs

From the above discussion we see that several methods for generating the canonical AND/OR graph of a given graphical model, or a given AND/OR graph may emerge. The method we focused on in this chapter is to generate the context minimal AND/OR graph first. Then we can process this graph from leaves to root, while computing the value of nodes, and additional nodes can be unified or pruned (if their value is “0”).

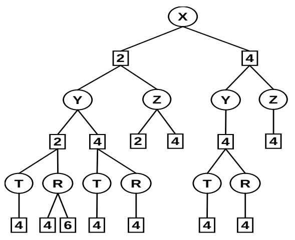
There is another approach that is based on processing the functions in a variable elimination style, when viewing the pseudo tree as a bucket tree or a cluster tree. The original functions can be expressed as AND/OR graphs and they will be combined pairwise until an AND/OR graph is generated. This phase allows computing the value of each node and therefore allows for semantic unification. Subsequently a forward phase will allow generating the backtrack-free representation as well as allow computing the full values associated with each node. The full details of this approach are out of the scope of the current chapter. For initial work restricted to constraint networks see [76].

2.3.4 Merging and Pruning: Orthogonal Concepts

Notice that the notion of minimality is orthogonal to that of pruning inconsistent subtrees (yielding the backtrack-free search space). We can merge two identical subtrees whose root value is “0” but still keep their common subtree. However, since our convention is that we don’t keep inconsistent subtrees we should completely prune them, irrespective of them rooting identical or non-identical subtrees. Therefore, we can have a minimal search graph that is *not* backtrack-free as well as a non-minimal search graph (*e.g.* a tree) that is

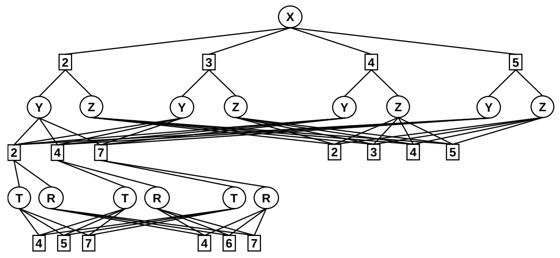


(a) Full AND/OR tree

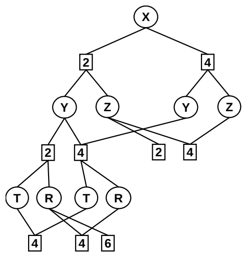


(b) Pruned backtrack-free AND/OR tree

Figure 2.13: AND/OR trees



(a) Context minimal unpruned AND/OR graph



(b) Context minimal pruned backtrack-free AND/OR graph

Figure 2.14: AND/OR graphs

backtrack-free.

When the search space is backtrack-free and if we seek a single solution, the size of the minimal AND/OR search graph and its being OR vs. AND/OR are both irrelevant. It will, however, affect a traversal algorithm that counts all solutions or computes an optimal solution as was often observed [50]. For counting and for optimization tasks, even when we record all no-goods and cache all nodes by context, the impact of the AND/OR graph search vs. the OR graph search can still be significant.

Example 2.3.13 *Consider the graph problem in Figure 2.6(a) when we add the value 4 to the domains of X and Z. Figure 2.13(a) gives the full AND/OR search tree and Figure 2.13(b) gives the backtrack-free search tree. Figure 2.14(a) gives the context minimal but unpruned search graph and Figure 2.14(b) gives the minimal and pruned search graph.*

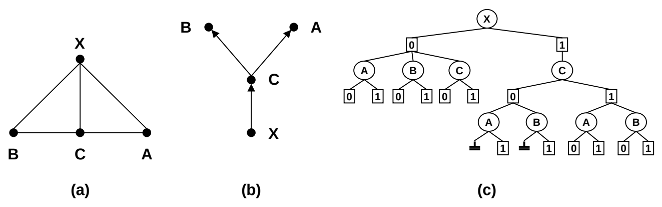


Figure 2.15: (a) A constraint graph; (b) a spanning tree; (c) a dynamic AND/OR tree

2.3.5 Using Dynamic Variable Ordering

The AND/OR search tree we defined uses a fixed variable ordering. It is known that exploring the search space in a dynamic variable ordering is highly beneficial. AND/OR search trees for graphical models can also be modified to allow dynamic variable ordering. A dynamic AND/OR tree that allows varied variable ordering has to satisfy that for every subtree rooted by the current path π, any arc of the primal graph that appears as a cross-arc (not a back-arc) in the subtree must be "inactive" conditioned on π.

Example 2.3.14 *Consider the propositional formula $X \rightarrow A \vee C$ and $X \rightarrow B \vee C$. The constraint graph is given in Figure 2.15(a) and a DFS tree in 2.15(b). However, the constraint subproblem conditioned on $\langle X, 0\rangle$, has no real constraint between A, B, C, so the effective spanning tree below $\langle X, 0\rangle$ is $\{\langle X, 0\rangle \rightarrow A, \langle X, 0\rangle \rightarrow B, \langle X, 0\rangle \rightarrow C\}$, yielding the AND/OR search tree in Figure 2.15(c). Note that while there is an arc between A and C in the constraint graph, the arc is* not *active when X is assigned the value 0.*

Clearly, the constraint graph conditioned on any partial assignment can only be sparser than the original graph and therefore may yield a smaller AND/OR search tree than with fixed ordering. In practice, after each new value assignment, the conditional constraint graph can be assessed as follows. For any constraint over the current variable X, if the current assignment $\langle X, x\rangle$ does not make the constraint *active* then the corresponding arcs can be removed from the graph. Then, a pseudo tree of the resulting graph is generated, its first variable is selected, and search continues. A full investigation of dynamic orderings is outside the scope of the current chapter.

2.4 Solving Reasoning Problems by AND/OR Search

2.4.1 Value Functions of Reasoning Problems

As we described earlier, there are a variety of reasoning problems over weighted graphical models. For constraint networks, the most popular tasks are to decide if the problem is consistent, to find a single solution or to count solutions. If there is a cost function defined we may also seek an optimal solution. The primary tasks over probabilistic networks are belief updating, finding the probability of the evidence and finding the most likely tuple given the evidence. Each of these reasoning problems can be expressed as finding the *value* of some nodes in the weighted AND/OR search space where different tasks call for different value definitions. For example, for the task of finding a solution to a constraint network, the value of every node is either "1" or "0". The value "1" means that the subtree rooted at the node is consistent and "0" otherwise. Therefore, the value of the root node answers the consistency query. For solutions-counting the value function of each node is the number of solutions rooted at that node.

DEFINITION **2.4.1 (value function for consistency and counting)** *Given a weighted AND/OR tree $S_{\mathcal{T}}(\mathcal{R})$ of a constraint network. The value of a node (AND or OR) for* deciding consistency *is "1" if it roots a consistent subproblem and "0" otherwise. The value of a node (AND or OR) for* counting solutions *is the number of solutions in its subtree.*

It is easy to see that the value of nodes in the search graph can be computed recursively from leaves to root.

Proposition 10 (recursive value computation) *(1) For the consistency task the value of AND leaves is their labels and the value of OR leaves is "0" (they are inconsistent). An internal OR node is labeled "1" if one of its successor nodes is "1" and an internal AND node has value "1" iff all its successor OR nodes have value "1".*

(2) The counting values of leaf AND nodes are "1" and of leaf OR nodes are "0". The counting value of an internal OR node is the sum of the counting-values of all its child nodes. The counting-value of an internal AND node is the product of the counting-values of all its child nodes.

Proof. The proof is by induction over the number of levels in the AND/OR graph.
Basis step: If the graph has only two levels, one OR and one AND, then the claim is straightforward because the AND leaves are labeled by "1" if consistent and the OR node accumulates "1" or the sum of consistent values below, or "0" if there is no consistent value.
Inductive step: Assuming the proposition holds for k pairs of levels (one AND and one OR in each pair), proving it holds for $k+1$ pairs of levels is similar to the basis step, only the labeling of the top AND nodes is the sum of solutions below in the case of counting. □

We can now generalize to any reasoning problem, focusing on the simplified case when $Z = \emptyset$, namely when the marginalization has to be applied to all the variables. This special case captures most tasks of interest. We will start with the recursive definition.

DEFINITION **2.4.2 (recursive definition of values)** *The value function of a reasoning problem $\mathcal{P} = \langle \mathcal{R}, \Downarrow_Y, Z\rangle$, where $\mathcal{R} = \langle X, D, F, \bigotimes\rangle$ and $Z = \emptyset$, is defined as follows: the value of leaf AND nodes is "1" and of leaf OR nodes is "0". The value of an internal OR node is obtained by* combining *the value of each AND child node with the weight (see Definition 2.2.6) on its incoming arc and then* marginalizing *over all AND children. The value of an AND node is the combination of the values of its OR children. Formally, if $children(n)$ denotes the children of node n in the AND/OR search graph, then:*

$$v(n) = \bigotimes_{n' \in children(n)} v(n'), \qquad \text{if } n = \langle X, x\rangle \text{ is an AND node,}$$

$$v(n) = \Downarrow_{n' \in children(n)} (w_{(n,n')} \bigotimes v(n')), \qquad \text{if } n = X \text{ is an OR node.}$$

The following proposition states that given a reasoning task, computing the value of the root node solves the given reasoning problem.

Proposition 11 *Let $\mathcal{P} = \langle \mathcal{R}, \Downarrow_Y, Z \rangle$, where $\mathcal{R} = \langle X, D, F, \bigotimes \rangle$ and $Z = \emptyset$, and let X_1 be the root node in any AND/OR search graph $S'_{\mathcal{T}}(\mathcal{R})$. Then $v(X_1) = \Downarrow_X \bigotimes_{i=1}^{r} f_i$ when v is defined in Definition 2.4.2.*

Proof. The proof is again by induction, similar to the proof of Proposition 10.

Basis step: If the model has only one variable, then the claim is obvious.

Inductive step: Let X be an OR node in the graph. Assume that the value of each OR node below it is the solution to the reasoning problem corresponding to the conditioned subproblem rooted by it. We need to prove that the value of X will be the solution to the reasoning problem of the conditioned subproblem rooted by X. Suppose X has children $Y_1, \ldots, Y_m$ in the pseudo tree. We have $v(Y_i) = \Downarrow_{Y_i \cup Desc(Y_i)} \bigotimes_{f \in F|_{\pi_{Y_i}}} f$, where $Desc(Y_i)$ are the descendants of Y_i, and the functions are restricted on the current path. Each AND node $\langle X, x \rangle$ will combine the values below. Because the sets $Y_i \cup Desc(Y_i)$ are pairwise disjoint, the marginalization operator commutes with the combination operator and we get:

$$v(\langle X, x \rangle) \;=\; \bigotimes_{i=1}^{m} \Downarrow_{Y_i \cup Desc(Y_i)} \bigotimes_{f \in F|_{\pi_{Y_i}}} f \;=\; \Downarrow_{\bigcup_{i=1}^{m}(Y_i \cup Desc(Y_i))} \bigotimes_{f \in F|_{\pi_x}} f.$$

The values $v(\langle X, x \rangle)$ are then combined with the values of the bucket of X, which are the weights $w_{(X, \langle X, x \rangle)}$. The functions that appear in the bucket of X do not contribute to any of the weights below Y_i, and therefore the marginalization over $\bigcup_{i=1}^{m}(Y_i \cup Desc(Y_i))$ can commute with the combination that we have just described:

$$w_{(X, \langle X, x \rangle)} \bigotimes v(\langle X, x \rangle) \;=\; \Downarrow_{\bigcup_{i=1}^{m}(Y_i \cup Desc(Y_i))} w_{(X, \langle X, x \rangle)} \bigotimes (\bigotimes_{f \in F|_{\pi_x}} f).$$

Finally, we get:

$$v(X) \;=\; \Downarrow_X w_{(X, \langle X, x \rangle)} \bigotimes v(\langle X, x \rangle) \;=\; \Downarrow_{X \cup Desc(X)} \bigotimes_{f \in F|_{\pi_X}} f. \quad \square$$

Search algorithms that traverse the AND/OR search space can compute the value of the root node yielding the answer to the problem. The following section discusses such algorithms. Algorithms that traverse the weighted AND/OR search tree in a depth-first manner or a breadth-first manner are guaranteed to have time bound exponential in the depth of the pseudo tree of the graphical model. Depth-first searches can be accomplished using either linear space only, or context based caching, bounded exponentially by the treewidth of the pseudo tree. Depth-first search is an anytime schemes and can, if terminated, provide an approximate solution for some tasks such as optimization. The next subsection presents typical depth-first algorithms that search AND/OR trees and graphs. We use *solution counting* as an example for a constraint query and the probability of evidence as an example for a probabilistic reasoning query. The algorithms compute the value of each node. For application of these ideas for combinatorial optimization tasks, such as MPE see [71].

2.4.2 Algorithm AND/OR Tree Search and Graph Search

Algorithm 1 presents the basic depth-first traversal of the AND/OR search tree (or graph, if caching is used) for counting the number of solutions of a constraint network, AO-COUNTING (or for probability of evidence for belief networks, AO-BELIEF-UPDATING).

The context based caching is done based on tables. We exemplify with OR caching. For each variable X_i, a table is reserved in memory for each possible assignment to its parent set pa_i. Initially each entry has a predefined value, in our case “-1”. The fringe of the search is maintained on a stack called `OPEN`. The current node is denoted by `n`, its parent by `p`, and the current path by π_n. The children of the current node are denoted by $successors(\mathtt{n})$.

The algorithm is based on two mutually recursive steps: EXPAND and PROPAGATE, which call each other (or themselves) until the search terminates.

Since we only use OR caching, before expanding an OR node, its cache table is checked (line 6). If the same context was encountered before, it is retrieved from cache, and $successors(\mathtt{n})$ is set to the empty set, which will trigger the PROPAGATE step.

If a node is not found in cache, it is expanded in the usual way, depending on whether it is an AND or OR node (lines 10-17). The only difference between counting and belief updating is line 12 vs. line 13. For counting, the value of a consistent AND node is initialized to 1 (line 12), while for belief updating, it is initialized to the bucket value for the current assignment (line 13). As long as the current node is not a dead-end and still has unevaluated successors, one of its successors is chosen (which is also the top node on OPEN), and the expansion step is repeated.

The bottom up propagation of values is triggered when a node has an empty set of successors (note that as each successor is evaluated, it is removed from the set of successors in line 31). This means that all its children have been evaluated, and its final value can now be computed. If the current node is the root, then the search terminates with its value (line 20). If it is an OR node, its value is saved in cache before propagating it up (line 22). If n is OR, then its parent p is AND and p updates its value by multiplication with the value of n (line 24). If the newly updated value of p is 0 (line 25), then p is a dead-end, and none of its other successors needs to be evaluated. An AND node n propagates its value to its parent p in a similar way, only by summation (line 30). Finally, the current node n is set to its parent p (line 32), because n was completely evaluated. The search continues either with a propagation step (if conditions are met) or with an expansion step.

2.4.3 General AND-OR Search - AO(i)

General AND/OR algorithms for evaluating the value of a root node for any reasoning problem using tree or graph AND/OR search are identical to the above algorithms when product is replaced by the combination operator and summation is replaced by the marginalization operator. We can view the AND/OR tree algorithm (which we will denote AOT) and the AND/OR graph algorithm (denoted AOG) as two extreme cases in a parameterized collection of algorithms that trade space for time via a controlling parameter i. We denote this class of algorithms as $AO(i)$ where i determines the size of contexts that the algorithm caches. Algorithm $AO(i)$ records nodes whose context size is i or smaller (the test in line 22 needs to be a bit more elaborate and check if the context size is smaller than i). Thus AO(0) is identical to AOT, while $AO(w)$ is identical to AOG, where w is the induced width of the used backbone tree. For any intermediate i we get an intermediate level of caching,

Algorithm 1: AO-COUNTING / AO-BELIEF-UPDATING

```
input  : A constraint network $\mathcal{R} = \langle X, D, C\rangle$, or a belief network $\mathcal{P} = \langle X, D, P\rangle$; a pseudo tree $\mathcal{T}$ rooted at $X_1$;
         parents $pa_i$ (OR-context) for every variable $X_i$; caching set to $true$ or $false$.
output : The number of solutions, or the updated belief, $v(X_1)$.
1  if caching $== true$ then                                                    // Initialize cache tables
2      Initialize cache tables with entries of "$-1$"
3  $v(X_1) \leftarrow 0$; OPEN $\leftarrow \{X_1\}$                                      // Initialize the stack OPEN
4  while OPEN $\neq \phi$ do
5      n $\leftarrow top($OPEN$)$; remove n from OPEN
6      if caching $== true$ and n is OR, labeled $X_i$ and $Cache(asgn(\pi_n)[pa_i]) \neq -1$ then   // In cache
7          $v($n$) \leftarrow Cache(asgn(\pi_n)[pa_i])$                                // Retrieve value
8          $successors($n$) \leftarrow \phi$                                          // No need to expand below
9      else                                                                     // EXPAND
10         if n is an OR node labeled $X_i$ then                                    // OR-expand
11             $successors($n$) \leftarrow \{\langle X_i, x_i\rangle \mid \langle X_i, x_i\rangle$ is consistent with $\pi_n$ \}
12             $v(\langle X_i, x_i\rangle) \leftarrow 1$,    for all $\langle X_i, x_i\rangle \in successors($n$)$
13             $v(\langle X_i, x_i\rangle) \leftarrow \prod_{f \in B_{\mathcal{T}}(X_i)} f(asgn(\pi_n)[pa_i])$,   for all $\langle X_i, x_i\rangle \in successors($n$)$   // AO-BU
14         if n is an AND node labeled $\langle X_i, x_i\rangle$ then                      // AND-expand
15             $successors($n$) \leftarrow children_{\mathcal{T}}(X_i)$
16             $v(X_i) \leftarrow 0$ for all $X_i \in successors($n$)$
17         Add $successors($n$)$ to top of OPEN
18     while $successors($n$) == \phi$ do                                           // PROPAGATE
19         if n is an OR node labeled $X_i$ then
20             if $X_i == X_1$ then                                              // Search is complete
21                 return $v($n$)$
22             if caching $== true$ then
23                 $Cache(asgn(\pi_n)[pa_i]) \leftarrow v($n$)$                             // Save in cache
24             $v($p$) \leftarrow v($p$) * v($c$)$
25             if $v($p$) == 0$ then                                              // Check if p is dead-end
26                 remove $successors($p$)$ from OPEN
27                 $successors($p$) \leftarrow \phi$
28         if n is an AND node labeled $\langle X_i, x_i\rangle$ then
29             let p be the parent of n
30             $v($p$) \leftarrow v($p$) + v($n$)$;
31         remove n from $successors($p$)$
32         n $\leftarrow$ p
```

which is space exponential in i and whose execution time will increase as i decreases.

2.4.4 Complexity

From Theorems 2.2.7 and 2.2.9 we can conclude that:

THEOREM **2.4.1** *For any reasoning problem,* AOT *runs in linear space and time $O(nk^m)$, when m is the depth of the pseudo tree of its graphical model and k is the maximum domain size. If the primal graph has a tree decomposition with treewidth w^*, there exists a pseudo tree $\mathcal{T}$ for which AOT is $O(nk^{w^* \cdot \log n})$.*

Obviously, the algorithm for constraint satisfaction, that would terminate early with first solution, would potentially be much faster than the rest of the AOT algorithms, in practice.

Based on Theorem 2.3.8 we get complexity bounds for graph searching algorithms.

THEOREM **2.4.2** *For any reasoning problem, the complexity of algorithm* AOG *is time and space $O(nk^w)$ where w is the induced width of the pseudo tree and k is the maximum domain size.*

Thus the complexity of AOG can be time and space exponential in the treewidth, while the complexity of any algorithm searching the OR space can be time and space exponential in its pathwidth.

The space complexity can often be less than exponential in the treewidth. This is similar to the well known space complexity of tree decomposition schemes which can operate in space exponential only in the size of the cluster separators, rather than exponential in the cluster size. It is also similar to the *dead caches* concept presented in [23, 2]. Intuitively, a node that has only one incoming arc will only be traversed once by search, and therefore its value does not need to be cached, because it will never be used again. For context based caching, such nodes can be recognized based only on the parents (or parent separators) sets.

DEFINITION **2.4.3 (dead cache)** *If X is the parent of Y in $\mathcal{T}$, and $pa_X \subset pa_Y$, then pa_Y is a* dead cache.

Given a pseudo tree $\mathcal{T}$, the induced graph along $\mathcal{T}$ can generate a tree decomposition based on the maximal cliques. The maximum separator size of the tree decomposition is the separator size of $\mathcal{T}$.

Proposition 12 *The space complexity of graph-caching algorithms can be reduced to being exponential in the separator's size only, while still being time exponential in the treewidth, if dead caches are not recorded.*

Proof. A bucket tree can be built by having a cluster for each variable X_i and its parents pa_i, and following the structure of the pseudo tree $\mathcal{T}$. Some of the clusters may not be maximal, and they have a one to one correspondence to the variables with dead caches. The parents pa_i that are not dead caches correspond to separators between maximal clusters in the bucket tree. $\square$

2.5 Related Work

2.5.1 Relationship with Variable Elimination

In Chapter 6 we extend the results in [74] and show that Variable Elimination can be understood as bottom up layer by layer traversal of the context minimal AND/OR search graph. If the graphical model is strictly positive (has no determinism), then context based AND/OR search and Variable Elimination are essentially identical. When determinism is present, they may differ, because they traverse the AND/OR graph in different directions and encounter determinism (and can take advantage of it) differently. Therefore, for graphical models with no determinism, there is no principled difference between memory-intensive AND/OR search with fixed variable ordering and inference beyond: (1) different direction of exploring a common search space (top down for search vs. bottom up for inference); (2) different assumption of control strategy (depth-first for search and breadth-first for inference).

Another interesting observation discussed in [74] is that many known advanced algorithms for constraint processing and satisfiability can be explained as traversing the AND/OR search tree, *e.g.* graph based backjumping [49, 26, 6]. For more details we refer the reader to [74].

2.5.2 Relationship with BTD (Backtracking with Tree-Decomposition)

BTD [99] is a memory intensive method for solving constraint satisfaction problems, which combines search techniques with the notion of tree decomposition. This mixed approach can in fact be viewed as searching an AND/OR graph, whose backbone pseudo tree is defined by and structured along the tree decomposition. What is defined in [99] as *structural goods*, that is parts of the search space that would not be visited again as soon as their consistency is known, corresponds precisely to the decomposition of the AND/OR space at the level of AND nodes, which root independent subproblems. Not surprisingly, the time and space guarantees of BTD are the same as those of AND/OR graph search. An optimization version of the algorithm is presented in [98].

2.5.3 Relationship with Recursive Conditioning

Recursive Conditioning (RC) [23] is based on the divide and conquer paradigm. Rather than instantiating variables to obtain a tree structured network like the cycle cutset scheme, RC instantiates variables with the purpose of breaking the network into independent subproblems, on which it can recurse using the same technique. The computation is driven by a data-structure called *dtree*, which is a full binary tree, the leaves of which correspond to the network CPTs.

It can be shown that RC explores an AND/OR space. Let's start with the example in Figure 2.16, which shows: (a) a belief network; (b) and (c), two dtrees and the corresponding pseudo-trees for the AND/OR search. The dtrees also show the variables that are instantiated at some of the internal nodes. The pseudo-trees can be generated from the static ordering of RC dictated by the dtree. This ensures that whenever RC splits the problem into independent subproblems, the same happens in the AND/OR space. It can also be shown that the context of the nodes in RC, as defined in [23] is identical to that in

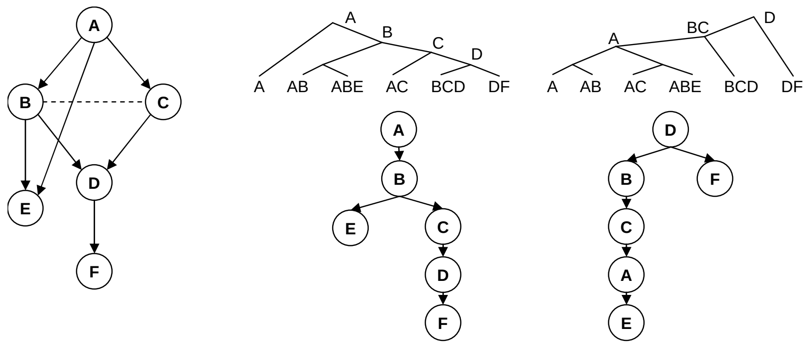


Figure 2.16: RC and AND/OR pseudo-trees

AND/OR.

2.5.4 Relationship with Value Elimination

Value Elimination [5] is a recently developed algorithm for Bayesian inference. It was already explained in [5] that, under static variable ordering, there is a strong relation between Value Elimination and Variable Elimination. From our paragraph on the relation between AND/OR search and VE we can derive the connection between Value Elimination and AND/OR search, under static orderings. But we can also analyze the connection directly. Given a static ordering d for Value Elimination, we can show that it actually traverses an AND/OR space. The pseudo-tree underlying the AND/OR search graph traversal by Value Elimination can be constructed as the bucket tree in reversed d. However, the traversal of the AND/OR space will be controlled by d, advancing the frontier in a hybrid depth or breadth first manner.

The most important part to analyze is the management of *goods* and the computation in which they are involved. Most backtracking algorithms for satisfiability and constraint processing exploit *no-goods*, or inconsistencies, by detecting and learning them, and then deriving new no-goods that help further prune the search space. For probabilistic networks (or weighted models in general) consistent tuples have an attached probability (or weight). The consistent tuples, and their associated probabilities are called *goods* in this context.

When Value Elimination computes a factor at a leaf node, it backs up the value to the deepest node in the dependency set `Dset`. The `Dset` is identical to the context in the AND/OR space. For clarity reasons, we chose to have the AND/OR algorithm back up the value to its parent in the pseudo-tree, which may be different than the deepest variable in the context. We can however accommodate the propagation of the value like in Value Elimination, and maintain bookkeeping of the summation set `Sset`, and this would amount to a constant factor saving. Value Elimination continues by unionizing `Dsets` and `Ssets` whenever values are propagated, and this is identical to computing the context of the corresponding node in the AND/OR space (which is in fact the induced ancestor set of graph-based backjumping [33]).

In the presence of determinism, any backjumping strategy and nogood learning used by Value Elimination can also be performed in the AND/OR space. Context specific structure that can be used by Value Elimination, can also be used in AND/OR. Dynamic variable orderings can also be used in AND/OR spaces, but here we limit the discussion to static orderings.

2.5.5 Relationship with Case-Factor Diagrams

Case-Factor Diagrams (CFD) were introduced in [80] and represent a probabilistic formalism subsuming Markov random fields of bounded treewidth and probabilistic context free grammars. Case-factor diagrams are based on a variant of BDDs (binary decision diagram [16]) with both zero suppression and “factor nodes”. Factor nodes are analogous to the AND nodes in an AND/OR search space. A case-factor diagram can be viewed as an AND/OR search space in which each outgoing arc from an OR node is explicitly labeled with an assignment of a value to a variable. Zero suppression is used to fix the value of variables not mentioned in a given solution. Zero suppression allows the formalism to concisely represent probabilistic context free grammars as functions from variable-value assignments to log probabilities (or energies).

2.5.6 AO-Search Graphs and Compilation

We dedicate Chapter 7 to presenting a compilation scheme based on AND/OR search spaces. Essentially, the AND/OR Multi-Valued Decision Diagrams (AOMDDs) is the strongly minimal AND/OR graph representation of a graphical model with redundant variables removed for conciseness. We will present two algorithms for compiling an AOMDD. The first is based on AND/OR search, and applies reduction rules to the trace of the search (i.e., the context minimal graph). The second algorithm is based on a Variable Elimination schedule. It uses a bottom up traversal of a bucket tree, and at each node an APPLY operator is used to combine all the AOMDDs of the bucket into another AOMDD. The APPLY is similar to the OBDD apply operator [16], but is adapted for AND/OR structures. The AOMDD extends an OBDD (or multi-valued decision diagram) with an AND/OR structure. We discuss further the relationship between AOMDDs and other compilation schemes.

Relationship with d-DNNF

An AND/OR structure restricted to propositional theories is very similar to d-DNNF [25]. One can show a one-to-one linear translation from an AND/OR bi-valued tree of a propositional CNF theory into a d-DNNF. The AND/OR structure is more restrictive allowing disjunction only on the variable's value while in d-DNNF disjunction is allowed on more complex expressions; see [55] for implications of this distinction. The AND/OR search graph is built on top of a graphical model and can be viewed as a compiled scheme of a CNF into an AND/OR structure. Since an AND/OR search can be expressed as a d-DNNF, the construction via pseudo tree yields a scheme for d-DNNF compilation. In other words, given a CNF theory, the algorithm can be applied using a pseudo tree to yield an AND/OR graph, which can be transformed in linear time and space into a d-DNNF.

Conversely, given a d-DNNF that is specialized to variable-based disjunction for OR nodes, it is easy to create an AND/OR graph or a tree that is equivalent having a polynomially equivalent size. The AND/OR search graph for probabilistic networks is also closely related to algebraic circuits of probabilistic networks [24] which is an extension of d-DNNF to this domain.

Relationship with OBDDs

The notion of minimal OR search graphs is also similar to the known concept of *Ordered Binary Decision Diagrams (OBDD)* in the literature of hardware and software design and verification The properties of OBDDs were studied extensively in the past two decades [16, 81].

It is well known that the size of the minimal OBDD is bounded exponentially by the *pathwidth* of the CNF's primal graph and that the OBDD is unique for a fixed variable ordering. Our notion of backtrack-free minimal AND/OR search graphs, if applied to CNFs, resembles *tree BDDs* [82]. Minimal AND/OR graphs are also related to Graph-driven BDDs (called G-FBDD) [51, 96] in that they are based on a partial order expressed in a directed graph. Still, a G-FBDD has an OR structure, whose ordering is restricted to some partial orders, but not an AND/OR structure. For example, the OBDD based on a DFS ordering of a pseudo tree is a G-FBDD. Some other relationships between graphical model compilation and OBDDs were studied in [25].

In summary, putting OBDDs within our terminology, an OBDD representation of a CNF formula is a strongly minimal OR search graph where redundant nodes are removed.

Relationship with Tree Driven Automata

Fargier and Vilarem [46] proposed the compilation of CSPs into tree-driven automata, which have many similarities to the work in [76]. In particular, the compiled tree-automata proposed there is essentially the same as the AND/OR multi-valued decision diagram. Their main focus is the transition from linear automata to tree automata (similar to that from OR to AND/OR), and the possible savings for tree-structured networks and hyper-trees of constraints due to decomposition. Their compilation approach is guided by a tree-decomposition while ours is guided by a variable-elimination based algorithms. And, it is well known that Variable Elimination and cluster-tree decomposition are in principle, the same [41].

Relationship with Disjoint Support Decomposition

The work on Disjoint Support Decompositions (DSD) [8] was proposed in the area of design automation [15], as an enhancement for BDDs aimed at exploiting function decomposition. The main common aspect of DSD and AOMDD [76] is that both approaches show how structure decomposition can be exploited in a BDD-like representation. DSD is focused on Boolean functions and can exploit more refined structural information that is inherent to Boolean functions. In contrast, AND/OR BDDs assume only the structure conveyed in the constraint graph, and are therefore more broadly applicable to any constraint expression and also to graphical models in general. They allow a simpler and higher level exposition that yields graph-based bounds on the overall size of the generated AOMDD.

Relationship with Semi-Ring BDDs

In recent work [101] OBDDs were extended to semi-ring BDDs. The semi-ring treatment is restricted to the OR search spaces, but allows dynamic variable ordering. It is otherwise very similar in aim and scope to our strongly minimal AND/OR graphs. When restricting the strongly minimal AND/OR graphs to OR graphs only, the two are closely related, except that we express BDDs using the Shenoy-Shafer axiomatization that is centered on the two operation of combination and marginalization rather then on the semi-ring formulation. Minimality in the formulation in [101] is more general allowing merging nodes having different values and therefore can capture symmetries (called interchangeability).

2.6 Conclusion to Chapter 2

The primary contribution of this chapter is in viewing search for graphical models in the context of AND/OR search spaces rather than OR spaces. We introduced the AND/OR search tree, and showed that its size can be bounded exponentially by the depth of its pseudo tree over the graphical model. This implies exponential savings for any linear space algorithms traversing the AND/OR search tree. Specifically, if the graphical model has treewidth w^*, the depth of the pseudo tree is $O(w^* \cdot \log n)$.

The AND/OR search tree was extended into a graph by merging identical subtrees. We showed that the size of the minimal AND/OR search graph is exponential in the treewidth while the size of the minimal OR search graph is exponential in the pathwidth. Since for some graphs the difference between treewidth and pathwidth is substantial (*e.g.*, balanced pseudo trees) the AND/OR representation implies substantial time and space savings for memory intensive algorithms traversing the AND/OR graph. Searching the AND/OR search *graph* can be implemented by goods caching during search, while no-good recording is interpreted as pruning portions of the search space independent of it being a tree or a graph, an OR or an AND/OR. For finding a single solution, pruning the search space is the most significant action. For counting and probabilistic inference, using AND/OR graphs can be of much help even on top of no-good recording.

Chapter 3

Mixed Networks

3.1 Introduction

Modeling real-life decision problems requires the specification and reasoning with probabilistic and deterministic information. The primary approach developed in artificial intelligence for representing and reasoning with partial information under conditions of uncertainty is Bayesian networks. They allow expressing information such as "if a person has flu, he is likely to have fever." Constraint networks and propositional theories are the most basic frameworks for representing and reasoning about deterministic information. Constraints often express resource conflicts frequently appearing in scheduling and planning applications, precedence relationships (e.g., "job 1 must follow job 2") and definitional information (e.g., "a block is clear iff there is no other block on top of it"). Most often the feasibility of an action is expressed using a deterministic rule between the pre-conditions (constraints) and post-conditions that must hold before and after executing an action (e.g., STRIPS for classical planning).

The two communities of probabilistic networks and constraint networks matured in parallel with only minor interaction. Nevertheless some of the algorithms and reasoning principles that emerged within both frameworks, especially those that are graph-based, are quite related. Both frameworks can be viewed as graphical models, a popular paradigm for knowledge representation in general.

Researchers within the logic-based and constraint communities have recognized for some time the need for augmenting deterministic languages with uncertainty information, leading to a variety of concepts and approaches such as non-monotonic reasoning, probabilistic constraint networks and fuzzy constraint networks. The belief networks community started only recently to look into the mixed representation [87, 84, 62, 35] perhaps because it is possible, in principle, to capture constraint information within belief networks [86].

In principle, constraints can be embedded within belief networks by modeling each constraint as a Conditional Probability Table (CPT). One approach is to add a new variable for each constraint that is perceived as its *effect* (child node) in the corresponding causal relationship and then to clamp its value to *true* [86, 21]. While this approach is semantically coherent and complies with the acyclic graph restriction of belief networks, it adds a substantial number of new variables, thus cluttering the problem's structure. An alternative approach is to designate one of the arguments of the constraint as a child node (namely, as its effect). This approach, although natural for functions (the arguments are the causes or parents and the function variable is the child node), is quite contrived for general relations (e.g., $x + 6 \neq y$). Such constraints may lead to cycles, which are disallowed in belief networks. Furthermore, if a variable is a child node of two different CPTs (one may be deterministic and one probabilistic) the belief network definition requires that they be combined into a single CPT.

3.1.1 Contributions

The main shortcoming of any of the above integrations is computational. Constraints have special properties that render them attractive computationally. When constraints are disguised as probabilistic relationships, their computational benefits may be hard to exploit. In particular, the power of constraint inference and constraint propagation may not be brought to bear.

Therefore, we propose a simple framework that combines deterministic and probabilistic networks, called a *mixed network*. In the mixed network framework identity of the respective relationships, as constraints or probabilities, will be maintained explicitly, so that their respective computational power and semantic differences can be vivid and easy to exploit. The mixed network approach allows two distinct representations: causal relationships that are directional and normally (but not necessarily) quantified by CPTs and symmetrical deterministic constraints. The proposed scheme's value is in providing: (1) semantic coherence; (2) user-interface convenience (the user can relate better to these two pieces of information if they are distinct); and most importantly, (3) computational efficiency.

The research presented in this chapter is based in part on [37, 44, 38].

3.2 Mixing Probabilities with Constraints

This section introduces the mixed network concept and discusses some of its properties.

3.2.1 Defining the Mixed Network

We next define the central concept of *mixed networks*.

DEFINITION **3.2.1 (mixed networks)** *Given a belief network $\mathcal{B} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P} \rangle$ that expresses the joint probability P_B and given a constraint network $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$ that expresses a set of solutions ρ, a mixed network based on $\mathcal{B}$ and $\mathcal{R}$ denoted $\mathcal{M}_{(\mathcal{B},\mathcal{R})} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}, \mathbf{C} \rangle$ is created from the respective components of the constraint network and the belief network as follows. The variables $\mathbf{X}$ and their domains are shared, (we could allow non-common variables and take the union), and the relationships include the CPTs in $\mathbf{P}$*

and the constraints in $\mathbf{C}$. *The mixed network expresses the conditional probability* $P_{\mathcal{M}}(\mathbf{X})$*:*

$$P_{\mathcal{M}}(\bar{x}) = \begin{cases} P_{\mathcal{B}}(\bar{x}|\bar{x} \in \rho), & if \;\; \bar{x} \in \rho \\ 0, & otherwise. \end{cases}$$

Clearly, $P_{\mathcal{B}}(\bar{x}|\bar{x} \in \rho) = \frac{P_{\mathcal{B}}(\bar{x})}{P_{\mathcal{B}}(\bar{x} \in \rho)}$.

The auxiliary network. We now define the belief network that expresses constraints as pure CPTs.

DEFINITION **3.2.2 (auxiliary network)** *Given a mixed network* $\mathcal{M}_{(\mathcal{B},\mathcal{R})}$ *we define the auxiliary network* $S_{(\mathcal{B},\mathcal{R})}$ *to be a belief network that has new auxiliary variables as follows. For every constraint* $C_i = (S_i, R_i)$ *in* $\mathcal{R}$, *we add the auxiliary variable* A_i *that has a domain of 2 values, "0" and "1". There is a CPT over* A_i *whose parent variables are* S_i, *defined as follows:*

$$P(A_i = 1|t_{S_i}) = \begin{cases} 1, & if \;\; t \in R_i \\ 0, & otherwise. \end{cases}$$

$S_{(\mathcal{B},\mathcal{R})}$ is a belief network that expresses a probability distribution P_S. It is easy to see that

Proposition 13 *Given a mixed network* $M_{(\mathcal{B},\mathcal{R})}$ *and an associated auxiliary network* $S = S_{(\mathcal{B},\mathcal{R})}$ *then:* $P_{\mathcal{M}}(\bar{x}) = P_S(\bar{x}|A_1 = 1, ..., A_t = 1)$.

3.2.2 Queries over Mixed Networks

Belief updating, MPE and MAP queries can be extended to mixed networks straightforwardly. They are well defined relative to the mixed probability $P_{\mathcal{M}}$. Since $P_{\mathcal{M}}$ is not well defined for inconsistent constraint networks we always assume that the constraint network portion is consistent. An additional relevant query over a mixed network is to find the probability of a consistent tuple relative to $\mathcal{B}$, namely determining $P_{\mathcal{B}}(\bar{x} \in \rho(\mathcal{R}))$ It is called *CNF or Constraint Probability Evaluation (CPE).* Note that the notion of evidence is a special type of constraint. We will ellaborate on this next.

The problem of evaluating the probability of CNF queries over belief networks has various applications. One application is to network reliability described as follows. Given a communication graph with a source and a destination, one seeks to diagnose failure of communication. Since several paths may be available, the reason for failure can be described by a CNF formula. Failure means that for all paths (conjunctions) there is a link on that path (disjunction) that fails. Given a probabilistic fault model of the network, the task is to assess the probability of a failure [88].

DEFINITION **3.2.3 (CPE)** *Given a mixed network $M_{(\mathcal{B},\mathcal{R})}$, where the belief network is defined over variables $\mathbf{X} = \{X_1, ..., X_n\}$ and where the constraint portion is a either a set of relational constraints or a CNF query ($\mathcal{R} = \varphi$) over a set of subsets $Q = \{Q_1, ...Q_r\}$, where $Q \subseteq \mathbf{X}$, the* constraint, *(respectively* CNF) Probability Evaluation (CPE) task *is the task to find the probability $P_{\mathcal{B}}(\bar{x} \in \rho(\mathcal{R}))$, respectively $P_{\mathcal{B}}(\bar{x} \in m(\varphi))$ where $m(\varphi)$ are the models (solutions of φ).*

Belief assessment conditioned on a constraint network or a CNF expression is the task of assessing $P(X|\varphi)$ for every variable X. Since $P(X|\varphi) = \alpha P(X \wedge \varphi)$ where α is a normalizing constant relative to X, computing $P(X|\varphi)$ reduces to a CPE task for the query $((X = x) \wedge \varphi)$. More generally, $P(\varphi|\psi)$ can be derived from $P(\varphi|\psi) = \alpha_\varphi \cdot P(\varphi \wedge \psi)$ where α_φ is a normalization constant relative to all the models of φ.

3.2.3 Examples

Java bugs Consider the classical naive-Bayes model or, more generally, two layer network. Often the root nodes in the first layer are desired to be mutually exclusive, a property that can be enforced by *all-different* constraints. For example, consider a bug diagnostics

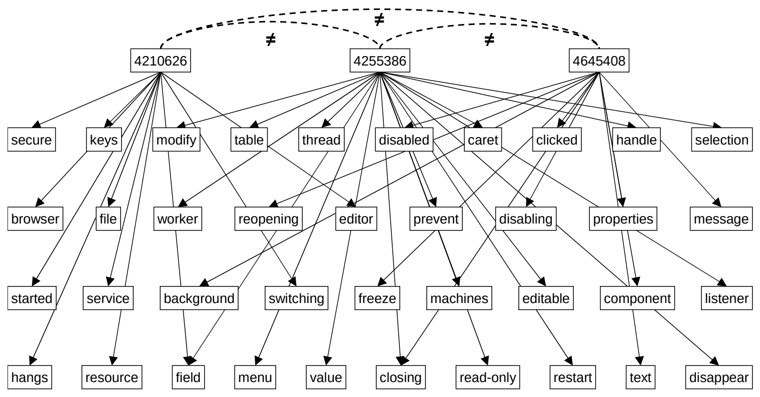


Figure 3.1: Two layer networks with root not-equal constraints (Java Bugs)

system for a software application such as Java Virtual Machine that contains numerous bug descriptions. When the user performs a search for the relevant bug reports, the system outputs a list of bugs, in decreasing likelihood of it being the problem's culprit. We can model the relationship between each bug identity and the key words that are likely to trigger this bug as a parent child relationship of a two layer belief network where the bug identities are root nodes and all the key words that may appear in each bug description are child nodes. Each bug has a directed edge to each relevant keyword (See Figure 3.1). In practice, a problem is caused by only one bug and thus, the bugs on the list are mutually exclusive. We may want to express this fact using a not-equal relationship between all (or some of) the root nodes. We could have taken care of this by putting all the bugs in one node. However, this will cause a huge inconvenience, having to express the conditional probability of each key word given each bug, even when it is not relevant. Java bug database contains thousands of bugs. It is hardly sensible to define a conditional probability table of that size. So, in the mixed network framework we can simply add one not-equal constraint over all the root variables.

Class scheduling Another source of examples is when reasoning about an agent's behavior. Consider a student's class scheduling activitiy. A relevant knowledge base can be built either from the student's point of view, the administration view or from the faculty point of view. Perhaps, the same knowledge-base can serve these multiple reasoning perspectives. The administration (e.g, the chair) tries to schedule the classes so as to meet the various requirements of the students (alow enough classes in each quarter for each concentration) while faculty may want to teach their classes in a particular quarter to maximize (or minimize) students attendance or to better distribute their research vs teaching time throuout the academic year.

In Figure 3.2 we demonstrate a scenario with 3 classes and 2 students. The variables are $T(S_i, C_j)$ meaning "student S_i takes course C_j", $P(S_i, C_j)$ denoting the performance (grade) of student S_i in course C_j. past-$P(S_i, C_j)$ is the past performance of student S_i in C_j (if the class was taken). The variable $teach(C_j)$ denotes the professor who teaches C_j in the current quarter, and $type(S_i)$ stands for a collection of variables denoting student S_i's characteristics (his strengths, goals and inclinations, time in the program etc.). If we have a restriction on the number of students that can take a class, we can impose a unary constraints ($N(C_i) \leq 10$). For each student and for each $P(S_i, C_j)$ we have a CPT from the parents node $T(S_i, C_j)$, $teaches(C_j)$ and $type(S_i)$. We then have constraints between various classes such as $T(S, C_1)$ and $T(S, C_2)$ indicating that both cannot be taken together due to scheduling conflicts. We can also have all-different constraints between pairs of $teachC_j$ since the same teacher may not teaches two classes even if those classes are not conflicting. (For clarity we do not express this constraints in Figure 3.2.) Finally, since a student may need to take at least 2 and at most 3 classes, we can have a variable $N(S_i)$ that is the number function of the classes taken by the student. If C_1 is a prerequisite to C_2 we can have a constraint between $T(S, C_1)$ and $past - P(S, C_2)$.

3.2.4 Processing Networks with Determinism

Often belief networks have a hybrid probabilistic and deterministic relationships. Such networks appear in medical applications in coding networks [89] and in networks having CPTs that are *causally independent* [53]. Recent work in dynamic decision networks reveals the

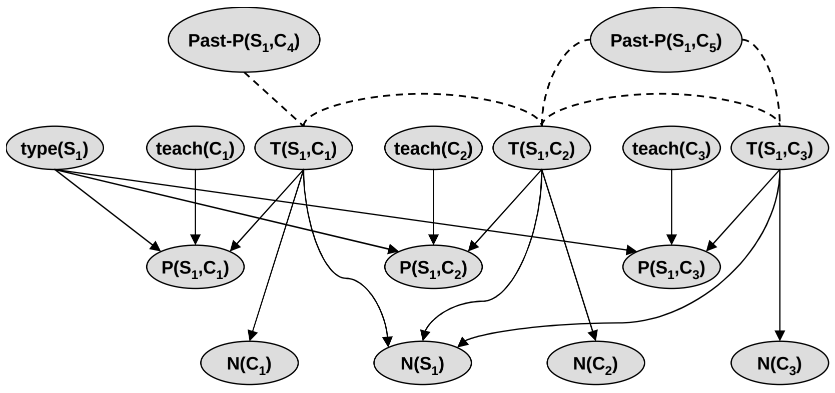


Figure 3.2: Mixed network for student's class taking

need to express large portion of the knowledge using deterministic constraints. We argue that treating such information in a special manner, using constraint processing methods is likely to yield significant computational benefit.

Belief assessment in belief networks having determinism translates to a CPE task over a mixed network. The idea is to collect together all the deterministic information appearing in the functions of F and to extract the deterministic information in the mixed CPTs, and then transform it all to one CNF or a constraint expression that will be treated as a constraint network part relative to the original belief network. Each entry in a mixed CPT $P(X_i|pa_i)$, having $P(x_i|x_{pa_i}) = 1$, (x is a tuple of variables in the family of X_i) can be translated to a constraint (not allowing tuples with zero probability) or to clauses $x_{pa_i} \rightarrow x_i$, and all such entries constitute a conjunction of clauses.

Let $B =< C, P, F >$ be a belief network having determinism. Given evidence e, assessing the posterior probability of a single variable X given evidence e is to compute $P(X|e) = \alpha P(X \wedge e)$. Let $cl(P)$ be the clauses extracted from the mixed CPTs. The network's deterministic portion is $cl(F) \wedge cl(P)$, and because this conjunction is redundant relative to the given network, namely since $P(cl(F) \wedge cl(P) = 1$ we can write:
$P((X = x) \wedge e) = P((X = x) \wedge e \wedge cl(F) \wedge cl(P))$ Therefore, to evaluate the belief of $X = x$ we can evaluate the probability of the CNF formula $\varphi = ((X = x) \wedge e \wedge cl(F) \wedge cl(P))$ over the original belief network.

3.2.5 Mixed Graphs as I-Maps

In this section we define the *mixed graph* of a mixed network and an accompanying separation criterion, extending d-separation. We show that a mixed graph is a minimal I-map (independency map) of a mixed network relative to an extended notion of separation, called *dm-separation*.

DEFINITION **3.2.4 (mixed graph)** *Given a mixed network $M_{(\mathcal{B},\mathcal{R})}$, the mixed graph $G_M = (G, D)$ is defined as follows. Its nodes correspond to the variables appearing either in $\mathcal{B}$ or in $\mathcal{R}$, and the arcs are the union of the undirected arcs in the constraint graph D of $\mathcal{R}$, and the directed arcs in the belief network $\mathcal{B}$, G. The moral mixed graph is the moral graph of the belief network union the constraint graph.*

The notion of d-speration in belief networks is known to capture conditional independence [86]. Namely any d-separation in the directed graph corresponds to a conditional independence in the corresponding probability distribution. Likewise, an undirected graph representation of probabilistic networks (i.e., Markov networks) allows reading valid conditional independence based on undirected graph separation.

In this section we define a *dm-separation* of mixed graphs and show that it provides a criterion for establishing minimal I-mapness for mixed networks.

DEFINITION **3.2.5 (ancestral mixed graph)** *Given a mixed graph $G_M = (G, D)$ of a mixed network $M_{(\mathcal{B},\mathcal{R})}$ where G is the directed acyclic graph of $\mathcal{B}$, and D is the undirected constraint graph of $\mathcal{R}$, the ancestral graph of X in G_M is the graph D union the ancestral graph of X in G.*

DEFINITION **3.2.6 (dm-separation)** *Given a mixed graph, G_M and given three subsets of variables X, Y and Z which are disjoint, we say that X and Y are dm-separated given Z in the mixed graph G_M, denoted $< X, Z, Y >_{dm}$, iff in the ancestral mixed graph of $X \cup Y \cup Z$, all the paths between X and Y are intercepted by variables in Z.*

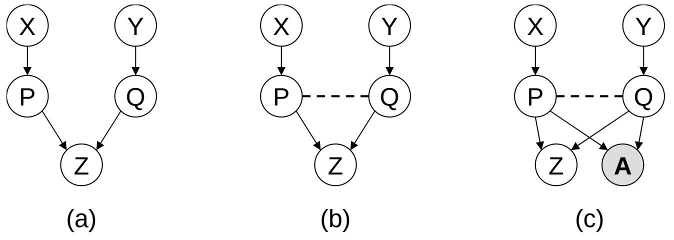


Figure 3.3: DM-separation

The following Theorem follows straightforwardly from the correspondance between mixed networks and auxiliary networks.

THEOREM **3.2.1 (I-map)** *Given a mixed network $M = M_{(\mathcal{B},\mathcal{R})}$ and its mixed graph G_M, then G_M is a minimal I-map relative to dm-separation. Namely, if $< X, Z, Y >_{dm}$ then $P_M(X|Y, Z) = P_M(X|Z)$ and no arc can be removed while maintaining this property.*

Proof. Assuming $< X, Z, Y >_{dm}$ we should prove $P_M(X|Y, Z) = P_M(X|Z)$. Namely, we should prove that $P_S(X|Y, Z, A = 1) = P_S(X|Z, A = 1)$, when $S = S_{(\mathcal{B},\mathcal{R})}$, and $A = 1$ is an abbreviation to assigning all auxiliary variables in S the value 1 (Proposition 13). Since $S = S_{(\mathcal{B},\mathcal{R})}$ is a regular belief network we can use the ancestral graph criterion to determine d-separation. It is easy to see that the ancestral graph of the directed graph of S given $X \cup Y \cup Z \cup A$ when moralized is identical to the corresponding ancestral mixed graph (if we ignore the edges going into the evidence variables A), and thus dm-separation translates to d-separation and provides a characterization of I-mapness of mixed networks. The minimality of mixed graphs as I-maps follows from the minimality of belief networks relative to d-separation applied to the auxiliary network. $\Box$

Example 3.2.2 *Figure 3.3a shows a regular belief network in which X and Y are d-separated given the empty set. If we add a constraint R_{PQ} between P and Q, we obtain the mixed network in Figure 3.3b. According to dm-separation X is no longer independent of Y, because of the path $XPQY$ in the ancestral graph. Figure 3.3c shows the auxialiary network, with variable A assigned to 1 corresponding to the constraint between P and Q. D-separation also dictates a dependency between X and Y.*

We will next see the first virtue of “mixed” vs “auxiliary” networks. It is now clear that the concept of constraint propagation has a clear meaning within the mixed network framework. That is, we can allow the constraint network to be processed by any constraint propagation algorithm to yield another, equivalent, well defined, mixed network.

DEFINITION **3.2.7 (equivalent mixed networks)** *Two mixed networks defined on the same set of variables $X = \{X_1, ..., X_n\}$ and the same domains, $D_1, ..., D_n$, denoted $M_1 = M_{(\mathcal{B}_1,\mathcal{R}_1)}$ and $M_2 = M_{(\mathcal{B}_2,\mathcal{R}_2)}$, are equivalent iff they are equivalent as probability distributions, namely iff $P_{M_1} = P_{M_2}$.*

Proposition 14 *If $\mathcal{R}_1$ and $\mathcal{R}_2$ are equivalent constraint networks (have the same set of solutions), then $M_{(\mathcal{B},\mathcal{R}_1)}$ is equivalent to $M_{(\mathcal{B},\mathcal{R}_2)}$.*

The above proposition shows one advantage of looking at mixed networks rather than at auxiliary networks. Due to the explicit representation of deterministic relationships, notions such as inference and constraint propagation are naturally defined and are exploitable in mixed network.

3.3 Inference Algorithms for Processing Mixed Networks

There are two primary approaches for processing mixed networks. One is the variable elimination approach which was presented in [35] and the other is search, also called *conditioning*. Search is based on enumerating the solutions of the constraint formula, and then assessing the belief of each solution. We will focus on the CPE task of computing $P(\varphi|e)$ where φ is the constraint or CNF formula. A number of related tasks can be easily derived by changing the appropriate operator (e.g. using maximization for maximum probable explanation - MPE, or summation and maximization for maximum a posteriori hypothesis - MAP).

3.3.1 A Bucket Elimination Method

In this section we will first derive a bucket elimination algorithm when the deterministic component is a CNF formula and then show how it generalizes to any constraint expression.

Given a mixed network $M_{(\mathcal{B},\varphi)}$, where φ is a CNF formula defined on a subset of variables Q, the CPE task is to compute:

$$P(\varphi) = \sum_{\bar{x}_Q \in models(\varphi)} P(\bar{x}_Q).$$

Using the belief-network product form we get:

$$P(\varphi) = \sum_{\{\bar{x} | \bar{x}_Q \in models(\varphi)\}} \prod_{i=1}^{n} P(x_i | x_{pa_i})$$

We assume that X_n is one of the CNF variables, and we separate the summation over X_n and $\mathbf{X} \setminus \{X_n\}$. We denote by γ_n the set of all clauses that are defined on X_n and by β_n all the rest of the clauses. The scope of γ_n is denoted Q_n, $S_n = \mathbf{X} \setminus Q_n$ and U_n is the set of all variables in the scopes of CPTs and clauses that are defined over X_n. We get:

$$P(\varphi) = \sum_{\{\bar{x}_{n-1} | \bar{x}_{S_n} \in models(\beta_n)\}} \sum_{\{x_n | \bar{x}_{Q_n} \in models(\gamma_n)\}} \prod_{i=1}^{n} P(x_i | x_{pa_i})$$

Denoting by t_n the set of indices of functions in the product that *do not* mention X_n and by $l_n = \{1, \ldots, n\} \setminus t_n$ we get:

$$P(\varphi) = \sum_{\{\bar{x}_{n-1} | \bar{x}_{S_n} \in models(\beta_n)\}} \prod_{j \in t_n} P_j \cdot \sum_{\{x_n | \bar{x}_{Q_n} \in models(\gamma_n)\}} \prod_{j \in l_n} P_j$$

Therefore:

$$P(\varphi) = \sum_{\{\bar{x}_{n-1} | \bar{x}_{S_n} \in models(\beta_n)\}} (\prod_{j \in t_n} P_j) \cdot \lambda^{X_n}$$

where λ^{X_n} is defined over $U_n - \{X_n\}$, by

$$\lambda^{X_n} = \sum_{\{x_n | \bar{x}_{Q_n} \in models(\gamma_n)\}} \prod_{j \in l_n} P_j \tag{3.1}$$

Case of observed variables. When X_n is observed, or constrained by a literal, the summation operation reduces to assigning the observed value to each of its CPTs *and* to each of the relevant clauses. In this case Equation (3.1) becomes (assume $X_n = x_n$ and $P_{=x_n}$ is the function instantiated by assigning x_n to X_n):

$$\lambda^{x_n} = \prod_{j \in l_n} P_{j_{=x_n}}, \quad if\ \bar{x}_{Q_n} \in m(\gamma_n \wedge (X_n = x_n)) \tag{3.2}$$

Otherwise, $\lambda^{x_n} = 0$. Since $\bar{x}_{Q_n}$ satisfies $\gamma_n \wedge (X_n = x_n)$ only if $\bar{x}_{Q_n - X_n}$ satisfies $\gamma^{x_n} = resolve(\gamma_n, (X_n = x_n))$, we get:

$$\lambda^{x_n} = \prod_{j \in l_n} P_{j_{=x_n}} \ \ if\ \bar{x}_{Q_n - X_n} \in m(\gamma_n^{x_n}) \tag{3.3}$$

Therefore, we can extend the case of observed variable in a natural way: CPTs are assigned the observed value as usual while clauses are individually resolved with the unit clause $(X_n = x_n)$, and both are moved to appropriate lower buckets.

Therefore, in the bucket of X_n we should compute λ^{X_n}. We need to place all CPTs and clauses mentioning X_n and then compute the function in Equation (3.1). The computation of the rest of the expression proceeds with X_{n-1} in the same manner. This yields algorithm *Elim-CPE*, described in Figures 3.4 and 3.5. The elimination operation is denoted by the general operator symbol $\otimes$ that instantiates to summation for the current query. Thus, for every ordering of the propositions, once all the CPTs and clauses are partitioned (each clause and CPT is placed in the bucket of the latest variable in their scope), we process the buckets from last to first, in each applying the following operation. Let $\lambda_1, ...\lambda_t$ be the probabilistic functions in bucket P over scopes $S_1, ..., S_t$ and $\alpha_1, ...\alpha_r$ be the clauses over

Algorithm Elim-CPE
Input: A belief network $BN = \{P_1, ..., P_n\}$; A cnf formula on k propositions $\varphi = \{\alpha_1, ...\alpha_m\}$ defined over k propositions, an ordering of the variables, d
Output: The belief $P(\varphi)$.
1. **Initialize:** Place buckets with unit clauses last in the ordering (to be processed first). Partition the BN and φ into $bucket_1, \ldots, bucket_n$, where $bucket_i$ contains all matrices and clauses whose highest variable is X_i. Put each observed variable into its appropriate bucket. Let $S_1, ..., S_j$ be the scopes of CPTs, and $Q_1, ...Q_r$ be the scopes of clauses. (We denote probabilistic functions as λs and clauses by αs).
2. **Backward:** Process from last to first.
Let p be the current bucket.
For $\lambda_1, ..., \lambda_j$, $\alpha_1, ..., \alpha_r$ in $bucket_p$, do
• **Process-bucket**$_p$**(**$\sum, (\lambda_1, ..., \lambda_j), (\alpha_1, ..., \alpha_r))$
3. **Return** $P(\varphi)$ as the results of the elimination function in the first bucket.

Figure 3.4: Algorithm *Elim-CPE*

Process-bucket$_p$**(**$\otimes, (\lambda_1, ..., \lambda_j), (\alpha_1, ..., \alpha_r)$**)**
• **If** $bucket_p$ contains $X_p = x_p$,
1. Assign $X_p = x_p$ to each λ_i and put each resulting function into its appropriate earlier bucket.
2. Resolve each α_i with the unit clause, put non-tautology resolvents in lower buckets and **move any bucket with unit clause to top of processing.**

• **Else,** generate λ^P:

$$\lambda^P = \otimes_{\{x_p|\bar{x}_{U_p} \in models(\alpha_1, \ldots, \alpha_r)\}} \prod_{i=1}^{j} \lambda_i$$

Add λ^p to the bucket of the largest-index variable in $U_p \leftarrow \bigcup_{i=1}^{j} S_i \bigcup_{i=1}^{r} Q_i - \{X_p\}$.

Figure 3.5: Process-bucket procedure

scopes $Q_1, ..., Q_r$. The algorithm computes a new function λ^P over $U_p = S \cup Q - \{X_p\}$ where $S = \cup_i S_i$, and $Q = \cup_j Q_j$, defined by:

$$\lambda^P = \sum_{\{x_p|\bar{x}_Q \in models(\alpha_1, \ldots, \alpha_r)\}} \prod_j \lambda_j$$

Example 3.3.1 *Consider the belief network in Figure 3.9, which is similar to the one in Figure 1.2, and the query $\varphi = (B \vee C) \wedge (G \vee D) \wedge (\neg D \vee \neg B)$. The initial partitioning into buckets along the ordering $d = A, C, B, D, F, G$, as well as the output buckets are given in Figure 3.6. We compute:*

In bucket G: $\lambda^G(f, d) = \sum_{\{g|g \vee d = true\}} P(g|f)$

In bucket F: $\lambda^F(b, c, d) = \sum_f P(f|b, c)\lambda^G(f, d)$

In bucket D: $\lambda^D(a, b, c) = \sum_{\{d|\neg d \vee \neg b = true\}} P(d|a, b)\lambda^F(b, c, d)$

In bucket B: $\lambda^B(a, c) = \sum_{\{b|b \vee c = true\}} P(b|a)\lambda^D(a, b, c)\lambda^F(b, c)$

In bucket C: $\lambda^C(a) = \sum_c P(c|a)\lambda^B(a, c)$

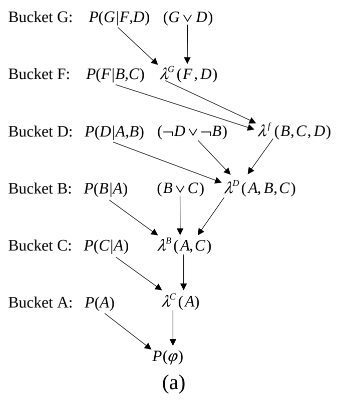

Bucket G: $P(G|F,D)$ $(G \vee D)$
Bucket F: $P(F|B,C)$ $\lambda^G(F,D)$
Bucket D: $P(D|A,B)$ $(\neg D \vee \neg B)$ $\lambda^f(B,C,D)$
Bucket B: $P(B|A)$ $(B \vee C)$ $\lambda^D(A,B,C)$
Bucket C: $P(C|A)$ $\lambda^B(A,C)$
Bucket A: $P(A)$ $\lambda^C(A)$
$P(\varphi)$
(a)

Figure 3.6: Execution of elim-CPE

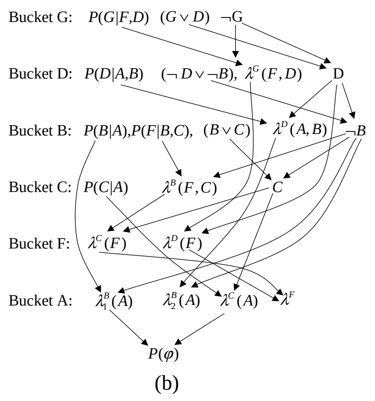

Bucket G: $P(G|F,D)$ $(G \vee D)$ $\neg G$
Bucket D: $P(D|A,B)$ $(\neg D \vee \neg B)$, $\lambda^G(F,D)$ D
Bucket B: $P(B|A)$,$P(F|B,C)$, $(B \vee C)$ $\lambda^D(A,B)$ $\neg B$
Bucket C: $P(C|A)$ $\lambda^B(F,C)$ C
Bucket F: $\lambda^C(F)$ $\lambda^D(F)$
Bucket A: $\lambda_1^B(A)$ $\lambda_2^B(A)$ $\lambda^C(A)$ λ^F
$P(\varphi)$
(b)

Figure 3.7: Execution of elim-bel-cnf

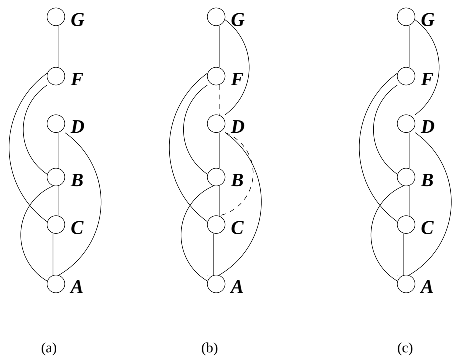


Figure 3.8: The induced augmented graph

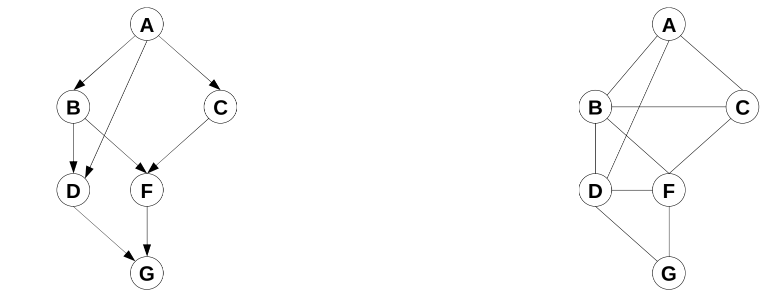
In bucket A: $\lambda^A = \sum_a P(a)\lambda^C(a)$

$P(\varphi) = \lambda^A$.

For example $\lambda^G(f, d = 0) = P(g = 1|f)$, because if $d = 0$ g must get the value "1", while $\lambda^G(f, d = 1) = P(g = 0|f) + P(g = 1|f)$. In summary,

THEOREM **3.3.2 (Correctness and Completeness)** *Algorithm Elim-CPE is sound and complete for the CPE task.*

Notice that algorithm Elim-CPE also includes a unit resolution step whenever possible (in step 2) and a dynamic reordering of the buckets that prefer processing buckets that include unit clauses. This may have a significant impact on efficiency because treating observations (namely unit clauses) specially can avoid creating new dependencies. In fact,



(a) Directed acyclic graph (b) Moral graph

Figure 3.9: Belief network

there exists a spectrum of feasible bounded inferences that can be applied to the clauses in the buckets and can enhance efficiency considerably.

Example 3.3.3 *Let's now extend the example by adding $\neg G$ to the query. This will place $\neg G$ in the bucket of G. When processing bucket G, unit resolution creates the unit clause D, which is then placed in bucket D. Next, processing bucket F creates a probabilistic function on the two variables B and C. Processing bucket D that now contains a unit clause will assign the value D to the CPT in that bucket and apply unit resolution, generating the unit clause $\neg B$ that is placed in bucket B. Subsequently, in bucket B we can apply unit resolution again, generating C placed in bucket C, and so on. In other words, aside from bucket F, we were able to process all buckets as observed buckets, by propagating the observations. (See Figure 3.7.) To incorporate dynamic variable ordering, after processing bucket G, we move bucket D to the top of the processing list (since it has a unit clause). Then, following its processing, we process bucket B and then bucket C, then F, and finally A.*

Since unit resolution increases the number of buckets having unit clauses, and since those are processed in linear time, it can improve performance substantially. Such buckets can be identified a priori by applying unit resolution on the CNF formula or arc-consistency on the constraint expression.

```
Algorithm Elim-ConsPE
Input: A belief network $BN = \{P_1, ..., P_n\}$; A constraint expression over $k$ variables, $\mathcal{R} = \{R_{Q_1}, ..., R_{Q_t}\}$ an ordering $d$
Output: The belief $P(\mathcal{R})$.

1. Initialize: Place buckets with observed variables last in the ordering (to be processed first). Partition the $BN$ and $\mathcal{R}$ into $bucket_1, \ldots, bucket_n$, where $bucket_i$ contains all matrices and constraints whose highest variable is $X_i$. Let $S_1, ..., S_j$ be the scopes of CPTs, and $Q_1, ...Q_t$ the scopes of constraints.
We denote probabilistic functions as $\lambda$s and constraints by $R$s.

2. Backward: Process from last to first. Let $p$ be the current bucket. For $\lambda_1, \lambda_2, ..., \lambda_j$, $R_1, .., R_r$ in $bucket_p$, do:
    PROCESS-BUCKET-REL$_p(\sum, (\lambda_1, ..., \lambda_j), (R_1, ..., R_r))$
    if $bucket_p$ contains $X_p = x_p$
      i) Assign $X_p = x_p$ to each $\lambda_i$ and put each resulting function into its appropriate bucket.
      ii) Apply arc-consistency (or any constraint propagation) over the constraints in the bucket.
      Put results in lower buckets and. Buckets with singleton domain go to top of processing.
    else
      Generate $\lambda^P = \sum_{\{x_p | \bar{x}_{U_p} \in \bowtie_j R_j\}} \prod_{i=1}^{j} \lambda_i$.
      Add $\lambda^p$ to the bucket of the largest-index variable in $U_p \leftarrow \bigcup_{i=1}^{j} S_i \cup Q_i - \{X_p\}$.
3. Return $P(\mathcal{R})$ as the results of the elimination function in the first bucket.
```

Figure 3.10: Algorithm *Elim-ConsPE*

3.3.2 Probability of Relational Constraints

When the variables in the belief network are multi-valued, the deterministic query can be expressed using relational operators and constraints. The set of solutions of a constraint network can be expressed using the join operator. The join of two relations R_{AB} and R_{BC} denoted $R_{AB} \bowtie R_{BC}$ is the largest set of solutions over A, B, C satisfying the two constraints R_{AB} and R_{BC}. The set of solutions of the constraint formula $\mathcal{R} = \{R_1, ...R_t\}$ is $sol(\mathcal{R}) = \bowtie_{i=1}^{t} R_i$.

Given a belief network and a constraint formula $\mathcal{R}$ we may be interested in computing $P(\bar{x} \in sol(\mathcal{R}))$. A bucket-elimination algorithm for computing this task is a simple generalization of Elim-CPE, except that it uses the relational operators as expressed in Figure 3.10.

3.3.3 Complexity

As usual, the complexity of bucket elimination algorithms is related to the number of variables appearing in each bucket, both in the scope of probability functions as well as in the scopes of constraints. The worst-case complexity is time and space exponential in the size of the maximal bucket, which is captured by the induced-width of the relevant graph. For the task at hand, the relevant graph is the moral mixed graph.

Clearly, the complexity of Elim-CPE and Elim-Conspe is $O(n \cdot exp(w^*))$, where w^* is the induced width of the moral mixed ordered graph.

In Figure 3.8 we see that while the induced width of the moral graph is just 2 (Figure 3.8a), the induced width of the mixed graph is 3 (Figure 3.8b).

To capture the simplification associated with observed variables or unit clauses, we can use the notion of an *adjusted induced graph*. The adjusted induced graph is created by processing the variables from last to first in the given ordering. Only parents of each non-observed variable are connected. The adjusted induced width is the width of the adjusted induced-graph. Figure 3.8c shows the adjusted induced-graph relative to the evidence in $\neg G$. We see that the induced width, adjusted for the observation, is just 2 (Figure 3.8c). Notice that adjusted induced-width can be obtained only after we obsereve those variables that were instantiated as a result of our propagation algorithm.

In summary,

THEOREM **3.3.4 ([35])** *Given a mixed network, $\mathcal{M}$, of a belief network over n variables, a constraint expression and an ordering o, algorithm Elim-CPE is time and space $O(n \cdot exp(w^*_{\mathcal{M}}(o)))$, where $w^*_{\mathcal{M}}(o)$ is the width along o of the adjusted moral mixed induced graph.*

3.3.4 Elim-CPE with General Constraint Propagation

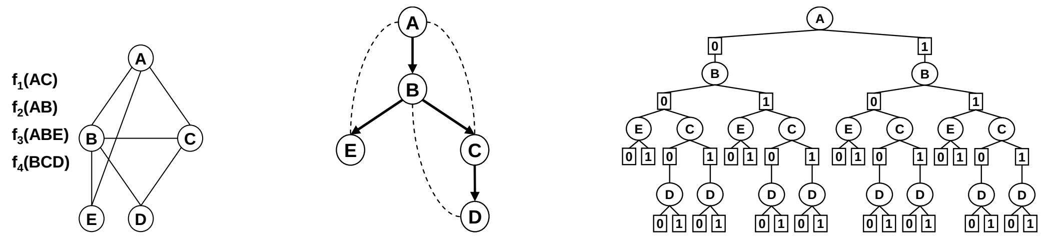
Constraint propagation can, in principle, improve Elim-CPE by inferring new unit clauses beyond the power of unit-resolution. Furthermore, inferred clauses correspond to infered conditional probabilities that are either “0” or “1”.

One form of constraint propagation is bounded resolution [90]. It applies pair-wise resolution to any two clauses in the CNF theory iff the resolvent does not exceed a bounding parameter, i. Bounded-resolution algorithms can be applied until quiesence or in a directional manner, called $BDR(i)$. After partitioning the clauses into ordered buckets, each is processed by resolution with bound i.

We extend Elim-CPE into a parameterized family of algorithms Elim-CPE(i) that incorporates $BDR(i)$. The added operation in $bucket_p$ is: (If the bucket does not have an observed variable)
For each pair $\{(\alpha \vee Q_i), (\beta \vee \neg Q_i)\} \subseteq bucket_i$. If the resolvent $\gamma = \alpha \cup \beta$ contains no more than i propositions, place the resolvents in the bucket of its highest index variable. Higher levels of propagation may infer more unit-clauses and general nogoods but require more computation. It is hard to assess in advance the right balance of constraint propagation. It is known that the complexity of $BDR(i)$ is $O(exp(i))$. Therefore, for small levels of i the computation in non-unit buckets is likely to be dominated by generating the probabilistic function rather than by $BDR(i)$.

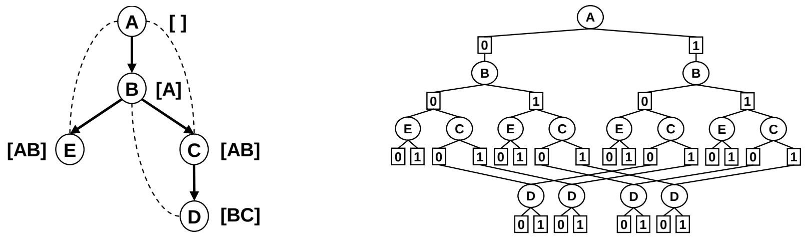
3.4 AND/OR Search Algorithms For Mixed Networks

Proposition 14 ensures the equivalence of mixed networks defined by the same belief network, and different constraint networks that have the same set of solutions. In particular, this implies that we can process the deterministic information separately (e.g., by enforcing some consistency level, which results in a tighter representation), without losing any solution. Conditioning algorithms (search) offer a natural approach for exploiting the de-



(a) Graphical model (b) Pseudo tree (c) Search tree

Figure 3.11: AND/OR search tree



(a) Pseudo tree (b) Context minimal graph

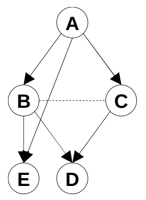
Figure 3.12: AND/OR search graph

terminism through constraint propagation techniques. The intuitive idea is to search in the space of partial variable assignments, and use the constraints to limit the actual traversed space.

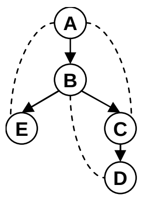
We will use the following examples to describe the algorithms.

Example 3.4.1 *Figure 3.11 shows an example of an AND/OR search tree. Figure 3.11(a) shows a graphical model defined by four functions, over binary variable, and assuming all tuples are consistent. When some tuples are inconsistent, some of the paths in the tree do not exists. Figure 3.11(b) gives the pseudo tree that guides the search, from top to bottom, as indicated by the arrows. The dotted arcs are backarcs from the primal graph. Figure 3.11(c) shows the AND/OR search tree, with the alternating levels of OR (circle) and AND (square) nodes, and having the structure indicated by the pseudo tree.*

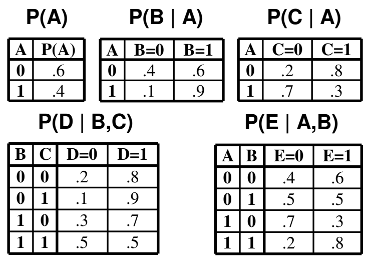
Example 3.4.2 *For Figure 3.12 we refer back to the model given in Figure 3.11(a), again assuming that all assignments are valid and that variables take binary values. Figure*



(a) Belif network



(b) Pseudo tree


P(A)

A	P(A)
0	.6
1	.4

P(B | A)

A	B=0	B=1
0	.4	.6
1	.1	.9

P(C | A)

A	C=0	C=1
0	.2	.8
1	.7	.3

P(D | B,C)

B	C	D=0	D=1
0	0	.2	.8
0	1	.1	.9
1	0	.3	.7
1	1	.5	.5

P(E | A,B)

A	B	E=0	E=1
0	0	.4	.6
0	1	.5	.5
1	0	.7	.3
1	1	.2	.8

(c) CPTs

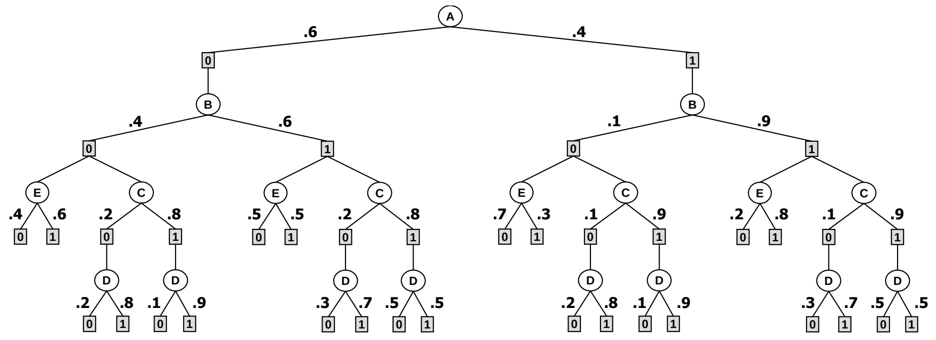

(d) Labeled AND/OR tree

Figure 3.13: Labeled AND/OR search tree for belief networks

3.12(a) shows the pseudo tree derived from ordering $d = (A, B, E, C, D)$, having the same structure as the bucket tree for this ordering. The (OR) context of each node appears in square brackets, and the dotted arcs are backarcs. The context of a node is also identical to the scope of the message sent from its bucket by Bucket Elimination. Figure 3.12(b) shows the context minimal AND/OR graph.

Example 3.4.3 *Figure 3.13 shows a weighted AND/OR tree for a belief network. Figure 3.13(a) shows the primal graph, 3.13(b) is the pseudo tree, and 3.13(c) shows the conditional probability tables. Figure 3.13(d) shows the weighted AND/OR search tree. Naturally, this tree could be transformed into the context minimal AND/OR graph, similar to the one in Figure 3.12(b).*

Algorithm 2: AND-OR-CPE

```
input  : A mixed network $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}, \mathbf{C} \rangle$; a pseudo tree $\mathcal{T}$ of the moral mixed graph, rooted at $X_1$; parents $pa_i$
         (OR-context) for every variable $X_i$; caching set to $true$ or $false$.
output : The probability $P(\bar{x} \in \rho(\mathcal{R}))$ that a tuple satisfies the constraint query.
1  if caching $== true$ then                                         // Initialize cache tables
2      Initialize cache tables with entries of "$-1$"
3  $v(X_1) \leftarrow 0$; OPEN $\leftarrow \{X_1\}$                   // Initialize the stack OPEN
4  while OPEN $\neq \phi$ do
5      n $\leftarrow top$(OPEN); remove n from OPEN
6      if caching $== true$ and n is OR, labeled $X_i$ and $Cache(asgn(\pi_n)[pa_i]) \neq -1$ then     // If in cache
7          $v$(n) $\leftarrow Cache(asgn(\pi_n)[pa_i])$                  // Retrieve value
8          $successors$(n) $\leftarrow \phi$                              // No need to expand below
9      else                                                          // Expand search (forward)
10         if n is an OR node labeled $X_i$ then                      // OR-expand
11             $\boxed{successors(n) \leftarrow \{\langle X_i, x_i \rangle \mid \langle X_i, x_i \rangle \textbf{ is consistent } \text{with } \pi_n\}}$   // CONSTRAINT PROPAGATION
12             $v(\langle X_i, x_i \rangle) \leftarrow \prod_{f \in B_{\mathcal{T}}(X_i)} f(asgn(\pi_n)[pa_i])$,   for all $\langle X_i, x_i \rangle \in successors$(n)
13         if n is an AND node labeled $\langle X_i, x_i \rangle$ then      // AND-expand
14             $successors$(n) $\leftarrow children_{\mathcal{T}}(X_i)$
15             $v(X_i) \leftarrow 0$ for all $X_i \in successors$(n)
16         Add $successors$(n) to top of OPEN
17     while $successors$(n) $== \phi$ do                          // Update values (backtrack)
18         if n is an OR node labeled $X_i$ then
19             if $X_i == X_1$ then                                    // Search is complete
20                 return $v$(n)
21             if caching $== true$ then
22                 $Cache(asgn(\pi_n)[pa_i]) \leftarrow v$(n)               // Save in cache
23             $v$(p) $\leftarrow v$(p) $* v$(c)
24             if $v$(p) $== 0$ then                                 // Check if p is dead-end
25                 remove $successors$(p) from OPEN
26                 $successors$(p) $\leftarrow \phi$
27         if n is an AND node labeled $\langle X_i, x_i \rangle$ then
28             let p be the parent of n
29             $v$(p) $\leftarrow v$(p) $+ v$(n);
30         remove n from $successors$(p)
31         n $\leftarrow$ p
```

3.4.1 AND/OR Search with Constraint Propagation

Algorithm 2, AND-OR-CPE, presents the basic depth-first traversal of the AND/OR search tree (or graph, if caching is used) for solving the CPE task over a mixed network (the algorithm is similar to the one presented in [38]). The algorithm is given as input a mixed network, a pseudo tree $\mathcal{T}$ of the moral mixed graph and the context of each variable. The output is the result of the CPE task, namely the probability that a random tuple satisfies the constraint query. AND-OR-CPE traverses the AND/OR search tree or graph corresponding to $\mathcal{T}$ in a DFS manner. Each node maintains a value v which accumulates the computation resulted from its subtree. OR nodes accumulate the summation of the product between each child's value and its OR-to-AND weight, while AND nodes accumulate the product of their children's values.

The context based caching is done based on tables. We exemplify with OR caching. For each variable X_i, a table is reserved in memory for each possible assignment to its parent set pa_i (context). Initially each entry has a predefined value, in our case "-1". The fringe of the search is maintained on a stack called `OPEN`. The current node is denoted by `n`, its parent by `p`, and the current path by π_n. The children of the current node are denoted by $successors(\mathtt{n})$.

The algorithm is based on two mutually recursive steps: *Expand search* (line 16) and *Update values* (line 31), which call each other (or themselves) until the search terminates.

Since we only use OR caching, before expanding an OR node, its cache table is checked (line 6). If the same context was encountered before, the value is retrieved from cache, and $successors(\mathtt{n})$ is set to the empty set, which will trigger the *Update values* step.

If a node is not found in cache, it is expanded in the usual way, depending on whether it is an AND or OR node (lines 10-16). As long as the current node is not a dead-end and still has unevaluated successors, one of its successors is chosen (which is also the top node on `OPEN`), and the expansion step is repeated.

The bottom up propagation of values is triggered when a node has an empty set of successors (note that as each successor is evaluated, it is removed from the set of successors in line 30). This means that all its children have been evaluated, and its final value can now be computed. If the current node is the root, then the search terminates with its value (line 19). If it is an OR node, its value is saved in cache before propagating it up (line 21). If `n` is OR, then its parent `p` is AND and `p` updates its value by multiplication with the value of `n` (line 23). If the newly updated value of `p` is 0 (line 24), then `p` is a dead-end, and none of its other successors needs to be evaluated. An AND node `n` propagates its value to its parent `p` in a similar way, only by summation (line 29). Finally, the current node `n` is set

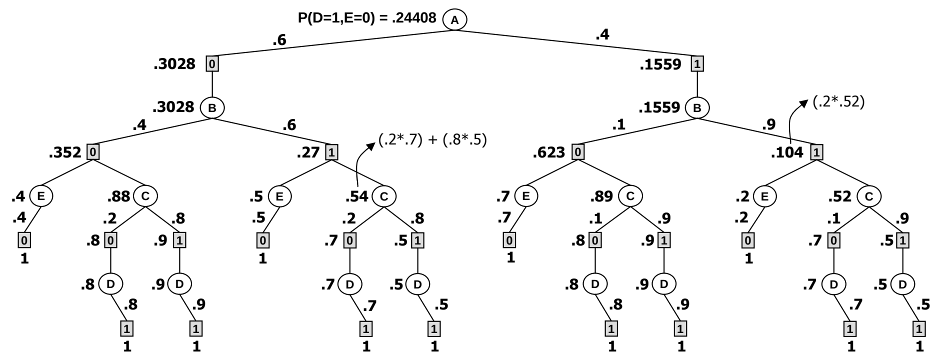
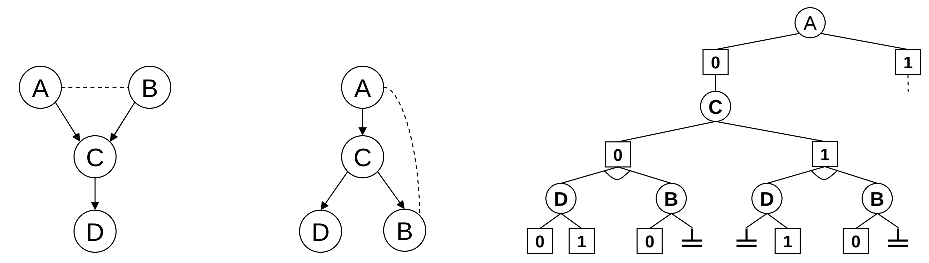


Figure 3.14: AND/OR search tree with final node values



(a) Mixed graph (b) Pseudo tree (c) AND/OR search tree

Figure 3.15: Mixed network; query $\varphi = (A \vee \neg B)(D \vee \neg C)$

to its parent p (line 31), because n was completely evaluated. The search continues either with a propagation step (if conditions are met) or with an expansion step.

Example 3.4.4 *We refer again to the example in Figure 3.13. Considering a constraint network that imposes that $D = 1$ and $E = 0$ (this can also be evidence in the belief network), the trace of the* AND-OR-CPE *algorithm without caching is given in Figure 3.14. To make the computation straightforward, the consistent leaf AND nodes are given a value of 1 (shown under the square node). The final value of each node is shown to its left, while the OR-to-AND weights are shown close to the arcs. The computation of the final value is detailed for one OR node and one AND node.*

Example 3.4.5 *Figure 3.15(a) shows a mixed binary network (the constraint part is given by the CNF formula φ). Figure 3.15(c) describes an AND/OR search tree based on the DFS tree given in Figure 3.15(b). Algorithm* AND-OR-CPE *starts from node A, and assigns*

$g(A) = 0$, then $g(\langle A, 0\rangle) = P(A = 0)$. It continues assigning $g(C) = 0$, and then $g(\langle C, 0\rangle) = 1$. B is not assigned yet, so $P(C|A, B)$ will participate in the label of a descendant node (the set A of step (3) of the algorithm is empty). The node D can take both values (φ is not violated), so by backing up the values of its descendents $g(D)$ becomes 1 ($g(D) = \sum_D P(D|C = 0) = 1$). Going on the branch of B, $g(B) = 0$, then B can only be extended to 0 (to satisfy $A \vee \neg B$), and the label becomes $g(\langle B, 0\rangle) = P(B = 0) \cdot P(C = 0|A = 0, B = 0)$. In general, a CPT participates in OR-to-AND weights at the highest level (closer to the root) of the tree where all the variables in its scope are assigned.

The following are implied immediately from the general properties of AND/OR search trees,

THEOREM **3.4.6** *Algorithm* AND-OR-CPE *is sound and exact for the CPE task.*

THEOREM **3.4.7** *Given a mixed network M with n variables with domain sizes bounded by k and a legal tree T of depth m of its moral mixed graph, the time complexity of* AND-OR-CPE *is $O(n \cdot k^m)$.*

Proposition 15 *A mixed network having induced width w^* has an AND/OR search tree whose size is $O(\exp(w^* \cdot \log n))$.*

Constraint Propagation in AND-OR-CPE

Proposition 14 provides an important justification for using mixed networks as opposed to auxiliary networks. The constraint portion can be processed by a wide range of constraint processing techniques, both statically before AND/OR search or dynamically during AND/OR search. The algorithms can combine consistency enforcing (e.g., arc-, path-, i-consistency) before or during search, directional consistency, look-ahead techniques, no-good learning etc.

The key to using constraint propagation is the boxed line 11 in Algorithm 2. Search only expands when the assignment of the last variable is consistent with the current path.

Table 3.1: AND/OR space vs. OR space

N=25, K=2, R=2, P=2, C=10, S=3, t=70%, 20 instances, w*=9, h=14				
	Time	Nodes	Dead-ends	Full space
AO-C	0.15	44,895	9,095	152,858
OR-C	11.81	3,147,577	266,215	67,108,862

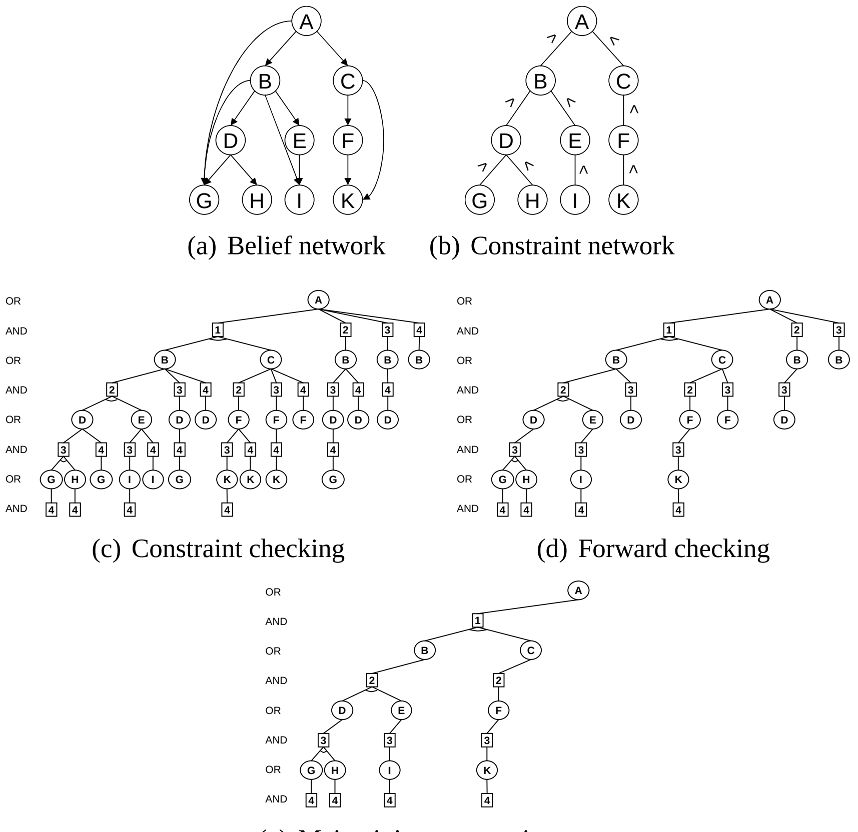
Here we have the freedom to employ any procedure for checking consistency, based on the constraints of the mixed network. The simplest case is when no constrain propagation is used, and only the initial constraints are checked for consistency, and we denote this algorithm by AO-C.

In the experimental evaluation, we used two forms of constraint propagation besides AO-C. The first, yielding algorithm AO-FC, is based on *forward checking*, which is one of the weakest forms of propagation. It propagates the effect of a value selection to each future uninstantiated variable separately, and checks consistency against the constraints whose scope would become fully instantiated by just one such future variable.

The second algorithm we used is called AO-RFC, and performs a variant of *relational forward checking*. Rather than checking only constraints whose scope becomes fully assigned, AO-RFC checks all the existing constraints by looking at their projection on the current path. If the projection is empty an inconsistency is detected. AO-RFC is computationally more expensive than AO-FC, but its search space is smaller.

Figure 3.16 shows the search spaces of AO-C and AO-FC.

Example 3.4.8 *Figure 3.16(a) shows the belief part of a mixed network, and Figure 3.16(b) the constraint part. All variables have the same domain,* $\{1,2,3,4\}$*, and the constraints express "less than" relations. Figure 3.16(c) shows the search space of* AO-C*. Figure 3.16(d) shows the space traversed by AO-FC. Figure 3.16(e) shows the space when consistency is enforced with Maintaining Arc Consistency.*



(a) Belief network (b) Constraint network

(c) Constraint checking (d) Forward checking

(e) Maintaining arc consistency

Figure 3.16: Example of AND-OR-CPE and AO-FC search spaces

3.4.2 Experimental Evaluation

We ran our algorithms on mixed networks generated randomly uniformly given a number of input parameters: N - number of variables; K - number of values per variable; R - number of root nodes for the belief network; P - number of parents for a CPT; C - number of constraints; S - the scope size of the constraints; t - the tightness (percentage of the allowed tuples per constraint). (N,K,R,P) defines the belief network and (N,K,C,S,t) defines the constraint network. We report the time in seconds, number of nodes expanded and number of dead-ends encountered (in thousands), and the number of consistent tuples of the mixed network ($\#sol$). In tables, w^* is the induced width and h is the height of the legal tree.

We compared four algorithms: 1) AND-OR-CPE, denoted here AO-C; 2) AO-FC and 3) AO-RFC (described in previous section); 4) BE - bucket elimination (which is equivalent to join tree clustering) on the auxiliary network; the version we used is the basic one for

Table 3.2: AND/OR Search Algorithms (1)

N=40, K=2, R=2, P=2, C=10, S=4, 20 instances, w*=12, h=19											
t	i	Time			Nodes (*1000)			Dead-ends (*1000)			#sol
		AO-			AO-			AO-			
		C	RC	RFC	C	FC	RFC	C	FC	RFC	
20	0	0.671	0.056	0.022	153	4	1	95	3	1	2E+05
	3	0.619	0.053	0.019	101	3	1	95	3	1	
	6	0.479	0.055	0.022	75	3	1	57	3	1	
	9	0.297	0.053	0.019	52	3	1	10	3	1	
	12	0.103	0.044	**0.016**	17	2	1	3	2	0	
40	0	2.877	0.791	1.094	775	168	158	240	40	36	8E+07
	3	2.426	0.663	0.894	330	57	52	240	40	36	
	6	1.409	0.445	0.544	183	35	32	107	28	24	
	9	0.739	0.301	0.338	119	24	21	20	12	10	
	12	0.189	**0.142**	0.149	28	9	7	3	4	3	
60	0	6.827	4.717	7.427	1,975	1,159	1,148	362	163	159	6E+09
	3	5.560	3.908	6.018	673	351	346	362	163	159	
	6	2.809	2.219	3.149	347	184	180	151	89	86	
	9	1.334	1.196	1.535	204	106	102	19	25	23	
	12	**0.255**	**0.331**	0.425	36	23	22	3	5	5	
80	0	14.181	14.199	21.791	4,283	3,704	3,703	370	278	277	1E+11
	3	11.334	11.797	17.916	1,320	1,109	1,107	370	278	277	
	6	5.305	6.286	9.061	626	519	518	128	98	97	
	9	2.204	2.890	3.725	336	274	273	17	21	20	
	12	**0.318**	0.543	0.714	44	40	40	1	3	3	
100	0	23.595	27.129	41.744	7,451	7,451	7,451	0	0	0	1E+12
	3	19.050	22.842	34.800	2,161	2,161	2,161	0	0	0	
	6	8.325	11.528	16.636	957	957	957	0	0	0	
	9	3.153	4.863	6.255	484	484	484	0	0	0	
	12	**0.366**	0.681	0.884	51	51	51	0	0	0	

belief networks, without any constraint propagation and any constraint testing. For the search algorithms we tried different levels of caching, denoted in the tables by i (i-bound, this is the maximum scope size of the tables that are stored). $i = 0$ stands for linear space search. Caching is implemented based on context as described in Section 3.4.

Table 3.1 gives a brief account for our choice of using AND/OR space instead of the traditional OR space. Given the same ordering, an algorithm that only checks constraints (without constraint propagation) always expands less nodes in the AND/OR space.

Tables 3.2, 3.3, and 3.4 show a comparison of the linear space and caching algorithms exploring the AND/OR space. We ran a large number of cases and this is a typical sample.

Table 3.2 shows a medium sized mixed network, across the full range of tightness for the constraint network. For linear space ($i = 0$), we see that more constraint propagation helps for tighter networks ($t = 20$), AO-RFC being faster than AO-FC. As the constraint network becomes loose, the effort of AO-RFC does not pay off anymore. When almost all

Table 3.3: AND/OR Search Algorithms (2)

t	i	Time		Nodes (*1000)		Dead-ends (*1000)		#sol
		AO-FC	AO-RFC	AO-FC	AO-RFC	AO-FC	AO-RFC	
N=100, K=2, R=10, P=2, C=30, S=3, 20 instances, w*=28, h=38								
10	0	**1.743**	**1.743**	15	15	15	15	0
	10	1.748	1.746	15	15	15	15	
	20	1.773	1.784	15	15	15	15	
20	0	**3.193**	3.201	28	28	28	28	0
	10	3.195	3.200	28	28	28	28	
	20	3.276	3.273	28	28	28	28	
30	0	69.585	62.911	805	659	805	659	0
	10	69.803	**62.908**	805	659	805	659	
	20	69.275	63.055	805	659	687	659	
N=100, K=2, R=5, P=3, C=40, S=3, 20 instances, w*=41, h=51								
10	0	1.251	0.382	7	2	7	2	0
	10	1.249	**0.379**	7	2	7	2	
	20	1.265	0.386	7	2	7	2	
20	0	22.992	**15.955**	164	113	163	111	0
	10	22.994	15.978	162	110	162	111	
	20	22.999	16.047	162	110	162	110	
30	0	253.289	43.255	2093	351	2046	304	0
	10	254.250	**42.858**	2026	283	2032	289	
	20	253.439	43.228	2020	278	2026	283	

tuples become consistent, any form of constraint propagation is not cost effective, AO-C being the best choice in such cases ($t = 80, 100$). For each type of algorithm, caching improves the performance. We can see the general trend given by the bolded figures.

Table 3.3 shows results for large mixed networks ($w^* = 28, 41$). These problems have an inconsistent constraint portion ($t = 10, 20, 30$). AO-C was much slower in this case, so we only include results for AO-FC and AO-RFC. For the smaller network ($w^* = 28$), AO-RFC is only slightly better than AO-FC. For the larger one ($w^* = 41$), we see that more propagation helps. Caching doesn't improve either of the algorithms here. This means that for these inconsistent problems, constraint propagation is able to detect many of the no-goods easily, so the overhead of caching cancels out its benefits (only no-goods can be cached for inconsistent problems). Note that these problems are infeasible for BE due to high induced width.

Table 3.4 shows a comparison between search algorithms and BE. All instances for $t < 40$ were inconsistent and the AO algorithms were much faster than BE, even with linear space. Between $t = 40 - 60$ we see that BE becomes more efficient than AO, and

Table 3.4: AND/OR Search vs. Bucket Elimination

t	i	Time			Nodes (*1000)		Dead-ends (*1000)		#sol
		BE	AO-FC	AO-RFC	AO-FC	AO-RFC	AO-FC	AO-RFC	
N=70, K=2, R=5, P=2, C=30, S=3, 20 instances, w*=22, h=30									
40	0	26.4	2.0	1.3	49	21	35	19	0
	10		1.9	**1.2**	30	18	29	18	
	20		1.9	1.3	26	17	21	16	
50	0		30.7	35.6	2,883	2,708	1,096	1,032	1E+12
	10		18.6	18.9	557	512	342	302	
	20		12.4	**12.1**	245	216	146	130	
60	0		396.8	511.4	51,223	50,089	13,200	12,845	7E+14
	10		167.9	182.5	5,881	5,708	2,319	2,241	
	20		**80.5**	83.6	1,723	1,655	718	697	
N=60, K=2, R=5, P=2, C=40, S=3, 20 instances, w*=23, h=31									
40	0	67.3	0.7	**0.6**	9	9	8	7	0
	10		0.6	0.6	6	5	5	5	
	20		0.6	0.6	5	5	4	4	
50	0		3.2	3.0	58	55	41	38	6E+04
	10		3.0	2.8	31	28	28	25	
	20		2.7	**2.6**	25	23	20	18	
60	0		65.2	70.2	2,302	2,292	1,206	1,195	8E+08
	10		54.1	56.4	791	781	660	649	
	20		**39.6**	40.7	459	449	319	309	

may be comparable only if AO is given the same amount of space as BE.

There is an expected trend with respect to the size of the traversed space and the dead-ends encountered. We see that the more advanced the constraint propagation technique, the less nodes the algorithm expands, and the less dead-ends it encounters. More caching also has a similar effect.

3.5 Conclusion to Chapter 3

We presented the new framework of *mixed networks* (Deterministic-Probabilistic networks) which combines belief and constraint networks. It allows for a more efficient and flexible exploitation of probabilistic and deterministic information by borrowing the specific strengths of each formalism that it builds upon. The *dm-separation* extends in a natural way the d-separation in belief networks, and we show that it provides a criterion for characterizing the minimal I-mapness of the mixed networks.

Proposition 14 which defines the equivalence of mixed networks gives the motivation for using the deterministic information by constraint propagation methods, rather than incorporating it in probability tables.

A number of algorithms based primarily on variable elimination and search are discussed. Many different tasks can be addresed by making only small changes to these algorithms, dictated by the operation that has to be performed (e.g. summation, maximization). The time and space complexities of these algorithms do not indicate a clear hierarchy. Rather, the problem itself may hint on which might be a better candidate, by the relative complexity of the belief network and constraint network.

The belief networks algorithms can benefit from the mixed representation in a number of ways. (1) Constraint propagation techniques can be applied straightforwardly, maintaining their properties of convergence and fixed point. (2) The semantics is much clearer by separating probabilistic and deterministic information. (3) The algorithms can be made more efficient. It is often the case that search based algorithms can benefit from pruning in the context of determinism, or when the number of solutions is small.

The relative advantages of the different algorithms presented here remain to be investigated empirically in future work. A wide variety of hybrid algorithms can be designed, based on search and variable elimination. Finally, they can also be adapted for the case of approximate computation.

Chapter 4

Iterative Algorithms for Mixed Networks

4.1 Introduction

Probabilistic inference is the principal task in Bayesian networks, and it is known to be an NP-hard problem [21]. Most of the commonly used exact algorithms such as join-tree clustering [66, 57] or variable-elimination [28, 103], exploit the network structure. Yet, they are time and space exponential in the *treewidth* of the graph, rendering them essentially intractable even for moderate size problems. Approximate algorithms are therefore necessary for most of the practical problems, although approximation within given error bounds is also NP-hard [22, 92].

The research presented in this chapter is focused primarily on graph algorithms for the task of belief updating. They are inspired by Pearl's belief propagation algorithm [86], which is known to be exact for poly-trees, and by Mini-Buckets algorithm [43], which performs bounded inference and is an anytime version of Variable Elimination. As a distributed algorithm, belief propagation is also well defined for networks that contain cycles, and it can be applied iteratively in the form of Iterative Belief Propagation (IBP), also known as loopy belief propagation. When the networks contain cycles, IBP is no longer guaranteed to be exact, but in many cases it provides very good approximations upon convergence, most notably in the case of coding networks [89] and satisfiability [69].

We are especially interested in the behavior of belief propagation algorithm on mixed networks, specifically on networks that contain zero, or extreme (close to zero or one) probabilities.

4.1.1 Contributions

In this chapter, we investigate: (1) the quality of bounded inference in anytime schemes such as Mini-Clustering, which is a generalization of Mini-Buckets to arbitrary tree-decompositions; (2) the virtues of iterating messages in belief propagation type algorithms, and the result of combining bounded inference with iterative message-passing in Iterative Join-Graph Propagation (IJGP); (3) we make connections with well understood consistency enforcing algorithms for constraint satisfaction, giving strong support for iterating messages, and helping identify cases of strong and weak inference power for IBP and IJGP.

Section 4.2 contains the Mini-Clustering (MC) algorithm, which is inspired by Mini-Buckets algorithm [43]. It is a message-passing algorithm guided by a user adjustable parameter called *i-bound*, offering a flexible tradeoff between accuracy and efficiency in anytime style (in general the higher the i-bound, the better the accuracy). MC algorithm operates on a tree-decomposition, and similar to Pearl's belief propagation algorithm [86] it converges in two passes, up and down the tree. Our contribution beyond other works in this area [43, 34] is in: (1) Extending the partition-based approximation for belief updating from mini-buckets to general tree-decompositions, thus allowing the computation of the updated beliefs for all the variables at once. This extension is similar to the one proposed in [34] but replaces optimization with probabilistic inference. (2) Providing for the first time empirical evaluation demonstrating the effectiveness of the partition-based idea for belief updating.

We were motivated by the success of Iterative Belief Propagation (IBP) in trying to make MC benefit from the apparent virtues of iterating. The resulting algorithm, Iterative Join-Graph Propagation (IJGP) is described in Section 4.3. IJGP is also a messages-passing algorithm, but it operates on a general join-graph decomposition which may contain cycles. It still provides a user adjustable *i-bound* that defines the maximum cluster size of the graph (and hence the complexity), so it is both anytime and iterative. Since both MC and IJGP are approximate schemes, empirical results on various classes of problems are included, shedding light on their average case performance.

Section 4.4 is based on some some simple observations that may shed light on IBP's behavior, and on the more general class of IJGP algorithms. Zero-beliefs are variable-value pairs that have zero conditional probability given the evidence. We show that: (1) if a value of a variable is assessed as having zero-belief in any iteration of IBP, it remains a zero-belief in all subsequent iterations; (2) that IBP converges in a finite number of iterations relative to its set of zero-beliefs; and, most importantly (3) that the set of zero-beliefs decided by any of the iterative belief propagation methods is sound. Namely any zero-belief determined by IBP corresponds to a true zero conditional probability relative to the given probability distribution expressed by the Bayesian network. While each of these claims can be proved directly, our approach is to associate a belief network with a constraint network and show a correspondence between IBP applied to the belief network and an arc-consistency algorithm applied to the corresponding constraint network. Since arc-consistency algorithms are well understood this correspondence not only proves right away the targeted claims, but may provide additional insight into the behavior of IBP and IJGP. In particular, not only it immediately justifies the iterative application of belief propagation algorithms on one hand, but it also illuminates its "distance" from being complete, on the other.

The research presented in this chapter is based in part on [79, 39, 36].

4.2 Mini-Clustering

In this section we present a parameterized anytime approximation scheme for probabilistic inference called *Mini-Clustering (MC)*, which extends the partition-based approximation offered by mini-bucket elimination [43], to general tree decompositions. The benefit of this algorithm is that all single-variable beliefs are computed (approximately) at once, using a two-phase message-passing process along the cluster tree. The resulting approximation scheme allows adjustable levels of accuracy and efficiency, in anytime style. Empirical evaluation against competing algorithms such as Iterative Belief Propagation and Gibbs Sampling demonstrates the potential of the Mini-Clustering approximation scheme: on several classes of problems (e.g. random noisy-or, grid networks and CPCS networks) Mini-Clustering exhibited superior performance. A similar scheme was presented in a general way in [58], and for optimization tasks in [34]. Here we adapt the scheme for the specific task of belief updating, and present the first empirical evaluation for this specific task, showing its effectiveness.

4.2.1 Tree-Decomposition Schemes

We will describe our algorithms relative to a unifying tree-decomposition framework based on [52]. It generalizes tree-decompositions to include join-trees, bucket-trees and other variants applicable to both constraint processing and probabilistic inference.

DEFINITION **4.2.1 (tree-decomposition, cluster tree)** *Let $BN =< X, D, G, P >$ be a belief network. A* tree-decomposition *for BN is a triple $< T, \chi, \psi >$, where $T = (V, E)$ is a tree, and χ and ψ are labeling functions which associate with each vertex $v \in V$ two sets, $\chi(v) \subseteq X$ and $\psi(v) \subseteq P$ satisfying:*

1. *For each function $p_i \in P$, there is* exactly *one vertex $v \in V$ such that $p_i \in \psi(v)$, and $scope(p_i) \subseteq \chi(v)$.*

2. *For each variable $X_i \in X$, the set $\{v \in V | X_i \in \chi(v)\}$ induces a connected subtree of T. This is also called the running intersection property.*

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

DEFINITION **4.2.2 (treewidth, hypertreewidth, separator, eliminator)** *The* treewidth *[3] of a tree-decomposition $< T, \chi, \psi >$ is $max_{v \in V} |\chi(v)|$, and its* hypertreewidth *is $max_{v \in V} |\psi(v)|$. Given two adjacent vertices u and v of a tree-decomposition, the* separator *of u and v is defined as $sep(u, v) = \chi(u) \cap \chi(v)$, and the* eliminator *of u with respect to v is $elim(u, v) = \chi(u) - \chi(v)$.*

Join-Trees and Cluster Tree Elimination

In both Bayesian network and constraint satisfaction communities, the most used tree decomposition method is called join-tree decomposition [66, 41]. Such decompositions can be generated by embedding the network's moral graph, G, in a chordal graph, often using a triangulation algorithm and using its maximal cliques as nodes in the join-tree. The triangulation algorithm assembles a join-tree by connecting the maximal cliques in the chordal graph in a tree. Subsequently, every CPT p_i is placed in one clique containing its scope. Using the previous terminology, a join-tree decomposition of a belief network (G, P) is a tree $T = (V, E)$, where V is the set of cliques of a chordal graph G' that contains G, and E is a set of edges that form a tree between cliques, satisfying the running intersection property [70]. Such a join-tree satisfies the properties of tree-decomposition, therefore our derivation using cluster trees is immediately applicable to join-trees.

There are a few variants for processing join-trees for belief updating [57, 94]. The variant which we use here, (similar to [34]), called cluster-tree-elimination (CTE) is applicable to tree-decompositions in general and is geared toward space savings. It is a message passing algorithm (either two-phase message passing, or in asynchronous mode). , where

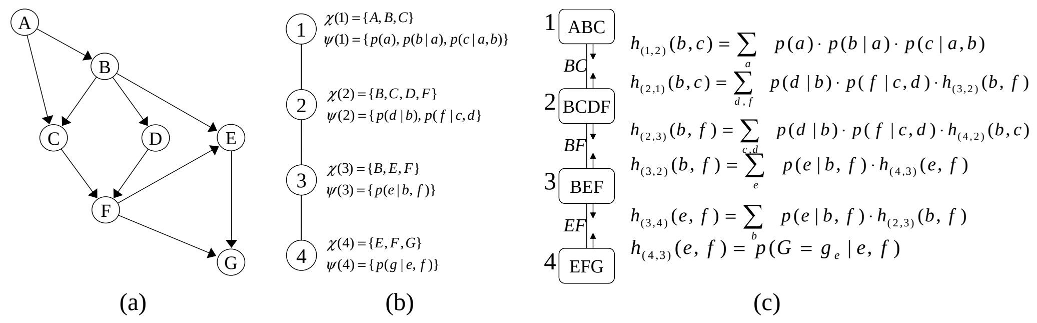

A
B
C
D
E
F
G
(a)
1
χ(1) = {A, B, C}
ψ(1) = {p(a), p(b|a), p(c|a,b)}
2
χ(2) = {B, C, D, F}
ψ(2) = {p(d|b), p(f|c,d)}
3
χ(3) = {B, E, F}
ψ(3) = {p(e|b, f)}
4
χ(4) = {E, F, G}
ψ(4) = {p(g|e, f)}
(b)
1 ABC
BC
2 BCDF
BF
3 BEF
EF
4 EFG
h_(1,2)(b,c) = Σ_a p(a)·p(b|a)·p(c|a,b)
h_(2,1)(b,c) = Σ_{d,f} p(d|b)·p(f|c,d)·h_(3,2)(b,f)
h_(2,3)(b,f) = Σ_{c,d} p(d|b)·p(f|c,d)·h_(4,2)(b,c)
h_(3,2)(b,f) = Σ_e p(e|b,f)·h_(4,3)(e,f)
h_(3,4)(e,f) = Σ_b p(e|b,f)·h_(2,3)(b,f)
h_(4,3)(e,f) = p(G = g_e | e,f)
(c)

Figure 4.1: $a)$ A belief network; $b)$ A join-tree decomposition; $c)$Execution of CTE-BU; no individual functions appear in this case

messages are computed by summation over the eliminator between the two clusters of the product of functions in the originating cluster. Algorithm CTE for belief updating denoted CTE-BU is given in Figure 4.2. The algorithm pays a special attention to the processing of observed variables since the presence of evidence is a central component in belief updating. When a cluster sends a message to a neighbor, the algorithm operates on all the functions in the cluster except the message from that particular neighbor. The message contains a single *combined* function and *individual* functions that do not share variables with the relevant eliminator. All the non-individual functions are *combined* in a product and summed over the eliminator.

Example 4.2.1 *Figure 4.1 describes a belief network (a) and a join-tree decomposition for it (b). Figure 4.1c shows the trace of running CTE-BU. In this case no individual functions appear between any of the clusters. To keep the figure simple, we only show the combined functions $h_{(u,v)}$ (each of them being in fact the only element of the set $H_{(u,v)}$ that represents the corresponding message between clusters u and v).*

THEOREM **4.2.2 (Complexity of CTE-BU)** *[34, 58] The time complexity of CTE-BU is $O(deg \cdot (n+N) \cdot d^{w^*+1})$ and the space complexity is $O(N \cdot d^{sep})$, where deg is the maximum degree of a node in the tree, n is the number of variables, N is the number of nodes in the tree decomposition, d is the maximum domain size of a variable, w^* is the treewidth and sep is the maximum separator size.*

```
Algorithm **CTE for Belief-Updating (CTE-BU)**
**Input:** A tree decomposition $< T, \chi, \psi >$, $T = (V, E)$ for $BN =< X, D, G, P >$.
Evidence variables $var(e)$.
**Output:** An augmented tree whose nodes are clusters containing the original CPTs and
the messages received from neighbors. $P(X_i, e)$, $\forall X_i \in X$.
Denote by $H_{(u,v)}$ the message from vertex $u$ to $v$, $ne_v(u)$ the neighbors of $u$ in $T$
excluding $v$.
$cluster(u) = \psi(u) \cup \{H_{(v,u)} | (v, u) \in E\}$.
$cluster_v(u) = cluster(u)$ excluding message from $v$ to $u$.

• **Compute messages:**
**For** every node $u$ in $T$, once $u$ has received messages from all $ne_v(u)$, compute message
to node $v$:
    1. **Process observed variables:**
       Assign relevant evidence to all $p_i \in \psi(u)$
    2. **Compute the combined function:**
            $h_{(u,v)} = \sum_{elim(u,v)} \prod_{f \in A} f.$
       where $A$ is the set of functions in $cluster_v(u)$ whose scope intersects $elim(u, v)$.
       Add $h_{(u,v)}$ to $H_{(u,v)}$ and add all the individual functions in $cluster_v(u) - A$
       Send $H_{(u,v)}$ to node $v$.

• **Compute** $P(X_i, e)$**:**
For every $X_i \in X$ let $u$ be a vertex in $T$ such that $X_i \in \chi(u)$. Compute $P(X_i, e) =$
$\sum_{\chi(u)-\{X_i\}} (\prod_{f \in cluster(u)} f)$
```

Figure 4.2: Algorithm Cluster-Tree-Elimination for Belief Updating (CTE-BU)

4.2.2 Mini-Clustering for Belief Updating

The time, and especially the space complexity of CTE-BU renders the algorithm infeasible for problems with large treewidth. In this section we introduce the Mini-Clustering, a partition-based anytime algorithm which computes approximate values or bounds on $P(X_i, e)$ for every variable X_i in the network. It is a natural extension of the mini-bucket idea to tree-decompositions. Rather than computing the mini-bucket approximation n times, one for each variable as would be required by the mini-bucket approach, the algorithm performs an equivalent computation with just two message passings along each arc

```
Procedure **MC for Belief Updating (MC-BU($i$))**

  2. **Compute the combined mini-functions:**
     Make an ($i$)-size mini-clusters partitioning of $cluster_v(u)$, $\{mc(1), \ldots, mc(p)\}$;
     $h^1_{(u,v)} = \sum_{elim(u,v)} \prod_{f \in mc(1)} f$
     $h^i_{(u,v)} = \max_{elim(u,v)} \prod_{f \in mc(i)} f \quad i = 2, \ldots, p$
     add $\{h^i_{(u,v)} | i = 1, \ldots, p\}$ to $H_{(u,v)}$. Send $H_{(u,v)}$ to $v$.

**Compute upper bounds on $P(X_i, e)$:**
For every $X_i \in X$ let $u \in V$ be a cluster such that $X_i \in \chi(u)$. Make ($i$) mini-clusters
from $cluster(u)$, $\{mc(1), \ldots, mc(p)\}$; Compute
$(\sum_{\chi(u)-X_i} \prod_{f \in mc(1)} f) \cdot (\prod_{k=2}^{p} \max_{\chi(u)-X_i} \prod_{f \in mc(k)} f)$.
```

Figure 4.3: Procedure Mini-Clustering for Belief Updating (MC-BU)

of the cluster tree. The idea is to partition each cluster into mini-clusters having at most i variables, where i is an accuracy parameter. Node u partitions its cluster into p mini-clusters $mc(1), \ldots, mc(p)$. Instead of computing $h_{(u,v)} = \sum_{elim(u,v)} \prod_{k=1}^{p} \prod_{f \in mc(k)} f$ as in CTE-BU, we can compute an upper bound by migrating the summation operator into each mini-cluster. However, this would give $\prod_{k=1}^{p} \sum_{elim(u,v)} \prod_{f \in mc(k)} f$ which is an unnecessarily large upper bound on $h_{(u,v)}$ in which each $\prod_{f \in mc(k)} f$ is bounded by its sum over $elim(u,v)$. Instead, we rewrite $h_{(u,v)} = \sum_{elim(u,v)} (\prod_{f \in mc(1)} f) \cdot (\prod_{i=2}^{p} \prod_{f \in mc(i)} f)$. Subsequently, instead of bounding $\prod_{f \in mc(i)} f, (i \geq 2)$ by summation over the eliminator, we bound it by its maximum over the eliminator, which yields $(\sum_{elim(u,v)} \prod_{f \in mc(1)} f) \cdot \prod_{k=2}^{p} (\max_{elim(u,v)} \prod_{f \in mc(k)} f)$. Therefore, if we are interested in an upper bound, we marginalize one mini-cluster by summation and the others by maximization. Note that the summation in the first mini-cluster must be over *all* variables in the eliminator, even if some of them might not appear in the scope of functions in $mc(1)$.

Consequently, the combined functions are approximated via mini-clusters, as follows. Suppose $u \in V$ has received messages from all its neighbors other than v (the message from v is ignored even if received). The functions in $cluster_v(u)$ that are to be combined are partitioned into mini-clusters $\{mc(1), \ldots, mc(p)\}$, each one containing at most i variables. One of the mini-clusters is processed by summation and the others by maximization over the eliminator, and the resulting combined functions as well as all the individual functions are sent to v.

Lower Bounds and Mean Approximations

We can also derive a lower-bound on beliefs by replacing the max operator with min operator (see above derivation for rationale). This allows, in principle, computing both an upper bound and a lower bound on the joint beliefs. Alternatively, if we yield the idea of deriving a bound (and indeed the empirical evaluation encourages that) we can replace max by a $mean$ operator (taking the sum and dividing by the number of elements in the sum), deriving an approximation of the joint belief.

Algorithm MC-BU for upper bounds can be obtained from CTE-BU by replacing step 2 of the main loop and the final part of computing the upper bounds on the joint belief by the procedure given in Figure 4.3.

Partitioning Strategies

In the implementation we used for the experiments reported here, the partitioning was done in greedy brute-force manner guided only by the sizes of the functions, and the choice of the first mini-cluster for upper bound computation was random. This had the advantage of adding a very small overhead to the Mini-Clustering algorithm. Clearly, more informed schemes that take into account the actual information in the tables of the functions may improve the overall accuracy.

Example 4.2.3 *Figure 4.4 shows the trace of running MC-BU(3) on the problem in Figure 4.1. First, evidence $G = g_e$ is assigned in all CPTs. There are no individual functions to be sent from cluster 1 to cluster 2. Cluster 1 contains only 3 variables, $\chi(1) = \{A, B, C\}$, therefore it is not partitioned. The combined function $h^1_{(1,2)}(b, c) = \sum_a p(a) \cdot p(b|a) \cdot$*

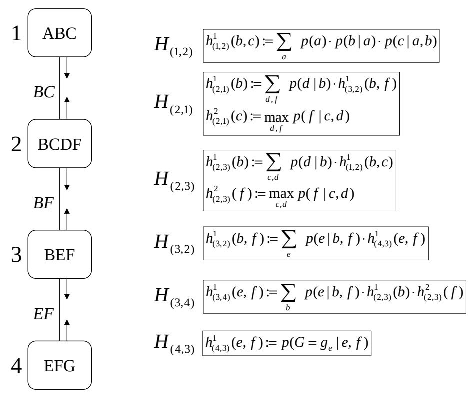

1
ABC
BC
2
BCDF
BF
3
BEF
EF
4
EFG
$H_{(1,2)}$
$h^1_{(1,2)}(b,c) := \sum_a p(a) \cdot p(b|a) \cdot p(c|a,b)$
$H_{(2,1)}$
$h^1_{(2,1)}(b) := \sum_{d,f} p(d|b) \cdot h^1_{(3,2)}(b,f)$
$h^2_{(2,1)}(c) := \max_{d,f} p(f|c,d)$
$H_{(2,3)}$
$h^1_{(2,3)}(b) := \sum_{c,d} p(d|b) \cdot h^1_{(1,2)}(b,c)$
$h^2_{(2,3)}(f) := \max_{c,d} p(f|c,d)$
$H_{(3,2)}$
$h^1_{(3,2)}(b,f) := \sum_e p(e|b,f) \cdot h^1_{(4,3)}(e,f)$
$H_{(3,4)}$
$h^1_{(3,4)}(e,f) := \sum_b p(e|b,f) \cdot h^1_{(2,3)}(b) \cdot h^2_{(2,3)}(f)$
$H_{(4,3)}$
$h^1_{(4,3)}(e,f) := p(G = g_e|e,f)$

Figure 4.4: Execution of MC-BU for $i = 3$

$p(c|a,b)$ is computed and the message $H_{(1,2)} = \{h^1_{(1,2)}(b,c)\}$ is sent to node 2. Now, node 2 can send its message to node 3. Again, there are no individual functions. Cluster 2 contains 4 variables, $\chi(2) = \{B, C, D, F\}$, and a partitioning is necessary: MC-BU(3) can choose $mc(1) = \{p(d|b), h_{(1,2)}(b,c)\}$ and $mc(2) = \{p(f|c,d)\}$. The combined functions $h^1_{(2,3)}(b) = \sum_{c,d} p(d|b) \cdot h_{(1,2)}(b,c)$ and $h^2_{(2,3)}(f) = \max_{c,d} p(f|c,d)$ are computed and the message $H_{(4,3)} = \{h^1_{(2,3)}(b), h^2_{(2,3)}(f)\}$ is sent to node 3. The algorithm continues until every node has received messages from all its neighbors. An upper bound on $p(a, G = g_e)$ can now be computed by choosing cluster 1, which contains variable A. It doesn't need partitioning, so the algorithm just computes $\sum_{b,c} p(a) \cdot p(b|a) \cdot p(c|a,b) \cdot h^1_{(2,1)}(b) \cdot h^2_{(2,1)}(c)$. Notice that unlike CTE-BU which processes 4 variables in cluster 2, MC-BU(3) never processes more than 3 variables at a time.

4.2.3 Properties of Mini-Clustering

THEOREM **4.2.4** *Algorithm MC-BU(i) with max (respectively min) computes an upper (respectively lower) bound on the joint probability $P(X, e)$ of each variable and each of its values.*

A similar mini-clustering scheme for combinatorial optimization was developed in [34] having similar performance properties as MC-BU.

THEOREM **4.2.5 (Complexity of MC-BU(i))** *[34] The time and space complexity of MC-BU(i) is $O(n \cdot hw^* \cdot d^i)$ where n is the number of variables, d is the maximum domain size of a variable and $hw^* = max_u|\{f|scope(f) \cap \chi(u) \neq \phi\}|$, which bounds the number of functions that may travel to a neighboring cluster via message-passing.*

Accuracy

For a given i, the accuracy of MC-BU(i) can be shown to be not worse than that of executing the mini-bucket algorithm MB(i) n times, once for each variable (an algorithm that we call nMB(i)). Given a specific execution of MC-BU(i), we can show that for every variable X_i, there exists an ordering of the variables and a corresponding partitioning such that MB(i) computes the same approximation value for $P(X_i, e)$ as does $MC - BU(i)$. In empirical analysis [58] it is shown that MC-BU has an up to linear speed-up over nMB(i).

Normalization

The MC-BU algorithm using max operator computes an upper bound $\overline{P}(X_i, e)$ on the joint probability $P(X_i, e)$. However, deriving a bound on the conditional probability $P(X_i|e)$ is not easy when the exact value of $P(e)$ is not available. If we just try to divide (multiply) $\overline{P}(X_i, e)$ by a constant, the result is not necessarily an upper bound on $P(X_i|e)$. In principle, if we can derive a lower bound $\underline{P}(e)$ on $P(e)$, then we can compute $\overline{P}(X_i, e)/\underline{P}(e)$ as an upper bound on $P(X_i|e)$. However, due to compound error, it is likely to be ineffective. In our empirical evaluation we experimented with normalizing the upper bound as $\overline{P}(X_i, e)/\sum_{X_i} \overline{P}(X_i, e)$ over the values of X_i. The result is not necessarily an upper bound on P$(X_i|e)$. Similarly, we can also normalize the values when using $mean$ or min operators. It is easy to show that normalization with the $mean$ operator is identical to normalization of MC-BU output when applying the summation operator in all the mini-clusters.

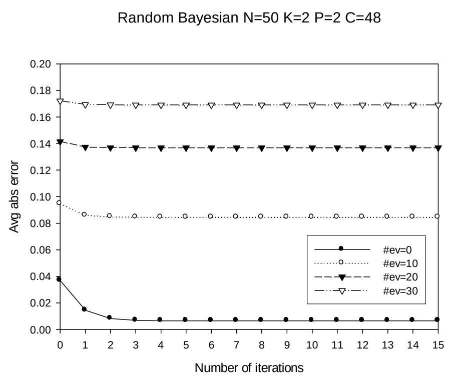


Figure 4.5: Convergence of IBP

4.2.4 Experimental Evaluation

We tested the performance of our scheme on random noisy-or networks, random coding networks, general random networks, grid networks, and three benchmark CPCS files with 54, 360 and 422 variables respectively (these are belief networks for medicine, derived from the Computer based Patient Case Simulation system, known to be hard for belief updating). On each type of network we ran Iterative Belief Propagation (IBP) - set to run at most 30 iterations, Gibbs Sampling (GS) and MC-BU(i), with i from 2 to the treewidth w^* to capture the anytime behavior of MC-BU.

We immediately observed that the quality of MC-BU in providing upper or lower bounds on the joint $P(X_i, e)$ was ineffective. Although the upper bound decreases as the accuracy parameter i increases, it still is in most cases greater than 1. Therefore, following the ideas explained in the previous subsection 4.2.3 we report the results with normalizing the upper bounds (called max) and normalizing the mean (called $mean$). We notice that MC-BU using the $mean$ operator is doing consistently better.

For noisy-or networks, general random networks, grid networks and for the CPCS networks we computed the exact solution and used three different measures of accuracy: 1. Normalized Hamming Distance (NHD) - We picked the most likely value for each variable

Noisy-OR networks, w*=10

N=50, P=2, 50 instances

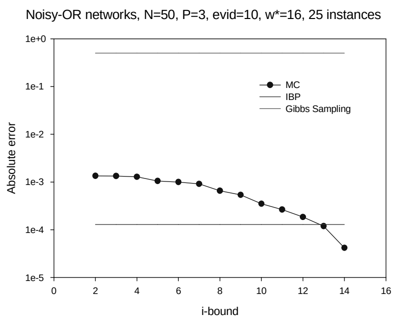
	|e|	NHD		Abs. Error		Rel. Error		Time	
		max	mean	max	mean	max	mean	max	mean
IBP	0		0		9.0E-09		1.1E-05		0.102
	10		0		3.4E-04		4.2E-01		0.081
	20		0		9.6E-04		1.2E+00		0.062
MC-BU(2)	0	0	0	1.6E-03	1.1E-03	1.9E+00	1.3E+00	0.056	0.057
	10	0	0	1.1E-03	8.4E-04	1.4E+00	1.0E+00	0.048	0.049
	20	0	0	5.7E-04	4.8E-04	7.1E-01	5.9E-01	0.039	0.039
MC-BU(5)	0	0	0	1.1E-03	9.4E-04	1.4E+00	1.2E+00	0.070	0.072
	10	0	0	7.7E-04	6.9E-04	9.3E-01	8.4E-01	0.063	0.066
	20	0	0	2.8E-04	2.7E-04	3.5E-01	3.3E-01	0.058	0.057
MC-BU(8)	0	0	0	3.6E-04	3.2E-04	4.4E-01	3.9E-01	0.214	0.221
	10	0	0	1.7E-04	1.5E-04	2.0E-01	1.9E-01	0.184	0.190
	20	0	0	3.5E-05	3.5E-05	4.3E-02	4.3E-02	0.123	0.127

Noisy-OR networks, w*=16

N=50, P=3, 25 instances

	|e|	NHD		Abs. Error		Rel. Error		Time	
		max	mean	max	mean	max	mean	max	mean
IBP	10		0		1.3E-04		7.9E-01		0.242
	20		0		3.6E-04		2.2E+00		0.184
	30		0		6.8E-04		4.2E+00		0.121
MC-BU(2)	10	0	0	1.3E-03	9.6E-04	8.2E+00	5.8E+00	0.107	0.108
	20	0	0	5.3E-04	4.0E-04	3.1E+00	2.4E+00	0.077	0.077
	30	0	0	2.3E-04	1.9E-04	1.4E+00	1.2E+00	0.064	0.064
MC-BU(5)	10	0	0	1.0E-03	8.3E-04	6.4E+00	5.1E+00	0.133	0.133
	20	0	0	4.6E-04	4.1E-04	2.7E+00	2.4E+00	0.104	0.105
	30	0	0	2.0E-04	1.9E-04	1.2E+00	1.2E+00	0.098	0.095
MC-BU(8)	10	0	0	6.6E-04	5.7E-04	4.0E+00	3.5E+00	0.498	0.509
	20	0	0	1.8E-04	1.8E-04	1.1E+00	1.0E+00	0.394	0.406
	30	0	0	3.4E-05	3.4E-05	2.1E-01	2.1E-01	0.300	0.308
MC-BU(11)	10	0	0	2.6E-04	2.4E-04	1.6E+00	1.5E+00	2.339	2.378
	20	0	0	3.8E-05	3.8E-05	2.3E-01	2.3E-01	1.421	1.439
	30	0	0	6.4E-07	6.4E-07	4.0E-03	4.0E-03	0.613	0.624
MC-BU(14)	10	0	0	4.2E-05	4.1E-05	2.5E-01	2.4E-01	7.805	7.875
	20	0	0	0	0	0	0	2.075	2.093
	30	0	0	0	0	0	0	0.630	0.638

Table 4.1: Performance on Noisy-OR networks;



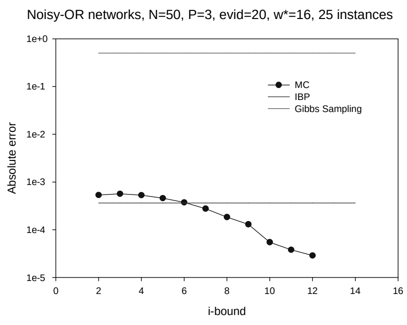


Figure 4.6: Absolute error for noisy-OR networks

for the approximate and for the exact, took the ratio between the number of disagreements and the total number of variables, and averaged over the number of problems that we ran for each class. 2. Absolute Error (Abs. Error) - is the absolute value of the difference between the approximate and the exact, averaged over all values (for each variable), all variables and all problems. 3. Relative Error (Rel. Error) - is the absolute value of the difference between the approximate and the exact, divided by the exact, averaged over all values (for each variable), all variables and all problems. For coding networks, we report only one measure, Bit Error Rate (BER). In terms of the measures defined above, BER is the normalized Hamming distance between the approximate (computed by an algorithm) and the actual input (which in the case of coding networks may be different from the solution given by exact algorithms), so we denote them differently to make this semantic distinction. We also show the time taken by each algorithm.

Random networks, w*=10

N=50, P=2, 50 instances

0 $\|e\|$ 10 20	NHD		Abs. Error		Rel. Error		Time	
	max	mean	max	mean	max	mean	max	mean
		0.01840		0.00696		0.01505		0.100
IBP		0.19550		0.09022		0.34608		0.080
		0.27467		0.13588		3.13327		0.062
		0.50400		0.10715		0.26621		13.023
GS		0.51400		0.15216		0.57262		12.978
		0.51267		0.18066		4.71805		13.321
	0.11400	0.08080	0.03598	0.02564	0.07950	0.05628	0.055	0.055
MC-BU(2)	0.10600	0.08800	0.04897	0.03957	0.12919	0.10579	0.047	0.048
	0.08667	0.07333	0.04443	0.03639	0.13096	0.10694	0.041	0.042
	0.10120	0.06480	0.03392	0.02242	0.07493	0.04937	0.071	0.072
MC-BU(5)	0.06950	0.05850	0.03254	0.02723	0.08613	0.07313	0.063	0.065
	0.03933	0.03400	0.02022	0.01831	0.05533	0.04984	0.059	0.060
	0.05080	0.02680	0.01872	0.01030	0.04103	0.02262	0.216	0.221
MC-BU(8)	0.01550	0.01450	0.00743	0.00587	0.01945	0.01547	0.178	0.180
	0.00600	0.00400	0.00228	0.00200	0.00597	0.00542	0.129	0.134

Random networks, w*=16

N=50, P=3, 25 instances

0 $\|e\|$ 10 20	NHD		Abs. Error		Rel. Error		Time	
	max	mean	max	mean	max	mean	max	mean
		0.03652		0.00907		0.01894		0.298
IBP		0.25200		0.08319		0.22335		0.240
		0.34000		0.13995		0.91671		0.183
	0.17304		0.04377		0.09395		0.140	
MC-BU(2)	0.17600	0.11600	0.05930	0.04558	0.14706	0.11034	0.100	0.103
	0.15067	0.14000	0.07658	0.06683	0.23155	0.19538	0.075	0.078
	0.15652		0.04380		0.09398		0.158	
MC-BU(5)	0.15600	0.11800	0.05665	0.04320	0.13484	0.10221	0.124	0.129
	0.09467	0.09467	0.05545	0.05049	0.15000	0.13706	0.105	0.107
	0.16783		0.04166		0.08904		0.602	
MC-BU(8)	0.09800	0.08100	0.04051	0.03254	0.09923	0.07942	0.481	0.491
	0.05467	0.04533	0.02939	0.02691	0.07865	0.07237	0.385	0.393
	0.12087		0.03076		0.06550		2.986	
MC-BU(11)	0.05500	0.04700	0.02425	0.01946	0.05644	0.04533	2.307	2.345
	0.00800	0.00533	0.00483	0.00431	0.01307	0.01156	1.564	1.585
	0.06348		0.01910		0.04071		14.910	
MC-BU(14)	0.01400	0.01200	0.00542	0.00434	0.01350	0.01108	8.548	8.578
	0.00000	0.00000	0.00089	0.00089	0.00212	0.00211	3.656	3.676

Table 4.2: Performance on random networks

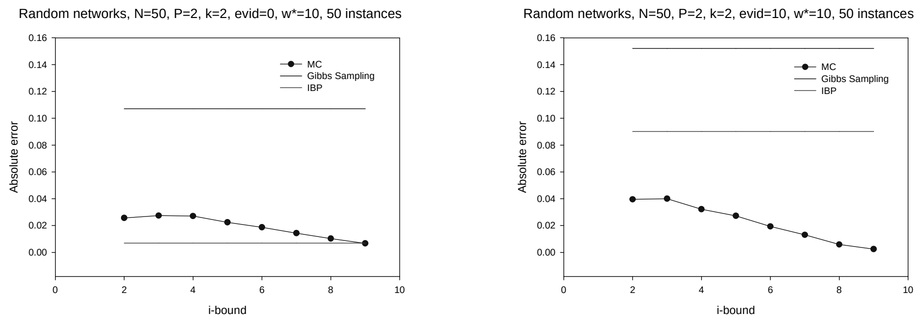


Figure 4.7: Absolute error for random networks

In Figure 4.5 we show that IBP converges after about 5 iterations. So, while in our experiments we report its time for 30 iterations, its time is even better when sophisticated termination is used. These results are typical of all runs.

The random noisy-or networks and the random networks were generated using parameters (N,K,C,P), where N is the number of variables (a square integer for grid networks), K is their domain size (we used only K=2), C is the number of conditional probability matrices and P is the number of parents in each conditional probability matrix. The grid networks have the structure of a square, with edges directed to form a diagonal flow (all parallel edges have the same direction). They were generated by specifying N (a square integer) and K (we used K=2). We also varied the number of evidence nodes, denoted by $|e|$ in the tables. The parameter values are reported in each table.

Comment: We should note that since our evaluation measures are based on comparing

	$\sigma = .22$		$\sigma = .26$		$\sigma = .32$		$\sigma = .40$		$\sigma = .51$		
BER	max	mean	max	mean	max	mean	max	mean	max	mean	Time
N=100, P=3, 50 instances, w*=7											
IBP	0.000	0.000	0.000	0.000	0.002	0.002	0.022	0.022	0.088	0.088	0.00
GS	0.483	0.483	0.483	0.483	0.483	0.483	0.483	0.483	0.483	0.483	31.36
MC-BU(2)	0.002	0.002	0.004	0.004	0.024	0.024	0.068	0.068	0.132	0.131	0.08
MC-BU(4)	0.001	0.001	0.002	0.002	0.018	0.018	0.046	0.045	0.110	0.110	0.08
MC-BU(6)	0.000	0.000	0.000	0.000	0.004	0.004	0.038	0.038	0.106	0.106	0.12
MC-BU(8)	0.000	0.000	0.000	0.000	0.002	0.002	0.023	0.023	0.091	0.091	0.19
N=100, P=4, 50 instances, w*=11											
IBP	0.000	0.000	0.000	0.000	0.002	0.002	0.013	0.013	0.075	0.075	0.00
GS	0.506	0.506	0.506	0.506	0.506	0.506	0.506	0.506	0.506	0.506	39.85
MC-BU(2)	0.006	0.006	0.015	0.015	0.043	0.043	0.093	0.094	0.157	0.157	0.19
MC-BU(4)	0.006	0.006	0.017	0.017	0.049	0.049	0.104	0.102	0.158	0.158	0.19
MC-BU(6)	0.005	0.005	0.011	0.011	0.035	0.034	0.071	0.074	0.151	0.150	0.29
MC-BU(8)	0.002	0.002	0.004	0.004	0.022	0.022	0.059	0.059	0.121	0.122	0.71
MC-BU(10)	0.001	0.001	0.001	0.001	0.008	0.008	0.033	0.032	0.101	0.102	1.87

Table 4.3: BER for coding networks

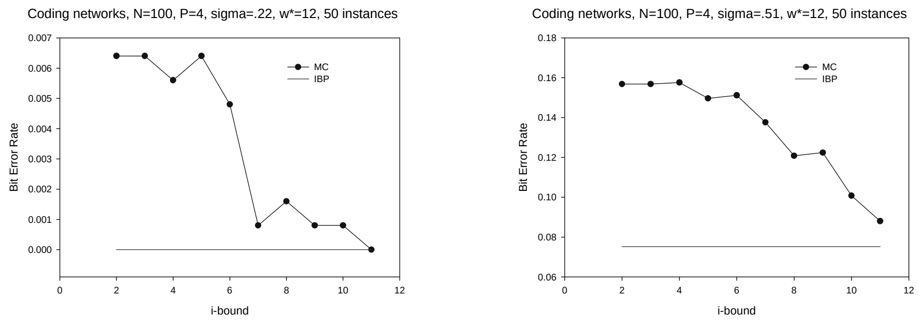


Figure 4.8: BER for coding networks

against exact figures, we had to restrict the instances to be relatively small or sparse enough to be managed by exact algorithms.

For all the problems, Gibbs sampling performed consistently poorly so we only include part of the results in the following tables and figures.

Random noisy-or networks results are summarized in Table 4.1 and Figure 4.6. For NHD, both IBP and MC-BU gave perfect results. For the other measures, we noticed that IBP is more accurate for no evidence by about an order of magnitude. However, as evidence is added, IBP's accuracy decreases, while MC-BU's increases and they give similar results. We also notice that MC-BU gets better as the accuracy parameter i increases, which shows its anytime behavior. We also observed a similar pattern of behavior when experimenting with smaller noisy-or networks, generated with P=2 (w*=10).

Grid 13x13, w*=21

N=169, 25 instances, $mean$ operator				
$\|e\|$ = 0, 10, 20, 30	NHD	Abs. Error	Rel. Error	Time
IBP	0.0102	0.0038	0.0083	0.053
	0.0745	0.0323	0.0865	0.047
	0.1350	0.0551	0.3652	0.044
	0.1672	0.0759	0.2910	0.044
GS	0.5172	0.1111	0.2892	6.634
	0.4901	0.1229	0.3393	6.667
	0.5205	0.1316	0.7320	6.787
	0.4921	0.1431	0.5455	6.806
MC-BU(2)	0.1330	0.0464	0.1034	0.044
	0.1263	0.0482	0.1103	0.028
	0.1388	0.0479	0.1117	0.026
	0.1168	0.0513	0.1256	0.024
MC-BU(6)	0.1001	0.0337	0.0731	0.044
	0.0863	0.0313	0.0697	0.040
	0.0805	0.0268	0.0605	0.041
	0.0581	0.0263	0.0610	0.036
MC-BU(10)	0.0402	0.0144	0.0310	0.235
	0.0330	0.0115	0.0252	0.220
	0.0223	0.0092	0.0211	0.206
	0.0224	0.0086	0.0195	0.191
MC-BU(14)	0.0151	0.0056	0.0123	1.246
	0.0151	0.0051	0.0113	1.340
	0.0137	0.0044	0.0101	1.306
	0.0124	0.0032	0.0073	1.256
MC-BU(17)	0.0088	0.0027	0.0059	6.916
	0.0045	0.0018	0.0040	5.889
	0.0030	0.0010	0.0022	5.219
	0.0023	0.0008	0.0018	4.354

Grid 15x15, w*=22

N=225, 10 instances, $mean$ operator				
$\|e\|$ = 0, 10, 20, 30	NHD	Abs. Error	Rel. Error	Time
	0.0094	0.0037	0.0080	0.071
IBP	0.0665	0.0665	0.0761	0.070
	0.1205	0.0463	0.1894	0.068
	0.1462	0.0632	0.1976	0.062
	0.5178	0.1096	0.2688	9.339
GS	0.5047	0.5047	0.3200	9.392
	0.4849	0.1232	0.4009	9.524
	0.4692	0.1335	0.4156	9.220
	0.1256	0.0474	0.1071	0.049
MC-BU(2)	0.1312	0.1312	0.1070	0.041
	0.1371	0.0523	0.1205	0.042
	0.1287	0.0512	0.1201	0.053
	0.1050	0.0356	0.0775	0.217
MC-BU(6)	0.0944	0.0944	0.0720	0.064
	0.0844	0.0313	0.0701	0.059
	0.0759	0.0286	0.0652	0.120
	0.0406	0.0146	0.0313	0.500
MC-BU(10)	0.0358	0.0358	0.0288	0.368
	0.0337	0.0122	0.0272	0.484
	0.0256	0.0116	0.0265	0.468
	0.0233	0.0081	0.0173	2.315
MC-BU(14)	0.0209	0.0209	0.0152	2.342
	0.0146	0.0055	0.0126	2.225
	0.0118	0.0046	0.0105	2.350
	0.0089	0.0031	0.0065	10.990
MC-BU(17)	0.0116	0.0116	0.0069	10.105
	0.0063	0.0022	0.0048	9.381
	0.0036	0.0017	0.0038	9.573

Table 4.4: Performance on grid networks;

General random networks results are summarized in Table 4.2 and Figure 4.7. They are in general similar to those for random noisy-or networks. NHD is non-zero in this case. Again, IBP has the best result only for few evidence variables. It is remarkable how quickly MC-BU surpasses the performance of IBP as evidence is added. We also experimented with larger networks generated with P=3 (w*=16) and observed a similar behavior.

Random coding networks results are given in Table 4.3 and Figure 4.8. The instances fall within the class of linear block codes, (σ is the channel noise level). It is known that IBP is very accurate for this class. Indeed, these are the only problems that we experimented with where IBP outperformed MC-BU throughout. The anytime behavior of MC-BU can again be seen in the variation of numbers in each column.

Grid networks results are given in Table 4.4 and Figure 4.9. We only report results with $mean$ operator for a 15x15 grid for which the induced width is w*=22. We notice that IBP is more accurate for no evidence and MC is better as more evidence is added. The same behavior was consistently manifested for smaller grid networks that we experimented with (from 7x7 up to 14x14).

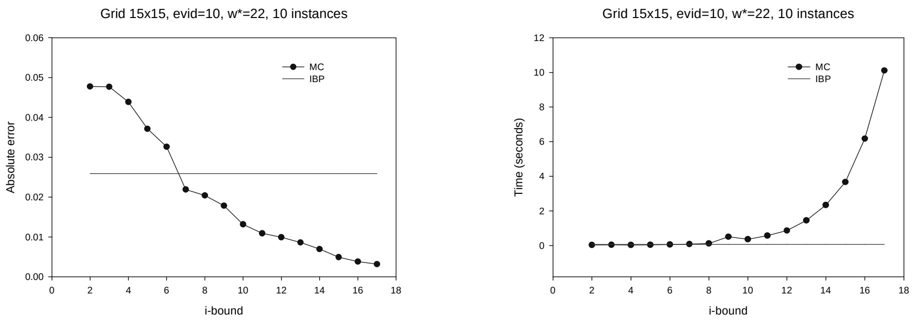


Figure 4.9: Absolute error and time for grid networks

N=54, 50 instances								
0 $\|e\|$ 10 20	NHD		Abs. Error		Rel. Error		Time	
	max	mean	max	mean	max	mean	max	mean
		0.01852		0.00032		0.00064		2.450
IBP		0.15727		0.03307		0.07349		2.191
		0.20765		0.05934		0.14202		1.561
		0.49444		0.07797		0.18034		17.247
GS		0.51409		0.09002		0.21298		17.208
		0.48706		0.10608		0.26853		17.335
	0.16667	0.07407	0.02722	0.01221	0.05648	0.02520	0.154	0.153
MC-BU(2)	0.11636	0.07636	0.02623	0.01843	0.05581	0.03943	0.096	0.095
	0.10529	0.07941	0.02876	0.02196	0.06357	0.04878	0.067	0.067
	0.18519	0.09259	0.02488	0.01183	0.05128	0.02454	0.157	0.155
MC-BU(5)	0.10727	0.07682	0.02464	0.01703	0.05239	0.03628	0.112	0.112
	0.08059	0.05941	0.02174	0.01705	0.04790	0.03778	0.090	0.087
	0.12963	0.07407	0.01487	0.00619	0.03047	0.01273	0.438	0.446
MC-BU(8)	0.06591	0.05000	0.01590	0.01040	0.03394	0.02227	0.369	0.370
	0.03235	0.02588	0.00977	0.00770	0.02165	0.01707	0.292	0.294
	0.11111	0.07407	0.01133	0.00688	0.02369	0.01434	2.038	2.032
MC-BU(11)	0.02818	0.01500	0.00600	0.00398	0.01295	0.00869	1.567	1.571
	0.00353	0.00353	0.00124	0.00101	0.00285	0.00236	0.867	0.869

Table 4.5: Performance on CPCS54 network, w*=15

CPCS networks results We also tested on three CPCS benchmark files. The results are given in Tables 4.5 and 4.6 and in Figure 4.10. It is interesting to notice that the MC scheme scales up even to fairly large networks, like the real life example of CPCS422 (induced width 23). IBP is again slightly better for no evidence, but is quickly surpassed by MC when evidence is added.

4.2.5 Discussion

We presented in this section an approximation scheme for probabilistic inference, one of the most important task over belief networks. The scheme, called Mini-Clustering, is governed by a controlling parameter that allows adjustable levels of accuracy and efficiency in an anytime style.

CPCS360, w*=20

N=360, 5 instances, $mean$ operator				
$\|e\|$ = 0, 20, 40	NHD	Abs. Error	Rel. Error	Time
IBP	0.0000	0.0027	0.0054	82
	0.0112	0.0256	3.4427	76
	0.0363	0.0629	736.1080	60
MC-BU(8)	0.0056	0.0125	0.0861	16
	0.0041	0.0079	0.0785	14
	0.0113	0.0109	0.2997	9
MC-BU(11)	0.0000	0.0080	0.0636	38
	0.0000	0.0048	0.0604	39
	0.0088	0.0102	0.1733	33
MC-BU(14)	0.0000	0.0030	0.0192	224
	0.0012	0.0045	0.0502	232
	0.0056	0.0070	0.0693	200
MC-BU(17)	0.0000	0.0016	0.0073	1433
	0.0006	0.0026	0.0266	1455
	0.0013	0.0006	0.0045	904

CPCS422, w*=23

N=422, 1 instance, $mean$ operator				
$\|e\|$ = 0, 20, 40	NHD	Abs. Error	Rel. Error	Time
IBP	0.0024	0.0062	0.0150	2838
	0.0721	0.0562	7.5626	2367
	0.0654	0.0744	37.5096	2150
MC-BU(3)	0.0687	0.0455	1.4341	161
	0.0373	0.0379	0.9792	85
	0.0366	0.0233	2.8384	48
MC-BU(7)	0.0545	0.0354	0.1531	146
	0.0249	0.0253	0.3112	77
	0.0262	0.0164	0.5781	45
MC-BU(11)	0.0166	0.0175	0.0738	152
	0.0448	0.0352	0.6113	95
	0.0340	0.0237	0.6978	63
MC-BU(15)	0.0024	0.0039	0.0145	526
	0.0398	0.0278	0.5338	564
	0.0183	0.0113	0.5248	547

Table 4.6: Performance on CPCS360 and CPCS422 networks

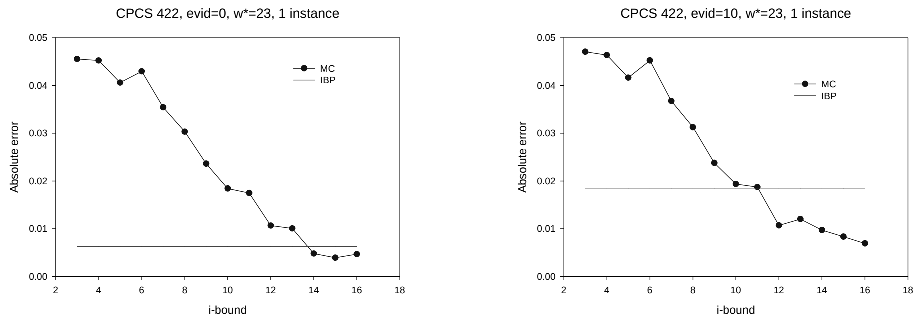


Figure 4.10: Absolute error for CPCS422

We presented empirical evaluation of mini-cluster approximation on several classes of networks, comparing its anytime performance with competing algorithms such as Gibbs Sampling and Iterative Belief Propagation, over benchmarks of noisy-or random networks, general random networks, grid networks, coding networks and CPCS type networks. Our results show that, as expected, IBP is superior to all other approximations for coding networks. However, for random noisy-or, general random networks, grid networks and the CPCS networks, in the presence of evidence, the mini-clustering scheme is often superior even in its weakest form. Gibbs sampling was particularly bad and we believe that enhanced variants of the Monte Carlo approach, such as likelihood weighting and importance sampling, should be compared with [17]. The empirical results are particularly encouraging as we use an unoptimized scheme that exploits a universal principle applicable to many reasoning tasks. Our contribution beyond recent works in this area [43, 34] is in: 1. Extending the partition-based approximation for belief updating from mini-buckets to general tree-decompositions, thus allowing the computation of the updated beliefs for all the variables at once. This extension is similar to the one proposed in [34] but replaces optimization with probabilistic inference. 2. Providing for the first time empirical evaluation demonstrating the effectiveness of the partition-based idea for belief updating.

There are many potential ways for improving the MC scheme. Among the most important, the partitioning step can be further elaborated. In the work presented here, we used only a brute-force approach for partitioning.

One extension of this work [39] is an iterative version of MC called Iterative Join-Graph Propagation (IJGP), which is both anytime and iterative and belongs to the class of generalized belief propagation methods [102]. Rather than assuming an underlying join-tree, IJGP works on a join-graph that may contain loops. IJGP is related to MC in a similar way as IBP is related to BP (Pearl's belief propagation). Experimental work shows that in most cases iterating improves the quality of the MC approximation even further, especially for low i-bounds. We will discuss this algorithm in detail in Section 4.3.

4.3 Iterative Join-Graph Propagation

This section contains our work on Iterative Join-Graph Propagation. The original motivation for designing this algorithm was in trying to combine the anytime feature of Mini-Clustering (MC) and the iterative virtues of Iterative Belief Propagation (IBP). MC is an anytime algorithm but it works on tree-decompositions and it converges in two passes, so iterating doesn't change the messages. IBP is an iterative algorithm that converges in most cases, and when it converges it does so very fast. Allowing it more time doesn't improve the accuracy. IJGP was designed to benefit from both these directions. It works on a general join-graph which may contain cycles. The cluster size of the graph is user adjustable by the *i-bound* (providing the anytime nature), and the cycles in the graph allow iterating. The precise mechanics of the algorithm are given in the following sections. Empirical results are also provided, showing that in many cases IJGP is superior to both MC and IBP on several classes of problems.

4.3.1 Join-Graphs

We will describe our algorithms relative to a join-graph decomposition framework using recent notation proposed by [52]. The notion of join-tree decompositions was introduced in relational databases [70].

DEFINITION **4.3.1 (join-graph decompositions)** *A* join-graph decomposition *for* $BN = < X, D, G, P >$ *is a triple* $D = < JG, \chi, \psi >$*, where* $JG = (V, E)$ *is a graph, and* χ *and* ψ *are labeling functions which associate with each vertex* $v \in V$ *two sets,* $\chi(v) \subseteq X$ *and* $\psi(v) \subseteq P$ *such that:*

1. *For each* $p_i \in P$*, there is* exactly *one vertex* $v \in V$ *such that* $p_i \in \psi(v)$*, and* $scope(p_i) \subseteq \chi(v)$.
2. *(connectedness) For each variable* $X_i \in X$*, the set* $\{v \in V | X_i \in \chi(v)\}$ *induces a connected subgraph of* G*. The connectedness requirement is also called the running intersection property.*

We will often refer to a node and its CPT functions as a *cluster*[1] and use the term *join-graph-decomposition* and *cluster graph* interchangeably. A *join-tree-decomposition* or a *cluster tree* is the special case when the join-graph *JG* is a tree.

Join-Tree Propagation

The well known join-tree clustering algorithm first converts the belief network into a cluster tree and then sends messages between clusters. We call the second message passing phase

[1]Note that a node may be associated with an empty set of CPTs

join-tree propagation. The complexity of join-tree clustering is exponential in the number of variables in a cluster (treewidth), and the number of variables in the intersections between adjacent clusters (separator-width), as defined below.

DEFINITION **4.3.2 (treewidth, separator-width)** *Let $D =< JT, \chi, \psi >$ be a tree decomposition of a belief network $< G, P >$. The* treewidth *of D [3] is $max_{v \in V} |\chi(v)|$. The treewidth of $< G, P >$ is the minimum treewidth over all its join-tree decompositions. Given two adjacent vertices u and v of JT, the* separator *of u and v is defined as $sep(u, v) = \chi(u) \cap \chi(v)$, and the* separator-width *is $max_{(u,v)} |sep(u, v)|$.*

The minimum treewidth of a graph G can be shown to be identical to a related parameter called *induced-width*. A join-graph decomposition D is *arc-minimal* if none of its arcs can be removed while still satisfying the connectedness property of Definition 4.3.1. If a graph-decomposition is not arc-minimal it is easy to remove some of its arcs until it becomes arc-minimal. In our preliminary experiments we observed immediately that when applying join-tree propagation on a join-graph iteratively, it is crucial to avoid cycling messages relative to every single variable. The property of arc-minimality is *not* sufficient to ensure such acyclicity though. What is required is that, for every node X, the arc-subgraph that contains X be a tree.

Example 4.3.1 *The example in Figure 4.11a shows an arc minimal join-graph which contains a cycle relative to variable 4, with arcs labeled with separators. Notice however that if we remove variable 4 from the label of one arc we will have no cycles (relative to single variables) while the connectedness property will still be maintained.*

To allow more flexible notions of connectedness we refine the definition of join-graph decompositions, when arcs can be labeled with a subset of their separator.

DEFINITION **4.3.3 ((minimal) arc-labeled join-graph decompositions)** *An* arc-labeled decomposition *for $BN =< X, D, G, P >$ is a four-tuple $D =< JG, \chi, \psi, \theta >$, where*

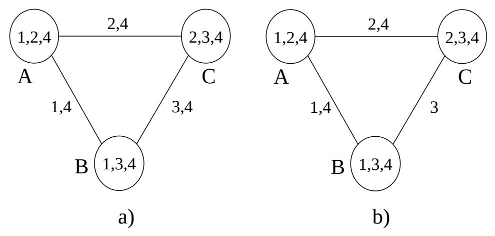


Figure 4.11: An arc-labeled decomposition

$JG = (V, E)$ *is a graph,* χ *and* ψ *associate with each vertex* $v \in V$ *the sets* $\chi(v) \subseteq X$ *and* $\psi(v) \subseteq P$ *and* θ *associates with each edge* $(v, u) \subset E$ *the set* $\theta((v, u)) \subseteq X$ *such that:*

1. *For each function* $p_i \in P$*, there is* exactly *one vertex* $v \in V$ *such that* $p_i \in \psi(v)$*, and* $scope(p_i) \subseteq \chi(v)$.
2. *(arc-connectedness) For each arc* (u, v), $\theta(u, v) \subseteq sep(u, v)$*, such that* $\forall X_i \in X$*, any two clusters containing* X_i *can be connected by a path whose every arc's label includes* X_i.

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

DEFINITION **4.3.4 (separator, eliminator)** *Given two adjacent vertices* u *and* v *of* JG*, the* separator *of* u *and* v *is defined as* $sep(u, v) = \theta((u, v))$*, and the* eliminator *of* u *with respect to* v *is* $elim(u, v) = \chi(u) - \theta((u, v))$.

Arc-labeled join-graphs can be made minimal by deleting variables from the labels. It is easy to see that a *minimal arc-labeled join-graph* does not contain any cycle relative to any single variable. That is, any two clusters containing the same variable are connected by exactly one path labeled with that variable.

4.3.2 Algorithm IJGP

Applying join-tree propagation iteratively to join-graphs yields algorithm *Iterative Join-Graph Propagation (IJGP)* described in Figure 4.12. One iteration of the algorithm applies message-passing in a topological order over the join-graph, forward and back.

```
Algorithm **Iterative Join Graph Propagation (IJGP)**

  Input   An arc-labeled join-graph decomposition $< JG, \chi, \psi, \theta >, JG = (V, E)$ for $BN =< X, D, G, P >$. Evidence variables $var(e)$.
  Output  An augmented graph whose nodes are clusters containing the original CPTs and the messages received from neighbors. Approximations of $P(X_i|e), \forall X_i \in X$.

Denote by $h_{(u,v)}$ the message from vertex $u$ to $v$, $ne_v(u)$ the neighbors of $u$ in $JG$ excluding $v$.
$cluster(u) = \psi(u) \cup \{h_{(v,u)}|(v,u) \in E\}$.
$cluster_v(u) = cluster(u)$ excluding message from $v$ to $u$.

• One iteration of IJGP:
For every node $u$ in $JG$ in some topological order $d$ and back,do
    1. Process observed variables:
       Assign relevant evidence to all $p_i \in \psi(u)$ $\chi(u) := \chi(u) - var(e), \ \forall u \in V$
    2. Compute individual functions:
       Include in $H_{(u,v)}$ each function in $cluster_v(u)$ whose scope does not contain variables in $elim(u,v)$. Denote by $A$ the remaining functions.
    3. Compute and send to $v$ the combined function: $h_{(u,v)} = \sum_{elim(u,v)} \prod_{f \in A} f$.
       Send $h_{(u,v)}$ and the individual functions $H_{(u,v)}$ to node $v$.
Endfor
• Compute $P(X_i, e)$:
       For every $X_i \in X$ let $u$ be a vertex in $T$ such that $X_i \in \chi(u)$.
       Compute $P(X_i|e) = \alpha \sum_{\chi(u)-\{X_i\}} (\prod_{f \in cluster(u)} f)$
```

Figure 4.12: Algorithm Iterative Join-Graph Propagation (IJGP)

When node i sends a message (or messages) to a neighbor node j it operates on all the CPTs in its cluster and on all the messages sent from its neighbors excluding the ones received from j. First, all individual functions that share no variables with the eliminator are collected and sent to j. All the rest of the functions are *combined* in a product and summed over the eliminator between i and j.

It is known that:

THEOREM **4.3.2** *1. [66] If $IJGP$ is applied to a join-tree decomposition it reduces to join-tree clustering and it therefore is guaranteed to compute the exact beliefs in one iteration.*

2. [64] The time complexity of one iteration of IJGP is $O(deg \cdot (n + N) \cdot d^{w^*+1})$ and its space complexity is $O(N \cdot d^{\theta})$, where deg is the maximum degree of a node in the join-graph, n is the number of variables, N is the number of nodes in the graph *decomposition, d is the maximum domain size, w^* is the maximum cluster size and θ is the maximum label size.*

However, when applied to a join-graph the algorithm is neither guaranteed to converge nor to find the exact posterior.

Proof. The number of cliques in the chordal graph G' corresponding to G is at most n, so the number of nodes in the join-tree is at most n. The complexity of processing a node u in the join-tree is $deg_u \cdot (|\psi(u)| + deg_u - 1) \cdot d^{|\chi(u)|}$, where deg_u is the degree of u. By bounding deg_u by deg, $|\psi(u)|$ by n and $\chi(u)$ by $w^* + 1$ and knowing that $deg < N$, by summing over all nodes, we can bound the entire time complexity by $O(deg \cdot (n + N) \cdot d^{w^*+1})$.

For each edge JTC records functions. Since the number of edges in bounded by n and the size of each message is bounded by d^{sep} we get space complexity of $O(n \cdot d^{sep})$. $\square$

4.3.3 I-Mappness of Arc-Labeled Join-Graphs

The success of IJGP, no doubt, will depend on the choice of cluster graphs it operates on. The following paragraphs provide some rationale to our choice of minimal arc-labeled join-graphs. First, we are committed to the use of an underlying graph structure that captures as many of the distribution independence relations as possible, without introducing new ones. That is, we restrict attention to cluster graphs that are I-maps of P [86]. Second, we wish to avoid cycles as much as possible in order to minimize computational over-counting.

Indeed, it can be shown that any join-graph of a belief network is an I-map of the underlying probability distribution relative to node-separation. It turns out that arc-labeled join-graphs display a richer set of independencies relative to arc-separation.

DEFINITION **4.3.5 (arc-separation in (arc-labeled) join-graphs)** *Let $D =< JG, \chi, \psi, \theta >$, $JG = (V, E)$ be an arc-labeled decomposition. Let $N_W, N_Y \subseteq V$ be two sets of nodes, and $E_Z \subseteq E$ be a set of edges in JG. Let W, Y, Z be their corresponding sets of variables ($W = \cup_{v \in N_W} \chi(v)$, $Z = \cup_{e \in E_Z} \theta(e)$). E_Z* arc-separates

N_W and N_Y in D if there is no path between N_W and N_Y in the graph JG with the edges in E_Z removed. In this case we also say that W is separated *from Y given Z in D, and write $< W|Z|Y >_D$. Arc-separation in a regular join-graph is defined relative to its separators.*

THEOREM **4.3.3** *Any arc-labeled join-graph decomposition $D =< JG, \chi, \psi, \theta >$ of a belief network $BN =< X, D, G, P >$ is an I-map of P relative to arc-separation.*

Proof. Let MG be the moral graph of BN. Since MG is an I-map of P, it is enough to prove that JG is and I-map of MG.

Let N_W, N_Z, N_Y be three disjoint set of nodes in JG, and W, Z, Y be their corresponding sets of variables in MG. We will prove:

$$< N_W|N_Z|N_Y >_{JG} \Longrightarrow < W|Z|Y >_{MG}$$

by contradiction.

Since the sets W, Z, Y may not be disjoint, we will actually prove that $< W - Z|Z|Y - Z >_G$ holds, this being equivalent to $< W|Z|Y >_G$.

Supposing $< W - Z|Z|Y - Z >_{MG}$ is false, then there exists a path $\alpha = \gamma_1, \gamma_2, \ldots, \gamma_{n-1}, \beta = \gamma_n$ in MG that goes from some variable $\alpha = \gamma_1 \in W - Z$ to some variable $\beta = \gamma_n \in Y - Z$ without intersecting variables in Z.

Let N_v be the set of all nodes in JG that contain variable $v \in X$, and let's consider the set of nodes:

$$S = \cup_{i=1}^{n} N_{\gamma_i} - N_Z$$

We argue that S forms a connected sub-graph in JG.

First, the running intersection property ensures that every $N_{\gamma_i}, i = 1, \ldots, n$, remains connected in JG after pulling out the nodes in N_Z (otherwise, it must be that there was a path between the two disconnected parts in the original JG, which implies that a γ_i is part of Z, which is a contradiction).

Second, the fact that $(\gamma_i, \gamma_{i+1}), i = 1, \ldots, n-1$, is an edge in the moral graph MG implies that there is a conditional probability table (CPT) on both γ_i and $\gamma_{i+1}, i = 1, \ldots, n-1$ (and perhaps other variables). From property 1 of the definition of the join-graph, it follows that for all $i = 1, \ldots, n-1$ there exists a node in JG that contains both γ_i and γ_{i+1}. This proves the existence of a path in the mutilated join-graph (JG with N_Z pulled out) from a node in N_W containing $\alpha = \gamma_1$ to the node containing both γ_1 and γ_2 (N_{γ_1} is connected), then from that node to the one containing both γ_2 and γ_3 (N_{γ_2} is connected), and so on until we reach a node in N_Y containing $\beta = \gamma_n$.

This shows that $< N_W|N_Z|N_Y >_{JG}$ is false, concluding the proof by contradiction. $\Box$

Interestingly however, removing arcs or labels from arc-labeled join-graphs whose clusters are fixed will not increase the independencies captured by arc-labeled join-graphs. That is:

Proposition 16 *Any two (arc-labeled) join-graphs defined on the same set of clusters, sharing (V, χ ψ), express exactly the same set of independencies relative to arc-separation.*

Consequently, all such decomposition are *correct* and are isomorphic I-maps.

THEOREM **4.3.4** *Any arc-labeled join-graph decomposition of a belief network $BN =< X, D, G, P >$ is a minimal I-map of P relative to arc-separation.*

Hence, the issue of minimizing computational over-counting due to cycles appears to be orthogonal to maximizing independencies via minimal I-mappness. Nevertheless, to avoid over-counting as much as possible, we still prefer join-graphs that minimize cycles relative to each variable. That is, we prefer to apply IJGP to *minimal* arc-labeled join-graphs.

4.3.4 Bounded Join-Graphs

Since we want to control the complexity of IJGP we will define it on decompositions having bounded cluster size. If the number of variables in a cluster is bounded by i, the time and

```
Algorithm Join-Graph Structuring(i)
    1. Apply procedure schematic mini-bucket(i).
    2. Associate each resulting mini-bucket with a node in the join-graph, the variables
       of the nodes are those appearing in the mini-bucket, the original functions are
       those in the mini-bucket.
    3. Keep the arcs created by the procedure (called out-edges) and label them by the
       regular separator.
    4. Connect the mini-bucket clusters belonging to the same bucket in a chain by
       in-edges labeled by the single variable of the bucket.
```

Figure 4.13: Algorithm Join-Graph Structuring(*i*)

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

Figure 4.14: Procedure Schematic Mini-Bucket(*i*)

space complexity of one full iteration of IJGP(i) is exponential in i. How can good graph-decompositions of bounded cluster size be generated?

Since we want the join-graph to be as close as possible to a tree, and since a tree has a treewidth 1, we may try to find a join-graph JG of bounded cluster size whose treewidth (as a graph) is minimized. While we will not attempt to optimally solve this task, we will propose one method for generating i-bounded graph-decompositions.

DEFINITION **4.3.6 (external and internal widths)** *Given an arc-labeled join-graph decomposition $D =< JG, \chi, \psi, \theta >$ of a network $< G, P >$, the internal width of D is $max_{v \in V} |\chi(v)|$, while the external width of D is the treewidth of JG as a graph.*

Clearly, if D is a tree-decomposition its external width is 1 and its internal width equals its treewidth. For example, an edge minimal dual decomposition has an internal width equal to the maximum scope of each function, m, and external width w^* which is the treewidth of the moral graph of G. On the other hand, a tree-decomposition has internal width of w^* and external width of 1.

Using this terminology we can now state our target decomposition more clearly. Given a graph G, and a bounding parameter i we wish to find a join-graph decomposition of G whose internal width is bounded by i and whose external width is minimized. The bound i controls the complexity of one iteration of $IJGP$ while the external width provides some measure of its accuracy.

One class of such decompositions is partition-based. It starts from a given tree-decomposition and then partitions the clusters until the decomposition has clusters bounded by i. The opposite approach is grouping-based. It starts from an arc-minimal dual-graph decomposition (where each cluster contains a single CPT) and groups clusters into larger clusters as long as the resulting clusters do not exceed the given bound. In both methods we should attempt to reduce the treewidth of the generated graph-decomposition. Our partition-based approach inspired by the mini-bucket idea [43] is as follows.

Given a bound i, algorithm *join-graph structuring(i)* applies procedure *schematic mini-bucket(i)*, described in Figure 4.14. The procedure only traces the scopes of the functions that would be generated by the full mini-bucket procedure, avoiding actual computation. The algorithm then connects the mini-buckets' scopes minimally to obtain the running intersection property, as described in Figure 4.13.

Example 4.3.5 *Figure 4.15a shows the trace of procedure schematic mini-bucket(3) applied to the problem described in Figure 4.1a. The decomposition in Figure 4.15b is created by the algorithm graph structuring. The only cluster partitioned is that of F into two scopes (FCD) and (BF), connected by an in-edge labeled with F.*

Procedure schematic mini-bucket ends with a collection of trees rooted in mini-buckets of the first variable. Each of these trees is minimally arc-labeled. Then, *in-edges* are labeled

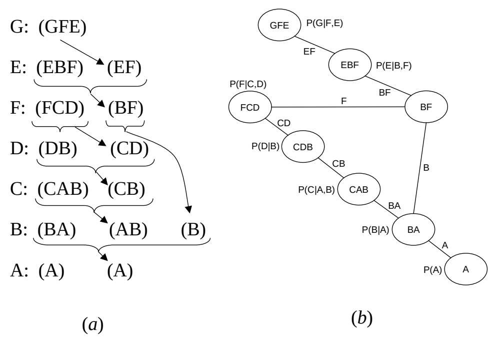


Figure 4.15: Join-graph decompositions

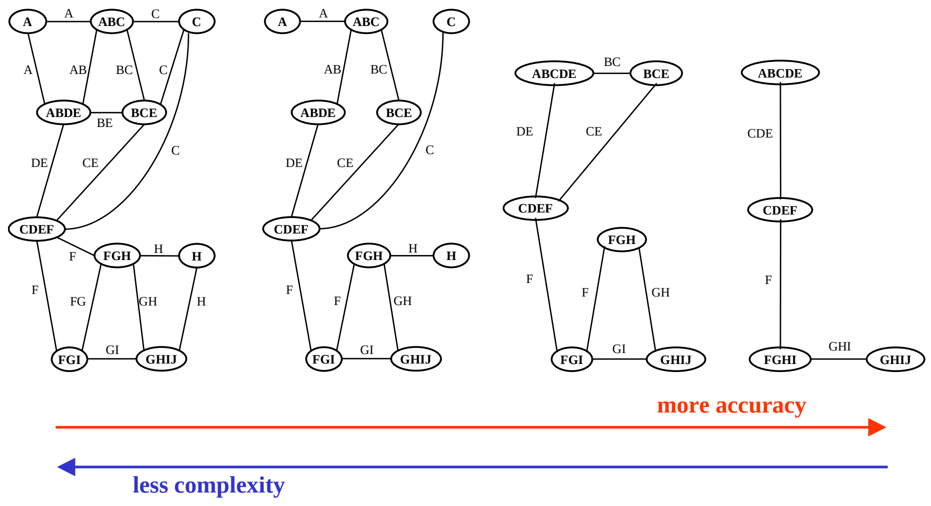


Figure 4.16: Join-graphs

with only one variable, and they are added only to obtain the running intersection property between branches of these trees. It can be shown that:

Proposition 17 *Algorithm join-graph structuring(i), generates a minimal arc-labeled join-graph decomposition having bound i.*

Example 4.3.6 *Figure 4.16 shows a range of arc-labeled join-graphs. On the left extreme we have a graph with smaller clusters, but more cycles. This is the type of graph IBP works on. On the right extreme we have a tree decomposition, which has no cycles but has bigger clusters. In between, there could be a number of join-graphs where maximum cluster size can be traded for number of cycles. Intuitively, the graphs on the left present*

Table 4.7: Random networks: N=50, K=2, C=45, P=3, 100 instances, w*=16

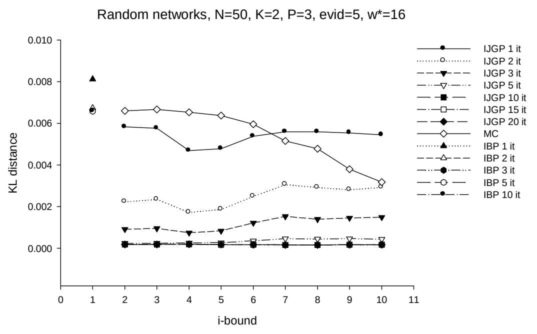
		Absolute error				Relative error				KL distance				Time			
		IBP	IJGP			IBP	IJGP			IBP	IJGP			IBP	IJGP		
#it	#evid		i=2	i=5	i=8		i=2	i=5	i=8		i=2	i=5	i=8		i=2	i=5	i=8
1	0	0.02988	0.03055	0.02623	0.02940	0.06388	0.15694	0.05677	0.07153	0.00213	0.00391	0.00208	0.00277	0.0017	0.0036	0.0058	0.0295
	5	0.06178	0.04434	0.04201	0.04554	0.15005	0.12340	0.12056	0.11154	0.00812	0.00582	0.00478	0.00558	0.0013	0.0040	0.0052	0.0200
	10	0.08762	0.05777	0.05409	0.05910	0.23777	0.18071	0.14278	0.15686	0.01547	0.00915	0.00768	0.00899	0.0013	0.0040	0.0036	0.0121
5	0	0.00829	0.00636	0.00592	0.00669	0.01726	0.01326	0.01239	0.01398	0.00021	0.00014	0.00015	0.00018	0.0066	0.0145	0.0226	0.1219
	5	0.05182	0.00886	0.00886	0.01123	0.12589	0.01967	0.01965	0.02494	0.00658	0.00024	0.00026	0.00044	0.0060	0.0120	0.0185	0.0840
	10	0.08039	0.01155	0.01073	0.01399	0.21781	0.03014	0.02553	0.03279	0.01382	0.00055	0.00042	0.00073	0.0048	0.0100	0.0138	0.0536
10	0	0.00828	0.00584	0.00514	0.00495	0.01725	0.01216	0.01069	0.01030	0.00021	0.00012	0.00010	0.00010	0.0130	0.0254	0.0436	0.2383
	5	0.05182	0.00774	0.00732	0.00708	0.12590	0.01727	0.01628	0.01575	0.00658	0.00018	0.00017	0.00016	0.0121	0.0223	0.0355	0.1639
	10	0.08040	0.00892	0.00808	0.00855	0.21782	0.02101	0.01907	0.02005	0.01382	0.00028	0.00024	0.00029	0.0109	0.0191	0.0271	0.1062
MC	0		0.04044	0.04287	0.03748		0.08811	0.09342	0.08117		0.00403	0.00435	0.00369		0.0159	0.0173	0.0552
	5		0.05303	0.05171	0.04250		0.12375	0.11775	0.09596		0.00659	0.00636	0.00477		0.0146	0.0158	0.0532
	10		0.06033	0.05489	0.04266		0.14702	0.13219	0.10074		0.00841	0.00729	0.00503		0.0119	0.0143	0.0470

less complexity for IJGP because the cluster size is small, but they are also likely to be less accurate. The graphs on the right side are computationally more complex, because of larger cluster size, but are likely to be more accurate.

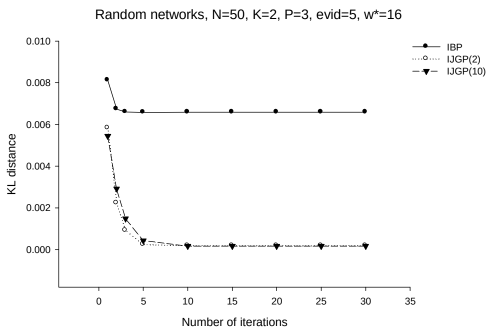
MC(i) vs. IJGP(i). As can be hinted by our structuring of a bounded join-graph, there is a close relationship between MC(i) and IJGP(i). In particular, one iteration of IJGP(i) is similar to MC(i) (MC(i) is an algorithm that approximates join-tree clustering and was shown to be competitive with IBP and Gibbs Sampling [79]). Indeed, while we view IJGP(i) as an iterative version of MC(i), the two algorithms differ in several technical points, some may be superficial, due to implementation, others may be more principled. We will leave the discussion at that and will observe the comparison of the two approaches in the empirical section.

4.3.5 Experimental Evaluation

We tested the performance of IJGP(i) on random networks, on M-by-M grids, on two benchmark CPCS files with 54 and 360 variables, respectively (these are belief networks for medicine, derived from the Computer based Patient Case Simulation system, known to be hard for belief updating) and on coding networks. On each type of networks, we ran Iterative Belief Propagation (IBP), MC(i) and IJGP(i), while giving IBP and IJGP(i) the same number of iterations.



a) Performance vs. i-bound



b) Convergence with iterations

Figure 4.17: Random networks: KL distance

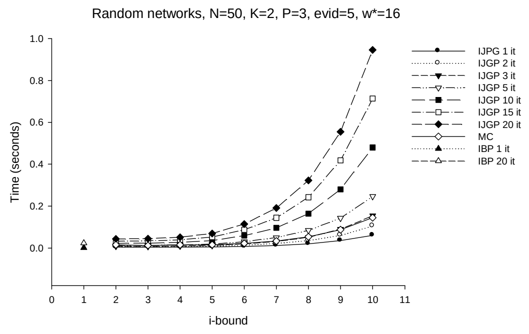


Figure 4.18: Random networks: Time

We use the partitioning method described in Section 4.3.4 to construct a join-graph. To determine the order of message computation, we recursively pick an edge (u,v), such that node u has the fewest incoming messages missing.

For each network except coding, we compute the exact solution and compare the accuracy of algorithms using: 1. Absolute error - the absolute value of the difference between the approximate and the exact, averaged over all values, all variables and all problems. 2. Relative error - the absolute value of the difference between the approximate and the exact, divided by the exact, averaged over all values, all variables and all problems. 3. KL (Kullback-Leibler) distance - $P_{exact}(X = a) \cdot log(P_{exact}(X = a)/P_{approximation}(X = a))$ averaged over all values, all variables and all problems. We also report the time taken by each algorithm. For coding networks we report Bit Error Rate (BER) computed as follows: for each approximate algorithm we pick the most likely value for each variable, take the number of disagreements with the exact input, divide by the total number of variables, and average over all the instances of the problem. We also report time.

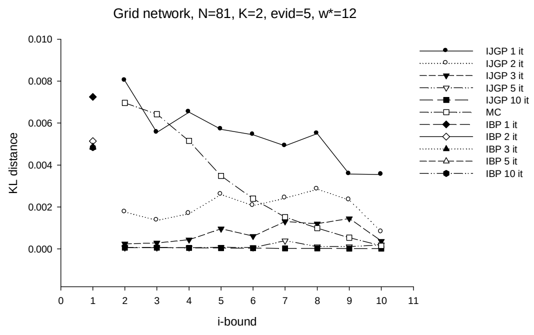
The random networks were generated using parameters (N,K,C,P), where N is the number of variables, K is their domain size, C is the number of conditional probability tables (CPTs) and P is the number of parents in each CPT. Parents in each CPT are picked randomly and each CPT is filled randomly. In grid networks, N is a square number and each CPT is filled randomly. In each problem class, we also tested different numbers of evidence variables. The coding networks are from the class of linear block codes, where σ is the channel noise level. Note that we are limited to relatively small and sparse problem instances since our evaluation measured are based on comparing against exact figures.

Random network results with networks of N=50, K=2, C=45 and P=3 are given in Table 4.7 and Figures 4.17 and 4.18. For IJGP(i) and MC(i) we report 3 different values of i-bound: 2, 5, 8; for IBP and IJGP(i) we report 3 different values of number of iterations: 1, 5, 10; for all algorithms we report 3 different values of number of evidence: 0, 5, 10. We notice that IJGP(i) is always better than IBP (except when i=2 and number of iterations is 1), sometimes as much as an order of magnitude, in terms of absolute and relative error and KL distance. IBP rarely changes after 5 iterations, whereas IJGP(i) solution can be improved up to 15-20 iterations. As we predicted, IJGP(i) is about equal to MC(i) in terms of accuracy for one iteration. But IJGP(i) improves as the number of iterations increases, and is eventually better than MC(i) by as much as an order of magnitude, although it clearly takes more time when the i-bound is large.

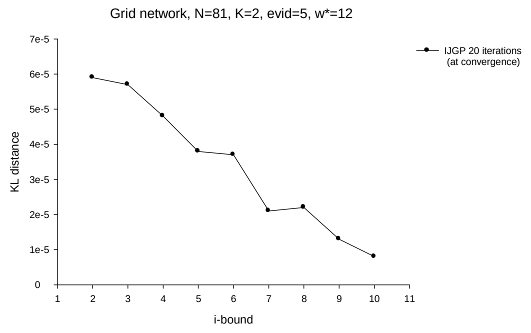
Figure 4.17a shows a comparison of all algorithms with different numbers of iterations, using the KL distance. Because the network structure changes with different i-bounds, we do not see monotonic improvement of IJGP with i-bound for a given number of iterations (as is the case with MC). Figure 4.17b shows how IJGP converges with iteration to smaller KL distance than IBP. As expected, the time taken by IJGP (and MC) varies exponentially with the i-bound (see Figure 4.18).

Table 4.8: 9x9 grid, K=2, 100 instances, w*=12

		Absolute error				Relative error				KL distance				Time			
		IBP	IJGP			IBP	IJGP			IBP	IJGP			IBP	IJGP		
#it	#evid		i=2	i=5	i=8		i=2	i=5	i=8		i=2	i=5	i=8		i=2	i=5	i=8
1	0	0.03524	0.05550	0.04292	0.03318	0.08075	0.13533	0.10252	0.07904	0.00289	0.00859	0.00602	0.00454	0.0010	0.0053	0.0106	0.0426
	5	0.05375	0.05284	0.04012	0.03661	0.16380	0.13225	0.09889	0.09116	0.00725	0.00802	0.00570	0.00549	0.0016	0.0041	0.0092	0.0315
	10	0.07094	0.05453	0.04304	0.03966	0.23624	0.14588	0.12492	0.12202	0.01232	0.00905	0.00681	0.00653	0.0013	0.0038	0.0072	0.0256
5	0	0.00358	0.00393	0.00325	0.00284	0.00775	0.00849	0.00702	0.00634	0.00005	0.00006	0.00007	0.00010	0.0049	0.0152	0.0347	0.1462
	5	0.03224	0.00379	0.00319	0.00296	0.11299	0.00844	0.00710	0.00669	0.00483	0.00006	0.00007	0.00010	0.0053	0.0131	0.0309	0.1127
	10	0.05503	0.00364	0.00316	0.00314	0.19403	0.00841	0.00756	0.01313	0.00994	0.00006	0.00009	0.00019	0.0036	0.0127	0.0271	0.0913
10	0	0.00352	0.00352	0.00232	0.00136	0.00760	0.00760	0.00502	0.00293	0.00005	0.00005	0.00003	0.00001	0.0090	0.0277	0.0671	0.2776
	5	0.03222	0.00357	0.00248	0.00149	0.11295	0.00796	0.00549	0.00330	0.00483	0.00005	0.00003	0.00002	0.0096	0.0246	0.0558	0.2149
	10	0.05503	0.00347	0.00239	0.00141	0.19401	0.00804	0.00556	0.00328	0.00994	0.00005	0.00003	0.00001	0.0090	0.0223	0.0495	0.1716
MC	0		0.05827	0.04036	0.01579		0.13204	0.08833	0.03440		0.00650	0.00387	0.00105		0.0106	0.0142	0.0382
	5		0.05973	0.03692	0.01355		0.13831	0.08213	0.03001		0.00696	0.00348	0.00099		0.0102	0.0130	0.0342
	10		0.05866	0.03416	0.01075		0.14120	0.07791	0.02488		0.00694	0.00326	0.00075		0.0099	0.0116	0.0321



a) Performance vs. i-bound



b) Fine granularity for KL

Figure 4.19: Grid 9x9: KL distance

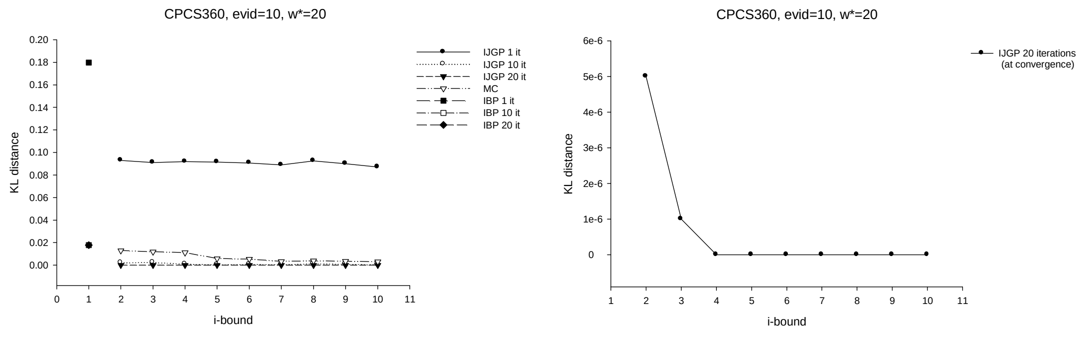
Grid network results with networks of N=81, K=2, 100 instances are very similar to those of random networks. They are reported in Table 4.8 and in Figure 4.19, where we can see the impact of having evidence (0 and 5 evidence variables) on the algorithms. IJGP at convergence gives the best performance in both cases, while IBP's performance deteriorates with more evidence and is surpassed by MC with i-bound 5 or larger.

CPCS network results with CPCS54 and CPCS360 are given in Table 4.9 and Figure 4.20, and are even more pronounced than those of random and grid networks. When evidence is added, IJGP(i) is more accurate than MC(i), which is more accurate than IBP, as can be seen in Figure 4.20a.

Coding network results are given in Table 4.10. We tested on large networks of 400 variables, with treewidth w*=43, with IJGP and IBP set to run 30 iterations (this is more

Table 4.9: CPCS54 50 instances, w*=15; CPCS360 10 instances, w*=20

		Absolute error				Relative error				KL distance				Time			
		IBP	IJGP			IBP	IJGP			IBP	IJGP			IBP	IJGP		
#it	#evid		i=2	i=5	i=8		i=2	i=5	i=8		i=2	i=5	i=8		i=2	i=5	i=8
CPCS54																	
1	0	0.01324	0.03747	0.03183	0.02233	0.02716	0.08966	0.07761	0.05616	0.00041	0.00583	0.00512	0.00378	0.0097	0.0137	0.0146	0.0275
	5	0.02684	0.03739	0.03124	0.02337	0.05736	0.09007	0.07676	0.05856	0.00199	0.00573	0.00493	0.00366	0.0072	0.0094	0.0087	0.0169
	10	0.03915	0.03843	0.03426	0.02747	0.08475	0.09156	0.08246	0.06687	0.00357	0.00567	0.00506	0.00390	0.005	0.0047	0.0052	0.0115
5	0	0.00031	0.00016	0.00123	0.00110	0.00064	0.00033	0.00255	0.00225	7.75e-7	0.00000	0.00002	0.00001	0.0371	0.0334	0.0384	0.0912
	5	0.01874	0.00058	0.00092	0.00098	0.04067	0.00124	0.00194	0.00203	0.00161	0.00000	0.00001	0.00001	0.0337	0.0215	0.0260	0.0631
	10	0.03348	0.00101	0.00139	0.00144	0.07302	0.00215	0.00298	0.00302	0.00321	0.00001	0.00003	0.00002	0.0290	0.0144	0.0178	0.0378
10	0	0.00031	0.00009	0.00014	0.00015	0.00064	0.00018	0.00029	0.00031	7.75e-7	0.0000	0.00000	0.00000	0.0736	0.0587	0.0667	0.1720
	5	0.01874	0.00037	0.00034	0.00038	0.04067	0.00078	0.00071	0.00080	0.00161	0.00000	0.00000	0.00000	0.0633	0.0389	0.0471	0.1178
	10	0.03348	0.00058	0.00051	0.00057	0.07302	0.00123	0.00109	0.00122	0.00321	4.0e-6	3.0e-6	4.0e-6	0.0575	0.0251	0.0297	0.0723
MC	0		0.02721	0.02487	0.01486		0.05648	0.05128	0.03047		0.00218	0.00171	0.00076		0.0144	0.0125	0.0333
	5		0.02702	0.02522	0.01760		0.05687	0.05314	0.03713		0.00201	0.00186	0.00098		0.0103	0.0126	0.0346
	10		0.02825	0.02504	0.01600		0.06002	0.05318	0.03409		0.00216	0.00177	0.00091		0.0094	0.0090	0.0295
CPCS360																	
1	10	0.26421	0.14222	0.13907	0.14334	7.78167	2119.20	2132.78	2133.84	0.17974	0.09297	0.09151	0.09255	0.7172	0.5486	0.5282	0.4593
	20	0.26326	0.12867	0.12937	0.13665	370.444	28720.38	30704.93	31689.59	0.17845	0.08212	0.08269	0.08568	0.6794	0.5547	0.5250	0.4578
10	10	0.01772	0.00694	0.00121	0.00258	1.06933	6.07399	0.01005	0.04330	0.017718	0.00203	0.00019	0.00116	7.2205	4.7781	4.5191	3.7906
	20	0.02413	0.00466	0.00115	0.00138	62.99310	26.04308	0.00886	0.01353	0.02027	0.00118	0.00015	0.00036	7.0830	4.8705	4.6468	3.8392
20	10	0.01772	0.00003	3.0e-6	3.0e-6	1.06933	0.00044	8.0e-6	7.0e-6	0.01771	5.0e-6	0.0	0.0	14.4379	9.5783	9.0770	7.6017
	20	0.02413	0.00001	9.0e-6	9.0e-6	62.9931	0.00014	0.00013	0.00004	0.02027	0.0	0.0	0.0	13.6064	9.4582	9.0423	7.4453
MC	10		0.03389	0.01984	0.01402		0.65600	0.20023	0.11990		0.01299	0.00590	0.00390		2.8077	2.7112	2.5188
	20		0.02715	0.01543	0.00957		0.81401	0.17345	0.09113		0.01007	0.00444	0.00234		2.8532	2.7032	2.5297



a) Performance vs. i-bound

b) Fine granularity for KL

Figure 4.20: CPCS360: KL distance

than enough to ensure convergence). IBP is known to be very accurate for this class of problems and it is indeed better than MC. It is remarkable however that IJGP converges to smaller BER than IBP even for small values of the i-bound. Both the coding network and CPCS360 show the scalability of IJGP for large size problems. Notice that here the anytime behavior of IJGP is not clear.

4.3.6 Discussion

In this section we presented an iterative anytime approximation algorithm called Iterative Join-Graph Propagation (IJGP(i)), that applies the message passing algorithm of join-tree

Table 4.10: Coding networks: N=400, P=4, 500 instances, 30 iterations, w*=43

Bit Error Rate

		i-bound					
σ		2	4	6	8	10	IBP
0.22	IJGP	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005
	MC	0.00501	0.00800	0.00586	0.00462	0.00392	
0.28	IJGP	0.00062	0.00062	0.00062	0.00062	0.00062	0.00064
	MC	0.02170	0.02968	0.02492	0.02048	0.01840	
0.32	IJGP	0.00238	0.00238	0.00238	0.00238	0.00238	0.00242
	MC	0.04018	0.05004	0.04480	0.03878	0.03558	
0.40	IJGP	0.01202	0.01188	0.01194	0.01210	0.01192	0.01220
	MC	0.08726	0.09762	0.09272	0.08766	0.08334	
0.51	IJGP	0.07664	0.07498	0.07524	0.07578	0.07554	0.07816
	MC	0.15396	0.16048	0.15710	0.15452	0.15180	
0.65	IJGP	0.19070	0.19056	0.19016	0.19030	0.19056	0.19142
	MC	0.21890	0.22056	0.21928	0.21904	0.21830	

Time

	IJGP	0.36262	0.41695	0.86213	2.62307	9.23610	0.019752
	MC	0.25281	0.21816	0.31094	0.74851	2.33257	

clustering to join-graphs rather than join-trees, iteratively. The algorithm borrows the iterative feature from Iterative Belief Propagation (IBP) on one hand and is inspired by the anytime virtues of mini-clustering MC(i) on the other. We show that the success of IJGP is facilitated by extending the notion of join-graphs to minimal arc-labeled join-graphs, and provide a structuring algorithm that generates minimal arc-labeled join-graphs of bounded size.

The empirical results are extremely encouraging. We experimented with randomly generated networks, grid-like networks, medical diagnosis CPCS networks and coding networks. We showed that IJGP is almost always superior to both IBP and MC(i) and is sometimes more accurate by an order of several magnitudes. One should note that IBP cannot be improved with more time, while MC(i) requires a large i-bound for many hard and large networks to achieve reasonable accuracy. There is no question that the iterative application of IJGP is instrumental to its success. In fact, IJGP(2) in isolation appears to be the most cost effective variant.

One question which we did not answer in this section is why propagating the messages iteratively helps. Why is IJGP upon convergence, superior to IJGP with one iteration and is superior to MC(i)? One clue can be provided when considering deterministic constraint networks which can be viewed as ”extreme probabilistic networks”. It is known that constraint propagation algorithms, which are analogous to the messages sent by belief propagation, are guaranteed to converge and are guaranteed to improve with convergence. The propagation scheme presented here works like constraint propagation relative to the flat network abstraction of P, (where all non-zero entries are normalized to a positive constant), and is guaranteed to be more accurate for that abstraction at least. It is precisely these issues that we address in Section 4.4.

4.4 The Inference Power of Iterative Belief Propagation

A good fraction of our current research is devoted to studying the properties of *Iterative Belief Propagation (IBP)*, and of the generalized belief propagation version *Iterative Join-Graph Propagation (IJGP)*. We are particularly interested in making connections to well known algorithms from constraint networks, like Arc-consistency, which may help explain when and why IBP has strong or weak inference power.

The belief propagation algorithm is a distributed algorithm that computes posterior beliefs for tree-structured Bayesian networks (poly-trees) [86]. However, in recent years it was shown to work surprisingly well in many applications involving networks with loops, including turbo codes, when applied iteratively [89]. Another recent result [69] shows impressive performance for an iterative message passing scheme used for very large satisfiability problems. While there is still very little understanding as to why and when IBP works well, some recent investigation shows that when IBP converges, it converges to a stationary point of the Bethe energy, thus making connections to approximation algorithms developed in statistical physics and to variational approaches to approximate inference [100, 102]. However, these approaches do not explain why IBP is successful where it is, and do not allow any performance guarantees on accuracy.

The work we present here is based on some some simple observations that may shed light on IBP's behavior, and on the more general class of IJGP algorithms. Zero-beliefs are variable-value pairs that have zero conditional probability given the evidence. We show that: if a value of a variable is assessed as having zero-belief in any iteration of IBP, it remains a zero-belief in all subsequent iterations; that IBP finitely converges relative to its set of zero-beliefs; and, most importantly that the set of zero-beliefs decided by any of the iterative belief propagation methods is sound. Namely any zero-belief determined by IBP corresponds to a true zero conditional probability relative to the given probability distribution expressed by the Bayesian network.

While each of these claims can be proved directly, our approach is to associate a belief network with a constraint network and show a correspondence between IBP applied to the belief network and an arc-consistency algorithm applied to the corresponding constraint network. Since arc-consistency algorithms are well understood this correspondence not only proves right away the targeted claims, but may provide additional insight into the behavior of IBP and IJGP. In particular, not only it immediately justifies the iterative application of belief propagation algorithms on one hand, but it also illuminates its ”distance” from being complete, on the other.

4.4.1 Arc-Consistency Algorithms

Constraint propagation algorithms is a class of polynomial time algorithms that are at the center of constraint processing techniques. They were investigated extensively in the past three decades and the most well known versions are *arc-*, *path-*, and *i-consistency* [27].

DEFINITION **4.4.1 (arc-consistency)** *[68] Given a binary constraint network* (X, D, C)*, the network is arc-consistent iff for every binary constraint* $R_{ij} \in C$*, every value* $v \in D_i$ *has a value* $u \in D_j$ *s.t.* $(v, u) \in R_{ij}$*.*

When a binary constraint network is not arc-consistent, arc-consistency algorithms can enforce arc-consistency. The algorithms remove values from the domains of the variables that violate arc-consistency until an arc-consistent network is generated. A variety of improved performance arc-consistency algorithms were developed over the years, however we will consider here a non-optimal distributed version, which we call *distributed arc-consistency*.

DEFINITION **4.4.2 (distributed arc consistency, DAC)** *Distributed arc consistency (DAC) is a message passing algorithm. Each node maintains a current set of viable values D_i. Let $ne(i)$ be the set of neighbors of X_i in the constraint graph. Every node X_i sends a message to any node $X_j \in ne(i)$, which consists of the values in X_j's domain that are consistent with the current D_i, relative to the constraint that they share. Namely, the message that X_i sends to X_j, denoted by D_i^j, is:*

$$D_i^j \leftarrow \pi_j(R_{ji} \bowtie D_i) \tag{4.1}$$

(where, join ($\bowtie$) and project (π) are the usual relational operators) and in addition node i computes:

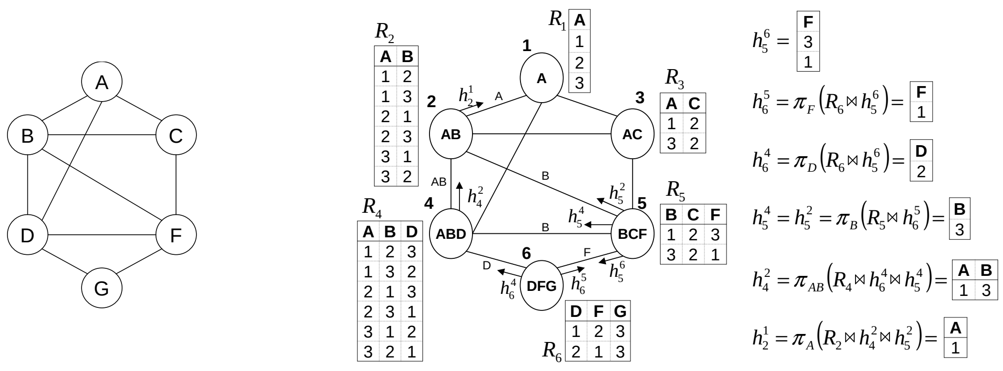
$$D_i \leftarrow D_i \cap (\bowtie_{k \in ne(i)} D_k^i) \tag{4.2}$$

Clearly the algorithm can be synchronized into iterations, where in each iteration every node computes its current domain based on all the messages received so far from its neighbors (eq. 4.2), and sends a new message to each neighbor (eq. 4.1). Alternatively, equations 4.1 and 4.2 can be combined. The message X_i sends to X_j is:

$$D_i^j \leftarrow \pi_j(R_{ji} \bowtie D_i \bowtie_{k \in ne(i)} D_k^i) \tag{4.3}$$

Let us mention again the definition of the dual graphs, which we will be using in this section:

DEFINITION **4.4.3 (dual graphs)** *Given a set of functions $F = \{f_1, ..., f_l\}$ over scopes $S_1, ..., S_l$, the dual graph of F is a graph $DG = (V, E, L)$ that associates a node with each function, namely $V = F$ and an arc connects any two nodes whose scope share a variable,*



R_1

A
1
2
3

R_2

A	B
1	2
1	3
2	1
2	3
3	1
3	2

R_3

A	C
1	2
3	2

R_4

A	B	D
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

R_5

B	C	F
1	2	3
3	2	1

R_6

D	F	G
1	2	3
2	1	3

$$h_5^6 = \begin{array}{|c|}\hline \mathbf{F} \\ \hline 3 \\ 1 \\ \hline\end{array}$$

$$h_6^5 = \pi_F(R_6 \bowtie h_5^6) = \begin{array}{|c|}\hline \mathbf{F} \\ \hline 1 \\ \hline\end{array}$$

$$h_6^4 = \pi_D(R_6 \bowtie h_5^6) = \begin{array}{|c|}\hline \mathbf{D} \\ \hline 2 \\ \hline\end{array}$$

$$h_5^4 = h_5^2 = \pi_B(R_5 \bowtie h_6^5) = \begin{array}{|c|}\hline \mathbf{B} \\ \hline 3 \\ \hline\end{array}$$

$$h_4^2 = \pi_{AB}(R_4 \bowtie h_6^4 \bowtie h_5^4) = \begin{array}{|c|c|}\hline \mathbf{A} & \mathbf{B} \\ \hline 1 & 3 \\ \hline\end{array}$$

$$h_2^1 = \pi_A(R_2 \bowtie h_4^2 \bowtie h_5^2) = \begin{array}{|c|}\hline \mathbf{A} \\ \hline 1 \\ \hline\end{array}$$

Figure 4.21: Part of the execution of RDAC algorithm

$E = \{(f_i, f_j) | S_i \cap S_j \neq \phi\}$. L is a set of labels for the arcs, each arc being labeled by the shared variables of its nodes, $L = \{l_{ij} = S_i \cap S_j | (i,j) \in E\}$.

The above distributed arc-consistency algorithm can be applied to the dual problem of any non-binary constraint network as well. This is accomplished by the following rule applied by each node in the dual graph. We call the algorithm relational distributed arc-consistency (RDAC).

DEFINITION **4.4.4 (relational distributed arc-consistency, RDAC)** *Let R_i and R_j be two constraints sharing scopes, whose arc in the dual graph is labeled by l_{ij}. The message R_i sends to R_j denoted h_i^j is defined by:*

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i)) \tag{4.4}$$

and each node updates its current relation according to:

$$R_i \leftarrow R_i \cap (\bowtie_{k \in ne(i)} h_k^i) \tag{4.5}$$

Example 4.4.1 *Figure 4.21 describes part of the execution of RDAC for a graph coloring problem, having the constraint graph shown on the left. All variables have the same domain, $\{$1,2,3$\}$, except for C which is 2, and G which is 3. The arcs correspond to* not equal *constraints, and the relations are R_A, R_{AB}, R_{AC}, R_{ABD}, R_{BCF}, R_{DFG}. The dual graph of this problem is given on the right side of the figure, and each table shows the initial constraints (there are unary, binary and ternary constraints). To initialize the algorithm, the first messages sent out by each node are universal relations over the labels. For this example, RDAC actually solves the problem and finds the unique solution A=1, B=3, C=2, D=2, F=1, G=3.*

Proposition 18 *Relational distributed arc-consistency converges after $O(t \cdot r)$ iterations to the largest arc-consistent network that is equivalent to the original network, where t bounds the number of tuples in each constraint and r is the number of constraints.*

Proposition 19 (complexity) *The complexity of distributed arc-consistency is $O(r^2 t^2 \log t)$.*

Proof. One iteration can be accomplished in $O(r \cdot t \cdot \log t)$, and there can be at most $r \cdot t$ iterations. $\square$

4.4.2 Iterative Belief Propagation over Dual Join-Graphs

Iterative belief propagation (IBP) is an iterative application of Pearl's algorithm that was defined for poly-trees [86]. Since it is a distributed algorithm, it is well defined for any network. In this section we will present IBP as an instance of join-graph propagation over variants of the *dual graph*.

Consider a Bayesian network $\mathcal{B} =< X, D, G, P >$. As defined earlier, the *dual graph* $\mathcal{D}_{\mathcal{G}}$ of the Belief network $\mathcal{B}$, is an arc-labeled graph defined over the CPTs as its functions. Namely, it has a node for each CPT and a labeled arc connecting any two nodes that share a variable in the CPT's scope. The arcs are labeled by the shared variables. A *dual join-graph* is a labeled arc subgraph of $\mathcal{D}_{\mathcal{G}}$ whose arc labels are subsets of the labels of $\mathcal{D}_{\mathcal{G}}$ such that the *running intersection property*, also called *connectedness property*, is satisfied. The running intersection property requires that any two nodes that share a variable in the

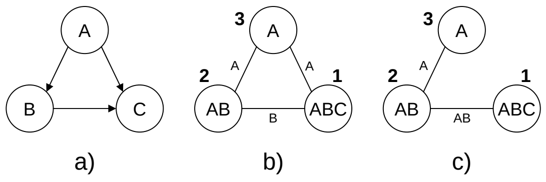


Figure 4.22: a) A belief network; b) A dual join-graph with singleton labels; c) A dual join-graph which is a join-tree

dual join-graph be connected by a path of arcs whose labels contain the shared variable. Clearly the dual graph itself is a dual join-graph. An *arc-minimal* dual join-graph is a dual join-graph for which none of the labels can be further reduced while maintaining the connectedness property.

Interestingly, there are many dual join-graphs of the same dual graph and many of them are arc-minimal. We define Iterative Belief Propagation on a dual join-graph. Each node sends a message over an arc whose scope is identical to the label on that arc. Since Pearl's algorithm sends messages whose scopes are singleton variables only, we highlight arc-minimal singleton dual join-graph. One such graph can be constructed directly from the graph of the Bayesian network, labeling each arc with the parent variable. It can be shown that:

Proposition 20 *The dual graph of any Bayesian network has an arc-minimal dual join-graph where each arc is labeled by a single variable.*

Example 4.4.2 *Consider the belief network on 3 variables A, B, C with CPTs 1.$P(C|A, B)$, 2.$P(B|A)$ and 3.$P(A)$, given in Figure 4.22a. Figure 4.22b shows a dual graph with singleton labels on the arcs. Figure 4.22c shows a dual graph which is a join tree, on which belief propagation can solve the problem exactly in one iteration (two passes up and down the tree).*

For complete reference, we will next present IBP algorithm that is applicable to any dual join-graph (Figure 4.23). The algorithm is a special case of IJGP introduced in [39].

```
Algorithm **IBP**
**Input:** An arc-labeled dual join-graph $DJ = (V, E, L)$ for a Bayesian network $BN =< X, D, G, P >$. Evidence $e$.
**Output:** An augmented graph whose nodes include the original CPTs and the messages received from neighbors. Approximations of $P(X_i|e)$, $\forall X_i \in X$. Approximations of $P(F_i|e)$, $\forall F_i \in \mathcal{B}$.
Denote by: $h_u^v$ the message from $u$ to $v$; $ne(u)$ the neighbors of $u$ in $V$; $ne_v(u) = ne(u) - \{v\}$; $l_{uv}$ the label of $(u,v) \in E$; $elim(u,v) = scope(u) - scope(v)$.
• **One iteration of IBP**
  **For** every node $u$ in $DJ$ in a topological order and back, do:
    1. **Process observed variables**
       Assign evidence variables to the each $p_i$ and remove them from the labeled arcs.
    2. **Compute and send to $v$ the function:**
       $$h_u^v = \sum_{elim(u,v)} (p_u \cdot \prod_{\{h_i^u, i \in ne_v(u)\}} h_i^u)$$
  **Endfor**
• **Compute approximations of $P(F_i|e)$, $P(X_i|e)$:**
  For every $X_i \in X$ let $u$ be the vertex of family $F_i$ in $DJ$,
  $P(F_i|e) = \alpha(\prod_{h_i^u, u \in ne(i)} h_i^u) \cdot p_u$;
  $P(X_i|e) = \alpha \sum_{scope(u)-\{X_i\}} P(F_i|e)$.
```

Figure 4.23: Algorithm Iterative Belief Propagation

It is easy to see that one iteration of IBP is time and space linear in the size of the belief network, and when IBP is applied to the singleton labeled dual graph it coincides with Pearl's belief propagation applied directly to the acyclic graph representation. For space reasons, we do not include the proof here. Also, when the dual join-graph is a tree IBP converges after one iteration (two passes, up and down the tree) to the exact beliefs.

4.4.3 The Flat Bayesian Network

Given a belief network $\mathcal{B}$ we will now define a flattening of the Bayesian network into a constraint network called $flat(\mathcal{B})$ where all the zero entries in the CPTs are removed from the corresponding relation. $flat(B)$ is a constraint network defined over the same set of variables and has the same set of domain values as $\mathcal{B}$. Formally, for every X_i and its CPT $P(X_i|pa_i) \in \mathcal{B}$ we define a constraint R_{F_i} over the family of X_i, $F_i = \{X_i\} \cup pa_i$ as follows: for every assignment $x = (x_i, x_{pa_i})$ to F_i,

$(x_i, x_{pa_i}) \in R_{F_i}$ iff $P(x_i|x_{pa_i}) > 0$.

The evidence set $e = \{e_1, ..., e_r\}$ is mapped into unary constraints that assign the corresponding values to the evidence variables.

THEOREM **4.4.3** *Given a belief network $\mathcal{B}$ and evidence e, for any tuple t: $P_{\mathcal{B}}(t|e) > 0 \Leftrightarrow t \in sol(flat(B,e))$.*

Proof. $P_{\mathcal{B}}(t|e) > 0 \Leftrightarrow \Pi_i P(x_i|x_{pa_i})|_t > 0 \Leftrightarrow \forall i,\ P(x_i|x_{pa_i})|_t > 0 \Leftrightarrow \forall i,\ (x_i, x_{pa_i})|_t \in R_{F_i}$ $\Leftrightarrow t \in sol(flat(B,e))$, where $|_t$ is the restriction to t. □

We next define an algorithm dependent notion of zero tuples.

DEFINITION **4.4.5 (IBP-zero)** *Given a CPT $P(X_i|pa_i)$, an assignment $x = (x_i, x_{pa_i})$ to its family F_i is* IBP-zero *if some iteration of IBP determines that $P(x_i|x_{pa_i}, e) = 0$.*

It is easy to see that when IBP is applied to a constraint network where sum and product are replaced by join and project, respectively, it becomes identical to distributed relational arc-consistency defined earlier. Therefore, a partial tuple is removed from a flat constraint by arc-consistency iff it is IBP-zero relative to the Bayesian network.

THEOREM **4.4.4** *When IBP is applied in a particular variable ordering to a dual join-graph of a Bayesian network $\mathcal{B}$, its trace is identical, relative to zero-tuples generation, to that of RDAC applied to the corresponding flat dual join-graph. Namely, taking a snapshot at identical steps, any IBP-zero tuple in the Bayesian network is a removed tuple in the corresponding step of RDAC over the flat dual join-graph.*

Proof. It suffices to prove that the first iteration of IBP and RDAC generates the same zero tuples and removed tuples, respectively. We prove the claim by induction over the topological ordering that defines the order in which messages are sent in the corresponding dual graphs.

Base case: By the definition of the flat network, when algorithms IBP and RDAC start, every zero probability tuple in one of the CPTs P_{X_i} in the dual graph of the Bayesian network, becomes a removed tuple in the corresponding constraint R_{F_i} in the dual graph of the flat network.

Inductive step: Suppose the claim is true after n correspondent messages are sent in IBP and RDAC. Suppose the $(n+1)th$ message is scheduled to be the one from node u to node v. Indexing messages by the name of the algorithm, in the dual graph of IBP, node u contains p_u and ${h_{IBP}}_i^u, i \in ne_v(u)$, and in the dual graph of RDAC, node u contains R_u and ${h_{RDAC}}_i^u, i \in ne_v(u)$. By the inductive hypothesis, the zero tuples in p_u and ${h_{IBP}}_i^u, i \in ne_v(u)$ are the removed tuples in R_u and ${h_{RDAC}}_i^u, i \in ne_v(u)$, respectively. Therefore, the zero tuples in the product $(p_u \cdot (\prod_{i \in ne_v(u)}) h_i^u)$ correspond to the removed tuples in the join $(R_u \bowtie (\bowtie_{i \in ne_v(u)}) h_i^u)$. This proves that the zero tuples in the message of IBP
${h_{IBP}}_u^v = \sum_{elim(u,v)} (p_u \cdot (\prod_{i \in ne_v(u)}) h_i^u)$, correspond to the removed tuples in the message of RDAC
${h_{RDAC}}_u^v = \pi_{l_{uv}} (R_u \bowtie (\bowtie_{i \in ne_v(u)}) h_i^u)$.
The same argument can now be extended for every iteration of the algorithms. □

Corollary 2 *Algorithm IBP zero-converges. Namely, its set of zero tuples does not change after $t \cdot r$ iterations.*

Proof. From Theorem 4.4.4 any IBP-zero is a no-good removed by arc-consistency over the flat network. Since arc-consistency converges, the claim follows. □

THEOREM **4.4.5** *When IBP is applied to a dual join-graph of a Bayesian network, any tuple t that is IBP-zero satisfies $P_{\mathcal{B}}(t|e) = 0$.*

Proof. From Theorem 4.4.4 if a tuple t is IBP zero, it is also removed from the corresponding relation by arc-consistency over $flat(\mathcal{B}, e)$. Therefore this tuple is a no-good of the network $flat(\mathcal{B}, e)$ and, from Theorem 4.4.3 it follows that $P_{\mathcal{B}}(t|e) = 0$. □

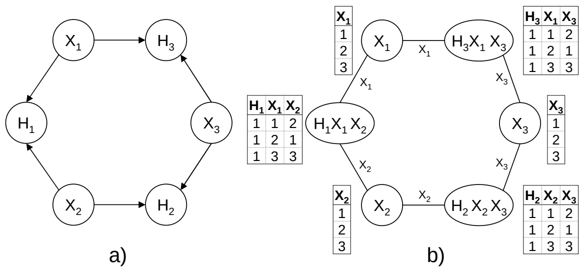

X1
H3
H1
X3
X2
H2
a)
X1 1 2 3
H3X1X3
H3 X1 X3 / 1 1 2 / 1 2 1 / 1 3 3
H1 X1 X2 / 1 1 2 / 1 2 1 / 1 3 3
H1X1X2
X3 1 2 3
X2 1 2 3
H2X2X3
H2 X2 X3 / 1 1 2 / 1 2 1 / 1 3 3
b)

Figure 4.24: a) A belief network; b) An arc-minimal dual join-graph

Zeros are Sound for any IJGP

The results for IBP can be extended to the more general class of algorithms called *iterative join-graph propagation,* IJGP [39]. IJGP can be viewed as a generalized belief propagation algorithm and was shown to benefit both from the virtues of iterative algorithms and from the anytime characteristics of bounded inference provided by mini-buckets schemes.

The message-passing of IJGP is identical to that of IBP. The difference is in the underlying graph that it uses. IJGP typically has an accuracy parameter i called i-bound, which restricts the maximum number of variables that can appear in a node (cluster). Each cluster contains a set of functions. IJGP performs message-passing on a graph called *minimal arc-labeled join-graph*.

It is easy to define a corresponding RDAC algorithm that operates on a similar minimal arc-label join-graph. Initially, each cluster of RDAC can contain a number of relations, which are just the flat correspondents of the CPTs in the clusters of IJGP. The identical mechanics of the message passing ensure that all the previous results for IBP can be extended to IJGP.

The Inference Power of IBP

We will next show that the inference power of IBP is sometimes very limited and other times strong, exactly wherever arc-consistency is weak or strong.

Cases of weak inference power

Example 4.4.6 *Consider a belief network over 6 variables $X_1, X_2, X_3, H_1, H_2, H_3$ where the domain of the X variables is $\{1, 2, 3\}$ and the domain of the H variables is $\{0, 1\}$ (see Figure4.24a). There are three CPTs over the scopes: $\{H_1, X_1, X_2\}$, $\{H_2, X_2, X_3\}$, and $\{H_3, X_1, X_3\}$. The values of the CPTs for every triplet of variables $\{H_k, X_i, X_j\}$ are:*

$$\begin{aligned} P(h_k = 1|x_i, x_j) &= \begin{cases} 1, & if \ \ (3 \neq x_i \neq x_j \neq 3); \\ 1, & if \ \ (x_i = x_j = 3); \\ 0, & otherwise \ ; \end{cases} \\ P(h_k = 0|x_i, x_j) &= 1 - P(h_k = 1|x_i, x_j). \end{aligned}$$

Consider the evidence set $e = \{H_1 = H_2 = H_3 = 1\}$. One can see that this Bayesian network expresses the probability distribution that is concentrated in a single tuple:

$$P(x_1, x_2, x_3|e) = \begin{cases} 1, & if \ \ x_1 = x_2 = x_3 = 3; \\ 0, & otherwise. \end{cases}$$

In other words, any tuple containing an assignment of "1" or "2" for any X variable has a zero probability. The flat constraint network of the above belief network is defined over the scopes $S_1 = \{H_1, X_1, X_2\}$, $S_2 = \{H_2, X_2, X_3\}$, $S_3 = \{H_3, X_1, X_3\}$. The constraints are defined by: $R_{H_k, X_i, X_j} = \{(1, 1, 2), (1, 2, 1), (1, 3, 3), (0, 1, 1), (0, 1, 3), (0, 2, 2), (0, 2, 3), (0, 3, 1), (0, 3, 2)\}$. Also, the prior probabilities for X_i's become unary constraints equal to the full domain {1,2,3} (assuming the priors are non-zero). An arc-minimal dual join-graph which is identical to the constraint network is given in Figure 4.24b.

In the flat constraint network, the constraints in each node are restricted after assigning the evidence values (see Figure 4.24b). In this case, RDAC sends as messages the full domains of the variables and therefore no tuple is removed from any constraint. Since IBP infers the same zeros as arc-consistency, IBP will also not *infer any zeros for any family or any single variable. However, since the true probability of most tuples is zero we can conclude that the inference power of IBP on this example is weak or non-existent.*

The weakness of arc-consistency as demonstrated in this example is not surprising. Arc-consistency is known to be a weak algorithm in general. It implies the same weakness for belief propagation and demonstrates that IBP is very far from completeness, at least as long as zero tuples are concerned.

The above example was constructed by taking a specific constraint network with known properties and expressing it as a belief network using a known transformation. We associate each constraint R_S with a bi-valued new hidden variable X_h, direct arcs from the constraint variables to this new hidden variable X_h, and create the CPT such that:

$$P(x_h = 1|x_{pa_h}) = 1 \ \ , iff \ \ x_{pa_h} \in R_S.$$

while zero otherwise [86]. The generated belief network conditioned on all the X_h variables being assigned "1" expresses the same set of solutions as the constraint network.

Cases of strong inference power The relationship between IBP and arc-consistency ensures that IBP is zero-complete whenever arc-consistency is. In general, if for a flat constraint network of a Bayesian network $\mathcal{B}$, arc-consistency removes all the inconsistent domain values (it creates minimal domains), then IBP will also discover all the true zeros of $\mathcal{B}$. We next consider several classes of constraints that are known to be tractable.

Acyclic belief networks. When the belief network is acyclic, namely when it has a dual join-graph that is a tree, the flat network is an acyclic constraint network that can be shown to be solvable by relational distributed arc-consistency [27]. Note that acyclic Bayesian networks is a strict superset of polytrees. The solution requires only one iteration (two passes) of IBP. Therefore:

Proposition 21 *IBP is complete for acyclic networks, when applied to the tree dual join-graph (and therefore it is also zero-complete).*

Example 4.4.7 *We refer back to the example of Figure 4.22. The network is acyclic because there is a dual join-graph that is a tree, given in Figure 4.22c, and IBP will be zero-complete on it. Moreover, IBP is known to be complete in this case.*

Belief networks with no evidence. Another interesting case is when the belief network has no evidence. In this case, the flat network always corresponds to the *causal constraint network* defined in [42]. The inconsistent tuples or domain values are already explicitly described in each relation, and new zeros do not exist. Indeed, it is easy to see (either directly or through the flat network) that:

Proposition 22 *IBP is zero-complete for any Bayesian network with no evidence.*

In fact, it can be shown [9] that IBP is also complete for non-zero posterior beliefs of many variables when there is no evidence.

Max-closed constraints. Consider next the class of Max-closed relations defined as follows. Given a domain D that is linearly ordered let *Max* be a binary operator that returns the largest element among 2. The operator can be applied to 2 tuples by taking the pair-wise operation [56].

DEFINITION **4.4.6 (Max-closed relations)** *A relation is Max-closed if whenever $t_1, t_2 \in R$ so is $Max(t_1, t_2)$. A constraint network is Max-closed if all its constraints are Max-closed.*

It turns out that if a constraint network is Max-closed, it can be solved by distributed arc-consistency. Namely, if no domain becomes empty by the arc-consistency algorithm, the network is consistent. While arc-consistency is not guaranteed to generate minimal domains, thus removing all inconsistent values, it can generate a solution by selecting the maximal value from the domain of each variable. Accordingly, while IBP will not necessarily discover all the zeros, all the largest non-zero values in the domains of each variable are true non-zeros.

Therefore, for a belief network whose flat network is Max-closed IBP is likely to be powerful for generating zero tuples.

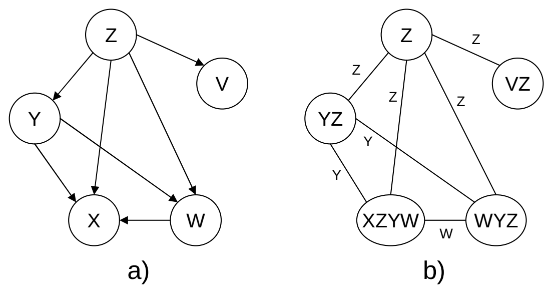


Figure 4.25: a) A belief network that corresponds to a Max-closed relation; b) An arc-minimal dual join-graph

Example 4.4.8 *Consider the following belief network: There are 5 variables* $\{V, W, X, Y, Z\}$ *over domains* $\{1, 2, 3, 4, 5\}$. *and the following CPTs:*

$$\begin{aligned} &P(x|z,y,w) \neq 0, && iff\ \ 3x + y + z \geq 5w + 1 \\ &P(w|y,z) \neq 0, && iff\ \ wz \geq 2y \\ &P(y|z) \neq 0, && iff\ \ y \geq z + 2 \\ &P(v|z) \neq 0, && iff\ \ 3v \leq z + 1 \\ &P(Z = i) = 1/4, && i \in \{1, 2, 3, 4\} \end{aligned}$$

All the other probabilities are zero. Also, the domain of W does not include 3 and the domain z does not include 5. The problem's acyclic graph is given in Figure 4.25a. It is easy to see that the flat network is the set of constraints over the above specified domains: $w \neq 3$, $z \neq 5$, $3v \leq z + 1$, $y \geq z + 2$, $3x + y + z \geq 5w + 1$, $wz \geq 2y$. *An arc-minimal dual join-graph with singleton labels is given in Figure 4.25b. It has 5 nodes, one for each family in the Bayesian network. If we apply relational distributed consistency we will get that the domains are:* $D_V = \{1\}$, $D_W = \{4\}$, $D_X = \{3, 4, 5\}$, $D_Y = \{4, 5\}$ *and* $D_Z = \{2, 3\}$. *Since all the constraints are Max-closed and since there is no empty domain the problem has a solution given by the maximal values in each domain:* $V = 1$, $W = 4$, $X = 5$, $Y = 5$, $Z = 3$. *The domains are not minimal however: there is no solution having* $X = 3$ *or* $X = 4$.

Based on the correspondence with arc-consistency, we know that applying IBP to the *dual join-graph will indeed infer all the zero domains except those of X, which validates that IBP is quite powerful for this example.*

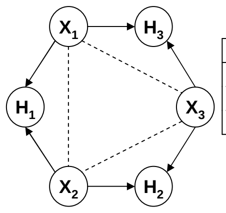
The above example is suggested by a general scheme for creating belief networks that correspond to Max-closed constraints (or any other language of constraints): First, create an acyclic graph, then, associate with each node and its parents a max-closed probability constraint.

An interesting case for propositional variables is the class of Horn clauses. A Horn clause can be shown to be Min-closed (by simply checking its models). If we have an acyclic graph, and we associate every family with a Horn clause expressed as a CPT in the obvious way, then applying Belief propagation on a dual join-graph can be shown to be nothing but the application of unit propagation until there is no change. It is well known that unit propagation decides the consistency of a set of Horn clauses (even if they are cyclic). However, unit propagation will not necessarily generate the minimal domains, and thus not infer all the zeros, but it is likely to behave well.

Implicational constraints. Finally, a class that is known to be solvable by path-consistency is implicational constraints, defined as follows:

DEFINITION **4.4.7** *A binary network is implicational, iff for every binary relation every value of one variable is consistent either with only one or with all the values of the other variable [60]. A Bayesian network is implicational if its flat constraint networks is.*

Clearly, a binary function is an implicational constraint. Since IBP is equivalent to arc-consistency only, we cannot conclude that IBP is zero-complete for implicational constraints. This raises the question of what corresponds to path-consistency in belief networks, a question which we do not attempt to answer at this point.



Prior for X_i

X_i	$P(X_i)$
1	.45
2	.45
3	.1

CPT for H_k

H_k	X_i	X_j	$P(H_k \mid X_i, X_j)$
1	1	2	1
1	2	1	1
1	3	3	1
1	...	...	0

#iter	$Bel(X_i = 1)$	$Bel(X_i = 2)$	$Bel(X_i = 3)$
1	.45	.45	.1
2	.49721	.49721	.00545
3	.49986	.49986	.00027
100	...	...	1e-129
200	...	...	1e-260
300	.5	.5	0
True belief	0	0	1

Figure 4.26: Example of a finite precision problem

A Finite Precision Problem

Algorithms should always be implemented with care on finite precision machines. We mention here a case where IBP's messages converge in the limit (i.e. in an infinite number of iterations), but they do not stabilize in any finite number of iterations. Consider again the example in Figure 4.24 with the priors on X_i's given in Figure 4.26. If all nodes H_k are set to value 1, the belief for any of the X_i variables as a function of iteration is given in the table in Figure 4.26. After about 300 iterations, the finite precision of our computer is not able to represent the value for $Bel(X_i = 3)$, and this appears to be zero, yielding the final updated belief $(.5, .5, 0)$, when in fact the true updated belief should be $(0, 0, 1)$. This does not contradict our theory, because mathematically, $Bel(X_i = 3)$ never becomes a true zero, and IBP never reaches a quiescent state.

4.4.4 Experimental Evaluation

We tested the performance of IBP and IJGP both on cases of strong and weak inference power. In particular, we looked at networks where probabilities are extreme and checked if the properties of IBP with respect to zeros also extend to ϵ small beliefs.

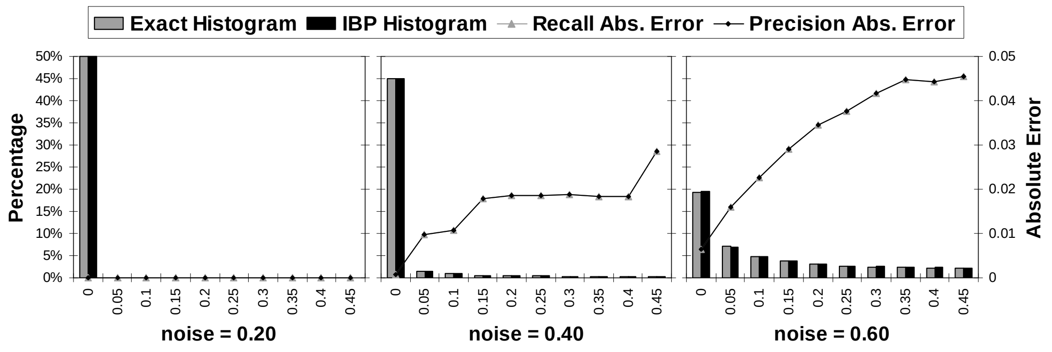


Figure 4.27: Coding, N=200, 1000 instances, w*=15

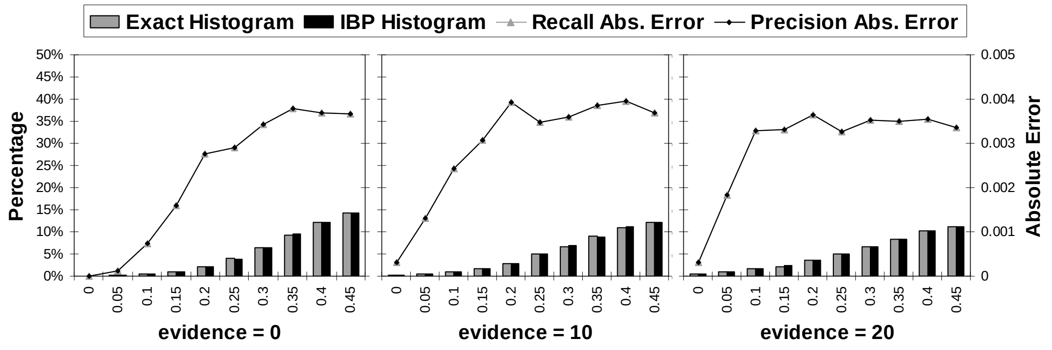


Figure 4.28: 10x10 grids, 100 instances, w*=15

Accuracy of IBP Across Belief Distribution

We investigated empirically the accuracy of IBP's prediction across the range of belief values from 0 to 1. Theoretically, zero values inferred by IBP are proved correct, and we hypothesize that this property extends to ϵ small beliefs. That is, if the flat network is easy for arc-consistency and IBP infers a posterior belief close to zero, then it is likely to be correct.

To capture the accuracy of IBP we computed its absolute error per intervals of $[0, 1]$. Using names inspired by the well known measures in information retrieval, we use *Recall Absolute Error* and *Precision Absolute Error*. *Recall* is the absolute error averaged over all the exact posterior beliefs that fall into the interval. For *Precision*, the average is taken over all the approximate posterior belief values computed by IBP that fall into the interval. Our experiments show that the two measures are strongly correlated. We also show the histograms of distribution of belief for each interval, for the exact and for IBP, which are

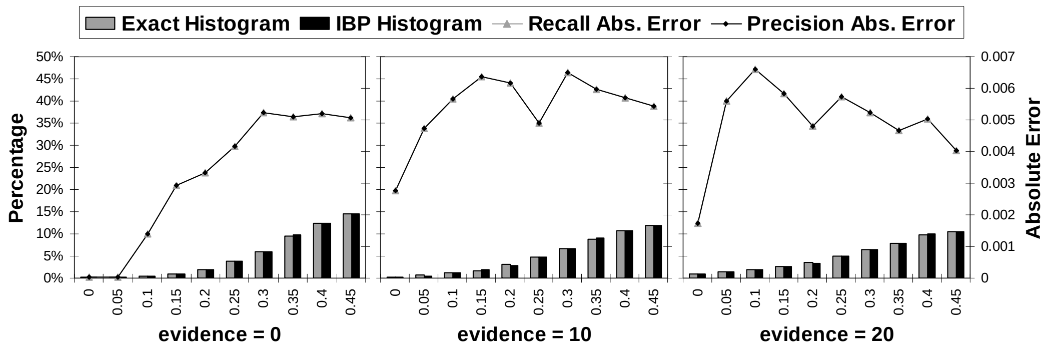


Figure 4.29: Random, N=80, 100 instances, w*=15

also strongly correlated. The results are given in Figures 4.27-4.30. The left Y axis corresponds to the histograms (the bars), the right Y axis corresponds to the absolute error (the lines). All problems have binary variables, so the graphs are symmetric about 0.5 and we only show the interval [0, 0.5]. The number of variables, number of iterations and induced width w* are reported for each graph.

Coding networks are the famous case where IBP has impressive performance. The problems are from the class of linear block codes, with 50 nodes per layer and 3 parent nodes. Figure 4.27 shows the results for three different values of channel noise: 0.2, 0.4 and 0.6. For noise 0.2, all the beliefs computed by IBP are extreme. The Recall and Precision are very small, of the order of 10^{-11}. So, in this case, all the beliefs are very small (ϵ small) and IBP is able to infer them correctly, resulting in almost perfect accuracy (IBP is indeed perfect in this case for the bit error rate). When the noise is increased, the Recall and Precision tend to get closer to a bell shape, indicating higher error for values close to 0.5 and smaller error for extreme values. The histograms also show that less belief values are extreme as the noise is increased, so all these factors account for an overall decrease in accuracy as the channel noise increases.

Grid networks results are given in Figure 4.28. Contrary to the case of coding networks, the histograms show higher concentration around 0.5. The absolute error peaks closer to 0 and maintains a plateau, as evidence is increased, indicating less accuracy for IBP.

Random networks results are given in Figure 4.29. The histograms are similar to those

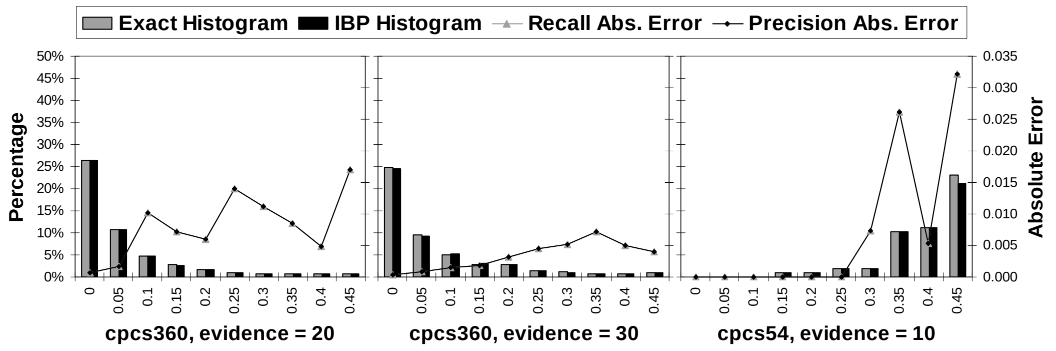


Figure 4.30: CPCS54, 100 instances, w*=15; CPCS360, 5 instances, w*=20

of the grids, but the absolute error has a tendency to decrease towards 0.5 as evidence increases. This may be due to the fact that the total number of nodes is smaller (80) than for grids (100), and the evidence can in many cases make the problem easier for IBP by breaking many of the loops (in the case of grids evidence has less impact in breaking the loops).

CPCS networks are belief networks for medicine, derived from the Computer based Patient Case Simulation system. We tested on two networks, with 54 and 360 variables. The histograms show opposing trends in the distribution of beliefs. Although irregular, the absolute error tends to increase towards 0.5 for cpcs54. For cpcs360 it is smaller around 0 and 0.5.

We note that for all these types of networks, IBP has very small absolute error for values close to zero, so it is able to infer them correctly.

Graph-coloring type problems

We also tested the behavior of IBP and IJGP on a special class of problems which were designed to be hard for belief propagation algorithms in general, based on the fact that arc-consistency is poor on the flat network.

We consider a graph coloring problem which is a generalization of example 4.4.6, with $N = 20$ X nodes, rather than 3, and a variable number of H nodes defining the density of the constraint graph. X variables are 3-valued root nodes, H variables are bi-valued and

Table 4.11: Graph coloring type problems: 20 root variables

Absolute error				
	ϵ	H=40, w*=5	H=60, w*=7	H=80, w*=9
IBP	0.0	0.4373	0.4501	0.4115
	0.1	0.3683	0.4497	0.3869
	0.2	0.2288	0.4258	0.3832
IJGP(2)	0.0	0.1800	0.1800	0.1533
	0.1	0.3043	0.3694	0.3189
	0.2	0.1591	0.3407	0.3022
IJGP(4)	0.0	0.0000	0.0000	0.0000
	0.1	0.1211	0.0266	0.0133
	0.2	0.0528	0.1370	0.0916
IJGP(6)	0.0	0.0000	0.0000	0.0000
	0.1	0.0043	0.0000	0.0132
	0.2	0.0123	0.0616	0.0256

each has two parents which are X variables, with the CPTs defined like in example 4.4.6. Each H CPT actually models a binary constraint between two X nodes. All H nodes are assigned value 1. The flat network of this kind of problems has only one solution, where every X has value 3. In our experiments we also added noise to the H CPTs, making probabilities ϵ and $1 - \epsilon$ rather than 0 and 1.

The results are given in Table 4.11. We varied parameters along two directions. One was increasing the number of H nodes, corresponding to higher densities of the constraint network (the average induced width $w*$ is reported for each column). The other was increasing the noise parameter ϵ. We averaged over 50 instances for each combination of these parameters. In each instance, the priors for nodes X were random uniform, and the parents for each node H were chosen randomly. We report the absolute error, averaged over all values, all variables and all instances. We should note that these are fairly small size networks (w*=5-9), yet they prove to be very hard for IBP and IJGP, because the flat network is hard for arc-consistency. It is interesting to note that even when ϵ is not extreme anymore (0.2) the performance is still poor, because the structure of the network is hard for arc-consistency. IJGP with higher i-bounds is good for $\epsilon = 0$ because it is able to infer some zeros in the bigger clusters, and these propagate in the network and in turn infer more zeros.

4.4.5 Discussion

The work presented in this section investigates the behavior of belief propagation algorithms by making analogies to well known and understood algorithms from constraint networks. By a simple transformation, called flattening of the Bayesian network, IBP (as well as any generalized belief propagation algorithm) can be shown to work in a manner that is similar to relational distributed arc-consistency relative to zero tuples generation. In particular we show that IBP's inference of zero beliefs converges and is sound.

Theorem 4.4.5 provides a justification for applying the belief propagation algorithm iteratively. We know that arc-consistency algorithms improve with iteration, generating the largest arc-consistent network that is equivalent to the original network. Therefore by applying IBP iteratively the set of zero tuples concluded grows monotonically until convergence.

While the theoretical results presented here are straightforward, they help identify new classes of problems that are easy or hard for IBP. Non-ergodic belief networks with no evidence, max-closed or implicational belief networks are expected to be cases of strong inference power for IBP. Based on empirical work, we observe that good performance of IBP and many small beliefs indicate that the flat network is likely to be easy for arc-consistency. On the other hand, when we generated hard networks for arc-consistency, IBP was very poor in spite of the presence of many zero beliefs. We believe that the success of IBP for coding networks can be explained by the presence of many extreme beliefs on one hand, and by an easy-for-arc-consistency flat network on the other. We plan to conduct more experiments on coding networks and study the influence of the good accuracy of IBP for extreme beliefs combined with the ϵ-cutset effect described in [9].

4.5 Conclusion to Chapter 4

In this chapter we investigated a family of approximation algorithms for mixed networks, that could also be extended to graphical models in general. We started with bounded inference algorithms and proposed Mini-Clustering (MC) scheme as a generalization of Mini-Buckets to arbitrary tree decompositions. Its power lies in being an anytime algorithm governed by a user adjustable i-bound parameter. MC can start with small i-bound and keep increasing it as long as it is given more time, and its accuracy usually improves with more time. If enough time is given to it, it is guaranteed to become exact.

Inspired by the success of iterative belief propagation (IBP), we extended MC into an iterative message-passing algorithm called Iterative Join-Graph Propagation (IJGP). IJGP operates on general join-graphs that can contain cycles, but it is sill governed by an i-bound parameter. Unlike IBP, IJGP is guaranteed to become exact if given enough time.

We also make connections with well understood consistency enforcing algorithms for constraint satisfaction, giving strong support for iterating messages, and helping identify cases of strong and weak inference power for IBP and IJGP. We show that: (1) if a value of a variable is assessed as having zero-belief in any iteration of IBP, then it remains a zero-belief in all subsequent iterations; (2) that IBP converges in a finite number of iterations relative to its set of zero-beliefs; and, most importantly (3) that the set of zero-beliefs decided by any of the iterative belief propagation methods is sound. Namely any zero-belief determined by IBP corresponds to a true zero conditional probability relative to the given probability distribution expressed by the Bayesian network.

Experimental evaluation is provided for all these schemes, and IJGP emerges as one of the most powerful approximate algorithms for belief networks.

Chapter 5

AND/OR Cutset Conditioning

5.1 Introduction

The complexity of a reasoning task over a graphical model depends on the induced width of the graph. For inference-type algorithms, the space complexity is exponential in the induced width in the worst case, which often makes them infeasible for large and densely connected problems. In such cases, space can be traded at the expense of time by conditioning (assigning values to variables).

Search algorithms perform conditioning on all the variables. Cycle cutset schemes [86, 26] only condition on a subset of variables such that the remaining network is singly connected (i.e., is a tree) and can be solved by inference tree algorithms. The more recent hybrid *w-cutset* scheme [90, 10] conditions on a subset of variables such that, when removed, the remaining network has induced width w or less, and can be solved by a variable elimination [29] type algorithm.

5.1.1 Contributions

In this chapter we revisit the well known conditioning method of cycle cutset and introduce the new concept of *AND/OR cycle cutset*. We show that the AND/OR cycle cutset is a strict improvement over the traditional cycle cutset method (and the same holds for the extended w-cutset version). The result goes beyond the simple organization of the traditional cutset in an AND/OR pseudo tree, which would be just the straightforward improvement. The complexity of exploring the traditional cutset is time exponential in the number of nodes in the cutset, and therefore it calls for finding a minimal cardinality cutset, denoted by $\mathbf{C}$. The complexity of exploring the AND/OR cutset is time exponential in its depth, and therefore it calls for finding a minimal depth *AND/OR cutset*, denoted by AO-$\mathbf{C}$. That is, a set of nodes that can be organized in a start pseudo tree of minimal depth. So, while the cardinality of the optimal AND/OR cutset, $|AO$-$\mathbf{C}|$, may be larger than that of the optimal traditional cutset, $|\mathbf{C}|$, the depth of AO-$\mathbf{C}$ is always smaller than or equal to $|\mathbf{C}|$.

The research presented in this chapter is based in part on [75, 38].

5.2 Traditional Cycle Cutset Explored by AND/OR Search

The AND/OR paradigm exploits the problem structure, by solving independent components separately. This fundamental idea can also be applied to the cycle cutset method, or reasoning by conditioning [86].

DEFINITION **5.2.1 (cycle cutset)** *Given a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, a* cycle cutset *is a subset $\mathbf{C} \subset \mathbf{X}$ such that the primal graph of $\mathcal{M}$ becomes singly connected (i.e., a tree) if all the nodes in $\mathbf{C}$ are removed from it. An* optimal cycle cutset *is one having the minimum number of variables.*

The cycle cutset method consists of enumerating all the possible instantiations of $\mathbf{C}$, and for each one of them solving the remaining singly connected network by a linear time and space tree algorithm. The instantiations of $\mathbf{C}$ are enumerated by regular OR search, yielding linear space complexity and $O(\exp |\mathbf{C}|)$ time complexity, therefore requiring a

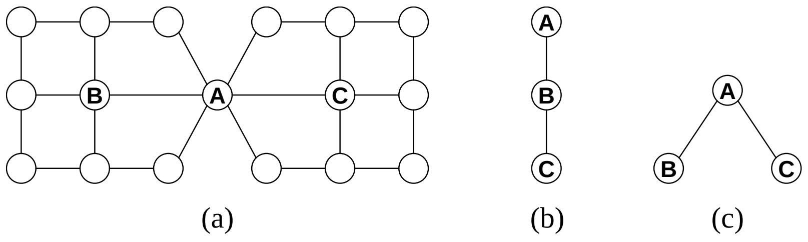


Figure 5.1: Traditional cycle cutset viewed as AND/OR tree

minimal cycle cutset to optimize complexity.

A first simple improvement to the traditional cycle cutset scheme described above would be the enumeration of **C** by AND/OR search.

Example 5.2.1 *Figure 5.2.1a shows two 3×3 grids, connected on the side node A. A cycle cutset must include at least two nodes from each grid, so the minimal cycle cutset contains three nodes: the common node A and one more node from each grid, for example B and C. The traditional way of solving the cycle cutset problem consists of enumerating all the assignments of the cycle cutset $\{A, B, C\}$, as if these variables form the chain pseudo tree in Figure 5.2.1b. However, if A is the first conditioning variable, the remaining subproblem is split into two independent portions, so the cycle cutset $\{A, B, C\}$ can be organized as an AND/OR search space based on the pseudo tree in Figure 5.2.1c. If k is the maximum domain size of variables, the complexity of solving Figure 5.2.1b is $O(k^3)$ while that of solving Figure 5.2.1c is $O(k^2)$.*

We can improve the general cycle cutset method, based on the previous example: first find the minimal cycle cutset **C**; then find the minimal depth *start pseudo tree* made of nodes in **C**:

DEFINITION **5.2.2 (start pseudo tree)** *Given an undirected graph $G = (\mathbf{X}, E)$, a directed rooted tree $T = (V, E')$, where $V \subseteq \mathbf{X}$, is called a* start pseudo tree *if it has the same root and is a subgraph of some pseudo tree of G.*

If a cycle cutset of cardinality $|\mathbf{C}| = c$ is explored by AND/OR search, based on a start pseudo tree T over the set $\mathcal{C}$, and the depth of T is m, then $m \leq c$. Therefore,

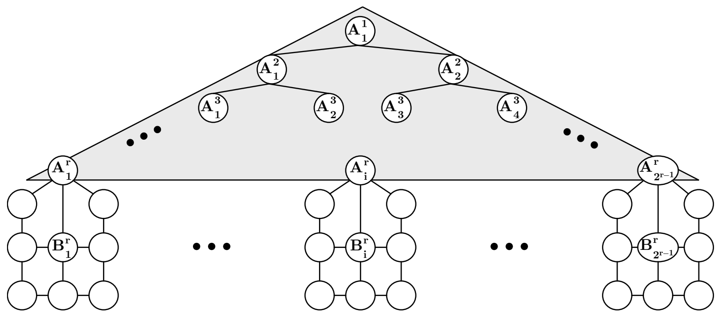


Figure 5.2: AND/OR cycle cutset

Proposition 23 *Exploring a cycle cutset by AND/OR search is always better than, or the same as, exploring it by OR search.*

5.3 AND/OR Cycle Cutset

The idea presented in section 5.2 is a straightforward application of the AND/OR paradigm to cycle cutsets. In the following we will describe a more powerful version of the *AND/OR cycle cutset*.

DEFINITION **5.3.1 (AND/OR cycle cutset)** *Given a graphical model $\mathcal{R} = (X, D, F)$, an* AND/OR cycle cutset *$AO\text{-}\mathcal{C}$ is a cycle cutset together with an associated start pseudo tree $T_{AO\text{-}\mathcal{C}}$ of depth m. An* optimal AND/OR cycle cutset *is one having the minimum depth m.*

Example 5.3.1 *Figure 5.3.1 shows a network for which the* optimal cycle cutset *contains fewer nodes than the* optimal AND/OR cycle cutset, *yet the latter yields an exponential improvement in time complexity. The network in the example is based on a complete binary tree of depth r, the nodes marked A_i^j shown on a gray background. The upper index j corresponds to the depth of the node in the binary tree, and the lower index i to the position in the level. Each of the leaf nodes, from A_1^r to $A_{2^{r-1}}^r$ is a side node in a 3×3 grid. A cycle cutset has to contain at least 2 nodes from each of the 2^{r-1} grids. An optimal cycle cutset is $\mathcal{C} = \{A_1^r, \ldots, A_{2^{r-1}}^r, B_1^r, \ldots, B_{2^{r-1}}^r\}$, containing 2^r nodes, so the complexity is $O(\exp|\mathcal{C}|) = O(\exp(2^r))$. We should note that the best organization of $\mathcal{C}$ as an*

AND/OR space would yield a pseudo tree of depth $2^{r-1}+1$. *This is because all the nodes in* $\{A^r_1,\ldots,A^r_{2^{r-1}}\}$ *are connected by the binary tree, so they all must appear along the same path in the pseudo tree (this observation also holds for any other optimal cycle cutset in this example). Exploring* $\mathcal{C}$ *by AND/OR search lowers the complexity from* $O(\exp(2^r))$ *to* $O(\exp(2^{r-1}+1))$.

Let's now look at the AND/OR cycle cutset $AO\text{-}\mathcal{C} = \{A^j_i \mid j=1,\ldots,r;\ i=1,\ldots,2^{j-1}\} \cup \{B^r_1,\ldots,B^r_{2^{r-1}}\}$, *containing all the* A *and* B *nodes. A pseudo tree in this case is formed by the binary tree of* A *nodes, and the* B *nodes exactly in the same position as in the figure. The depth in this case is* $r+1$, *so the complexity is* $O(\exp(r+1))$, *even though the number of nodes is* $|AO\text{-}\mathcal{C}| = |\mathcal{C}| + 2^{r-1} - 1$.

The previous example highlights the conceptual difference between the *cycle cutset method* and what we will call the *AND/OR cycle cutset method*. In *cycle cutset*, the objective is to identify the smallest cardinality cutset. Subsequently, the exploration can be improved from OR search to AND/OR search. In *AND/OR cycle cutset* the objective is to find a cutset that forms a start pseudo tree of smallest depth.

THEOREM **5.3.2** *Given a graphical model* $\mathcal{R}$, *an optimal cycle cutset* $\mathcal{C}$, *its corresponding smallest depth start pseudo tree* $T_{\mathcal{C}}$, *and the optimal AND/OR cycle cutset* $AO\text{-}\mathcal{C}$ *with the start pseudo tree* $T_{AO\text{-}\mathcal{C}}$, *then:*

$$|\mathcal{C}| \geq depth(T_{\mathcal{C}}) \geq depth(T_{AO\text{-}\mathcal{C}}) \tag{5.1}$$

There exist instances for which the inequalities are strict.

Proof. The leftmost inequality follows from Prop. 23. The rightmost inequality follows from the definition of AND/OR cycle cutsets. Example 5.3.1 is an instance where the inequalities are strict. □

We should note that strict inequalities in Eq. 5.1 could translate into exponential differences in time complexities.

5.4 AND/OR w-Cutset

The principle of cutset conditioning can be generalized using the notion of *w-cutset*. A *w-cutset* of a graph is a set of nodes such that, when removed, the remaining graph has induced width at most w. A hybrid algorithmic scheme combining conditioning and *w-bounded* inference was presented in [90, 63]. More recently, *w-cutset* sampling was investigated in [10], and the complexity of finding the minimal *w-cutset* was discussed in [11].

The hybrid *w-cutset* algorithm performs search on the cutset variables and exact inference (e.g., bucket elimination [29]) on each of the conditioned subproblems. If the *w-cutset* C_w is explored by linear space OR search, the time complexity is $O(\exp(|C_w| + w))$, and the space complexity is $O(\exp w)$.

The AND/OR cycle cutset idea can be extended naturally to *AND/OR w-cutset*. To show an example of the difference between the traditional w-cutset and the *AND/OR w-cutset* we refer again to the example in Figure 5.3.1. Consider each 3×3 grid replaced by a network which has a minimal w-cutset C_w. The minimal w-cutset of the whole graph contains in this case $2^{r-1} \cdot |C_w|$ nodes. If this w-cutset is explored by OR search, it yields a time complexity exponential in $(2^{r-1} \cdot |C_w| + w)$. If the w-cutset is explored by AND/OR search it yields a time complexity exponential in $(2^{r-1} + |C_w| + w)$ (similar to Example 5.3.1). In contrast to this, the *AND/OR w-cutset*, which contains the A nodes and the w-cutsets of each leaf network, yields a time complexity exponential only in $(r + |C_w| + w)$, or possibly even less if the nodes in C_w can be organized in a start pseudo tree which is not a chain (i.e., has depth smaller than $|C_w|$).

5.5 Algorithm Description

The idea of w-cutset schemes is to define an algorithm that can run in space $O(\exp w)$. The *AND/OR w-cutset algorithm* is a hybrid scheme. The cutset portion, which is organized in a start pseudo tree, is explored by AND/OR search. The remaining w-bounded subproblems can be solved either by a variable elimination type algorithm, or by search with w-bounded caching - in particular, AND/OR search with full caching is feasible for these subproblems.

5.5.1 Adaptive AND/OR Caching Scheme

In [45], the caching scheme of AND/OR search is based on *contexts* [23], which are precomputed based on the pseudo tree before search begins. Algorithm $AO(i)$ performs caching only at the variables for which the context size is smaller than or equal to i (called *i-bound*).

The cutset principle inspires a more refined caching scheme for AO, which caches some values even at nodes with contexts greater than the *i-bound*. Lets assume the context of the node X_k is $context(X_k) = \{X_1, \ldots, X_k\}$, where $k > i$. During the search, when variables $X_1, \ldots, X_{k-i}$ are instantiated, they can be regarded as part of a cutset. The problem rooted by X_{k-i+1} can be solved in isolation, like a subproblem in the cutset scheme, after the variables $X_1, \ldots, X_{k-i}$ are assigned their current values in all the functions. In this subproblem, $context(X_k) = \{X_{k-i+1}, \ldots, X_k\}$, so it can be cached within i-bounded space. However, when the search retracts to X_{k-i} or above, the cache table for variable X_k needs to be purged, and will be used again when a new subproblem rooted at X_{k-i+1} is solved.

This improved caching scheme only increases the space requirements linearly, compared to $AO(i)$, but the time savings can be exponential. We will show results in section 5.6.

5.5.2 Algorithm $AO\text{-}C(i)$

We can now define the different versions of *AND/OR i-cutset algorithm* that we experimented with. We chose to explore the cutset portion either by linear space AND/OR search (no caching) or by AND/OR search with improved caching. For the i-bounded subproblems, we chose either Bucket Elimination (BE) or AND/OR search with full caching (which coincides with the improved caching on the bounded subproblems). The four resulting algorithms are: 1) $AO\text{-}LC(i)$ - linear space cutset and full caching for subproblems; 2) $AO\text{-}LC\text{-}BE(i)$ - linear space cutset and BE for subproblems; $AO\text{-}C(i)$ - improved caching everywhere; 4) $AO\text{-}C\text{-}BE(i)$ - improved caching on cutset and BE on subproblems.

5.5.3 Finding a Start Pseudo Tree

The performance of $AO\text{-}C(i)$ is influenced by the quality of the start pseudo tree. Finding the minimal depth start pseudo tree for the given i-bound is a hard problem, and it is beyond the scope of this chapter to address its complexity and solution. We will only describe the heuristic we used in creating the pseudo trees for our experiments.

Min-Fill [61] is one of the best and most widely used heuristics for creating small induced width orderings. The ordering defines a unique pseudo tree. The minimal start pseudo for an i-bound contains the nodes for which some descendant has adjusted context (i.e., context without the variables instantiated on the current path) greater than i. Min-Fill heuristic tends to minimize context size, rather than pseudo tree depth. Nevertheless, we chose to try it and discovered that it provides one of the best pseudo trees for higher values of i.

Min-Depth We developed a heuristic to produce a balanced start pseudo tree, resulting in smaller depth. We start from a Min-Fill tree decomposition and then iteratively search for the separator that would break the tree in parts that are as balanced as possible, relative to the following measure: on either side of the separator eliminate the separator variables, count the number of remaining clusters, say n, and then add the sizes of the largest $\log n$ clusters.

GWC [11] is a greedy algorithm to build a minimal cardinality cutset. In the process, we also arranged the minimal cardinality cutset as AND/OR cutset, to compare with the minimal depth cutset that we could find.

CPCS 422 - $f(i)$											
i	1	2	3	4	5	6	7	8	9	10	11
$d(AO\text{-}C)$	32	32	32	32	31	31	31	31	31	30	29
$d(C)$	40	37	32	32	38	37	36	34	32	30	29
$\|C\|$	79	71	65	59	54	50	46	41	37	34	32
GWCA	79	67	60	55	50	46	42	38	34	31	29

Table 5.1: CPCS 422 - Cutsets Comparison

5.6 Experimental Evaluation

We investigated two directions. One was to empirically test the quality of the start pseudo trees, and the other was to compare actual runs of the different versions of $AO\text{-}C(i)$.

5.6.1 The Quality of Start Pseudo Trees

We report here the results on the CPCS 422b network from the UAI repository. It has 422 nodes and induced width 22. Table 5.1 shows the values of $f(i)$, which expresses the total complexity of a cutset scheme. For a cardinality cutset, $f(i) = i + |\mathcal{C}|$ and for an AND/OR cutset of depth d, $f(i) = i + d$. The row $d(AO\text{-}C)$ shows the depth of the best AND/OR cutset we could find. $|C|$ shows the number of nodes in the best cutset found by GWC, and $d(C)$ shows its depth when organized as AND/OR cutset. GWCA is taken from [11]. The best complexity, expressed by small values of $f(i)$, is always given by the AND/OR cutset, and for smaller values of i they translate into impressive savings over the cardinality cutset $\mathcal{C}$.

In all our experiments described in the following, we refrained from comparing the new cutset scheme with the old cardinality cutset scheme (equivalent to an OR search on the cutset), because the latter was too slow.

5.6.2 Performance of $AO\text{-}C(i)$

We tested the different version of the $AO\text{-}C(i)$ family primarily on Bayesian networks with strictly positive distributions, for the task of belief updating. This is necessary to grasp the

N=40, K=3, P=2, 20 instances, w*=7							
i	Algorithms	d		Time(sec)		# nodes	
		MF	MD	MF	MD	MD	MF
1	AO(i)	12	9	610.14	27.12	50,171,141	1,950,539
	AO-LC(i)			174.53	8.75	13,335,595	575,936
	AO-C(i)			67.99	7.61	4,789,569	499,391
	AO-C-BE(i)			16.95	**2.18**	-	-
3	AO(i)	7	6	71.68	8.13	5,707,323	595,484
	AO-LC(i)			5.73	0.84	501,793	69,357
	AO-C(i)			2.94	0.84	248,652	69,357
	AO-C-BE(i)			0.69	**0.25**	-	-
5	AO(i)	4	3	11.28	2.77	999,441	24,396
	AO-LC(i)			0.55	0.54	50,024	4,670
	AO-C(i)			0.55	0.55	49,991	4,670
	AO-C-BE(i)			0.10	**0.04**	-	-

N=60, K=3, P=2, 20 instances, w*=11							
i	Algorithms	d		Time(sec)		# nodes	
		MF	MD	MF	MD	MD	MF
6	AO-LC(i)	7	6	159.79	63.01	14,076,416	5,165,486
	AO-C(i)			112.43	62.98	9,925,855	5,165,486
	AO-C-BE(i)			27.33	**5.50**	-	-
9	AO-LC(i)	3	3	24.40	41.45	2,140,791	3,509,709
	AO-C(i)			24.15	40.93	2,140,791	3,509,709
	AO-C-BE(i)			4.27	**2.89**	-	-
11	AO-LC(i)	0	1	17.39	38.46	1,562,111	3,173,129
	AO-C(i)			17.66	38.22	1,562,111	3,173,129
	AO-C-BE(i)			**1.29**	2.81	-	-

Table 5.2: Random Networks

power of the scheme when no pruning is involved in search.

In all the tables N is the number of nodes, K is the maximum domain size, P is the number of parents of a variable, w^* is the induced width, i is the i-bound, d is the depth of the i-cutset. For most problems, we tested a min-fill pseudo-tree (MF) and one based on the depth minimizing heuristic (MD). The time and the number of nodes expanded in the search are shown for the two pseudo trees correspondingly.

Random networks. Table 5.2 shows results for random networks, generated based on N, K and P and averaged over 20 instances. Note that K=3, which makes the problems harder, even though w^* seems small. For N=40 we see that the old scheme AO(i) is always outperformed. Using improved caching on the cutset is almost always beneficial. For i very close to $w*$, caching on the cutset doesn't save much, and in some cases when no caching is possible, the extra overhead may actually make it slightly slower. Also, for strictly positive distributions, switching to BE is faster than running AO search with caching on the remaining problems.

CPCS networks. CPCS are real life networks for medical diagnoses, which are hard for belief updating. Table 5.3 shows results for CPCS 360 file, having induced width 20. For $i = 20$, AO-C-$BE(i)$ is actually BE. It is interesting to note that AO-LC-$BE(i)$, for $i = 12$ is actually faster than BE on the whole problem, while requiring much less space ($\exp(12)$ compared to $\exp(20)$), due to smaller overhead in caching (smaller cache tables) and a good ordering that doesn't require recomputing the same problems again. We also mention that AO(i) was much slower on this problem and therefore not included in the table.

In the above experiments, the values of d show that MF heuristic provided a better cutset for large values of i, while the MD heuristic provided good cutsets when i was small.

Genetic linkage network. We include in Table 5.4 results for the genetic linkage network EA4 [47]. This is a large network, with N=1173, but relatively small induced width, $w^* = 15$. This network contains a lot of determinism (zero probability tuples). We did not use in AO search any form of constraint propagation, limiting the algorithm to prune only the zero value nodes (their subproblems do not contribute to the updated belief). We note here that for i-bound 13 and 9, AO-$C(i)$ is faster than AO-C-$BE(i)$ because it is able to prune the search space. We used a version of BE which is insensitive to determinism.

Large networks. Memory limitations are the main drawback of BE. In Table 5.5 we show results for hard networks, solved by AO-C-$BE(i)$, where $i = 12$ is set to the maximum value that we could use on a 2.4 GHz Pentium IV with 1 GB of RAM. For N=100, the space requirements of BE would be about 100 times bigger than the RAM (note K=3), yet AO-C-$BE(12)$ could solve it in about six and a half hours, showing the scalability of the AND/OR cutset scheme.

CPCS 360b, N=360, K=2, w* = 20				
i	Algorithms	d (MF)	Time	# nodes
1	AO-LC(i)	23	2,507.6	406,322,117
	AO-LC-BE(i)		1,756.4	-
	AO-C(i)		1,495.2	243,268,549
	AO-C-BE(i)		1,019.4	-
12	AO-LC(i)	8	186.8	14,209,057
	AO-LC-BE(i)		**10.3**	-
	AO-C(i)		185.1	14,209,057
	AO-C-BE(i)		10.4	-
20	AO-LC(i)	0	167.8	12,046,369
	AO-LC-BE(i)		**11.5**	-
	AO-C(i)		170.9	12,046,369
	AO-C-BE(i)		**11.6**	-

Table 5.3: CPCS 360

EA4 - N=1173, K=5, w*=15							
i	Algorithms	d		Time(sec)		# nodes	
		MF	MD	MF	MD	MD	MF
6	AO(i)	23	21	10.0	103.4	1,855,490	15,312,582
	AO-LC(i)			22.5	76.4	3,157,012	9,928,754
	AO-C(i)			**2.0**	51.3	281,896	6,666,210
	AO-C-BE(i)			8.4	82.3	-	-
9	AO(i)	18	17	3.3	9.3	410,934	1,466,338
	AO-LC(i)			1.6	4.7	196,662	617,138
	AO-C(i)			**1.5**	4.8	196,662	616,802
	AO-C-BE(i)			3.5	7.0	-	-
13	AO(i)	3	8	2.0	5.9	235,062	887,138
	AO-LC(i)			1.4	3.6	172,854	431,458
	AO-C(i)			1.6	3.4	172,854	431,458
	AO-C-BE(i)			**0.7**	5.3	-	-

Table 5.4: Genetic Linkage Network

5.7 Conclusion to Chapter 5

This section presents the *AND/OR w-cutset scheme*, which combines the newly developed AND/OR search for graphical models [45] with the w-cutset scheme [10]. Theorem 5.3.2 shows that the new scheme is always at least as good as the existing cutset schemes, but it often provides exponential improvements.

The new AND/OR cutset inspired an improved caching scheme for the AND/OR search, which is always better than the one used by AO(i) [45], based on context.

The experimental evaluation showed, first, that the theoretical expectations of getting exponential improvements over the traditional cardinality cutset are actually met in practice.

Second, it showed the power and flexibility of the new hybrid scheme. Our conclusion

K=3, P=2; AO-C-BE(i), i=12			
N	w*	d (MF)	Time(sec)
70	13	2	12
80	15	3	61
90	17	6	2,072
100	18	9	22,529

Table 5.5: Networks with high memory requirements for BE

is that improved caching on the cutset is in most cases beneficial. For the remaining problems, if the task is belief updating (or counting solutions) and there is little determinism, then switching to BE is faster. In the presence of determinism, solving the remaining problems with search with full caching may be better. We leave for future work the investigation of using look-ahead and no-good learning in the presence of determinism for the AND/OR w-cutset scheme.

Finally, the new scheme is scalable to memory intensive problems, where inference type algorithms are infeasible.

Chapter 6

AND/OR Search and Inference Algorithms

6.1 Introduction

It is convenient to classify algorithms that solve reasoning problems of graphical models as either search (*e.g.*, depth first, branch and bound) or inference (*e.g.*, variable elimination, join-tree clustering). Search is time exponential in the number of variables, yet it can be accomplished in linear memory. Inference exploits the graph structure of the model and can be accomplished in time and space exponential in the *treewidth* of the problem. When the treewidth is big, inference must be augmented with search to reduce the memory requirements. In the past three decades search methods were enhanced with structure exploiting techniques. These improvements often require substantial memory, making the distinction between search and inference fuzzy. Recently, claims regarding the superiority of memory-intensive search over inference or vice-versa were made [5]. Our aim is to clarify this relationship and to create cross-fertilization using the strengths of both schemes.

6.1.1 Contributions

First, we compare pure search with pure inference algorithms in graphical models through the new framework of AND/OR search. Specifically, we compare Variable Elimination (**VE**) against memory-intensive AND/OR Search (**AO**), and place algorithms such as graph-based backjumping, no-good and good learning, and look-ahead schemes [31] within the AND/OR search framework. We show that there is no principled difference between memory-intensive search restricted to fixed variable ordering and inference beyond: (1) different direction of exploring a common search space (top down for search vs. bottom-up for inference); (2) different assumption of control strategy (depth first for search and breadth first for inference). We also show that those differences have no practical effect, except under the presence of determinism. Our analysis assumes a fixed variable ordering. When variable ordering is dynamic in search, some of these conclusions may not hold.

Second, we address some long-standing questions regarding the computational merits of several time-space sensitive algorithms for graphical models. In the past ten years, four types of algorithms have emerged, based on: (1) cycle-cutset and w-cutset [86, 26]; (2) alternating conditioning and elimination controlled by induced-width w [90, 63, 47]; (3) recursive conditioning [23], which was recently recast as context-based AND/OR search [45]; (4) varied separator-sets for tree decompositions [32]. The question is how do all these methods compare and, in particular, is there one that is superior? A brute-force analysis of time and space complexities of the respective schemes does not settle the question. For example, if we restrict the available space to be linear, the cycle-cutset scheme is exponential in the cycle-cutset size while recursive conditioning is exponential in the depth of the pseudo tree (or d-tree) that drives the computation. However some graphs have small cycle-cutset and larger tree depth, while others have large cycle-cutsets and small tree depth (e.g., grid-like chains). The immediate conclusion seems to be that the methods are not comparable.

We show that by looking at all these schemes side by side, and analyzing them using the context minimal AND/OR graph data structure [74], each of these schemes can be improved via the AND/OR search principle and by careful caching, to the point that they all become identically good. Specifically, we show that the new algorithm *Adaptive Caching* (**AOC(i)**), inspired by AND/OR cutset conditioning [75] (improving cutset, and w-cutset schemes), can simulate any execution of alternating elimination and conditioning, if the latter is augmented with AND/OR search over the conditioning variables, and can also simulate any execution of separator controlled tree-clustering schemes [32], if the clusters are augmented with AND/OR cutset search, rather than regular search, as was initially proposed.

All the analysis is again done assuming that the problem contains no determinism. When the problem has determinism all these schemes become incomparable, as was shown in [74], because they are all different in their variable ordering approach and this accounts for differences in exploiting determinism as we observed in the simplest case comparing Variable Elimination to AND/OR search [74].

The research presented in this chapter is based in part on [74, 78, 38].

6.2 AND/OR Search (AO) vs. Variable Elimination (VE)

We will compare Variable Elimination and search by the portions of a common search space that they traverse and record. Since VE's execution is uniquely defined by a bucket-tree, and since every bucket tree corresponds to a pseudo tree, and a pseudo tree uniquely defines the context-minimal AND/OR search graph, we can compare both schemes on this common search space. Furthermore, we choose the context-minimal AND/OR search graph (CM) because algorithms that traverse the full CM need memory which is comparable to that used by VE, namely, space exponential in the treewidth of their pseudo/bucket trees.

Algorithm AO denotes any traversal of the CM search graph, AO-DF is a depth-first traversal and AO-BF is a breadth-first traversal. We will compare VE and AO via the

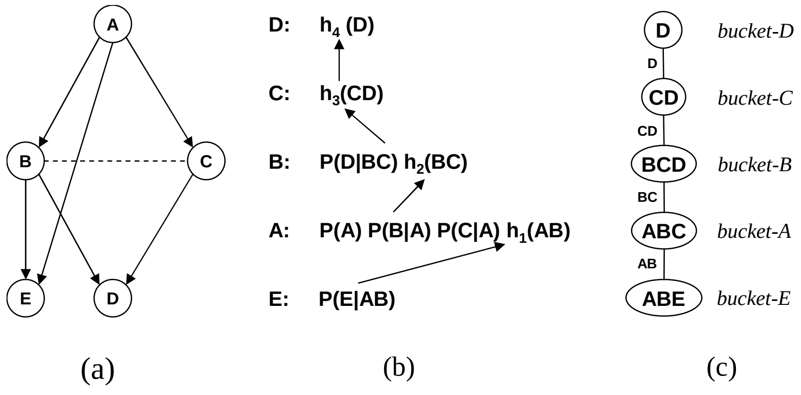


Figure 6.1: Variable Elimination

portion of this graph that they generate and by the order of node generation. The task's value computation performed during search adds only a constant factor.

We distinguish graphical models with or without determinism, namely, graphical models that have inconsistencies vs. those that have none. We compare *brute-force* versions of VE and AO, as well as versions enhanced by various known features. We assume that the task requires the examination of all solutions (e.g. belief updating, counting solutions).

6.2.1 AO vs. BE with No Determinism

We start with the simplest case in which the graphical model contains no determinism and the bucket tree (pseudo tree) is a chain.

OR Search Spaces

Figure 6.1a shows a Bayesian network. Let's consider the ordering $d = (D, C, B, A, E)$ which has the treewidth $w(d) = w^* = 2$. Figure 6.1b shows the bucket-chain and a schematic application of VE along this ordering (the bucket of E is processed first, and the bucket of D last). The buckets include the initial CPTs and the functions that are generated and sent (as messages) during the processing. Figure 6.1c shows the bucket tree.

If we use the chain bucket tree as pseudo tree for the AND/OR search along d, we get the *full CM graph* given in Figure 6.2. Since this is an OR space, we can eliminate the OR

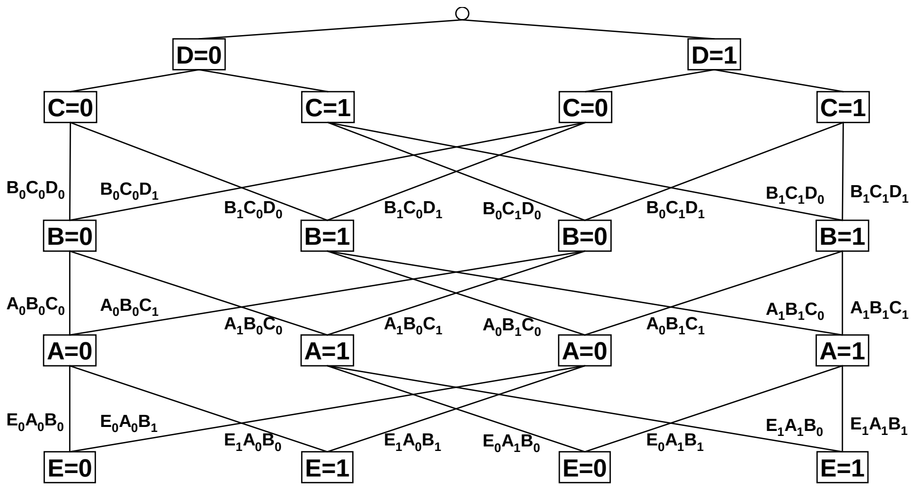


Figure 6.2: Context-minimal AND/OR search space

levels as shown. Each level of the graph corresponds to one variable. The edges should be labeled with the product of the values of the functions that have just been instantiated on the current path. We note on the arc just the assignment to the relevant variables (e.g., B_1 denotes $B = 1$). For example, the edges between C and B in the search graph are labeled with the function valuation on (BCD), namely $P(D|B, C)$, where for each individual edge this function is instantiated as dictated on the arcs.

AO-DF computes the value (e.g., updated belief) of the root node by generating and traversing the context-minimal graph in a depth-first manner and accumulating the partial value (e.g., probabilities) using combination (products) and marginalization (summation). The first two paths generated by AO-DF are $(D_0, C_0, B_0, A_0, E_0)$ and $(D_0, C_0, B_0, A_0, E_1)$, which allow the first accumulation of value $h_1(A_0B_0) = P(E_0|A_0B_0)+P(E_1|A_0B_0)$. AO-DF subsequently generates the two paths $(D_0, C_0, B_0, A_1, E_0)$ and $(D_0, C_0, B_0, A_1, E_1)$ and accumulates the next partial value $h_1(A_1B_0) = P(E_0|A_1B_0)+P(E_1|A_1B_0)$. Subsequently it computes the summation $h_2(B_0C_0) = P(A_0)\cdot P(B_0|A_0)\cdot P(C_0|A_0)\cdot h_1(A_0B_0)+P(A_1)\cdot P(B_0|A_1) \cdot P(C_0|A_1) \cdot h_1(A_1B_0)$. Notice that due to caching each arc is generated and traversed just once (in each direction). For example when the partial path (D_1, C_0, B_0) is created, it is recognized (via context) that the subtree below was already explored and its compiled value will be reused.

In contrast, VE generates the full context-minimal graph by layers, from the *bottom of the search graph up*, in a manner that can be viewed as dynamic programming or as breadth-first search on the explicated search graph. VE's execution can be viewed as first generating all the edges between E and A (in some order), and then all the edges between A and B (in some order), and so on up to the top. We can see that there are 8 edges between E and A. They correspond to the 8 tuples in the bucket of E (the function on (ABE)). There are 8 edges between A and B, corresponding to the 8 tuples in the bucket of A. And there are 8 edges between B and C, corresponding to the 8 tuples in the bucket of B. Similarly, 4 edges between C and D correspond to the 4 tuples in the bucket of C, and 2 edges between D and the rood correspond to the 2 tuples in the bucket of D.

Since the computation is performed from bottom to top, the nodes of A store the result of *eliminating* E (namely the function $h_1(AB)$ resulting by summing out E). There are 4 nodes labeled with A, corresponding to the 4 tuples in the message sent by VE from bucket of E to bucket of A (the message on (AB)). And so on, each level of nodes corresponds to the number of tuples in the message sent on the separator (the common variables) between two buckets.

AND/OR Search Spaces

The above correspondence between **AO** and **VE** is also maintained in non-chain pseudo/bucket trees, as is demonstrated next. We refer again to the example in Figures 3.12, and assume the task is belief updating (the CPTs are $P(A), P(B|A), P(C|A), P(D|B,C), P(E|A,B)$. The bucket tree in Figure 7.3(c) has the same structure as the pseudo tree in Figure 3.12(a). We will show that VE traverses the AND/OR search graph in Figure 3.12(b) bottom up, while AO-DF traverses the same graph in depth first manner, top down.

AO-DF first sums $h_3(A_0, B_0) = P(E_0|A_0, B_0) + P(E_1|A_0, B_0)$ and then goes depth first to $h_1(B_0, C_0) = P(D_0|B_0, C_0) + P(D_1|B_0, C_0)$ and $h_1(B_0, C_1) = P(D_0|B_0, C_1) +$

$P(D_1|B_0, C_1)$. Then it computes $h_2(A_0, B_0) = (P(C_0|A_0) \cdot h_1(B_0, C_0)) + (P(C_1|A_0) \cdot h_1(B_0, C_1))$. All the computation of AO-DF is precisely the same as the one performed in the buckets of VE. Namely, h_1 is computed in the bucket of D and placed in the bucket of C. h_2 is computed in the bucket of C and placed in the bucket of B, h_3 is computed in the bucket of E and also placed in the bucket of B and so on, as shown in Figure 7.3b. All this corresponds to traversing the AND/OR graph from leaves to root. Thus, both algorithms traverse the same graph, only the control strategy is different.

We can generalize both the OR and AND/OR examples,

THEOREM **6.2.1 (VE and AO-DF are identical)** *Given a graphical model having no determinism, and given the same bucket/pseudo tree VE applied to the bucket-tree is a (breadth-first) bottom-up search that will explore all the full CM search graph, while AO-DF is a depth-first top-down search that explores (and records) the full CM graph as well.*

Breadth-first on AND/OR Since one important difference between AO search and VE is the order by which they explore the search space (top-down vs. bottom-up) we wish to remove this distinction and consider a VE-like algorithm that goes top-down. One obvious choice is breadth-first search, yielding AO-BF. That is, in Figure 3.11 we can process the layer of variable A first, then B, then E and C, and then D. General *breadth-first* or *best-first* search of AND/OR graphs for computing the optimal cost solution subtrees are well defined procedures. The process involves expanding all solution subtrees in layers of depth. Whenever a new node is generated and added to the search frontier the value of all relevant partial solution subtrees are updated. A well known Best-first version of AND/OR spaces is the AO* algorithm [85]. Algorithm AO-BF can be viewed as a top-down inference algorithm. We can now extend the comparison to AO-BF.

Proposition 24 (VE and AO-BF are identical) *Given a graphical model with no determinism and a bucket/pseudo tree, VE and AO-BF explore the same full CM graph, one* *bottom-up (VE) and the other top-down; both perform identical value computation.*

Terminology for the comparison of algorithms Let A and B be two algorithms over graphical models, whose performance is determined by an underlying bucket/pseudo tree.

DEFINITION **6.2.1 (comparison of algorithms)** *We say that: 1. algorithms A* and *B* are *identical if for every graphical model and when given the same bucket-tree they traverse an identical search space. Namely, every node is explored by A iff it is explored by B; 2. A* is weakly better than *B if there exists a graphical model and a bucket-tree, for which A explores a strict subset of the nodes explored by B; 3. A* is better than *B if A is weakly better than B but B is not weakly better than A; 4. The relation of "weakly-better" defines a partial order between algorithms. A and B are* not comparable *if they are not comparable w.r.t to the "weakly-better" partial order.*

Clearly, any two algorithms for graphical models are either 1. identical, 2. one is better than the other, or 3. they are not comparable. We can now summarize our observations so far using the new terminology.

THEOREM **6.2.2** *For a graphical model having no determinism AO-DF, AO-BF and VE are identical.*

Note that our terminology captures the time complexity but may not capture the space complexity, as we show next.

Space Complexity

To make the complete correspondence between VE and AO search, we can look not only at the computational effort, but also at the space required. Both VE and AO search traverse the context minimal graph, but they may require different amounts of memory to do so. So, we can distinguish between the portion of the graph that is traversed and the portion that should be recorded and maintained. If the whole graph is recorded, then the space is $O(n \cdot \exp(w^*))$, which we will call the base case.

VE can forget layers Sometimes, the task to be solved can allow VE to use less space by deallocating the memory for messages that are not necessary anymore. Forgetting previously traversed layers of the graph is a well known property of dynamic programming. In such a case, the space complexity for VE becomes $O(d_{BT} \cdot \exp(w*))$, where d_{BT} is the *depth* of the bucket tree (assuming constant degree in the bucket tree). In most cases, the above bound is not tight. If the bucket tree is a chain, then $d_{BT} = n$, but forgetting layers yields an $O(n)$ improvement over the base case. AO-DF cannot take advantage of this property of VE. It is easy to construct examples where the bucket tree is a chain, for which VE requires $O(n)$ less space than AO-DF.

AO dead caches The straightforward way of caching is to have a table for each variable, recording its context. However, some tables might never get cache hits. We call these *dead-caches*. In the AND/OR search graph, dead-caches appear at nodes that have only one incoming arc. AO search needs to record only nodes that are likely to have additional incoming arcs, and these nodes can be determined by inspection from the pseudo tree. Namely, if the context of a node includes that of its parent, then AO need not store anything for that node, because it would be a dead-cache.

In some cases, VE can also take advantage of dead caches. If the dead caches appear along a chain in the pseudo tree, then avoiding the storage of dead-caches in AO corresponds to collapsing the subsumed neighboring buckets in the bucket tree (remember that the computation within a bucket can be done with linear space algorithms, while space reflects the message size that needs to be communicated. Thus if the clique sizes are bounded by r and the separators by s, time is $exp(r)$ while space is $exp(s)$). This results in having cache tables of the size of the separators, rather than the cliques. The time savings are within a constant factor from the complexity of solving the largest clique, but the space complexity can be reduced from exponential in the size of the maximal cique to exponential in the maximal separator.

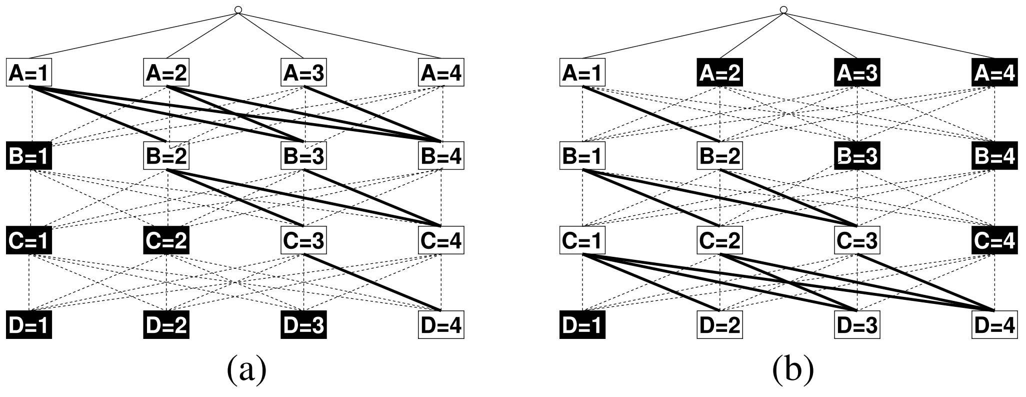


Figure 6.3: CM graphs with determinism: a) AO; b) VE

However, if the variables having dead caches form connected components that are subtrees (rather than chains) in the pseudo tree, then the space savings of AO cannot be achieved by VE. Consider the following example:

Example 6.2.3 *Let variables $\{X_1, \ldots, X_n\}$ be divided in three sets: $\mathcal{A} = \{X_1, \ldots, X_{\frac{n}{3}}\}$, $\mathcal{B} = \{X_{\frac{n}{3}+1}, \ldots, X_{\frac{2n}{3}}\}$ and $\mathcal{C} = \{X_{\frac{2n}{3}+1}, \ldots, X_n\}$. There are two cliques on $\mathcal{A} \cup \mathcal{B}$ and $\mathcal{A} \cup \mathcal{C}$ defined by all possibile binary functions over variables in those respective cliques. The input is therefore $O(n^2)$. Consider the bucket tree (pseudo tree) defined by the ordering $d = (X_1, \ldots, X_n)$, where X_n is eliminated first by VE. In this pseudo tree, all the caches are dead, and as a result the AO search graph coincides with the AO search tree. Therefore, AO can solve the problem using space $O(\frac{2n}{3})$. VE can collapse some neighboring buckets (for variables in $\mathcal{B}$ and $\mathcal{C}$), but needs to store at least one message on the variables in $\mathcal{A}$, which yields space complexity $O(\exp(\frac{n}{3}))$. In this example, AO and VE have the same time complexity, but AO uses space linear in the number of variables while VE needs space exponential in the number of variables (and exponential in the input too).*

The above observation is similar to the known properties of depth-first vs. breadth-first search in general. When the search space is close to a tree, the benefit from the inherent memory use of breadth-first search is nil.

VE vs. AND/OR Search with Determinism

When the graphical model contains determinism the AND/OR trees and graphs are dependant not only on the primal graph but also on the (flat) constraints, namely on the consistency and inconsistency of certain relationships (no-good tuples) in each relation. In such cases AO and VE, may explore different portions of the context-minimal graphs because the order of variables plays a central role, dictating where the determinism reveals itself.

Example 6.2.4 *Let's consider a problem on four variables: A, B, C, D, each having the domain $\{1, 2, 3, 4\}$, and the constraints $A < B$, $B < C$ and $C < D$. The primal graph of the problem is a chain. Let's consider the natural ordering from A to D, which gives rise to a chain pseudo tree (and bucket-tree) rooted at A. Figure 6.3a shows the full CM graph with determinism generated by AO search, and Figure 6.3b the graph generated and traversed by VE in reverse order. The thick lines and the white nodes are the ones traversed. The dotted lines and the black nodes are not explored (when VE is executed from D, the constraint between D and C implies that $C = 4$ is pruned, and therefore not further explored). Note that the intersection of the graphs explored by both algorithms is the* backtrack-free AND/OR context graph, *corresponding to the unique solution (A=1,B=2,C=3,D=4).*

As we saw in the example, AO and VE explore different parts of the inconsistent portion of the full CM. Therefore, in the presence of determinism, AO-DF and AO-BF are both un-comparable to VE, as they differ in the direction they explore the CM space.

THEOREM **6.2.5** *Given a graphical model with determinism, then AO-DF and AO-BF are identical and both are un-comparable to VE.*

This observation is in contrast with claims of superiority of one scheme or the other [5], at least for the case when variable ordering is fixed and no advanced constraint propagation schemes are used and assuming no exploitation of context independence.

6.2.2 Algorithmic Advances and Their Effect

So far we compared brute-force VE to brute-force AO search. We will now consider the impact of some enhancements on this relationship. Clearly, both VE and AO explore the portion of the context-minimal graph that is backtrack-free. Thus they can differ only on the portion that is included in full CM and not in the backtrack-free CM. Indeed, constraint propagation, backjumping and no-good recording just reduce the exploration of that portion of the graph that is *inconsistent*. Here we compare those schemes against bare VE and against VE augmented with similar enhancements whenever relevant.

VE vs. AND/OR Search with Look-Ahead

In the presence of determinism AO-DF and AO-BF can naturally accommodate look-ahead schemes which may avoid parts of the context-minimal search graph using some level of constraint propagation. It is easy to compare AO-BF against AO-DF when both use the same look-ahead because the notion of constraint propagation as look-ahead is well defined for search and because both algorithms explore the search space top down. Not surprisingly when both algorithms have the same level of look-ahead propagation, they explore an identical search space.

We can also augment VE with look-ahead constraint propagation (e.g., unit resolution, arc consistency), yielding VE-LAH as follows. Once VE-LAH processes a single bucket, it then applies constraint propagation as dictated by the look-ahead propagation scheme (bottom-up), then continues with the next bucket applied over the modified set of functions and so on. We can show that:

THEOREM **6.2.6** *Given a graphical model with determinism and given a look-ahead propagation scheme, LAH,*

1. AO-DF-LAH and AO-BF-LAH are identical.

2. VE and VE-LAH are each un-comparable with each of AO-DF-LAH and AO-BF-LAH.

Proof. 1. The search graph is traversed in the same direction by both AO-DF-LAH and AO-BF-LAH, so the look-ahead has the same effect for both. 2. Determinism can still impact differently in different variable orderings. □

Graph-Based No-Good Learning

AO search can be augmented with no-good learning [31]. Graph-based no-good learning means recording that some nodes are inconsistent based on their context. This is automatically accomplished when we explore the CM graph which actually amounts to recording no-goods and goods by their contexts. Therefore AO-DF is identical to AO-BF and both already exploit no-goods, we get that (AO-NG denotes AO with graph-based no-good learning):

THEOREM **6.2.7** *For every graphical model the relationship between AO-NG and VE is the same as the relationship between AO (Depth-first or breadth-first) and VE.*

Combined no-goods and look-ahead No-goods that are generated during search can also participate in the constraint propagation of the look-ahead and strengthen the ability to prune the search-space further. The graphical model itself is modified during search and this affects the rest of the look-ahead. It is interesting to note that AO-BF is not able to simulate the same pruned search space as AO-DF in this case because of its breadth-first manner. While AO-DF can discover deep no-goods due to its depth-first nature, AO-BF has no access to such deep no-goods and cannot use them within a constraint propagation scheme in shallower levels. However, even when AO exploits no-goods within its look-ahead propagation scheme, VE and AO remain un-comparable. Any example that does not allow effective no-good learning can illustrate this.

Example 6.2.8 *Consider a constraint problem over n variables. Variables $X_1, \ldots, X_{n-1}$ have the domain $\{1, 2, \ldots, n-2, *\}$, made of n-2 integer values and a special $*$ value. Between any pair of the $n-1$ variables there is a not-equal constraint between the integers*

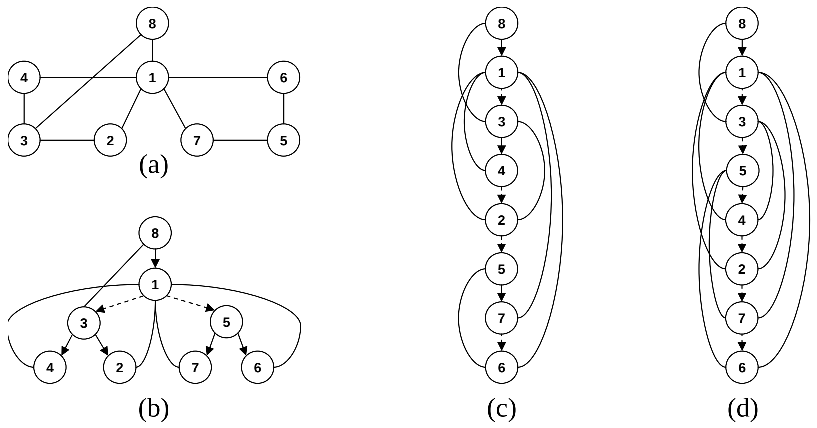


Figure 6.4: GBJ vs. AND/OR search

and equality between stars. There is an additional variable X_n which has a constraint with each variable, whose values are consistent only with the $*$ of the other n-1 variables. Clearly if the ordering is $d = (X_1, \ldots, X_{n-1}, X_n)$, AO may search all the exponential search space over the first $n-1$ variables (the inconsistent portion) before it reaches the $*$ of the $n-th$ variable. On the other hand, if we apply VE starting from the $n-th$ variable, we will reveal the only solution immediately. No constraint propagation, nor no-good learning can help any AO search in this case.

THEOREM **6.2.9** *Given a graphical model with determinism and a particular look-ahead propagation scheme LAH:*
1. AO-DF-LAH-NG is better than AO-BF-LAH-NG.
2. VE and AO-DF-LAH-NG are not comparable.

Graph-Based Backjumping

Backjumping algorithms [31] are backtracking search applied to the OR space, which uses the problem structure to jump back from a dead-end as far back as possible. In *graph-based backjumping* (GBJ) each variable maintains a graph-based induced ancestor set which ensures that no solutions are missed by jumping back to its deepest variable.

DFS orderings If the ordering of the OR space is a DFS ordering of the primal graph, it is known [31] that all the backjumps are from a variable to its DFS parent. This means that *naive AO-DF* automatically incorporates GBJ jumping-back character.

Pseudo tree orderings In the case of pseudo tree orderings that are not DFS-trees, there is a slight difference between OR-GBJ and AO-DF and GBJ may sometime perform deeper backjumps than those implicitly done by AO. Figure 6.4a shows a probabilistic model, 6.4b a pseudo tree and 6.4c a chain driving the OR search (top down). If a deadend is encountered at variable 3, GBJ retreats to 8 (see 6.4c), while naive AO-DF retreats to 1, the pseudo tree parent. When the deadend is encountered at 2, both algorithms backtrack to 3 and then to 1. Therefore, in such cases, augmenting AO with GBJ can provide additional pruning on top of the AND/OR structure. In other words, GBJ on OR space along a pseudo tree is never stronger than GBJ on AND/OR and it is sometimes weaker.

GBJ can be applied using an arbitrary order d for the OR space. The ordering d can be used to generate a pseudo tree. In this case, however, to mimic GBJ on d, the AO traversal will be controlled by d. In Figure 6.4d we show an arbitrary order $d = (8, 1, 3, 5, 4, 2, 7, 6)$ which generates the pseudo tree in 6.4b. When AO search reaches 3, it goes in a breadth first manner to 5, according to d. It can be shown that GBJ in order d on OR space corresponds to the GBJ-based AND/OR search based on the associated pseudo tree. All the backjumps have a one to one correspondence.

Since VE is not comparable with AO-DF, it is also un-comparable with AO-DF-GBJ. Note that backjumping is not relevant to AO-BF or VE. In summary,

THEOREM **6.2.10** *1. When the pseudo tree is a DFS tree AO-DF is identical to AO-DF-GBJ. This is also true when the AND/OR search* **tree** *is explored (rather than the CM-***graph***). 2. AO-DF-GBJ is superior to AO-DF for general pseudo trees. 3. VE is not comparable to AO-DF-GBJ.*

Proof. 1. For DFS trees, backjumps go to DFS parent. 2. See example in Figure 6.4b. 3.

Determinism reveals itself differently in reversed orderings. □

6.2.3 Discussion

In this section we compare search and inference in graphical models through the new framework of AND/OR search spaces. We show that there is no principled difference between memory-intensive search with fixed variable ordering and inference beyond: (1) different direction of exploring a common search space (top down for search vs. bottom-up for inference); (2) different assumption of control strategy (depth-first for search and breadth-first for inference). We also show that those differences occur only in the presence of determinism. We show the relationship between algorithms such as graph-based backjumping and no-good learning [31] within the AND/OR search space. AND/OR search spaces can also accommodate dynamic variable and value ordering which can affect algorithmic efficiency significantly. Variable Elimination and general inference methods however require static variable ordering. This issue will be addressed in future work.

6.3 A Comparison of Hybrid Time-Space Schemes

6.3.1 Defining the Algorithms

In this section we describe the three algorithms that will be compared. They are all parameterized memory intensive algorithms that need to use space in order to achieve the worst case time complexity of $O(n\ k^{w^*})$, where k bounds domain size, and w^* is the treewidth of the primal graph. The task that we consider is one that is #P-hard (e.g., belief updating in Bayesian networks, counting solutions in SAT or constraint networks). We also assume that the model has no determinism (i.e., all tuples have a strictly positive probability).

The algorithms we discuss work by processing variables either by *elimination* or by *conditioning*. These operations have an impact on the primal graph of the problem. When a variable is eliminated, it is removed from the graph along with its incident edges, and its neighbors are connected in a clique. When it is conditioned, it is simply removed from the graph along with its incident edges.

The algorithms we discuss typically depend on a variable ordering $d = (X_1, ..., X_n)$. Search proceeds by instantiating variables from X_1 to X_n, while Variable Elimination processes the variables backwards, from X_n to X_1. Given a graph G and an ordering d, an elimination tree, denoted by $\mathcal{T}(G, d)$, is uniquely defined by the Variable Elimination process. $\mathcal{T}(G, d)$ is also a valid pseudo tree to drive the AND/OR search. Note however that several orderings can give rise to the same elimination tree.

AND/OR Cutset Conditioning - AOCutset(i)

AND/OR Cutset Conditioning (**AOCutset(i)**) [75] is a search algorithm that combines AND/OR search spaces with cutset conditioning. The conditioning (cutset) variables form a *start* pseudo tree. The remaining variables (not belonging to the cutset), have bounded conditioned context size that can fit in memory.

DEFINITION **6.3.1 (start pseudo tree)** *Given a primal graph G and a pseudo tree $\mathcal{T}$ of G, a* start pseudo tree *$\mathcal{T}_{start}$ is a connected subgraph of $\mathcal{T}$ that contains the root of $\mathcal{T}$.*

Algorithm **AOCutset(i)** depends on a parameter i that bounds the maximum size of a context that can fit in memory. Given a graphical model and a pseudo tree $\mathcal{T}$, we first find a start pseudo tree $\mathcal{T}_{start}$ such that the context of any node not in $\mathcal{T}_{start}$ contains at most i variables that are not in $\mathcal{T}_{start}$. This can be done by starting with the root of $\mathcal{T}$ and then including as many descendants as necessary in the start pseudo tree until the previous condition is met. $\mathcal{T}_{start}$ now forms the cutset, and when its variables are instantiated, the remaining conditioned subproblem has induced width bounded by i. The cutset variables can be explored by linear space (no caching) AND/OR search, and the remaining variables by using full caching, of size bounded by i. The cache tables need to be deleted and reallocated for each new conditioned subproblem (i.e., each new instantiation of the cutset variables).

Algorithm AOC(i) - Adaptive Caching

The cutset principle inspires a new algorithm, based on a more refined caching scheme for AND/OR search, which we call *Adaptive Caching* - **AOC(i)** (in the sense that it adapts to the available memory), that caches some values even at nodes with contexts greater than the bound i that defines the memory limit. Lets assume that $context(X) = [X_1 \dots X_k]$ and $k > i$. During search, when variables $X_1, \dots, X_{k-i}$ are instantiated, they can be regarded as part of a cutset. The problem rooted by X_{k-i+1} can be solved in isolation, like a subproblem in the cutset scheme, after variables $X_1, \dots, X_{k-i}$ are assigned their current values in all the functions. In this subproblem, $context(X) = [X_{k-i+1} \dots X_k]$, so it can be cached within space bounded by i. However, when the search retracts to X_{k-i} or above, the cache table for X needs to be deleted and will be reallocated when a new subproblem rooted at X_{k-i+1} is solved.

DEFINITION **6.3.2 (i-context, flag)** *Given a graphical model, a pseudo tree $\mathcal{T}$, a variable X and $context(X) = [X_1 \dots X_k]$, the* i-context *of X is:*

$$i\text{-}context(X) = \begin{cases} [X_{k-i+1} \dots X_k], & if \quad i < k \\ context(X), & if \quad i \geq k \end{cases}$$

*X_i is called the **flag** of $i\text{-}context(X)$.*

The high level pseudocode for **AOC(i)** is given here. The algorithm works similar to AND/OR search based on full context. The difference is in the management of cache tables. Whenever a variable X is instantiated (when an AND node is reached), the cache table is purged (reinitialized with a neutral value) for any variable Y such that X is the flag of $i\text{-}context(Y)$ (line 6). Otherwise, the search proceeds as usual, retrieving values

```
Algorithm AOC(i)
    input  : $\mathcal{M}=\langle \mathbf{X},\mathbf{D},\mathbf{F}\rangle$; $G=(\mathbf{X},E)$; $d=(X_1,\ldots,X_n)$; $i$
    output : Updated belief for $X_1$
1   Let $\mathcal{T}$ = $\mathcal{T}(G,d)$        // create elimination tree for each $X \in \mathbf{X}$ do
2       allocate a table for $i$-$context(X)$
3   Initialize search with root of $\mathcal{T}$;
4   while search not finished do
5       Pick next successor not yet visited                       // EXPAND;
6       Purge cache tables that are not valid;
7       if value in cache then
8           retrieve value; mark successors as visited;
9       while all successors visited do                       // PROPAGATE
10          Save value in cache;
11          Propagate value to parent;
```

from cache if possible (line 8) or else continuing to expand, and propagating the values up when the search is completed for subproblem below (line 11). We do not detail here the alternation of OR and AND type of nodes.

Example 6.3.1 *We will clarify here the distinction between AND/OR with full caching, AND/OR Cutset and AND/OR Adaptive Caching. We should note that the scope of a cache table is always a subset of the variables on the current path in the pseudo tree. Therefore, the caching method (e.g., full caching based on context, cutset conditioning cache, adaptive caching) is an orthogonal issue to that of the search space decomposition. We will show an example based on an OR search space (pseudo tree is a chain), and the results will carry over to the AND/OR search space.*

Figure 6.5 shows a pseudo tree, with binary valued variables, the context for each variable, and the context minimal graph. If we assume the bound $i = 2$, some of the cache tables don't fit in memory. We could in this case use ***AOCutset(2)****, shown in Figure 6.7, that takes more time, but can execute in the bounded memory. The cutset in this case is made of variables A and B, and we see four conditioned subproblems, the four columns, that are solved independently from one another (there is no sharing of subgraphs). Figure 6.6*

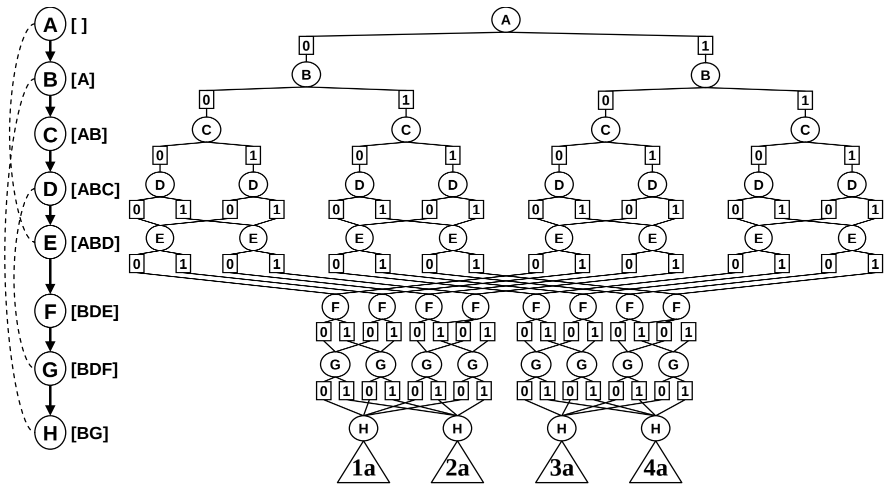


Figure 6.5: Context minimal graph (full caching)

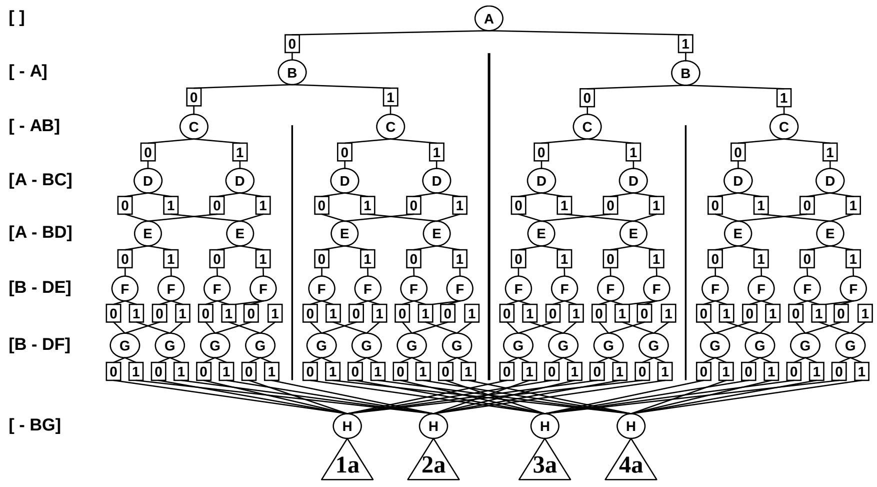


Figure 6.6: **AOC(2)** graph (Adaptive Caching)

*shows **AOC(2)**, which falls between the previous two. It uses bounded memory, takes more time than full caching (as expected), but less time than **AOCutset(2)** (because the graph is smaller). This can be achieved because Adaptive Caching allows the sharing of subgraphs. Note that the cache table of H has the scope $[BG]$, which allows merging.*

Variable Elimination and Conditioning - VEC(i)

Variable Elimination and Conditioning (**VEC**) [90, 63] is an algorithm that combines the virtues of both inference and search. One of its remarkably successful applications is the

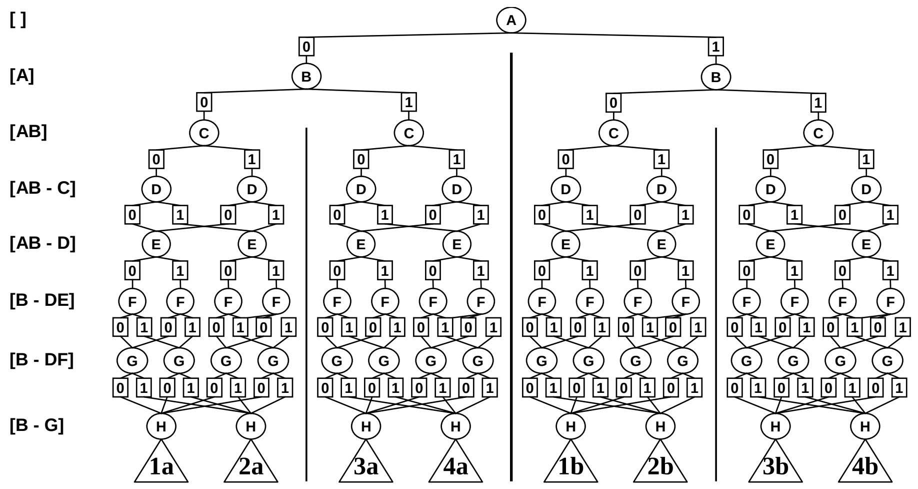


Figure 6.7: **AOCutset(2)** graph (AND/OR Cutset)

```
Algorithm VEC-OR(i)
   input  : $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$; $d = (X_1, \ldots, X_n)$
   output: Updated belief for $X_1$
1  if ($context(X_n) \leq i$) then
2      eliminate $X_i$;
3      call VEC-OR(i) on reduced problem
4  else for each $x_n \in D_n$ do
5      assign $X_n = x_n$;
6      call VEC-OR(i) on the conditioned subproblem
```

genetic linkage analysis software Superlink [47]. **VEC** works by interleaving elimination and conditioning of variables. Typically, given an ordering, it prefers the elimination of a variable whenever possible, and switches to conditioning whenever space limitations require it, and continues in the same manner until all variables have been processed. We say that the conditioning variables form a *conditioning set*, or *cutset* (this can be regarded as a w-*cutset*, where the w defines the induced width of the problems that can be handled by elimination). The pseudocode for the vanilla version, called **VEC-OR(i)** because the cutset is explored by OR search rather than AND/OR, is shown below:

When there are no conditioning variables, **VEC** becomes the well known Variable Elimination (**VE**) algorithm. In this case **AOC** also becomes the usual AND/OR graph search (**AO**), and it was shown in Theorem 6.2.1 that **VE** and **AO** are identical.

Tree Decomposition with Conditioning - TDC

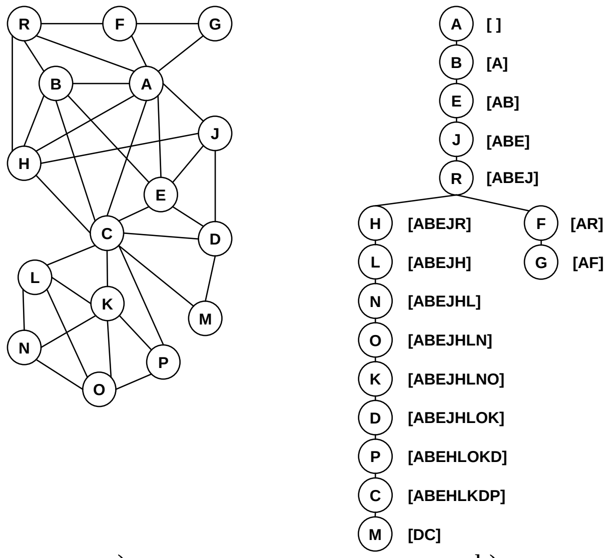
One of the most widely used methods of processing graphical models, especially belief networks, is tree clustering (also known as join tree or junction tree algorithm [66]). The work in [32] presents an algorithm called *directional join tree clustering*, that corresponds to an inward pass of messages towards a root in the regular tree clustering algorithm. If space is not sufficient for the separators in the tree decomposition, then [32] proposes the use of secondary join trees, which simply combine any two neighboring clusters whose separator is too big to be stored. The resulting algorithm that uses less memory at the expense of more time is called *space based join tree clustering*,

The computation in each cluster can be done by any suitable method. The obvious one would be to simply enumerate all the instantiations of the cluster, which corresponds to an OR search over the variables of the cluster. A more advanced method advocated by [32] is the use of cycle cutset inside each cluster. We can improve the cycle cutset scheme first by using an AND/OR search space, and second by using Adaptive Caching bounded by i, rather than simple AND/OR Cutset in each cluster. We call the resulting method *tree decomposition with conditioning* (**TDC(i)**).

6.3.2 AOC(i) Compared to VEC(i)

We will begin by following an example. Consider the graphical model given in Figure 6.8a having binary variables, the ordering $d_1 = (A, B, E, J, R, H, L, N, O, K, D, P, C, M, F-, G)$, and the space limitation $i = 2$. The pseudo tree corresponding to this ordering is given in Figure 6.8b. The context of each node is shown in square brackets.

If we apply **VEC** along d_1 (eliminating from last to first), variables G, F and M can be eliminated. However, C cannot be eliminated, because it would produce a function with scope equal to its context, $[ABEHLKDP]$, violating the bound $i = 2$. **VEC** switches to conditioning on C and all the functions that remain to be processed are modified accordingly, by instantiating C. The primal graph has two connected components now, shown



a) b)

Figure 6.8: Primal graph and pseudo tree

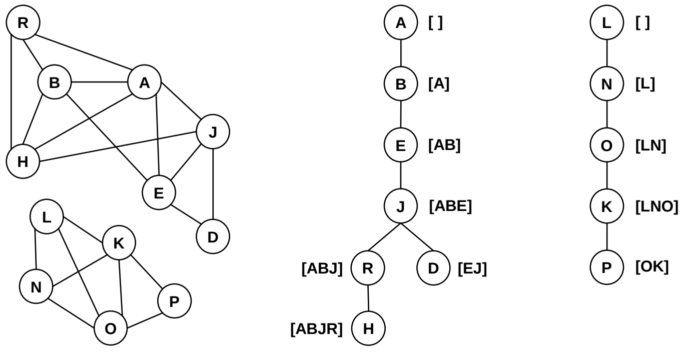


Figure 6.9: Components after conditioning on C

in Figure 6.9. Notice that the pseudo trees are based on this new graph, and their shape changes from the original pseudo tree.

Continuing with the ordering, P and D can be eliminated (one variable from each component), but then K cannot be eliminated. After conditioning on K, variables O, N and L can be eliminated (all from the same component), then H is conditioned (from the other component) and the rest of the variables are eliminated. To highlight the conditioning set, we will box its variables when writing the ordering, $d_1 = (A, B, E, J, R, \boxed{H}, L, N, O$-$, \boxed{K}, D, P, \boxed{C}, M, F, G)$.

If we take the conditioning set $[HKC]$ in the order imposed on it by d_1, reverse it and

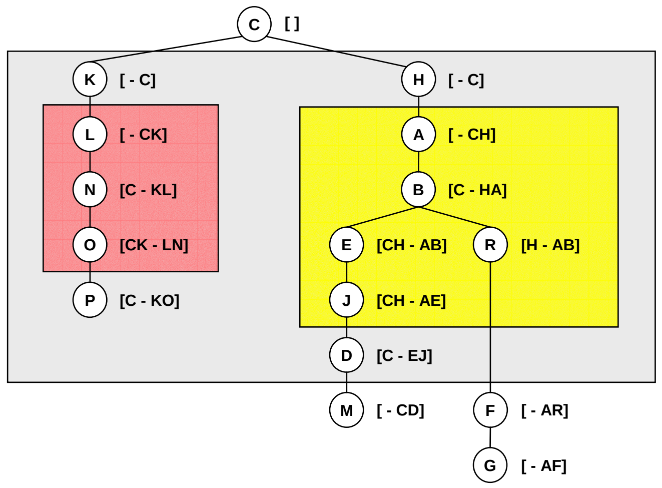


Figure 6.10: Pseudo tree for **AOC(2)**

put it at the beginning of the ordering d_1, then we obtain:

$$d_2 = \left(\boxed{C}, \underline{\left[\boxed{K}, \underline{\left[\boxed{H}, \underline{[A,B,E,J,R]}_H, L,N,O\right]}_K, D,P\right]}_C, M,F,G\right)$$

where the indexed squared brackets together with the underlines represent subproblems that need to be solved multiple times, for each instantiation of the index variable.

So we started with d_1 and bound $i = 2$, then we identified the corresponding conditioning set $[HKC]$ for **VEC**, and from this we arrived at d_2. We are now going to use d_2 to build the pseudo tree that guides **AOC(2)**, given in Figure 6.10. The outer box corresponds to the conditioning of C. The inner boxes correspond to conditioning on K and H, respectively. The context of each node is given in square brackets, and the *2-context* is on the right side of the dash. For example, $context(J) = [CH\text{-}AE]$, and 2-$context(J) = [AE]$. The context minimal graph corresponding to the execution of **AOC(2)** is shown in Figure 6.11.

We can follow the execution of both **AOC** and **VEC** along this context minimal graph. After conditioning on C, **VEC** solves two subproblems (one for each value of C), which are the ones shown on the large rectangles. The vanilla version **VEC-OR** is less efficient than **AOC**, because it uses an OR search over the cutset variables, rather than AND/OR. In

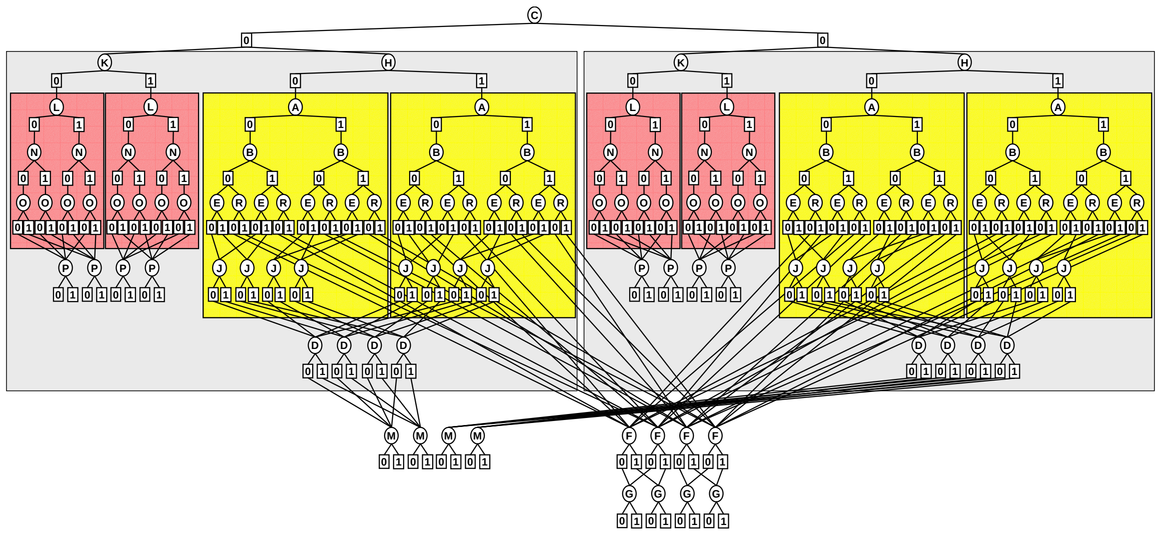


Figure 6.11: Context minimal graph

our example, the subproblem on A, B, E, J, R would be solved eight times by **VEC-OR**, once for each instantiation of C, K and H, rather than four times. It is now easy to make the first improvement to **VEC**, so that it uses an AND/OR search over the conditioning variables, an algorithm we call **VEC-AO(i)**, by changing line 6 of **VEC-OR** to:

```
Algorithm VEC-AO(i)

    ...
6   call VEC-AO(i) on each connected component of conditioned
    subproblem separately;
```

Let's look at one more condition that needs to be satisfied for the two algorithms to be identical. If we change the ordering to $d_3 = (A, B, E, J, R, \boxed{H}, L, N, O, \boxed{K}, D, P, F\text{-}, G, \boxed{C}, M)$, ($F$ and G are eliminated after conditioning on C), then the pseudo tree is the same as before, and the context minimal graph for **AOC** is still the one shown in Figure 6.11. However, **VEC-AO** would require more effort, because the elimination of G and F is done twice now (once for each instantiation of C), rather than once as was for ordering d_1. This shortcoming can be eliminated by defining a pseudo tree based version for **VEC**, rather than one based on an ordering. The final algorithm, **VEC(i)** is given below (where $N_G(X_i)$ is the set of neighbors of X_i in the graph G). Note that the guiding pseudo tree is regenerated after each conditioning.

Algorithm **VEC(i)**

```
input  : $\mathcal{M}=\langle \mathbf{X},\mathbf{D},\mathbf{F}\rangle$; $G=(\mathbf{X},E)$; $d=(X_1,\ldots,X_n)$; $i$
output : Updated belief for $X_1$
1 Let $\mathcal{T}$ = $\mathcal{T}(G,d)$                    // create elimination tree;
2 while $\mathcal{T}$ not empty do
3     if (($\exists X_i$ leaf in $\mathcal{T}$)$\wedge$($|N_G(X_i)|\le i$)) then eliminate $X_i$ else pick $X_i$ leaf from $\mathcal{T}$;
4     for each $x_i \in D_i$ do
5         assign $X_i = x_i$;
6         call VEC(i) on each connected component of conditioned subproblem
7     break;
```

Based on the previous example, we can prove:

THEOREM **6.3.2 (AOC(i) simulates VEC(i))** *Given a graphical model $\mathcal{M} = \langle \mathbf{X},\mathbf{D},\mathbf{F}\rangle$ with no determinism and an execution of* ***VEC(i)****, there exists a pseudo tree that guides an execution of* ***AOC(i)*** *that traverses the same context minimal graph.*

Proof. The pseudo tree of **AOC(i)** is obtained by reversing the conditioning set of **VEC(i)** and placing it at the beginning of the ordering. The proof is by induction on the number of conditioning variables, by comparing the corresponding contexts of each variable.

Basis step. If there is no conditioning variable, Theorem 6.2.1 applies. If there is only one conditioning variable. Given the ordering $d = (X_1,\ldots,X_j,\ldots,X_n)$, let's say X_j is the conditioning variable.

a) Consider $X \in \{X_{j+1},\ldots,X_n\}$. The function recorded by **VEC(i)** when eliminating X has the scope equal to the context of X in **AOC(i)**.

b) For X_j, both **VEC(i)** and **AOC(i)** will enumerate its domain, thus making the same effort.

c) After X_j is instantiated by **VEC(i)**, the reduced subproblem (which may contain

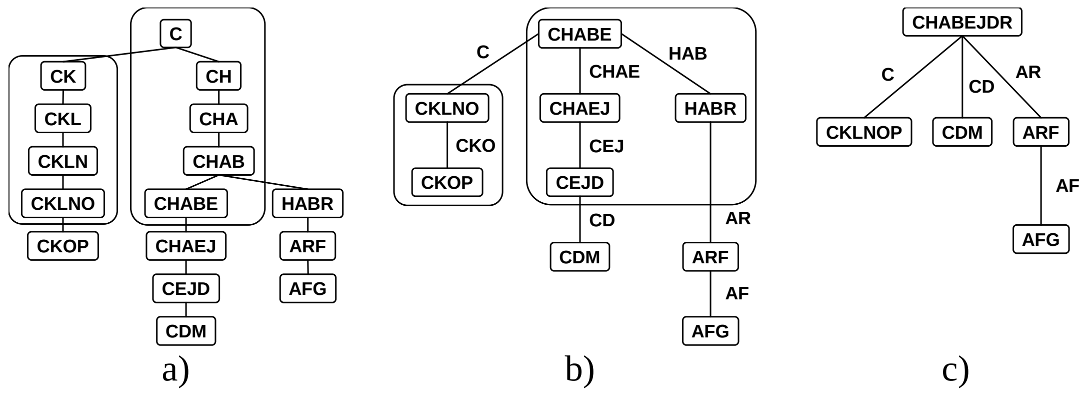


Figure 6.12: Tree decompositions: a) for d_2; b) maximal cliques only; c) secondary tree for $i = 2$

multiple connected components) can be solved by variable elimination alone. By Theorem 6.2.1, variable elimination on this portion is identical to AND/OR search with full caching, which is exactly **VEC(i)** on the reduced subproblem.

From a), b) and c) it follows that **VEC(i)** and **AOC(i)** are identical if there is only one conditioning variable.

Inductive step. We assume that **VEC(i)** and **AOC(i)** are identical for any graphical model if there are at most k conditioning variables, and have to prove that the same is true for $k + 1$ conditioning variables.

If the ordering is $d = (X_1, \ldots, X_j, \ldots, X_n)$ and X_j is the last conditioning variable in the ordering, it follows (similar to the basis step) that **VEC(i)** and **AOC(i)** traverse the same search space with respect to variables $\{X_{j+1}, \ldots, X_n\}$, and also for X_j. The remaining conditioned subproblem now falls under the inductive hypothesis, which concludes the proof. Note that it is essential that **VEC(i)** uses AND/OR over cutset, and is pseudo tree based, otherwise **AOC(i)** is better. $\square$

6.3.3 AOC(i) Compared to TDC(i)

We will look again at the example from Figures 6.10 and 6.11, and the ordering d_2. It is well known that a tree decomposition corresponding to d_2 can be obtained by inducing the graph along d_2 (from last to first), and then picking as clusters each node together with its parents in the induced graph, and connecting each cluster to that of its latest (in the ordering) induced parent. Because the induced parent set is equal to the context of a node, the method above is equivalent to creating a cluster for each node in the pseudo tree from Figure 6.10, and labeling it with the variable and its context. The result is shown in Figure 6.12a. A better way to build a tree decomposition is to pick only the maximal cliques in the induced graph, and this is equivalent to collapsing neighboring subsumed clusters from Figure 6.12a, resulting in the tree decomposition in Figure 6.12b. If we want to run **TDC** with bound $i = 2$, some of the separators are bigger than 2, so a secondary tree is obtained by merging clusters adjacent to large separators, obtaining the tree in Figure 6.12c. **TDC(2)** now runs by sending messages upwards, toward the root. Its execution, when augmented with AND/OR cutset in each cluster, can also be followed on the context minimal graph in Figure 6.11. The separators $[AF]$, $[AR]$ and $[CD]$ correspond to the contexts of G, F and M. The root cluster $[CHABEJDR]$ corresponds to the part of the context minimal graph that contains all these variables. If this cluster would be processed by enumeration (OR search), it would result in a tree with $2^8 = 256$ leaves. However, when explored by AND/OR search with adaptive caching the context minimal graph of the cluster is much smaller, as can be seen in Figure 6.11. By comparing the underlying context minimal graphs, it can be shown that:

THEOREM **6.3.3** *Given a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ with no determinism, and an execution of* ***TDC(i)****, there exists a pseudo tree that guides an execution of* ***AOC(i)*** *that traverses the same context minimal graph.*

Proof. Algorithm **TDC(i)** is already designed to be an improvement over *space based join tree clustering* (Section 6.3.1), in that it uses AND/OR Adaptive Caching (rather than cutset conditioning) inside each cluster to compute the messages that are sent. **TDC(i)** is based on a rooted tree decomposition, which can serve as the skeleton for the underlying pseudo tree. Each cluster has its own pseudo tree to guide the AND/OR Adaptive Caching. Each message sent between neighboring clusters couples all the variables in its scope, therefore the scope variables have to appear on a chain at the top of the pseudo tree of the cluster where the message is generated, and also they have to appear on a chain in the pseudo tree of the cluster where the message is received. This implies that two neighboring clusters can agree on an ordering of the common variables (since we assume no determinism, the order of variables on a chain doesn't change the size of the context minimal graph). All this allows us to build a common pseudo tree for two neighboring clusters, by superimposing the common variables in their respective pseudo trees (which are the variables in the separator between the clusters). In this way we can build the pseudo tree for the entire problem. Now, for any variable X_i in the problem pseudo tree, $i\text{-}context(X_i)$ is the same as it was in the highest cluster (closest to the root of the tree decomposition) where X_i is mentioned. From this, it follows that **AOC(i)** based on the pseudo tree for the entire problem traverses the same context minimal graph as **TDC(i)**. □

6.3.4 Discussion

We have compared three parameterized algorithmic schemes for graphical models that can accommodate time-space trade-offs. They have all emerged from seemingly different principles: **AOC(i)** is search based, **TDC(i)** is inference based and **VEC(i)** combines search and inference.

We show that if the graphical models contain no determinism, **AOC(i)** can have a smaller time complexity than the vanilla versions of both **VEC(i)** and **TDC(i)**. This is due to a more efficient exploitation of the graphical structure of the problem through AND/OR search, and the adaptive caching scheme that benefits from the cutset principle. These ideas can be used to enhance **VEC(i)** and **TDC(i)**. We show that if **VEC(i)** uses AND/OR search over the conditioning set and is guided by the pseudo tree data structure, then there exists an execution of **AOC(i)** that is identical to it. We also show that if **TDC(i)** processes clusters by AND/OR search with adaptive caching, then there exists an execution of **AOC(i)** identical to it. AND/OR search with adaptive caching (**AOC(i)**) emerges therefore as a unifying scheme, never worse than the other two. All the analysis was done by using the context minimal data structure, which provides a powerful methodology for comparing the algorithms.

When the graphical model contains determinism, all the above schemes become incomparable. This is due to the fact that they process variables in reverse orderings, and will encounter and exploit deterministic information differently.

6.4 Conclusion to Chapter 6

This chapter was dedicated to the analysis of search and inference algorithms in graphical models. Analogies between top down and bottom up traversals of a tree or graph go a long way back and appear in many areas of computer science.

Our main contribution in this chapter was the comparison of AND/OR search (as a top down method) with inference algorithms (as bottom up methods) in graphical models. We develop a methodology of comparison by describing the *context minimal graph* (CM graph). Memory intensive AND/OR search (i.e., with full context-based caching) traverses the CM graph top down, in a depth first (DFS) manner. Variable Elimination, on the other hand, can be shown to traverse the same CM graph, if the elimination order is the reverse of the DFS traversal of the pseduo tree that guides the AND/OR search. Each multiplication and summation performed by Variable Elimination can be associated with the traversal of and edge in the CM graph. We show that there is no principled difference between memory-intensive search with fixed variable ordering and inference beyond: (1) different direction of exploring a common search space (top down for search vs. bottom-up for inference); (2) different assumption of control strategy (depth-first for search and breadth-first for inference). We also show that those differences occur only in the presence of determinism.

We also extend the same type of analysis to hybrid algorithms, that combine search and inference. We propose the *Adaptive Caching* algorithm (**AOC(i)**) as the most efficient AND/OR search algorithm, that exploits the available memory in the best way. We show that **AOC(i)** is never worse than two other schemes, **VEC(i)** that interleaves elimination and conditioning, and **TDC(i)** that is based on tree decompositions.

Chapter 7

AND/OR Multi-Valued Decision Diagrams (AOMDDs)

7.1 Introduction

The work presented in this chapter is based on two existing frameworks: (1) AND/OR search spaces for graphical models (see Chapter 2) and (2) decision diagrams (DD).

Decision diagrams are widely used in many areas of research, especially in software and hardware verification [18, 81]. A BDD represents a Boolean function by a directed acyclic graph with two sink nodes (labeled 0 and 1), and every internal node is labeled with a variable and has exactly two children: *low* for 0 and *high* for 1. If isomorphic nodes were not merged, on one extreme we would have the full search *tree*, also called Shannon tree, which is the usual full tree explored by backtracking algorithm. The tree can be ordered if we impose that variables be encountered in the same order along every branch. It can then be compressed by merging isomorphic nodes (i.e., with the same label and identical children), and by eliminating redundant nodes (i.e., whose *low* and *high* children are identical). The result is the celebrated *reduced ordered binary decision diagram*, or OBDD for short, introduced by Bryant [16]. However, the underlying structure is OR, because the initial Shannon tree is an OR tree. If AND/OR search trees are reduced by node merging and redundant nodes elimination we get a compact search graph that can be viewed as a BDD representation augmented with AND nodes.

7.1.1 Contributions

We combine here the ideas of AND/OR search and decision diagrams, in order to create a decision diagram that has an AND/OR structure, thus exploiting problem decomposition. As a detail, the number of values is also increased from two to any constant, but this is less significant for the algorithms. Our proposal is closely related to two earlier research lines within the BDD literature. The first is the work on Disjoint Support Decompositions (DSD) investigated within the area of design automation [15], that were proposed recently as enhancements for BDDs aimed at exploiting function decomposition [8]. The second is the work on BDDs trees [82]. Another related proposal is the recent work by Fargier and Vilarem [46] on compiling CSPs into tree-driven automata.

A decision diagram offers a compilation of a problem. It typically requires an extended offline effort in order to be able to support polynomial (in its size) or constant time online queries. In the context of constraint networks, it could be used to represent the whole set of solutions, to give the solutions count or solution enumeration and to test satisfiability or equivalence of constraint networks. The benefit of moving from OR structure to AND/OR is in a lower complexity of the algorithms and size of the compiled structure. It typically moves from being bounded exponentially in *pathwidth* pw^*, which is characteristic to chain decompositions or linear structures, to being exponentially bounded in *treewidth* w^*, which is characteristic of tree structures (it always holds that $w^* \leq pw^*$ and $pw^* \leq w^* \cdot \log n$). In both cases, the compactness achieved in practice is often far smaller than what the bounds suggest.

The contributions made in this chapter are the following:

(1) We formally describe the AND/OR Multi-Valued Decision Diagram (AOMDD) and

prove that it is a canonical representation for constraint networks.

(2) We extend the AOMDD to general weighted graphical models.

(3) We give a compilation algorithm based on AND/OR search, that saves the trace of the memory intensive search (which is a subset of the context minimal graph), and then reduces it in one bottom up pass.

(4) We describe the APPLY operator that combines two AOMDDs by an operation, and show that its complexity is at most quadratic in the input.

(5) We give a scheduling of building the AOMDD of a graphical model starting with the AOMDDs of its functions. It is based on an ordering of variables, which gives rise to a pseudo tree according to the execution of Variable Elimination algorithm. This guarantees that the complexity is at most exponential in the the *induced width* along the ordering (equal to the treewidth of the corresponding decomposition).

(6) We show how AOMDDs relate to various earlier and recent compilation frameworks, providing a unifying perspective for all these methods.

(7) We also introduce the concept of *semantic treewidth*, which helps explain why the size of a decision diagram is often much smaller than the worst case bound.

The research presented in this chapter is based in part on [76, 77, 38, 73].

7.2 Motivation

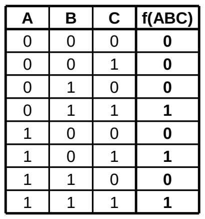
Before we proceed with the technical details, we provide the motivation for *compiling* a graphical model, and give examples of how it would be useful. Without elaborating on the definition now, what we mean by *compilation* is a compact representation of a graphical model, which allows fast response time to queries.

The first aim is to divide of the computational effort between an *offline* and an *online* phase. Most of the work is pushed offline, with the tradeoff that online responses to querries are be fast. The second purpose of compilation is that of obtaining the most compact representation possible for a function (or graphical model), and to have efficient operations between functions. This can have an impact on any existing algorithm that can benefit from such a compact and efficient data structure.

A typical example where one would want to divide the work between offline computation and online query answering might be product configuration. Imagine a user that choses sequential options to configure a product. There may be several constraints that do not permit certain combinations. In a naive system, the user would be allowed to choose any valid option at the current level based only on the initial constraints, until either the product is configured, or else, when a dead-end is encountered, the system would backtrack to some previous state and continue from there. This would in fact be a search through the space of possible partial configurations. Needless to say, it would be very unpractical, and would offer the user no guarantee of finishing in a limited time. A system based on compilation would actually build the *backtrack-free* search space in the offline phase, and represent it in a compact manner. In the online phase, only valid partial configurations (i.e., that can be extended to a full valid configuration) are allowed, and depending on the query type, response time guarantees can be offered in terms of the size of the compiled structure.

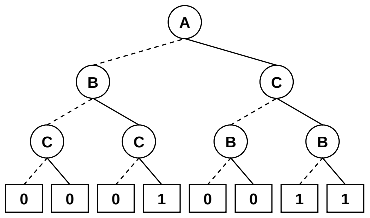
There are numerous other examples that can be formulated as graphical models, and where compilation would be useful. In diagnosis, the problem is to detect the possible faults or explanations for some unusual behaviour. Planning problems can also be formulated as graphical models, and a compilation would alow swift adjustments according to changes in the environment. Probabilistic models are one of the most used types of graphical models, and the basic query is to compute conditional probabilities of some variables given the evidence. A compact compilation of a probabilistic model would allow fast response for any change in the evidence along time.

Formal verification is another example where compilation is heavily used to compare equivalence of circuit design, or to check the behaviour of a circuit. *Binary Decision Diagram* (BDD) [16] are arguably the most widely known and used compiled structure.

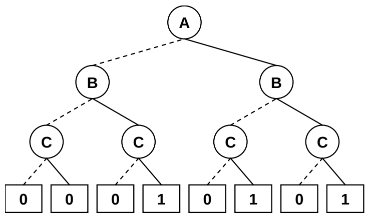


A	B	C	f(ABC)
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**0**
1	1	1	**1**

(a) Table



(b) Unordered tree



(c) Ordered tree

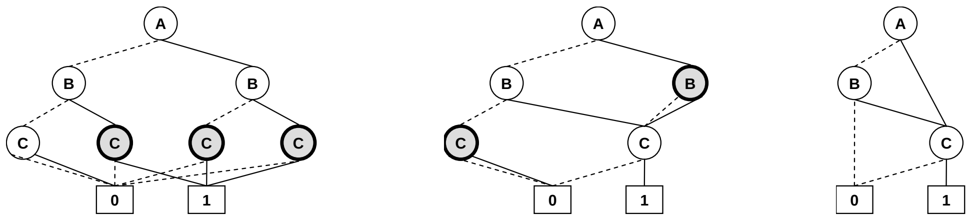
Figure 7.1: Boolean function representations

7.3 Binary Decision Diagrams Review

Decision diagrams are widely used in many areas of research to represent decision processes. In particular, they can be used to represent functions. Due to the fundamental importance of Boolean functions, a lot of effort has been dedicated to the study of *Binary Decision Diagrams* (BDDs), which are extensively used in software and hardware verification [18, 81]. The earliest work on BDDs is due to Lee [67], who introduced binary-decision *programs,* that can be understood as a linear representation of a BDD (e.g., a depth first search ordering of the nodes), where each node is a branching instruction indicating the address of the next instruction for both the 0 and the 1 value of the test variable. Akers [1] presented the actual graphical representation and further developed the BDD idea. However, it was Bryant [16] that introduced what is now called the *Ordered Binary Decision Diagram* (OBDD). He restricted the order of variables along any path of the diagram, and presented algorithms (most importantly the *apply* procedure, that combines two OBDDs by an operation) that have time complexity at most quadratic in the sizes of the input diagrams. OBDDs are fundamental for applications with large binary functions, especially because in many practical cases they provide very compact representations.

A BDD is a representation of a Boolean function. Given $\mathbf{B} = \{0, 1\}$, a Boolean function $f : \mathbf{B}^n \to \mathbf{B}$, has n arguments, $X_1, \cdots, X_n$, which are Boolean variables, and takes Boolean values.

Example 7.3.1 *Figure 7.1(a) shows a table representation of a Boolean function of three*



(a) Isomorphic nodes (b) Redundant nodes (c) OBDD

Figure 7.2: Reduction rules

variables. This explicit representation is the most straightforward, but also the most costly due to its exponential requirements. The same function can also be represented by a binary tree, shown in Figure 7.1(b), that has the same exponential size in the number of variables. The internal round nodes represent the variables, the solid edges are the 1 (or high) value, and the dotted edges are the 0 (or low) value. The leaf square nodes show the value of the function for each assignment along a path. The tree shown in 7.1(b) is unordered, because variables do not appear in the same order along each path.

In building an OBDD, the first condition is to have variables appear in the same order (A,B,C) along every path from root to leaves. Figure 7.1(c) shows an ordered binary tree for our function. Once an order is imposed, there are two reduction rules that transform a decision diagram into an equivalent one:

(1) isomorphism: merge nodes that have the same label and the same children.

(2) redundancy: eliminate nodes whose low and high edges point to the same node, and connect parent of removed node directly to child of removed node.

Applying the two reduction rules exhaustively yields a *reduced* OBDD, sometimes denoted rOBDD. We will just use OBDD and assume that it is completely reduced.

Example 7.3.2 *Figure 7.2(a) shows the binary tree from Figure 7.1(c) after the isomorphic terminal nodes (leaves) have been merged. The highlighted nodes, labeled with C, are also isomomorphic, and Figure 7.2(b) shows the result after they are merged. Now, the highlighted nodes labeled with C and B are redundant, and removing them gives the OBDD in Figure 7.2(c).*

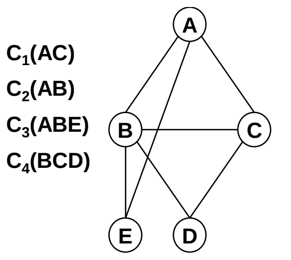
7.4 Bucket Elimination (BE)

Bucket Elimination (**BE**) [29] is a well known variable elimination algorithm for inference in graphical models. We will describe it using the terminology for constraint networks, but **BE** can also be applied to any graphical model. Consider a constraint network $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$ and an ordering $d = (X_1, X_2, \ldots, X_n)$. The ordering d dictates an elimination order for **VE**, from last to first. Each variable is associated with a bucket. Each constraint from $\mathbf{C}$ is placed in the bucket of its latest variable in d. Buckets are processed from X_n to X_1 by eliminating the bucket variable (the constraints residing in the bucket are joined together, and the bucket variable is projected out) and placing the resulting constraint (also called *message*) in the bucket of its latest variable in d. After its execution, **VE** renders the network backtrack free, and a solution can be produced by assigning variables along d. **VE** can also produce the solutions count if marginalization is done by summation (rather than projection) over the functional representation of the constraints, and join is substituted by multiplication.

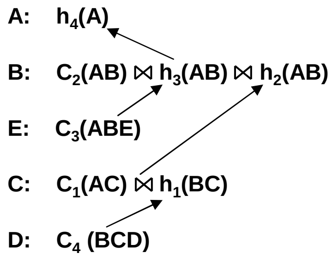
VE also constructs a bucket tree, by linking the bucket of each X_i to the destination bucket of its message (called the parent bucket). A node in the bucket tree typically has a *bucket variable*, a *collection of constraints*, and a *scope* (the union of the scopes of its constraints). If the nodes of the bucket tree are replaced by their respective bucket variables, it is easy to see that we obtain a pseudo tree.

Example 7.4.1 *Figure 7.3(a) shows a network with four constraints. Figure7.3(b) shows the execution of Bucket Elimination along $d = (A, B, E, C, D)$. The buckets are processed from D to A* [1]*. Figure 7.3(c) shows the bucket tree. The pseudo tree corresponding to the order d is given in Fig. 3.11(b).*

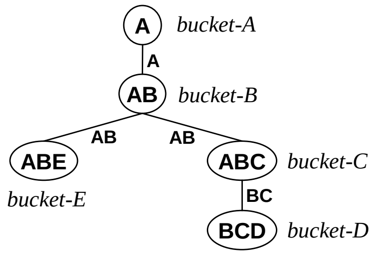
[1]The representation in Figure 7.3 reverses the top down bucket processing described in earlier papers.



(a) Constraint network



(b) **VE** execution



(c) Bucket tree

Figure 7.3: Bucket Elimination

7.5 AND/OR Multi-Valued Decision Diagrams (AOMDDs)

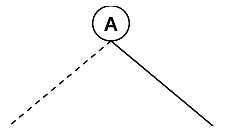
The *context minimal* AND/OR graph (Definition 2.3.9) offers an effective way of identifying some unifiable nodes during the execution of the search algorithm. Namely, context unifiable nodes are discovered based only on their paths from the root, without actually solving their corresponding subproblems. However, merging based on context is not complete, which means that there may still exist unifiable nodes in the search graph that do not have identical contexts. Moreover, some of the nodes in the context minimal AND/OR graph may be redundant, for example when the set of solutions rooted at variable X_i is not dependant on the specific value assigned to X_i (this situation is not detectable based on context). This is sometimes termed as “interchangeable values” or “symmetrical values”.

As overviewed earlier, in [45] we defined the complete *minimal AND/OR graph* which is an AND/OR graph whose all unifiable nodes are merged and we also proved canonicity [38]. Here we propose to augment the minimal AND/OR search graph with removing redundant variables as is common in OBDD representation as well as adopt some notational conventions common in this community. This yields a data structure that we call AND/OR BDD, that exploits decomposition by using AND nodes. We present the extension over multi-valued variables yielding AND/OR MDD or AOMDD and define them for general weighted graphical models. We will also present two algorithms for compiling the canonical AOMDD of a graphical model: the first is search based, and used the memory intensive AND/OR graph search to generate the context minimal AND/OR graph, and then reduces it bottom up by applying reduction rules; the second is inference based, and uses a Bucket Elimination schedule to combine the AOMDDs of initial functions by APPLY operations (similar to the *apply* for OBDDs). As we will show, both approaches have the same worst case complexity as the AND/OR graph search with context based caching, and also the same complexity as Bucket Elimination, namely time and space exponential in the treewidth of the problem, $O(n\ k^{w^*})$.

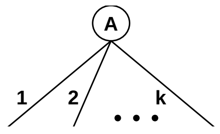
7.5.1 From AND/OR Search Graphs to Decision Diagrams

An AND/OR search graph $\mathcal{G}$ of a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \otimes \rangle$ represents the set of all possible assignments to the problem variables (all solutions and their costs). In this sense, $\mathcal{G}$ can be viewed as representing the function $f = \otimes_{f_i \in \mathbf{F}} f_i$ that defines the universal equivalent graphical model $u(\mathcal{M})$ (Definition 1.2.6). For each full assignment $x = (x_1, \ldots, x_n)$, if x forms a solution subtree t, then $f(x) = w(t) = \otimes_{e \in arcs(t)} w(e)$ (Definition 2.2.7); otherwise $f(x) = 0$ (the assignment is inconsistent). The solution subtree t of a consistent assignment x can be read from $\mathcal{G}$ in linear time by following the assignments from the root. If x is inconsistent, then a deadend is encountered in $\mathcal{G}$ when attempting to read the solution subtree t, and $f(x) = 0$. Therefore, $\mathcal{G}$ can be viewed as a decision diagram that determines the values of f for every complete assignment x.

We will now see how we can process an AND/OR search graph by reduction rules similar to the case of OBDDs, in order to obtain a representation of minimal size. In the case of OBDDs, a node is labeled with a variable name, for example A, and the *low*

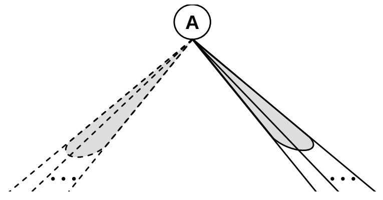


(a) OBDD

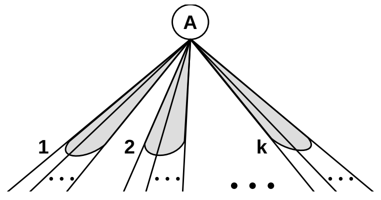


(b) MDD

Figure 7.4: Decision diagram nodes (OR)



(a) AOBDD



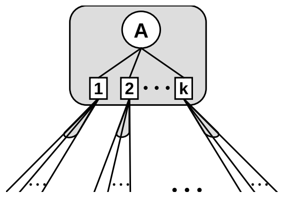
(b) AOMDD

Figure 7.5: Decision diagram nodes (AND/OR)

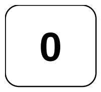
(dotted line) and *high* (solid line) outgoing arcs capture the restriction of the function to the assignments $A = 0$ or $A = 1$. To determine the value of the function, one needs to follow either one or the other (but not both) of the outgoing arcs from A (see Figure 7.4(a)). The straightforward extension of OBDDs to multi-valued variables (multi-valued decision diagrams, or MDDs) was presented in [97], and the nodes that they use are given in Figure 7.4(b). Here, each outgoing arc is associated with one of the k values of variable A.

In this chapter we generalize the OBDD and MDD representations in Figures 7.4(a) and 7.4(b) by allowing each outgoing arc of a node to be an AND arc. An AND arc connects a node to a set of nodes, and captures the decomposition of the problem into independent components. The number of AND arcs emanating from a node is two in the case of AOBDDs (Figure 7.5(a)), or the domain size of the variable in the general case (Figure 7.5(b)). For a given node A, each of its k AND arcs can connect it to possibly different number of nodes, depending on how the problem decomposes based on each particular assignment of A. The AND arcs are depicted by a shaded sector that connects the outgoing lines corresponding to the independent components.

We derive the AND/OR Decision Diagram representation based on AND/OR search graphs. We find that it is useful to maintain the semantics of Figure 7.5 especially when we



(a) Nonterminal meta-node



(b) Terminal meta-node **0**

(c) Terminal meta-node **1**

Figure 7.6: Meta-nodes

need to express the redundancy of nodes, and therefore we introduce the *meta-node* data structure, which defines small portions of any AND/OR graph, based on a an OR node and its AND children:

DEFINITION **7.5.1 (meta-node)** *A* meta-node u *in an AND/OR search graph consists of an OR node labeled X (therefore $var(u) = X$) and its k AND children labeled $x_1, \ldots, x_k$ that correspond to the value assignments of X. Each AND node labeled x_i stores a list of pointers to child meta-nodes, denoted by $u.children_i$. In the case of weighted graphical models, the AND node x_i also stores the OR-to-AND arc weight $w(X, x_i)$.*

The rectangle in Figure 7.6(a) is a meta-node for variable A, that has a domain of size k. Note that this is very similar to Figure 7.5, with the small difference that the information about the value of A that corresponds to each outgoing AND arc is now stored in the AND nodes of the meta-node. We are not showing the weights in that figure. A larger example of an AND/OR graph with meta-nodes appears later in Figure 7.8.

We also define two special meta-nodes, that will play the role of the terminal nodes in OBDDs. The terminal meta-node **0**, shown in Figure 7.6(b), indicates inconsistent assignments, while the terminal meta-node **1**, shown in figure 7.6(c) indicates consistent ones.

Any AND/OR search graph can now be viewed as a diagram of meta-nodes, simply by grouping OR nodes with their AND children, and adding the terminal meta-nodes appropriately.

Once we have defined the meta-nodes, it is easier to see when a variable is redundant with respect to the outcome of the function based on the current patial assignment. Intuitiveley, any assignment to a redundant variable should lead to the same set of solutions.

DEFINITION **7.5.2 (redundant meta-node)** *Given a weighted AND/OR search graph $\mathcal{G}$ represented with meta-nodes, a meta-node u with $var(u) = X$ and $|D(X)| = k$ is* redundant *iff:*

(a) $u.children_1 = \ldots = u.children_k$ *and*

(b) $w(X, x_1) = \ldots = w(X, x_k)$.

An AND/OR graph $\mathcal{G}$, that contains a redundant meta-node u, can be transformed into an equivalent graph $\mathcal{G}'$ by replacing any incoming arc into u with its common list of children $u.children_1$, absorbing the common weight $w(X, x_1)$ by combination into the weight of the parent meta-node corresponding to the incoming arc, and then removing u and its outgoing arcs from $\mathcal{G}$. If u is the root of the graph, then the common weight $w(X, x_1)$ has to be stored separateley as a constant.

The notion of isomorphism is extended naturally from AND/OR graphs to meta-nodes.

DEFINITION **7.5.3 (isomorphic meta-nodes)** *Given a weighted AND/OR search graph $\mathcal{G}$ represented with meta-nodes, two meta-nodes u and v having $var(u) = var(v) = X$ and $|D(X)| = k$ are* isomorphic *iff:*

(a) $u.children_i = v.children_i \; \forall i \in \{1, \ldots, k\}$ *and*

(b) $w^u(X, x_i) = w^v(X, x_i) \; \forall i \in \{1, \ldots, k\}$, *(where w^u, w^v are the weights of u and v).*

Naturally, the AND/OR graph obtained by merging isomorphic meta-nodes is equivalent to the original one.

We can now define the AND/OR Multi-Valued Decision Diagram:

DEFINITION **7.5.4 (AOMDD)** *An AND/OR Multi-Valued Decision Diagram (AOMDD) is a weighted AND/OR search graph that is completely reduced by isomorphic merging and redundancy removal, namely:*

(1) it contains no isomorphic meta-nodes; and

(2) it contains no redundant meta-nodes.

The AOMDD of a function f is denoted by $\mathcal{G}_f^{aomdd}$.

Examples of AOMDDs appear in Figures 7.8 and 7.9. We will next discuss two approaches for generating the AOMDD of a graphical model: in Section 7.6 we present a search based algorithm, that performs AND/OR search and then applies the reduction rules (Definitions 7.5.2 and 7.5.3) bottom up to the trace of the search (i.e., the traversed space) to obtain the AOMDD; in Section 7.7 we present an inference algorithm based on a Bucket Elimination schedule, that uses the APPLY operation inside each bucket to combine the AOMDDs of the graphical model functions.

7.6 Using AND/OR Search to Generate AOMDDs

In Section 7.5.1 we described how we can transform an AND/OR graph into an AOMDD by applying reduction rules. In this section we describe the explicit algorithm that takes as input a graphical model, performs AND/OR search with context based caching to obtain the context minimal AND/OR graph, and then applies the reduction rules bottom up to obtain the AOMDD.

7.6.1 AND/OR Search Algorithm

For completeness, Algorithm 7 shows the AND/OR search algorithm [38]. The input is a graphical model $\mathcal{M}$ and a pseudo tree $\mathcal{T}$. Algorithm 7 associates a value to each node in the AND/OR search space. This is necessary when solving a reasoning task, but can be avoided when we just need to generate an AOMDD for $\mathcal{M}$. The only important information that needs to be propagated in the backtrack phase of the search when generating an AOMDD is the distinction between consistent and inconsistent assignments, namely to propagate the dead-ends up to the highest level. Therefore, an AOMDD does not need to be associated with a reasoning task, since it represents the universal function of $\mathcal{M}$, and as such can be traversed to obtain answers to different reasoning problems. It is up to the user to decide if more information is stored in each meta-node, for example the value corresponding to a reasoning task restricted to its subproblem.

Algorithm 7 describes both a depth first tree AND/OR search and a graph AND/OR search. The switch is realized through the variable `caching`, which we set to *true* to obtain the memory intensive version. Each variable X_i has an associated cache table, whose scope is the context of X_i in $\mathcal{T}$. This will ensure that the trace of the search is the context minimal AND/OR graph.

We will also use a list for each variable, to save pointers to meta-nodes corresponding to that variable level. The lists will be used by the procedure that performs the bottom up traversal, per layers of the AND/OR graph, to apply the reduction rules. The list for variable X_i is denoted by $List^{X_i}$.

The fringe of the search is maintained on a stack called `OPEN`. The current node is denoted by `n`, its parent by `p`, and the current path by π_n. The children of the current node are denoted by $successors(\texttt{n})$.

The algorithm is based on two mutually recursive steps: EXPAND and PROPAGATE, which call each other (or themselves) until the search terminates.

Since we only use OR caching, before expanding an OR node, its cache table is checked (line 6). If the same context was encountered before, the node and its value are retrieved from cache, and $successors(\texttt{n})$ is set to the empty set, which will trigger the PROPAGATE step.

If a node is not found in cache, it is expanded in the usual way, depending on whether it is an AND or OR node (lines 10-19). In our description, we assume a sum-product reasoning task (e.g., belief updating in belief networks, or solutions counting in constraint networks). For belief updating, the value of an AND node is initialized to the bucket value for the current assignment, namely the weight of the OR-to-AND arc (line 15). When an OR node is expanded, its possible variable assignments are checked for consistency (line 14). The deterministic information (inconsistent assignments) in $\mathcal{M}$ can be extracted to form a constraint network. Any level of constraint propagation can be performed in this step (e.g., look ahead, arc consistency, path consistency, i-consistency etc.). At the minimum, a value $\langle X_i, x_i \rangle$ together with the current path assignement $asgn(\pi_n)$ can be checked to be consistent with the initial functions in $\mathbf{F}$. As long as the current node is not a dead-end and still has unevaluated successors, one of its successors is chosen (which is also the top node on `OPEN`), and the expansion step is repeated.

The bottom up propagation of values is triggered when a node has an empty set of successors (note that, as each successor is evaluated, it is removed from the set of successors in line 34). This means that all its children have been evaluated, and its final value can now be computed. If the current node is the root, then the search terminates with its value (line 22). If it is an OR node, its value is saved in cache before propagating it up (line 24). If `n` is OR, then its parent `p` is AND and `p` updates its value by multiplication with the value of `n` (line 27). If the newly updated value of `p` is 0 (line 28), then `p` is a dead-end, and none of its other successors needs to be evaluated. An AND node `n` propagates its value to its parent `p` in a similar way, only by summation (line 33). Finally, the current node `n` is set to its parent `p` (line 35), because `n` was completely evaluated. The search continues either with a propagation step (if conditions are met) or with an expansion step.

When Algorithm 7 terminates, the context minimal AND/OR graph of $\mathcal{M}$ is obtained, and can also be viewed as the trace of the search algorithm. To avoid cluttering the algorithm, we did not describe explicitly how pointers are maintained between OR and AND nodes, and how meta-nodes are formed, but this is straightforwad from the execution of the depth first search. A list L^{H_i} contains all the metanodes of X_i that appear in the context minimal AND/OR graph.

Algorithm 7: AND/OR SEARCH

```
input  : A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$; a pseudo tree $\mathcal{T}$ of the primal graph, rooted at $X_1$; parents $pa_i$
         (OR-context) for every variable $X_i$; caching set to $true$.
output : The context minimal graph of $\mathcal{M}$.
1  if caching $== true$ then                                          // Initialize cache tables
2      Initialize cache tables with entries of "$-1$"
3  $v(X_1) \leftarrow 0$; OPEN $\leftarrow \{X_1\}$                               // Initialize the stack OPEN
4  while OPEN $\neq \phi$ do
5      n $\leftarrow top$(OPEN); remove n from OPEN
6      if caching $== true$ and n is OR, labeled $X_i$ and $Cache(asgn(\pi_n)[pa_i]) \neq -1$ then    // If in cache
7          $v$(n) $\leftarrow Cache(asgn(\pi_n)[pa_i])$                              // Retrieve value
8          $successors$(n) $\leftarrow \phi$                                     // No need to expand below
9      else                                                             // EXPAND (forward)
10         if n is an OR node labeled $X_i$ then                                  // OR-expand
11             $successors$(n) $\leftarrow \phi$
12             forall $x_i \in D_i$ do                                    // Constraint Propagation
13                 if $\langle X_i, x_i \rangle$ is consistent with $\pi_n$ then
14                     $successors$(n) $\leftarrow successors$(n) $\cup \langle X_i, x_i \rangle$
15             $v(\langle X_i, x_i \rangle) \leftarrow \prod_{f \in B_{\mathcal{T}}(X_i)} f(asgn(\pi_n)[pa_i])$,   for all $\langle X_i, x_i \rangle \in successors$(n)
16         if n is an AND node labeled $\langle X_i, x_i \rangle$ then                     // AND-expand
17             $successors$(n) $\leftarrow children_{\mathcal{T}}(X_i)$
18             $v(X_i) \leftarrow 0$ for all $X_i \in successors$(n)
19         Add $successors$(n) to top of OPEN
20     while $successors$(n) $== \phi$ do                             // UPDATE VALUES (backtrack)
21         if n is an OR node labeled $X_i$ then
22             if $X_i == X_1$ then                                        // Search is complete
23                 return $v$(n)
24             if caching $== true$ then
25                 $Cache(asgn(\pi_n)[pa_i]) \leftarrow v$(n)                            // Save in cache
26                 Add meta-node of n to list $L^{H_i}$
27             $v$(p) $\leftarrow v$(p) $* v$(c)
28             if $v$(p) $== 0$ then                                     // Check if p is dead-end
29                 remove $successors$(p) from OPEN
30                 $successors$(p) $\leftarrow \phi$
31         if n is an AND node labeled $\langle X_i, x_i \rangle$ then
32             let p be the parent of n
33             $v$(p) $\leftarrow v$(p) $+ v$(n);
34         remove n from $successors$(p)
35         n $\leftarrow$ p
36 return trace of search and meta-nodes lists $L^{H_i}$
```

7.6.2 Reducing the Context Minimal AND/OR Graph to an AOMDD

We describe in Procedure 8 the bottom up reduction of the context minimal graph, and will prove that the result is the AOMDD of $\mathcal{M}$.

Procedure 8 processes the variables bottom up relative to the pseudo tree $\mathcal{T}$. We use the depth first traversal ordering of $\mathcal{T}$ (line 1), but any other bottom up ordering is as good. The outer for loop (starting at line 10) goes through all each level of the context minimal AND/OR graph. For efficiency, and to ensure the complexity guarantees that we will prove, a hash table, initially empty, is used for each level. The inner for loop (starting at line 10) goes through all the metanodes of a level, that are also saved (or pointers to them are saved) in the list L^{H_i}. For each new meta-node n in the list, if it was already encountered before (as meta-node p, namely if it is found in the cache table, then n is merged with p. Otherwise, if the new meta-node n is redundant, then it is eliminated from the AND/OR graph. If none of the previous two conditions is met, then the new meta-node n is hashed into the table H.

Procedure 8: `Bottom Up Reduction`

```
input   : A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$; a pseudo tree $\mathcal{T}$ of the primal graph, rooted at $X_1$;
          Context minimal AND/OR graph, and lists $L^{X_i}$ of meta-nodes for each level $X_i$.
output  : AOMDD of $\mathcal{M}$.
1  Let $d = \{X_1, \ldots, X_n\}$ be the depth first traversal ordering of $\mathcal{T}$
2  for $i \leftarrow n$ down to $1$ do
3      Let $H$ be a hash table, initially empty
4      forall meta-nodes n in $L^{H_i}$ do
5          if $H(X_i, n.children_1, \ldots, n.children_{k_i}, w^u(X_i, x_1), \ldots, w^u(X_{k_i}, x_{k_i}))$ returns p then
6              merge n with p in the AND/OR graph
7          else if n is redundant then
8              eliminate n from the AND/OR graph
9          else
10             $H(X_i, n.children_1, \ldots, n.children_{k_i}, w^u(X_i, x_1), \ldots, w^u(X_{k_i}, x_{k_i})) \leftarrow$ p
11 return reduced AND/OR graph
```

Proposition 25 *The output of Procedure 8 is the AOMDD of $\mathcal{M}$ along the pseudo tree $\mathcal{T}$, namely the resulting AND/OR graph is completely reduced.*

Proof. Consider the level of variable X_i, and the meta-nodes in the list L^{X_i}. After one pass through the meta-nodes in L^{X_i} (the inner for loop), there can be no two meta-nodes at the level of X_i in the AND/OR graph that are isomorphic, because they would have been merged in line 6. Also, during the same pass through the meta-nodes in L^{X_i} all the redundant meta-nodes in L^{X_i} are eliminated in line 8. Processing the meta-nodes in the level of X_i will not create new redundant or isomorphic meta-nodes in the levels that have been processed before. It follows that the resulting AND/OR graph is completely reduced. □

Note that we explicated Procedure 8 separately only for clarity. In practice, it can actually be included in Algorithm 7. We can maintain a hash table for each variable, during the AND/OR search, to store pointers to meta-nodes. When the search backtracks out of an OR node (in the Update value phase), it can already check the redundancy of that meta-node, and also look up in the hash table to check for isomorphism. Therefore, the reduction of the AND/OR graph can be done during the AND/OR search, and the output will be the AOMDD of $\mathcal{M}$.

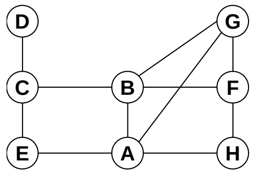
From Theorem 2.3.8 and Proposition 25 we can conclude:

THEOREM **7.6.1** *Given a graphical model $\mathcal{M}$ and a pseudo tree $\mathcal{T}$ of its primal graph G, the AOMDD of $\mathcal{M}$ corresponding to $\mathcal{T}$ has size bounded by $O(n\ k^{w^*_{\mathcal{T}}(G)})$ and it can be computed by the AND/OR search algorithm in time $O(n\ k^{w^*_{\mathcal{T}}(G)})$, where $w^*_{\mathcal{T}}(G)$ is the induced width of G over the depth first traversal of $\mathcal{T}$, and k bounds the domain size.*

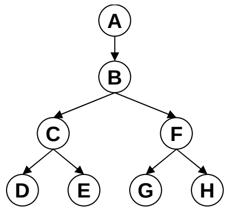
Proof. The bound on the size follows directly from Theorem 2.3.8. The AOMDD size can only be smaller than the size of the context minimal AND/OR graph, which is bounded by $O(n\ k^{w^*_{\mathcal{T}}(G)})$. To prove the time bound, we have to rely on the use of hash table, and the assumption that an efficient implementation allows an access time that we assume to be constant. The time bound of Algorithm 7 is $O(n\ k^{w^*_{\mathcal{T}}(G)})$, from Theorem 2.3.8, because it takes time linear in the output (we assume here that no constraint propagation is performed during search). Procedure 25 takes time linear in the size of the context minimal AND/OR graph. Therefore, the AOMDD can be computed in time $O(n\ k^{w^*_{\mathcal{T}}(G)})$, and the result is the same for the algorithm that performs the reduction during the search. □

7.7 Using Bucket Elimination to Generate AOMDDs

In this section we propose to use a Bucket Elimination (**BE**) type algorithm to guide the compilation of a graphical model into an AOMDD. The basic idea is to express the graphical model functions as AOMDDs, and then combine them with APPLY operations based



(a)



(b)

Figure 7.7: (a) Constraint graph for $C = \{C_1, \ldots, C_9\}$, where $C_1 = F \vee H$, $C_2 = A \vee \neg H$, $C_3 = A \oplus B \oplus G$, $C_4 = F \vee G$, $C_5 = B \vee F$, $C_6 = A \vee E$, $C_7 = C \vee E$, $C_8 = C \oplus D$, $C_9 = B \vee C$; (b) Pseudo tree (bucket tree) for ordering $d = (A, B, C, D, E, F, G, H)$

on a **BE** schedule. The APPLY is very similar to that from OBDDs [16], but it is adapted to AND/OR search graphs. It takes as input two functions represented as AOMDDs based on the same pseudo tree, and outputs the combination of initial functions, also represented as an AOMDD based on the same pseudo tree. We will describe it in detail in Section 7.7.2.

We will start with an example based on constraint networks. This is easier to understand because the weights on the arcs are all 1 or 0, and therefore are depicted in the figures by solid and dashed lines, respectively.

Example 7.7.1 *Consider the network defined by* $\mathbf{X} = \{A, B, \ldots, H\}$*,* $D_A = \ldots = D_H = \{0, 1\}$ *and the constraints (where* $\oplus$ *denotes XOR):* $C_1 = F \vee H$*,* $C_2 = A \vee \neg H$*,* $C_3 = A \oplus B \oplus G$*,* $C_4 = F \vee G$*,* $C_5 = B \vee F$*,* $C_6 = A \vee E$*,* $C_7 = C \vee E$*,* $C_8 = C \oplus D$*,* $C_9 = B \vee C$*. The constraint graph is shown in Figure 7.7(a). Consider the ordering* $d = (A, B, C, D, E, F, G, H)$*. The pseudo tree (or bucket tree) induced by* d *is given in Fig. 7.7(b). Figure 7.8 shows the execution of* ***VE*** *with AOMDDs along ordering* d*. Initially, the constraints* C_1 *through* C_9 *are represented as AOMDDs and placed in the bucket of their latest variable in* d*. The scope of any original constraint always appears on a path from root to a leaf in the pseudo tree. Therefore, each* original *constraint is represented by an AOMDD based on a chain. (i.e. there is no branching into independent components at any point). The chain is just the scope of the constraint, ordered according to* d*. For bi-valued variables, the original constraints are represented by OBDDs, for multiple-valued variables they are MDDs. Note that we depict meta-nodes: one OR node and its two AND*

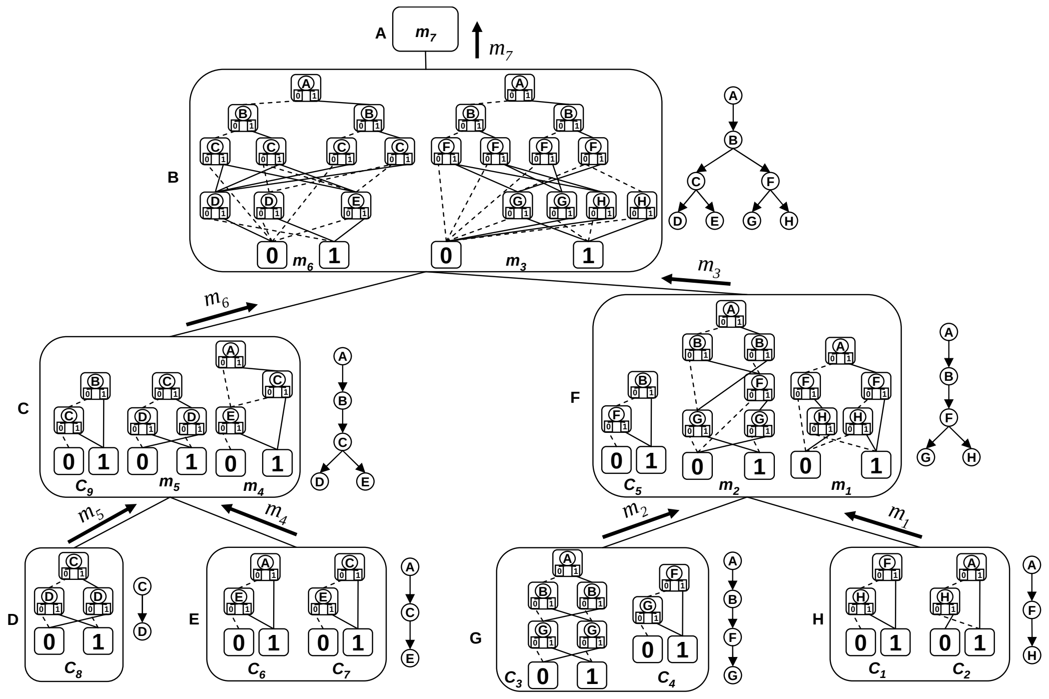


Figure 7.8: Execution of VE with AOMDDs

children, that appear inside each gray node. The dotted edge corresponds to the 0 value (the low *edge in OBDDs), the solid edge to the 1 value (the* high *edge). We have some redundancy in our notation, keeping both AND value nodes and arc-types (doted arcs from "0" and solid arcs from "1").*

*The **VE** scheduling is used to process the buckets in reverse order of* d*. A bucket is processed by* joining *all the AOMDDs inside it, using the* APPLY *operator. However, the step of elimination of the bucket variable is omitted because we want to generate the full AOMDD. In our example, the messages* $m_1 = C_1 \bowtie C_2$ *and* $m_2 = C_3 \bowtie C_4$ *are still based on chains, so they are still OBDDs. Note that they still contain the variables* H *and* G*, which have not been eliminated. However, the message* $m_3 = C_5 \bowtie m_1 \bowtie m_2$ *is not an OBDD anymore. We can see that it follows the structure of the pseudo tree, where* F *has two children,* G *and* H*. Some of the nodes corresponding to* F *have two outgoing edges for value 1.*

The processing continues in the same manner The final output of the algorithm, which

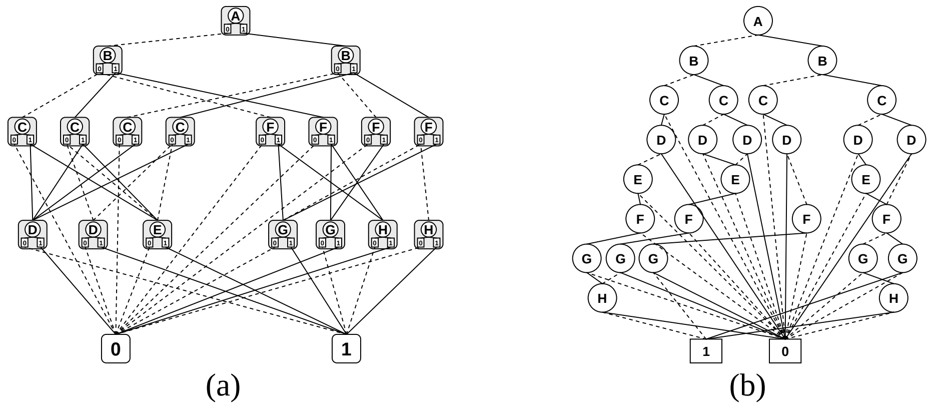


Figure 7.9: (a) The final AOMDD; (b) The OBDD corresponding to d

coincides with m_7, is shown in Figure 7.9(a). The OBDD based on the same ordering d is shown in Fig. 7.9(b). Notice that the AOMDD has 18 nonterminal nodes and 47 edges, while the OBDD has 27 nonterminal nodes and 54 edges.

7.7.1 Algorithm VE-AOMDD

Given an ordering d, the structural information captured in the primal graph through the scopes of the functions $\mathbf{F} = \{f_1, \ldots, f_r\}$ can be used to create the unique pseudo tree that corresponds to d. This is precisely the bucket tree (or elimination tree), that is created by **BE** (when variables are processed in reverse d). The same pseudo tree can be created by conditioning on the primal graph, and processing variables in the order d, as described in Procedure 9 (*GeneratePseudoTree*). In the following, $d|_{G_i}$ is the restriction of the order d to the nodes of the graph G_i.

Each constraint C_i is compiled into an AOMDD that is compatible with $\mathcal{T}$ and placed into the appropriate bucket. The buckets are processed from last variable to first as usual. Each bucket contains AOMDDs that are either initial constraints or AOMDDs received from previously processed buckets. The scope of all the variables that are mentioned in a bucket include *relevant* variables, i.e. the ones whose buckets were not yet processed (note that they are identical to the OR context), and *superfluous* variables, the ones whose buckets had been proceessed. The number of relevant variables is bounded by the induced width

```
Procedure 9:  GeneratePseudoTree($G, d$)
  input  : graph $G = (\mathbf{X}, E)$; order $d = (X_1, \ldots, X_n)$
  output : Pseudo tree $\mathcal{T}$
1 Make $X_1$ the root of $\mathcal{T}$;
2 Condition on $X_1$ (eliminate $X_1$ and its incident edges from $G$). Let $G_1, \ldots, G_p$ be
  the resulting connected components of $G$;
3 for $i = 1$ to $p$ do
4     $\mathcal{T}_i$ = GeneratePseudoTree ($G_i, d|_{G_i}$);
5     Make root of $\mathcal{T}_i$ a child of $X_1$
6 return $\mathcal{T}$;
```

```
Algorithm 10: BE-AOMDD
  input  : Graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, where $\mathbf{X} = \{X_1, \ldots, X_n\}$,
           $\mathbf{F} = \{f_1, \ldots, f_r\}$ ; order $d = (X_1, \ldots, X_n)$
  output : AOMDD representing $\otimes_{i \in \mathbf{F}} f_i$
1  $\mathcal{T}$ = GeneratePseudoTree($G, d$);
2  for $i \leftarrow 1$ to $r$ do                    // place functions in buckets
3      place $\mathcal{G}_{f_i}^{aomdd}$ in the bucket of its latest variable in $d$
4  for $i \leftarrow n$ down to 1 do                       // process buckets
5      $message(X_i) \leftarrow \mathcal{G}_{\mathbf{1}}^{aomdd}$          // initialize with AOMDD of 1;
6      while $bucket(X_i) \neq \phi$ do     // combine AOMDDs in bucket of $X_i$
7          pick $\mathcal{G}_f^{aomdd}$ from $bucket(X_i)$;
8          $bucket(X_i) \leftarrow bucket(X_i) \setminus \{\mathcal{G}_f^{aomdd}\}$;
9          $message(X_i) \leftarrow$ APPLY$(message(X_i), \mathcal{G}_f^{aomdd})$
10     add $message(X_i)$ to the bucket of the parent of $X_i$ in $\mathcal{T}$
11 return $message(X_1)$
```

(because so is the OR context). It is easy to see that any two AOMDDs in a bucket only have in common relevant variables, which reside on the top chain portion of the bucket pseudo tree. The superfluous variables appear in disjoint branches of the bucket pseudo tree. These observations will be important later on when we present the APPLY algorithm, and analyze the complexity.

Algorithm 10, called **BE-AOMDD**, creates the AOMDD of a graphical model by using a **BE** schedule for APPLY operations. Given an order d of the variables, first a pseudo tree is created based on the primal graph. Each initial function f_i is then represented as an AOMDD, denoted by $\mathcal{G}_{f_i}^{aomdd}$, and placed in its bucket. To obtain the AOMDD of a function, the scope of the function is ordered according to d, a search tree (based on a chain) that represents f_i is generated, and then reduced by Procedure 8. The algorithm proceeds exactly like **BE**, with the only difference that combination is realized by the APPLY algorithm, and variables are not eliminated but carried arround to the destination bucket. We defer the complexity analysis until we present the APPLY algorithm, and just observe that the complexity is bounded by that of **BE** (namely exponential in treewidth), provided that APPLY is an efficient operation. This will also become clear after we prove the canonicity of the AOMDD, since the complexity bounds were already given by the search based generation algorithm in Section 7.6.

7.7.2 The AOMDD APPLY Operation

We describe here how to combine two AOMDDs. The apply operator takes as input two AOMDDs representing functions f_1 and f_2 and returns an AOMDD representing $f_1 \otimes f_2$.

In OBDDs the *apply* operator combines two input diagrams based on the same variable ordering. Likewise, in order to combine two AOMDDs we assume that their backbone pseudo trees are *identical*. This condition is satisfied by any two AOMDDs in the same bucket of **VE-AOMDD**. However, it we present here a version of APPLY that is more general, by relaxing the previous condition from *identical* to *compatible* psedo trees. Namely, there should be a pseudo tree in which both can be embedded. In general, a pseudo tree induces a strict partial order between the variables where a parent node always precedes its child nodes.

DEFINITION **7.7.1 (compatible pseudo trees)** *A strict partial order $d_1 = (\mathbf{X}, <_1)$ over a set $\mathbf{X}$ is* consistent *with a strict partial order $d_2 = (\mathbf{Y}, <_2)$ over a set $\mathbf{Y}$, if for all $x_1, x_2 \in \mathbf{X} \cap \mathbf{Y}$, if $x_1 <_2 x_2$ then $x_1 <_1 x_2$. Two partial orders d_1 and d_2 are* compatible *iff there exists a partial order d that is consistent with both. Two pseudo trees are* compatible *iff the partial orders induced via the parent-child relationship, are compatible.*

For simplicity, we focus on a more restricted notion of compatibility, which is sufficient when using a **VE** like schedule for the *apply* operator to combine the input AOMDDs (as described in Section 7.7). The *apply* algorithm that we will present can be extended to the more general notion of compatibility.

DEFINITION **7.7.2 (strictly compatible pseudo trees)** *A pseudo tree $\mathcal{T}_1$ having the set of nodes $\mathbf{X}_1$ can be* embedded *in a pseudo tree $\mathcal{T}$ having the set of nodes $\mathbf{X}$ if $\mathbf{X}_1 \subseteq \mathbf{X}$ and $\mathcal{T}_1$ can be obtained from $\mathcal{T}$ by deleting each node in $\mathbf{X} \setminus \mathbf{X}_1$ and connecting its parent to each of its descendents. Two pseudo trees $\mathcal{T}_1$ and $\mathcal{T}_2$ are* compatible *if there exists $\mathcal{T}$ such that both $\mathcal{T}_1$ and $\mathcal{T}_2$ can be embedded in $\mathcal{T}$.*

Algorithm 11, called APPLY, takes as input one node from $\mathcal{G}_f^{aomdd}$ and a list of nodes from $\mathcal{G}_g^{aomdd}$. Initially, the node from $\mathcal{G}_f^{aomdd}$ is its, and the list of nodes from $\mathcal{G}_g^{aomdd}$ is in fact just one node, its root. We will sometimes identify an AOMDD by its root node. The backbone pseudo trees, $\mathcal{T}_f$ and $\mathcal{T}_g$ are strictly compatible, having a target pseudo tree $\mathcal{T}$.

The list of nodes from $\mathcal{G}_f^{aomdd}$ always has a special property: there is no node in it that can be the ancestor in $\mathcal{T}$ of another (we refer to the variable of the meta-node). Therefore, the list $w_1, \ldots, w_m$ from g expresses a decomposition with respect to $\mathcal{T}$, so all those nodes appear on different branches. We will employ the usual techniques from OBDDs to make the operation efficient. First, if one of the arguments is $\mathbf{0}$, then we can safely return $\mathbf{0}$. Second, a hash table H_1 is used to store the nodes that have already been processed, based on the nodes $(v_1, w_1, \ldots, w_r)$. Therefore, we never need to make multiple recursive calls on the same arguments. Third, a hash table H_2 is used to detect isomorphic nodes. This is typically split in separate tables for each variable. If at the end of the recursion, before returning a value, we discover that a meta-node with the same variable, the same children and the same weights has already been created, then we don't need to store it and we simply return the existing node. And fourth, if at the end of the recursion we discover we created a redundant node (all children are the same and all weights are the same), then we don't

```
Algorithm 11: APPLY($v_1$; $w_1, \dots, w_m$)
input  : AOMDDs $\mathcal{G}_f^{aomdd}$ with nodes $v_i$ and $\mathcal{G}_g^{aomdd}$ with nodes $w_j$, based on strictly compatible
         pseudo trees $\mathcal{T}_f$, $\mathcal{T}_g$ that can be embedded in $\mathcal{T}$.
         $var(v_1)$ is an ancestor of all $var(w_1), \dots, var(w_m)$ in $\mathcal{T}$.
         $var(w_i)$ and $var(w_j)$ are not in ancestor-descendant relation in $\mathcal{T}$.
output : $v_1 \otimes (w_1 \wedge \dots \wedge w_m)$, based on $\mathcal{T}$.
1  if $H_1(v_1, w_1, \dots, w_m) \neq null$ then return $H_1(v_1, w_1, \dots, w_m)$;        // is in cache
2  if (any of $v_1, w_1, \dots, w_m$ is $\mathbf{0}$) then return $\mathbf{0}$
3  if ($v_1 = \mathbf{1}$) then return $\mathbf{1}$
4  if ($m = 0$) then return $v_1$                                                    // nothing to combine
5  create new nonterminal meta-node $u$
6  $var(u) \leftarrow var(v_1)$ (call it $X_i$, with domain $D_i = \{x_1, \dots, x_{k_i}\}$ )
7  for $j \leftarrow 1$ to $k_i$ do
8      $u.children_j \leftarrow \phi$                        // children of the j-th AND node of $u$
9      $w^u(X_i, x_j) \leftarrow w^{v_1}(X_i, x_j)$                           // assign weight from $v_1$
10     if ( ($m = 1$) and ($var(v_1) = var(w_1) = X_i$) ) then
11         $tempChildren \leftarrow w_1.children_j$
12         $w^u(X_i, x_j) \leftarrow w^{v_1}(X_i, x_j) \otimes w^{w_1}(X_i, x_j)$          // combine input weights
13     else
14         $tempChildren \leftarrow \{w_1, \dots, w_m\}$
15     group nodes from $v_1.children_j \cup tempChildren$ in several $\{v^1; w^1, \dots, w^r\}$
16     for each $\{v^1; w^1, \dots, w^r\}$ do
17         $y \leftarrow$ APPLY($v^1$; $w^1, \dots, w^r$)
18         if ($y = \mathbf{0}$) then
19             $u.children_j \leftarrow \mathbf{0}$; break
20         else
21             $u.children_j \leftarrow u.children_j \cup \{y\}$
22     if ($u.children_1 = \dots = u.children_{k_i}$) and ($w^u(X_i, x_1) = \dots = w^u(X_i, x_{k_i})$) then
23         promote $w^u(X_i, x_1)$ to parent
24         return $u.children_1$                                                  // redundancy
25     if ($H_2(X_i, u.children_1, \dots, u.children_{k_i}, w^u(X_i, x_1), \dots, w^u(X_{k_i}, x_{k_i})) \neq null$) then
26         return $H_2(X_i, u.children_1, \dots, u.children_{k_i}, w^u(X_i, x_1), \dots, w^u(X_{k_i}, x_{k_i}))$
           // isomorphism
27 Let $H_1(v_1, w_1, \dots, w_m) = u$                                              // add $u$ to $H_1$
28 Let $H_2(X_i, u.children_1, \dots, u.children_{k_i}, w^u(X_i, x_1), \dots, w^u(X_{k_i}, x_{k_i})) = u$   // add $u$ to
   $H_2$
29 return $u$
```

store it, and return instead one of its identical lists of children, and promote the common weight.

Note that v_1 is always an ancestor of all $w_1, \dots, w_m$ in $\mathcal{T}$. We consider a variable in $\mathcal{T}$ to be an ancestor of itself. A few self explaining checks are performed in lines 1-4. Line 2 is specific for multiplication, and needs to be changed for other operations. The algorithm creates a new meta-node u, whose variable is $var(v_1) = X_i$ - recall that $var(v_1)$ is highest (closest to root) in $\mathcal{T}$ among $v_1, w_1, \ldots, w_m$. Then, for each possible value of X_i, line 7, it starts building its list of children.

One of the important steps happens in line 15. There are two lists of meta-nodes, one from each original AOMDD f and g, and we will refer only to their variables, as they appear in $\mathcal{T}$. Each of these lists has the important property mentioned above, that its nodes are not ancestors of each other. The union of the two lists is grouped into maximal sets of nodes, such that the highest node in each set is an ancestor of all the others. It follows that the root node in each set belongs to one of the original AOMDD, say v^1 is from f, and the others, say $w^1, \ldots, w^r$ are from g. As an example, suppose $\mathcal{T}$ is the pseudo tree from Fig. 7.7(b), and the two lists are $\{C, G, H\}$ from f and $\{E, F\}$ from g. The grouping from line 15 will create $\{C; E\}$ and $\{F; G, H\}$. Sometimes, it may be the case that a newly created group contains only one node. This means there is nothing more to join in recursive calls, so the algorithm will return, via line 4, the single node. From there on, only one of the input AOMDDs is traversed, and this is important for the complexity of APPLY, discussed below.

7.7.3 Complexity of APPLY

An AOMDD along a pseudo tree can be regarded as a union of regular MDDs, each restricted to a full path from root to a leaf in the pseudo tree. Let $\pi_{\mathcal{T}}$ be such a path in $\mathcal{T}$. Based on the definition of strictly compatible pseudo trees, $\pi_{\mathcal{T}}$ has corresponding paths $\pi_{\mathcal{T}_f}$ in $\mathcal{T}_f$ and $\pi_{\mathcal{T}_g}$ in $\mathcal{T}_g$. The MDDs from f and g corresponding to $\pi_{\mathcal{T}_f}$ and $\pi_{\mathcal{T}_g}$ can be combined using the regular MDD *apply*. This process can be repeated for every path $\pi_{\mathcal{T}}$. The resulting MDDs, one for each path in $\mathcal{T}$ need to be synchronized on their common parts (on the intersection of the paths). The algorithm we proposed does all this processing at once, in a depth first search traversal over the inputs. Based on our construction, we can give a first characterization of the complexity of AOMDD APPLY as being governed by the complexity of MDD apply.

Proposition 26 *Let $\pi_1, \ldots, \pi_l$ be the set of paths in $\mathcal{T}$ enumerated from left to right and let $\mathcal{G}_f^i$ and $\mathcal{G}_g^i$ be the MDDs restricted to path π_i, then the size of the output of AOMDD* apply *is bounded by $\sum_i |\mathcal{G}_f^i| \cdot |\mathcal{G}_g^i| \leq n \cdot max_i |\mathcal{G}_f^i| \cdot |\mathcal{G}_g^i|$. The time complexity is also bounded by $\sum_i |\mathcal{G}_f^i| \cdot |\mathcal{G}_g^i| \leq n \cdot max_i |\mathcal{G}_f^i| \cdot |\mathcal{G}_g^i|$.*

Proof. The complexity of OBDD (and MDD) apply is known to be quadratic in the input. Namely, the number of nodes in the output is at most the product of number of nodes in the input. Therefore, the number of nodes that can appear along one path in the output AOMDD can be at most the product of the number of nodes in each input, along the same path, $|\mathcal{G}_f^i| \cdot |\mathcal{G}_g^i|$. Summing over all the paths in $\mathcal{T}$ gives the result. $\square$

A second characterization of the complexity can be given, similar to the MDD case, in terms of total number of nodes of the inputs:

Proposition 27 *Given two AOMDDs $\mathcal{G}_f^{aomdd}$ and $\mathcal{G}_g^{aomdd}$ based on strictly compatible pseudo trees, the size of the output of* APPLY *is at most $O(\mid \mathcal{G}_f^{aomdd} \mid \cdot \mid \mathcal{G}_g^{aomdd} \mid)$.*

Proof. The argument is identical to the case of MDDs. The recursive calls in APPLY lead to combinations of one node from $\mathcal{G}_f^{aomdd}$ and one node from $\mathcal{G}_g^{aomdd}$ (rather than a list of nodes). The number of total possible such combinations is $O(\mid \mathcal{G}_f^{aomdd} \mid \cdot \mid \mathcal{G}_g^{aomdd} \mid)$. $\square$

We can further detail the previous proposition as follows. Given AOMDDs $\mathcal{G}_f^{aomdd}$ and $\mathcal{G}_g^{aomdd}$, based on compatible pseudo trees $\mathcal{T}_f$ and $\mathcal{T}_g$ and the common pseudo tree $\mathcal{T}$, we define the *intersection pseudo tree* $\mathcal{T}_{f \cap g}$ as being obtained from $\mathcal{T}$ by the following two steps: (1) mark all the subtrees whose nodes belong to either $\mathcal{T}_f$ or $\mathcal{T}_g$ but not to both (the leaves of each subtree should be leaves in $\mathcal{T}$); (2) remove the subtrees marked in step (1) from $\mathcal{T}$. Steps (1) and (2) are applied just once (that is, not recursively). The part of AOMDD $\mathcal{G}_f^{aomdd}$ corresponding to the variables in $\mathcal{T}_\cap$ is denoted by $\mathcal{G}_f^{f \cap g}$, and similarly for $\mathcal{G}_g^{aomdd}$ it is denoted by $\mathcal{G}_g^{f \cap g}$.

Proposition 28 *The time complexity of* APPLY *and the size of the output are $O(|\mathcal{G}_f^{f \cap g}| \cdot |\mathcal{G}_g^{f \cap g}| + |\mathcal{G}_f^{aomdd}| + |\mathcal{G}_g^{aomdd}|)$.*

Proof. The recursive calls of APPLY can generate one meta-node in the output for each combination of nodes from $\mathcal{G}_f^{f \cap g}$ and $\mathcal{G}_g^{f \cap g}$. Let's look at combinations of nodes from $\mathcal{G}_f^{f \cap g}$ and $\mathcal{G}_g^{aomdd} \setminus \mathcal{G}_g^{f \cap g}$. The meta-nodes from $\mathcal{G}_g^{aomdd} \setminus \mathcal{G}_g^{f \cap g}$ that can participate in such combinations (let's call this set $\mathcal{A}$) are only those from levels (of variables) right below $\mathcal{T}_{f \cap g}$. This is because of the mechanics of the recursive calls in APPLY. Whenever a node from f that belongs to $\mathcal{G}_f^{f \cap g}$ is combined with a node from g that belongs to $\mathcal{A}$, line 15 of APPLY expands the node from f, and the node (or nodes) from $\mathcal{A}$ remain the same. This will happen until there are no more nodes from f that can be combined with the node (or nodes) from $\mathcal{A}$, and at that point APPLY will simply copy the remaining portion of its output from $\mathcal{G}_g^{aomdd}$. The size of $\mathcal{A}$ is therefore proportional to $\mid \mathcal{G}_g^{f \cap g} \mid$ (because it is the layer of metanodes immediately below $\mathcal{G}_g^{f \cap g}$). A similar argument is valid for the symmetrical case. And there are no combinations between nodes in $\mathcal{G}_g^{aomdd} \setminus \mathcal{G}_g^{f \cap g}$ and $\mathcal{G}_g^{aomdd} \setminus \mathcal{G}_g^{f \cap g}$. The bound follows from all these arguments. □

Having clarified the APPLY operation, we can now return to the complexity of the **VE-AOMDD** algorithm. Each bucket has an associated bucket pseudo tree. The top chain of the bucket pseudo tree for variable X_i contains all and only the variables in $context(X_i)$. For any other variables that appear in the bucket pseudo tree, their associated buckets have already been processed. The original functions that belong to the bucket of X_i have their scope included in $context(X_i)$, and therefore their associated AOMDDs are based on chains. Any other functions that appear in bucket of X_i are messages received from independent branches below. Therefore, any two functions in bucket of X_i only share variables in the $context(X_i)$, which forms the top chain of the bucket pseudo tree. We can therefore bound the complexity of **VE-AOMDD** and the output size:

THEOREM **7.7.2** *The space complexity of* ***BE-AOMDD****and the size of the output AOMDD are $O(n\ k^{w^*})$, where n is the number of variables, k is the maximum domain size and w^* is the treewidth of the bucket tree. The time complexity is bounded by $O(n\ k^{w^*})$ and $O(r\ k^{w^*})$, where r is the number of initial functions.*

Proof. The space complexity is governed by that of **BE**. Since an AOMDD never requires more space than that of a full exponential table (or a tree), it follows that **BE-AOMDD**only needs space $O(n\ k^{w^*})$. The size of the output AOMDD is also bounded, per layers, by the number of assignments to the context of that layer (namely, by the size of the context minimal AND/OR graph). Therefore, because context size is bounded by treewidth, it follows that the output has size $O(n\ k^{w^*})$. For the time complexity, the APPLY could be modified to combine all the AOMDDs in a bucket at once, rather than two at a time. By an argument similar to that of Proposition 28, the number of recursive calls is proportional to the size of the intersection portion of all the AOMDDs in a bucket, which amounts to that corresponding to the top chain. Even if the common portion can not be reduced at all, and needs to be represented by a tree, the size of that tree would be $O(k^{w^*})$, because the treewidth is equal to the context. The remaining independent portions of the AOMDDs are just copied (or linked through pointers) and need no more processing. Therefore, the total time for such an algorithm would be $O(n\ k^{w^*})$. If we maintain the APPLY to process two AOMDDs at a time, we can observe that we only need r APPLY operations in total. And each such operation is again bounded by $O(k^{w^*})$, therefore the bound $O(r\ k^{w^*})$ follows. □

7.8 AOMDDs Are Canonical Representations

It is well known that OBDDs are canonical representations of Boolean functions given an ordering of the variables [16], and this property extends to MDDs [97]. In the case of AOBDDs and AOMDDs, the canonicity is with respect to a pseudo tree, following the transition from total orders (that correspond to a linear ordering) to partial orders (that correspond to a pseudo tree ordering).

A pseudo tree $\mathcal{T}$ of the variables $\mathbf{X}$ defines a partial order relation $<_{\mathcal{T}}$, where $X_i <_{\mathcal{T}} X_j$ iff X_i is an ancestor of X_j in $\mathcal{T}$.

Many of the algorithms for graphical models are based on a linear ordering d of variables. The structural information captured in the primal graph through the scopes of the functions $\mathbf{F} = \{f_1, \ldots, f_r\}$ can be used to create a natural pseudo tree that corresponds to d. This is precisely the bucket tree (or elimination tree), that is created by **BE**(variables are processed in reverse d). The same pseudo tree can be created by conditioning on the primal graph, and processing variables in the order d, as described in Procedure 9.

We should note that given a graphical model and a pseudo tree $\mathcal{T}$ of it, there may be several linear orderings that correspond to it. In fact, any topological ordering of the variables in $\mathcal{T}$ (where ancestors appear before descendants), is a linear ordering that generates $\mathcal{T}$. Therefore, once we have structural information for the universal function of a graphical model $F = \otimes_i f_i$, given by the scopes of all f_i, then all the linear orderings of $\mathbf{X}$ can be grouped into equivalence classes, based on the pseudo tree that they generate.

We will discuss the canonicity of AOMDD for constraint networks and for general weighted graphical models separately.

7.8.1 Canonicity of AOMDDs for Constraint Networks

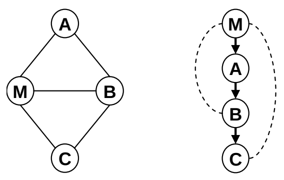
The case of constraint networks is easier, because the weights on the OR-to-AND arcs are only 0 or 1.

THEOREM **7.8.1 (AOMDDs are canonical for a given pseudo tree)** *Given a constraint network, and a pseudo tree $\mathcal{T}$ of its constraint graph, there is a unique (up to isomorphism) AOMDD that represents it, and it has the minimal number of meta-nodes.*

Proof. The proof is by structural induction over the pseudo tree $\mathcal{T}$. □

7.8.2 Canonicity of AOMDDs for Weighted Graphical Models

In this section we raise the issue of recognizing nodes that root AND/OR graphs that represent the same universal function, even though the graphical representation is different. We



M	A	B	f(M,A,B)
0	0	0	12
0	0	1	5
0	1	0	18
0	1	1	2
1	0	0	4
1	0	1	10
1	1	0	6
1	1	1	4

M	B	C	g(M,B,C)
0	0	0	3
0	0	1	5
0	1	0	14
0	1	1	12
1	0	0	9
1	0	1	15
1	1	0	7
1	1	1	6

Figure 7.10: Weighted graphical model

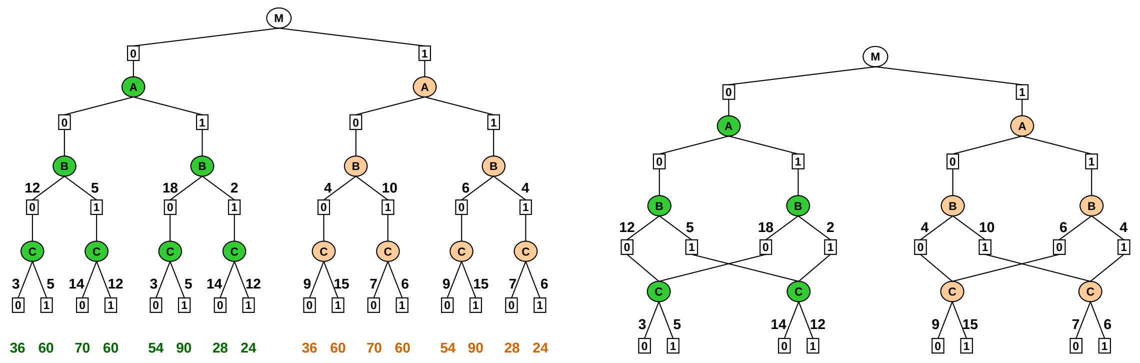


Figure 7.11: AND/OR search tree and context minimal graph

will see that the AOMDD for a weighted graphical model is not unique under the current definitions, but we can slightly modify them to obtain canonicity again. We have to note that canonicity of AOMDDs for weighted graphical models (e.g., belief networks) is far less crucial than in the case of OBDDs that are used in formal verification. Even more than that, sometimes it may be useful not to eliminate the redundant nodes, in order to maintain a simpler semantics of the AND/OR graph that represents the model.

The loss of canonicity of AOMDD for weighted graphical models can happen because of the weights on the OR-to-AND arcs, and we suggest a possible way of re-enforcing it if a more compact and canonical representation is needed.

Example 7.8.2 *Figure 7.10 shows a weighted graphical model, defined by two (cost) functions, $f(M, A, B)$ and $g(M, B, C)$. Assuming the order (M,A,B,C), Figure 7.11 shows the AND/OR search tree on the left. The arcs are labeled with function values, and the leaves show the value of the corresponding full assignment (which is the product of numbers on the arcs of the path). We can see that either value of M (0 or 1) gives rise to the same*

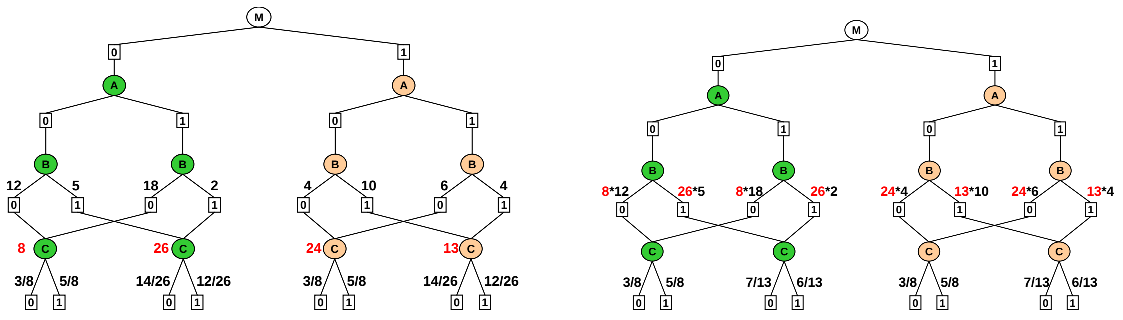


Figure 7.12: Normalizing values bottom up

function (because the leaves in the two subtrees have the same values). However, the two subtrees can not be identified as representing the same function by the usual reduction rules. The right part of the figure shows the context minimal graph, which has a compact representation of each subtree, but does not share any of their parts.

What we would like in this case is to have a method of recognizing that the left and right subtrees corresponding to $M = 0$ and $M = 1$ represent the same function. We can do this by normalizing the values in each level, and processing bottom up. In Figure 7.12 left, the values on the OR-to-AND arcs have been normalized, for each OR variable, and the normalization constant was promoted up to the OR value. In Figure 7.12 right, the normalization constant are promoted upwards again by multiplication. This process does not change the value of each full assignment, and therefore produces equivalent graphs.

We can see already that some of the nodes labeled by C can now be merged, producing the graph in Figure 7.13 on the left. Continuing the same process we obtain the AOMDD for the weighted graph, shown in Figure 7.13 on the right.

THEOREM **7.8.3** *Given two equivalent weighted graphical models that accept a common pseudo tree $\mathcal{T}$, normalizing arc values together with exhaustive application of reduction rules yields the same AND/OR graphs.*

Proof. By structural induction over layers of the graph, bottom up. □

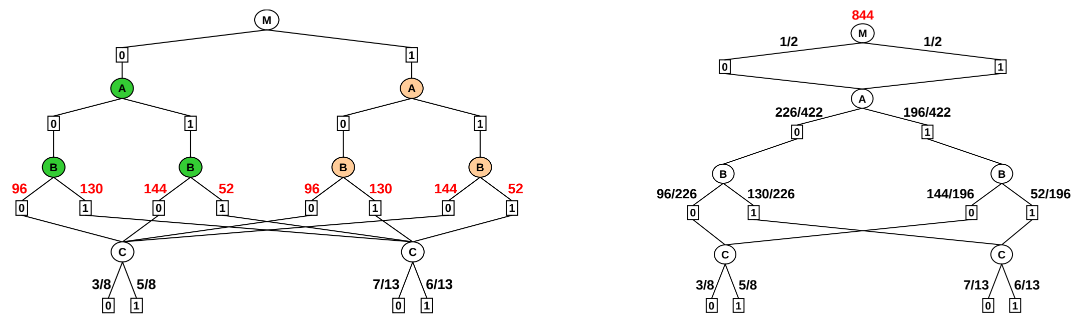
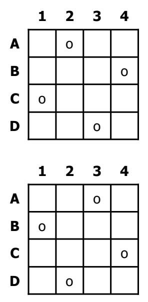
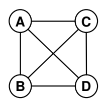


Figure 7.13: AOMDD for the weighted graph



(a) The two solutions



A	B
1	3
1	4
2	4
3	1
4	1
4	2

A	C
1	2
1	4
2	1
2	3
3	2
3	4
4	1
4	3

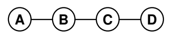
A	D
1	2
1	3
2	1
2	3
2	4
3	1
3	2
3	4
4	2
4	3

B	C
1	3
1	4
2	4
3	1
4	1
4	2

B	D
1	2
1	4
2	1
2	3
3	2
3	4
4	1
4	3

C	D
1	3
1	4
2	4
3	1
4	1
4	2

(b) First model



A	B
2	4
3	1

B	C
1	4
4	1

C	D
1	3
4	2

(c) Second model

Figure 7.14: The 4-queen problem

7.9 Semantic treewidth

A graphical model $\mathcal{M}$ represents a universal function $F = \otimes f_i$. The function F may be represented by different graphical models. Given a particular pseudo tree $\mathcal{T}$, that captures some of the structural information of F, we are interested in all the graphical models that accept $\mathcal{T}$ as a pseudo tree, namely their primal graphs only contain edges that are back-arcs in $\mathcal{T}$. Since the size of the AOMDD for F based on $\mathcal{T}$ is bounded in the worst case by the induced width of the graphical model along $\mathcal{T}$, we define the *semantic treewidth* to be:

DEFINITION **7.9.1 (semantic treewidth)** *The* semantic treewidth *of a graphical model* $\mathcal{M}$ *relative to a pseudo tree* $\mathcal{T}$ *denoted by* $sw_{\mathcal{T}}(\mathcal{M})$*, is defined by* $sw_{\mathcal{T}}(\mathcal{M}) = min_{\mathcal{R},u(\mathcal{R})=u(\mathcal{M})} w_{\mathcal{T}}(\mathcal{R})$*, where* $u(\mathcal{M})$ *is the universal function of* $\mathcal{M}$*, and* $w_{\mathcal{T}}(\mathcal{R})$ *is the induced width of* $\mathcal{R}$ *along* $\mathcal{T}$*. The* semantic treewidth *of a graphical model,* $\mathcal{M}$*, is the minimal semantic treewidth over all the pseudo trees that can express its universal function.*

Computing the semantic treewidth is obviously a hard problem. However, the semantic treewidth can explain why sometimes the minimal AND/OR graph or OBDD are much smaller than the upper bounds exponential in treewidth or pathwidth. In many cases, there could be a huge disparity between the treewidth of $\mathcal{M}$ and the semantic treewidth along $\mathcal{T}$.

Example 7.9.1 *Figure 7.14(a) shows the two solutions of the 4-queen problem. The problem is expressed by a complete graph of treewidth 3, given in Figure 7.14(b). Figure 7.14(c) shows an equivalent problem, which has treewidth 1. The semantic treewidth of the 4-queen problem is 1.*

Based on the fact that AOMDDs are canonical representation of the universal function of a graphical model, we can conclude that the size of the AOMDD is bounded exponentially by the semantic treewidth along the pseudo tree, rather than the treewidth of the given graphical model representation.

Proposition 29 *The size of the AOMDD of a graphical model $\mathcal{M}$ is bounded by $O(n\ k^{sw_{\mathcal{T}}(\mathcal{M})})$, where n is the number of variables, k is the maximum domain size and $sw_{\mathcal{T}}(\mathcal{M})$ is the semantic treewidth of $\mathcal{M}$ along the pseudo tree $\mathcal{T}$.*

Example 7.9.2 *Consider a constraint network on n variables such that every two variables are the equality constraint ($X = Y$). One graph representation is a complete graph, another is a chain and another is a tree. If the problem is specified as a complete graph, and if we use a linear order, the OBDD will have a linear size because there exists a representation having a pathwidth of 1 (rather than n).*

7.10 Related Work

There are various lines of related research. The formal verification literature, beginning with [16] contains a very large number of papers dedicated to the study of BDDs. However, BDDs are in fact OR structures (the underlying pseudo tree is a chain) and do not take advantage of the problem decomposition in an explicit way. The complexity bounds for OBDDs are based on *pathwidth* rather than *treewidth*.

As noted earlier, the work on Disjoint Support Decomposition (DSD) is related to AND/OR BDDs in various ways [8]. The main common aspect is that both approaches show how structure decomposition can be exploited in a BDD-like representation. DSD is focused on Boolean functions and can exploit more refined structural information that is inherent to Boolean functions. In contrast, AND/OR BDDs assumes only the structure conveyed in the constraint graph, they are therefore more broadly applicable to any constraint expression and also to graphical models in general. They allow a simpler and higher level exposition that yields graph-based bounds on the overall size of the generated AOMDD. The full relationship between these two formalisms should be studied further.

McMillan introduced the BDD trees [82], along with the operations for combining them. For circuits of bounded tree width, BDD trees have linear upper space bound $O(|g|2^{w2^{2w}})$, where $|g|$ is the size of the circuit g (typically linear in the number of variables) and w is the treewidth. This bound hides some very large constants to claim the linear dependence on $|g|$ when w is bounded. However, McMillan maintains that when the input function is a CNF expression BDD-trees have the same bounds as AND/OR BDDs, namely they are exponential in the treewidth only.

Darwiche has done much research on compilation, using insights from the AI community. The AND/OR structure restricted to propositional theories is very similar to deterministic decomposable negation normal form (d-DNNF) [25]. More recently, in [55], the trace of the DPLL algorithm is used to generate an OBDD, and compared with the bottom up approach of combining the OBDDs of the input function according to some schedule (as is typical in formal verification). The structures that are investigated are still OR. The idea can nevertheless be extended to AND/OR search. We could run the depth first AND/OR search with caching, generating the *context minimal* AND/OR graph, which can then be processed bottom up by layers to be reduced even further by eliminating isomorphic subgraphs and redundant nodes.

McAllester [80] introduced the case factor diagrams (CFD) which subsume Markov random fields of bounded tree width and probabilistic context free grammars (PCFG). CFDs are very much related to the AND/OR graphs. The CFDs target the minimal representation, by exploiting decomposition (similar to AND nodes) but also by exploiting context sensitive information and allowing dynamic ordering of variables based on context. CFDs do not eliminate the redundant nodes, and part of the cause is that they use zero suppression. There is no claim about CFDs being a canonical form, and also there is no description of how to combine two CFDs.

More recently, independently and in parallel to our work on AND/OR graphs [45, 44], Fargier and Vilarem [46] proposed the compilation of CSPs into tree-driven automata, which have many similarities to our work. Their main focus is the transition from linear automata to tree automata (similar to that from OR to AND/OR), and the possible savings for tree-structured networks and hyper-trees of constraints due to decomposition. Their compilation approach is guided by a tree-decomposition while ours is guided by a variable elimination based algorithm, or by AND/OR search directly. And, it is well known that Bucket Elimination and cluster-tree decomposition are in principle, the same [41].

We see that our work using AND/OR search graphs has a unifying quality that helps make connections among seemingly different compilation approaches.

7.11 Conclusion to Chapter 7

We propose the AND/OR multi-valued decision diagram (AOMDD), which emerges from the study of AND/OR search for graphical models [45, 44, 74] and ordered binary decision diagrams (OBDDs) [16]. This data-structure can be used to compile any set of relations over multi-valued variables as well as any CNF Boolean expression.

The approach we take in this chapter may seem to go against the current trend in model checking, which moves away from BDD-based algorithms into CSP/SAT based approaches. However, constraint processing algorithms that are search-based and compiled data-structures such as BDDs differ primarily by their choices of time vs memory. When we move from regular OR search space to an AND/OR search space the spectrum of algorithms available is improved for all time vs memory decisions. We believe that the AND/OR search space clarifies the available choices and helps guide the user into making an informed selection of the algorithm that would fit best the particular query asked, the specific input function and the available computational resources.

In summary, the contribution of our work is: (1) We formally describe the AOMDD and prove that it is a canonical representation of a constraint network; (2) We describe the APPLY operator that combines two AOMDDs by an operation and give its complexity bounded by the product of the sizes of the inputs; (3) We give a scheduling of building the AOMDD of a constraint network starting with the AOMDDs of its constraints. It is based on an ordering of variables, which gives rise to a pseudo tree (or bucket tree) according to the execution of Bucket Elimination algorithm. This gives the complexity guarantees in terms of the *induced width* along the ordering (equal to the treewidth of the corresponding decomposition); 4) We show how AOMDDs relate to various earlier and recent works, providing a unifying perspective for all these methods.

Chapter 8

Software

The main algorithms described in this dissertation have been implemented in software packages developed in C++. This chapter contains a short overview and description of the implementation and discusses future directions.

8.1 Iterative Algorithms

The algorithms presented in Chapter 4, namely Mini-Clustering (MC) and Iterative Join-Graph Propagation (IJGP), produce approximate results by performing bounded inference on tree decompositions or on the more general join graphs. If allowed enough resources (memory and time), they become exact. IJGP is also an iterative algorithm, that can be viewed as generalized belief propagation.

These algorithms have been implemented by Kalev Kask, with some contributions from the author. The package, called CSP, was originally developed for constraint networks, and then extended to belief networks and mixed networks. CSP can either load problems in standard formats (e.g., *.bif for belief networks, *.erg for CPCS networks), or it can generate random networks based on user defined parameters. For example, it can generate constraint networks defined by: N, the number of variables; k, the domain size; C, the number of constraints; P, the size of a function scope. It can also generate linear block coding networks, based on the length of the block and the channel noise, or noisy-OR networks.

Both MC and IJGP work on a decomposition of the interaction graph, which is derived based on an ordering of the variables. Typically, the heuristic min-fill is used to obtain the ordering, but other options such as min-degree or weighted min-fill are available.

The CSP package was integrated in the REES system (Reasoning Engine(s) Evaluation Shell) developed by Radu Marinescu. REES is a software environment to support research and development in the area of both deterministic and non-deterministic reasoning. REES has a plug-in oriented architecture that promotes reuse of existing software components and allows for the comparison and evaluation of alternative technologies.

All the experiments presented in Chapter 4 were done using the CSP package integrated in REES. For most of the experiments, an exact result was also necessary in order to compute the various measures, such as absolute or relative error, Kullback-Leibler divergence, bit error rate etc. We used a Bucket Elimination algorithm to compute the exact answer.

8.2 AND/OR Search

The AND/OR search algorithms have been implemented from scratch by Radu Marinescu and the author, in a package called AOLIB. The new system uses its own input format file (*.simple), but can potentially load any other usual format file. Also, it can still generate random networks based on user defined parameters.

The AOLIB package contains a family of algorithms, based on a combination of AND/OR search, constraint propagation, AND/OR w-cutset conditioning, memory limit and exact inference (Bucket Elimination). AOLIB is especially suitable for mixed networks, because of the straightforward exploitation of deterministic information.

The pseudo tree that guides the AND/OR search is computed in accordance with the available resources and the user instructions. If only a linear amount of memory is available (tree search), the pseudo tree is optimized for depth. We use a min-depth heuristic that creates a balanced tree decomposition, resulting in a likely small depth. If more memory is available, then the heuristic tends to trade the small depth target for a smaller context size (size of a cache table). If full caching is possible (i.e., the memory available is exponential in the treewidth), then we typically use a weighted min-fill heuristic. We have implemented all these heuristics and reported the results in Chapter 5, where we presented the AND/OR cutset.

The current version of AOLIB runs the Adaptive Caching algorithm, based on the AND/OR cutset idea. We found that it is the most flexible and powerful, taking advantage of the available memory in the most efficient way.

One of the ingredients of AOLIB is the constraint propagation. The user can choose between different levels of consistency enforcing algorithms, from forward checking, arc consistency to relational forward checking. Each of the methods guarantees more pruning of the search space, but at the cost of more computation, therefore an appropriate level should be chosen based on the problem.

Finally, AOLIB offers the possibility to combine AND/OR search and inference algorithms. When the conditoned subproblem has a sufficiently small treewidth, it can be solved exactly. One option is to continue with AND/OR search with full caching. The other would be to solve the conditioned subproblem by an inference-based method, such as Bucket Elimination. They are in principle equivalent as shown in Chapter 6, unless there is determinism. In practice we discovered that inference-based methods tend to have a smaller overhead and are faster in the case of no determinism. The user can choose what method to use for the conditioned subproblem.

The task that AOLIB addresses is #P-hard. It can report the number of valid assignments, or the probability of the evidence.

The AOLIB system participated in the UAI'06 (Uncertainty in Artificial Intelligence) Evaluation of Probabilistic Inference Systems, both for the probability of evidence task (PE) and for maximum probable explanation (MPE). Results are available at *http://ssli.ee.washington.edu/∼bilmes/uai06InferenceEvaluation/*.

8.3 Future Work

The AOLIB package has been the focus of our experimental work. There are a number of future directions that can be explored.

The pseudo tree that guides the AND/OR search can play a crucial role in the efficiency of the algorithms. There is still a lot of potential in investigating new heuristics for the generation of the pseudo tree in the context of Adaptive Caching.

The constraint propagation also plays an important role for mixed networks with substantial deterministic information. The algorithms can be improved by extending the consistency enforcing schemes, possibly by integrating already existing libraries with AOLIB.

The experimental evaluation of AOMDDs (Chapter 7) is still under way. We intend to have a functional AND/OR BDD package (also with the multi-valued version), and to test it on the existing formal verification benchmarks. The Bucket Elimination schedule for compilation is also promising for the case of genetic linkage analysis, where the networks contain a lot of determinism.

All these software packages will be made available online in a short time, on the web page of the research group of Professor Rina Dechter, at the University of California, Irvine (*http://csp.ics.uci.edu/*).

Chapter 9

Conclusion

The research presented in this dissertation is concerned with graphical model algorithms that leverage the structure of the problem. We investigated techniques that capitalize on the independencies expressed by the model's graph by decomposing the problem into independent components, resulting in often exponentially reduced computational costs. The algorithms that we presented can be characterized along three main dimensions: (1) search vs. dynamic programming methods; (2) deterministic vs. probabilistic information; (3) approximate vs. exact algorithms.

The first and main contribution of this dissertation is the introduction of AND/OR search spaces for graphical models. In contrast to the traditional OR search, the new AND/OR search is sensitive to problem decomposition. The linear space AND/OR tree search can be exponentially better (and never worse) than the linear space OR tree search. The AND/OR search graph is exponential in the treewidth w^* of the graph, while the OR search graph is exponential in the pathwidth pw^*, and it is known that $w^* \leq pw^* \leq w^* \log n$, where n is the number of variables. Therefore, the savings with respect to memory intensive schemes are more modest when moving from OR to AND/OR, $O(\exp w^*)$ vs. $O(\exp(w^* \log n))$, but can still be significant in practice when n is large.

The second contribution is the framework of *mixed networks*, a new graphical model that combines belief and constraint networks. By keeping the probabilistic and deterministic information separate we are able to more efficiently exploit them by specific methods. We describe the primary algorithms for processing such networks, based on inference and on AND/OR search. We also present experimental evaluation showing the benefit of exploiting the deterministic information during search, coupled with the efficiency of the AND/OR scheme.

The third contribution is in the area of approximate algorithms for graphical models, and mixed networks in particular. We investigated message-passing schemes based on join tree clustering and belief propagation. We introduced Mini-Clustering (MC), which performs bounded inference on a tree decomposition. We then combine MC with the iterative version of Pearl's belief propagation (IBP), creating Iterative Join-Graph Propagation (IJGP). IJGP is both anytime (controlled by a bounding parameter) and iterative, and we showed empirically that IJGP is one of the most powerful approximate schemes for belief networks. Through analogy with arc consistency algorithms from constraint networks, we proved that IBP and IJGP infer zero-beliefs correctly, and empirically showed that this property also extends to extreme beliefs. This gives an explanation of why and when iterative algorithm perform well, in particular giving a strong explanation of their tremendous performance on coding networks.

The fourth contribution is the application of AND/OR search spaces to the problem of cutset and w-cutset conditioning. We showed that the new concept of *AND/OR cutset* (or w-cutset) is a strict improvement over the old one. Rather than trying to minimize the number of variables that form a cutset, the new method needs to minimize the depth of the pseudo tree that spans the AND/OR cutset. The new method also inspires our most flexible and powerful *Adaptive Caching* algorithm.

The fifth contribution is the creation of a methodology for the comparison of algorithms. Using the context minimal graph traversed by full caching AND/OR search, we compared pure search and pure dynamic programming algorithms for graphical models. We showed that there is no principled difference between AND/OR search and Variable Elimination besides different directions of exploring a common search space (top down vs. bottom up) and different control strategies (depth first vs. breadth first).

The sixth contribution is in the domain of compilation of graphical models. We combined AND/OR search with decision diagrams and created the AND/OR Multi-Valued Decision Diagram (AOMDD), that is sensitive to function decomposition structure. We gave two compilation algorithms for AOMDDs, one search based and the other inference based. Both algorithms and the size of the AOMDD are bounded exponentially by the treewidth, in contrast to the bound exponential in pathwidth known for ordered binary decision diagrams (OBDDs). We also introduced the concept of semantic treewidth, which helps explain why the size of decision diagrams is often much smaller than the worst case bound.

The examples that we used throughout the dissertation were primarily based on constraint networks and belief networks. The reasoning task that we considered was usually #P-hard, or equivalent to solutions counting or belief updating. The exposition of AND/OR search spaces was however at a general level, and therefore easily extendable to other types of graphical models and reasoning tasks (e.g., optimization tasks).

Bibliography

[1] S.B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, 1978.

[2] D. Allen and A. Darwiche. New advances in inference by recursive conditioning. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI'03)*, pages 2–10, 2003.

[3] S. A. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. *BIT*, 25:2–23, 1985.

[4] F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and complexity results for #SAT and bayesian inference. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, pages 340–351, 2003.

[5] F. Bacchus, S. Dalmao, and T. Pitassi. Value elimination: Bayesian inference via backtracking search. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI'03)*, pages 20–28, 2003.

[6] R. Bayardo and D. Miranker. A complexity analysis of space-bound learning algorithms for the constraint satisfaction problem. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, pages 298–304, 1996.

[7] R. J. Bayardo and R. C. Schrag. Using CSP look-back techniques to solve real world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208, 1997.

[8] V. Bertacco and M. Damiani. The disjunctive decomposition of logic functions. In *ICCAD, International Conference on Computer-Aided Design*, pages 78–82, 1997.

[9] B. Bidyuk and R. Dechter. The epsilon-cuset effect in bayesian networks. Technical report, University of California, Irvine, 2001.

[10] B. Bidyuk and R. Dechter. Cycle-cutset sampling for bayesian networks. In *Proceedings of the Sixteenth Canadian Conference on Artificial Intelligence (CAAI'03)*, pages 297–312, 2003.

[11] B. Bidyuk and R. Dechter. On finding minimal w-cutset problem. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI'04)*, pages 43–50, 2004.

[12] D. Bienstock, N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a forest. *J. Combin. Theory Ser. B*, 52:274–283, 1991.

[13] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *The Twenty Second International Symposium on Mathematical Foundations of Computer Science (MFCS'97)*, pages 19–36, 1997.

[14] H. L. Bodlaender and J. R. Gilbert. Approximating treewidth, pathwidth and minimum elimination tree-height. Technical report, Utrecht University, 1991.

[15] R. K. Brayton and C. McMullen. The decomposition and factorization of boolean expressions. In *ISCAS, Proceedings of the International Symposium on Circuits and Systems*, pages 49–54, 1982.

[16] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, 35:677–691, 1986.

[17] J. Cheng and M. Druzdzel. AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large bayesian networks. *Journal of Artificial Intelligence Research (JAIR)*, 13:155–188, 2000.

[18] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[19] Z. Collin, R. Dechter, and S. Katz. On the feasibility of distributed constraint satisfaction. In *Proceedings of the Twelfth International Joint Conferences on Artificial Intelligence (IJCAI'91)*, pages 318–324, 1991.

[20] Z. Collin, R. Dechter, and S. Katz. Self-stabilizing distributed constraint satisfaction. *The Chicago Journal of Theoretical Computer Science*, 3(4), special issue on self-stabilization, 1999.

[21] G. F. Cooper. The computational complexity of probabistic inferences. *Artificial Intelligence*, 42:393–405, 1990.

[22] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, 1993.

[23] A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 125(1-2):5–41, 2001.

[24] A. Darwiche. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 50(3):280–305, 2003.

[25] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research (JAIR)*, 17:229–264, 2002.

[26] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[27] R. Dechter. Constraint networks. *Encyclopedia of Artificial Intelligence*, pages 276–285, 1992.

[28] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference algorithms. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence (UAI'96)*, pages 211–219, 1996.

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

[30] R. Dechter. A new perspective on algorithms for optimizing policies under uncertainty. In *International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pages 72–81, 2000.

[31] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

[32] R. Dechter and Y. El Fattah. Topological parameters for time-space tradeoff. *Artificial Intelligence*, 125:93–188, 2001.

[33] R. Dechter and D. Frost. Backjump-based backtracking for constraint satisfaction problems. *Artificial Intelligence*, 136(2):147–188, 2002.

[34] R. Dechter, K. Kask, and J. Larrosa. A general scheme for multiple lower bound computation in constraint optimization. *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP'01)*, pages 346–360, 2001.

[35] R. Dechter and D. Larkin. Hybrid processing of belief and constraints. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI'01)*, pages 112–119, 2001.

[36] R. Dechter and R. Mateescu. A simple insight into iterative belief propagation's success. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI'03)*, pages 175–183, 2003.

[37] R. Dechter and R. Mateescu. Mixtures of deterministic-probabilistic networks and their AND/OR search space. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI'04)*, pages 120–129, 2004.

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

[39] R. Dechter, R. Mateescu, and K. Kask. Iterative join-graph propagation. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI'02)*, pages 128–136, 2002.

[40] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.

[41] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[42] R. Dechter and J. Pearl. Directed constraint networks: A relational framework for causal reasoning. In *Proceedings of the Twelfth International Joint Conferences on Artificial Intelligence (IJCAI'91)*, pages 1164–1170, 1991.

[43] R. Dechter and I. Rish. A scheme for approximating probabilistic inference. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI'97)*, pages 132–141, 1997.

[44] Rina Dechter and Robert Mateescu. The impact of AND/OR search spaces on constraint satisfaction and counting. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP'04)*, pages 731–736, 2004.

[45] Rina Dechter and Robert Mateescu. Mixtures of deterministic-probabilistic networks and their AND/OR search space. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI'04)*, pages 120–129, 2004.

[46] H. Fargier and M. Vilarem. Compiling CSPs into tree-driven automata for interactive solving. *Constraints*, 9(4):263–287, 2004.

[47] M. Fishelson and D. Geiger. Exact genetic linkage computations for general pedigrees. *Bioinformatics*, 18(1):189–198, 2002.

[48] E. C. Freuder and M. J. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proceedings of the Ninth International Joint Conferences on Artificial Intelligence (IJCAI'85)*, pages 1076–1078, 1985.

[49] E. C. Freuder and M. J. Quinn. The use of lineal spanning trees to represent constraint satisfaction problems. Technical Report 87-41, University of New Hampshire, Durham, 1987.

[50] D. H. Frost. *Algorithms and Heuristics for constraint satisfaction problems*. PhD thesis, Information and Computer Science, University of California, Irvine, 1997.

[51] J. Gergov and C. Meinel. Efficient boolean manipulation with OBDDs can be extended to FBDDs. *IEEE Trans. Computers*, 43:1197–1209, 1994.

[52] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, pages 243–282, 2000.

[53] D. Heckerman. A tractable inference algorithm for diagnosing multiple diseases. In *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence (UAI'89)*, pages 171–181, 1989.

[54] R. A. Howard and J. E. Matheson. *Influence diagrams*, volume 2 of *Readings on the Principles and Applications of Decision Analysis*, pages 719–762. Strategic decisions Group, Menlo Park, CA, USA, 1984.

[55] J. Huang and A. Darwiche. DPLL with a trace: From SAT to knowledge compilation. In *Proceedings of the Nineteenth International Joint Conferences on Artificial Intelligence (IJCAI'05)*, pages 156–162, 2005.

[56] P. G. Jeavons and M. C. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79:327–339, 1996.

[57] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic networks by local computation. *Computational Statistics Quarterly*, 4:269–282, 1990.

[58] K. Kask. *Approximation algorithms for graphical models*. PhD thesis, Information and Computer Science, University of California, Irvine, 2001.

[59] K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying cluster-tree decompositions for reasoning in graphical models. *Artificial Intelligence*, 166 (1-2):165–193, 2005.

[60] L. M. Kirousis. Fast parallel constraint satisfaction. *Artificial Intelligence*, 64:147–160, 1993.

[61] U. Kjæaerulff. Triangulation of graph-based algorithms giving small total state space. Technical report, University of Aalborg, Denmark, 1990.

[62] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proceedings of the Fifteenth National Conference of Artificial Intelligence (AAAI'98)*, pages 580–587, 1998.

[63] J. Larrosa and R. Dechter. Boosting search with variable elimination. *Constraints*, 7(3-4):407–419, 2002.

[64] J. Larrosa, K. Kask, and R. Dechter. Up and down mini-buckets: a scheme for approximating combinatorial optimization tasks. Technical report, University of California, Irvine, 2001.

[65] J. Larrosa, P. Meseguer, and M. Sanchez. Pseudo-tree search with soft constraints. In *Proceedings of the European Conference on Artificial Intelligence (ECAI'02)*, pages 131–135, 2002.

[66] S. L. Lauritzen and D. J. Spiegelhalter. Local computation with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50(2):157–224, 1988.

[67] C.Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38:985–999, 1959.

[68] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[69] M. Mézard, G. Parisi, and R. Zecchina. Analytic and algorithmic solution of random satisfiability problems. *Science*, 297:812–815, 2002.

[70] D. Maier. The theory of relational databases. In *Computer Science Press, Rockville, MD*, 1983.

[71] R. Marinescu and R. Dechter. AND/OR branch-and-bound for graphical models. In *Proceedings of the Nineteenth International Joint Conferences on Artificial Intelligence (IJCAI'05)*, pages 224–229, 2005.

[72] J. P. Marques-Silva and K. A. Sakalla. GRASP: A search algorithm for propositional satisfiability. *IEEE Transaction on Computers*, 48 (5):506–521, 1999.

[73] R. Mateescu and R. Dechter. AND/OR search spaces and the semantic width of constraint networks. In *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP'05), Doctoral Program*, page 860.

[74] R. Mateescu and R. Dechter. The relationship between AND/OR search and variable elimination. In *Proceedings of the Twenty First Conference on Uncertainty in Artificial Intelligence (UAI'05)*, pages 380–387, 2005.

[75] R. Mateescu and R. Dechter. AND/OR cutset conditioning. In *Proceedings of the Nineteenth International Joint Conferences on Artificial Intelligence (IJCAI'05)*, pages 230–235, 2005.

[76] R. Mateescu and R. Dechter. Compiling constraint networks into AND/OR multi-valued decision diagrams (AOMDDs). In *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP'06)*, pages 329–343, 2006.

[77] R. Mateescu and R. Dechter. And/or multi-valued decision diagrams (aomdds) for-weighted graphical models. In *Proceedings of the Twenty Third Conference on Uncertainty in Artificial Intelligence (UAI'07)*, 2007.

[78] R. Mateescu and R. Dechter. A comparison of time-space schemes for graphical models. In *Proceedings of the Twentieth International Joint Conferences on Artificial Intelligence (IJCAI'07)*, pages 2346–2352, 2007.

[79] R. Mateescu, R. Dechter, and K. Kask. Tree approximation for belief updating. In *Proceedings of The Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, pages 553–559, 2002.

[80] D. McAllester, M. Collins, and F. Pereira. Case-factor diagrams for structured probabilistic modeling. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI'04)*, pages 382–391, 2004.

[81] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic, 1993.

[82] K. L. McMillan. Hierarchical representation of discrete functions with application to model checking. In *Computer Aided Verification*, pages 41–54, 1994.

[83] P. J. Modi, W. Shena, M. Tambea, and M. Yokoo. ADOPT: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2005.

[84] L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–177, 1977.

[85] N. J. Nilsson. *Principles of Artificial Intelligence*. Tioga, Palo Alto, CA, 1980.

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

[87] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64:81–129, 1993.

[88] L. Portinale and A. Bobbio. Bayesian networks for dependency analysis: an application to digital control. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI'99)*, pages 551–558, 1999.

[89] D. J. C. MacKay R. J. McEliece and J. F. Cheng. Turbo decoding as an instance of Pearl's belief propagation algorithm. *IEEE J. Selected Areas in Communication*, 1997.

[90] I. Rish and R. Dechter. Resolution versus search: Two strategies for SAT. *Journal of Automated Reasoning*, 24(1/2):225–275, 2000.

[91] N. Robertson and P. Seymour. Graph minors I. Excluding a forest. *J. Combin. Theory, Ser. B*, 35:39–61, 1983.

[92] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, 1996.

[93] T. Sang, F. Bacchus, P. Beam, H. Kautz, and T. Pitassi. Combining component caching and clause learning for effective model counting. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, 2004.

[94] G. R. Shafer and P. P. Shenoy. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2:327–352, 1990.

[95] P. P. Shenoy. Valuation-based systems for bayesian decision analysis. *Operations Research*, 40:463–484, 1992.

[96] D. Sieling and I. Wegener. Graph driven BDDs - a new data structure for boolean functions. *Theoretical Computer Science*, 141:283–310, 1994.

[97] A. Srinivasan, T. Kam, S. Malik, and R. K. Brayton. Algorithms for discrete function manipulation. In *International conference on CAD*, pages 92–95, 1990.

[98] C. Terrioux and P. Jégou. Bounded backtracking for the valued constraint satisfaction problems. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP'03)*, pages 709–723, 2003.

[99] C. Terrioux and P. Jégou. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146:43–75, 2003.

[100] M. Welling and Y. W. Teh. Belief optimization for binary networks: a stable alternative to loopy belief propagation. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI'01)*, pages 554–561, 2001.

[101] Nic Wilson. Decision diagrams for the computation of semiring valuations. In *Proceedings of the Nineteenth International Joint Conferences on Artificial Intelligence (IJCAI'05)*, pages 331–336, 2005.

[102] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Generalized belief propagation. In *Advances in Neural Information Processing Systems 13 (NIPS'00)*, pages 689–695, 2000.

[103] N. L. Zhang, R. Qi, and D. Poole. A computational theory of decision networks. *International Journal of Approximate Reasoning*, 11:83–158, 1994.