Generalized Constraint-Based Inference

by

Le Chang

B.E., Tsinghua University, P.R.China, 2000
M.E., Tsinghua University, P.R.China, 2002

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

April 20, 2005

© Le Chang, 2005
Abstract

Constraint-Based Inference (CBI) is a unified framework that subsumes many practical problems in different research communities. These problems include probabilistic inference, decision-making under uncertainty, constraint satisfaction, propositional satisfiability, decoding problems, and possibility inference. Solving them efficiently is important for both research and practical applications.

Along with the development of inference approaches for concrete CBI problems, researchers are increasingly aware that these problems share common features in representation and essentially identical inference approaches. As the first contribution of this thesis, we explicitly use the semiring concept to generalize various CBI problems into a single formal representation framework with a broader coverage of the problem space based on the synthesis of existing generalized frameworks. Second, the proposed semiring-based unified framework is also a single formal algorithmic framework that provides a broader coverage of both exact and approximate inference algorithms, including variable elimination, junction tree, and loopy message propagation methods. Third, we discuss inference algorithm design and complexity issues based on the abstract representations of CBI problems and inference algorithms. These discussions are examples of applying the abstract knowledge to the concrete application domains. Researchers from various fields can (1) study the most important common characteristics of various CBI problems without representation barriers; (2) analyze and compare different inference approaches; (3) borrow design ideas from other fields and improve the inference approaches' efficiency in their own domains; and (4) significantly reduce the amount of implementation work targeted previously at the individual problems.

Finally, we present a software toolkit named the Generalized Constraint-Based Inference Toolkit in Java (GCBIJ) as the last contribution of this thesis. GCBIJ is the first concrete software toolkit that implements the abstract semiring approach to unify the CBI problem representations and the inference algorithms. Users can design their own task-specific semirings or simply use ones provided by the toolkit to solve their own concrete CBI problems through instantiating various already provided abstract inference algorithms. Users can also design their own inference algorithms on the abstract level and apply them to solve different problems. Furthermore, researchers can test, verify, compare, and analyze inference approaches based on this toolkit. The experimental results based on GCBIJ show that the generalized CBI framework is a useful tool for both research and practical problem-solving.
## Contents

Abstract ................................................................. ii

Contents ............................................................... iii

List of Tables ........................................................ vi

List of Figures ........................................................ vii

List of Algorithms ...................................................... viii

Acknowledgements ...................................................... ix

1 Introduction ........................................................... 1

1.1 Motivation ......................................................... 1

1.2 Related Work ....................................................... 2

1.2.1 Generalized CBI Representations ............................... 2

1.2.2 Generalized Inference Algorithms .............................. 3

1.3 Thesis Outline ..................................................... 3

2 Background ............................................................ 5

2.1 Graphical Models for Constraint-Based Inference .................... 5

2.1.1 Hypergraph Representations .................................... 5

2.1.2 Primal Graph Representations .................................. 6

2.1.3 Junction Tree Representations .................................. 6

2.1.4 Factor Graph Representations .................................. 7

2.2 Topological Parameters to Characterize Graphs ....................... 8

2.2.1 Treewidth and Induced Width ................................... 8

2.2.2 Triangulated Graph and Treewidth Computing ................... 9

2.2.3 Small World and Scale-Free Topology ........................... 10

2.3 The Basis of Abstract Algebra ...................................... 12

2.3.1 Set, Group, and Semigroup .................................... 13

2.3.2 Ring, Semiring, and k-Semiring ................................ 13

2.4 Summary ........................................................... 16

3 A Semiring-Based Generalized Framework for CBI .................... 18

3.1 A Semiring-Based Generalized Framework .......................... 18

3.2 Instances of the Semiring-Based Generalized Framework ............. 20
### Contents

3.2.1 Constraint Satisfaction Problems .............................................. 20
3.2.2 Propositional Satisfiability Problems ....................................... 22
3.2.3 Probability Inference Problems ................................................ 23
3.2.4 Dynamic Bayesian Networks ...................................................... 25
3.2.5 Decision-Making Problems ........................................................ 25
3.2.6 Possibility Inference Problems .................................................. 26
3.2.7 Coordination Graph .................................................................... 27
3.2.8 Maximum Likelihood Decoding ..................................................... 28
3.3 Summary ......................................................................................... 29

4 Exact Inference Algorithms ................................................................. 31
4.1 Variable Elimination Algorithm ....................................................... 31
  4.1.1 Generalized Variable Elimination Algorithm ............................... 31
  4.1.2 Generalized VE Algorithm for k-Semiring .................................. 34
  4.1.3 Instances of Variable Elimination Algorithm .............................. 34
  4.1.4 Space and Time Complexity ..................................................... 35
  4.1.5 Discussion of Applying the VE Algorithm .................................... 36
4.2 Junction Tree Algorithm .................................................................. 36
  4.2.1 Generalized JT Algorithm ......................................................... 36
  4.2.2 Generalized JT Algorithm for k-Semiring .................................. 39
  4.2.3 Generalized JT Algorithm for Idempotent Semirings .................... 39
  4.2.4 Generalized JT Algorithm for Invertible Semirings ...................... 41
  4.2.5 Instances of Junction Tree Algorithm ......................................... 42
  4.2.6 Space and Time Complexity ..................................................... 46
  4.2.7 Discussion of Applying the JT Algorithm ..................................... 52
4.3 Summary ......................................................................................... 54

5 Approximate Inference Algorithms .................................................... 57
5.1 Motivation for the Design of Approximate Algorithms ...................... 57
5.2 Algebraic Approximation for VE and JT ......................................... 57
  5.2.1 Approximate Variable Elimination Algorithm .............................. 58
  5.2.2 Approximate Junction Tree Algorithm ....................................... 60
  5.2.3 Discussion ............................................................................... 60
5.3 Thin Junction Tree Algorithm .......................................................... 61
  5.3.1 Thinning through Variable Contraction ...................................... 62
  5.3.2 Generalized Thin Junction Tree Algorithm .................................. 63
  5.3.3 Discussion ............................................................................... 64
5.4 Loopy Message Propagation ............................................................ 64
  5.4.1 Building a Junction Graph ......................................................... 65
  5.4.2 Generalized Loopy Message Propagation Algorithm .................... 66
  5.4.3 Loopy Message Propagation As an Exact Inference ..................... 67
  5.4.4 Discussion ............................................................................... 68
5.5 Hybrid Junction Tree Algorithm ....................................................... 68
5.6 Summary ......................................................................................... 69
6 Generalized CBI Toolkit in Java - GCBIJ ........................................ 71
6.1 GCBIJ Motivation .............................................................. 71
6.2 GCBIJ System Architecture .................................................. 72
6.3 GCBIJ Implementation and Interfaces ...................................... 74
  6.3.1 Semiring ................................................................. 74
  6.3.2 Parser ................................................................. 75
  6.3.3 CBITask ............................................................... 76
  6.3.4 Inference ............................................................ 76
  6.3.5 Graph ............................................................... 79
  6.3.6 Utilities ............................................................ 80
  6.3.7 GCBIJ.Main ......................................................... 81
6.4 Summary ........................................................................ 81

7 GCBIJ Applications: Case Studies .............................................. 83
  7.1 Test Suites .................................................................. 83
  7.2 Studies of Heuristic Triangulation Algorithms ...................... 83
  7.3 Empirical Studies of Exact Inference Algorithms .................... 86
    7.3.1 Exact Inference Algorithms with Invertible Semiring .......... 86
    7.3.2 Exact Inference Algorithms with Idempotent Semiring ....... 88
    7.3.3 Exact Inference Algorithms for General CBI Problems ....... 89
  7.4 Empirical Studies of Approximate Inference Algorithms .......... 89
    7.4.1 Approximate VE and JT for CSPs ................................. 89
    7.4.2 Approximate VE and JT for Probability Inference .......... 92
  7.5 Loopy Message Propagation As Exact Inference Algorithm ....... 95
  7.6 Impact of Small World Topology ....................................... 96
    7.6.1 Small World Topology of CBI problems ....................... 96
    7.6.2 Inference in the Small World .................................... 99
  7.7 Summary .................................................................. 101

8 Conclusions and Future Work .................................................. 102
  8.1 Conclusions .................................................................. 102
  8.2 Possible Improvements to Inference Approaches .................... 103
    8.2.1 From Variables to Values ........................................ 103
    8.2.2 From Systematic to Stochastic ................................... 105
  8.3 Future Work .................................................................. 105

Bibliography ........................................................................... 107

A Heuristics Search Algorithms for Triangulation .......................... 113
  A.1 Minimal Width Heuristic Triangulation ............................... 113
  A.2 Minimal Induced Width Heuristic Triangulation ................. 114
  A.3 Minimal Fill-in Heuristic Triangulation ............................. 115
  A.4 Minimal Induced Fill-in Heuristic Triangulation ................. 116
  A.5 Lexicographic BFS Heuristic Triangulation, variant Perfect ... 117
  A.6 Lexicographic BFS Heuristic Triangulation, variant Minimal ... 118
  A.7 Maximum Cardinality Heuristic Triangulation .................... 119
## List of Tables

2.1 Heuristic Triangulation Algorithms ........................................ 11  
2.2 Summary of Different Commutative Semirings ............................ 16  

4.1 Computation Costs of Two Equivalent Tasks ............................ 31  
4.2 VE and JT Complexity Comparisons ....................................... 53  
4.3 Upper Bounds of VE and JT Running Times .............................. 54  
4.4 Running Times for Different Operations in Java ....................... 54  
4.5 Concrete VE and JT Algorithms in Different Fields .................... 56  

6.1 Public Methods of the Class *Semiring* .................................. 74  
6.2 Public Methods of the Interface *Idempotent* ........................... 75  
6.3 Public Methods of the Interface *IReversible* .......................... 75  
6.4 Public Methods of the Class *Parser* .................................... 76  
6.5 Public Methods of the Class *InferAlg* .................................. 77  
6.6 Public Methods of the Class *ApproxInferAlg* ......................... 79  
6.7 Public Methods of the Class *Cluster* .................................... 82  

7.1 The Basic Information of Test Suites .................................... 84  
7.2 Upper Bounds of Treewidth for Graphs, Different Heuristic Searches 85  
7.3 Running Times of Exact Inference Algorithms with Inverses ........ 88  
7.4 Running Times of Exact Inference Algorithms with Idempotency ..... 89  
7.5 Running Times of General Exact Inference Algorithms ............... 89  
7.6 The Maximum Number of Satisfiable Constraints for a CSP ........... 90  
7.7 Parameters of the Small World Topology for Graphs .................. 98  

8.1 Detecting Elements of Some Commutative Semirings .................... 105
List of Figures

2.1 Graphical Representations of an Example CBI Problem .......... 7
2.2 Axioms in the Definitions of Group and Related Concepts ..... 14
2.3 Axioms in the Definitions of Ring and Related Concepts ..... 15

5.1 Graphical Interpretation of Approximate VE. ................. 59
5.2 Graphical Interpretation of Approximate JT. ................. 60

6.1 System Architecture of GCBIJ .................................. 73

7.1 Upper Bounds of Treewidth, Based on Various Heuristics .... 86
7.2 Bayesian Network of the Diagnosis Problem .................. 87
7.3 Accuracy and Running Time of the Approx. VE for a CSP .... 91
7.4 Accuracy and Running Time of the Approx. JT for a CSP .... 92
7.5 Bayesian Network of the Insurance Problem .................. 93
7.6 Marginal of the Approx. VE for a Probability Inference Problem 94
7.7 Marginal of the Approx. JT for a Probability Inference Problem 95
7.8 Accuracy and Time Complexity of loopy MP for a CSP ....... 96
7.9 Small World Parameter Distributions for the Pigs Network .... 97
7.10 Small World Parameter Distributions for the Link Network ... 99
7.11 Small World Parameters for Different Re-written Probabilities 100
7.12 Time Complexity of loopy MP in the Small World .......... 100
List of Algorithms

2.1 Triangulating a Graph, Given an arbitrary Ordering (RAND) . . 10
2.2 Small World Modelling [59] ........................................... 12
2.3 Scale-Free Modelling .................................................. 12
4.1 Generalized Variable Elimination Algorithm (GVE) .......... 33
4.2 Generalized Bucket Elimination Algorithm (GBE) .......... 33
4.3 Generalized VE Algorithm for k-Semiring (kGVE) ........ 34
4.4 Generalized JT Algorithm (GJT) ................................ 38
4.5 Generalized JT Algorithm for k-Semiring (kGJT) ........ 40
4.6 Generalized JT Algorithm for Idempotent Semirings (GJT-Idemp) .......................................................... 41
4.7 Generalized JT Algorithm for Invertible Semirings (GJT-Inv) 43
4.8 Generalized JT Algorithm for LS Architecture (GJT-LS) ... 44
4.9 Generalized JT Algorithm for HUGIN Architecture (GJT-HUGIN) 45
5.1 The Approximate Marginal of Combination Algorithm (ApproxMC) 58
5.2 Generalized Approximate VE Algorithm (GVE-Approx) .... 59
5.3 Generalized Approximate JT Algorithm (GJT-Approx) ..... 61
5.4 Variable Contraction (Contract(v, C_i)) ......................... 62
5.5 Generalized Thin Junction Tree Algorithm (GThinJT) .... 63
5.6 Building a Junction Graph for a CBI Problem ............... 65
5.7 Generalized Loopy Message Propagation Algorithm (GLoopy) 66
5.8 Generalized Loopy Message Propagation Algorithm with Inverses (GLoopy-Inv) .................................................. 67
5.9 Hybrid Junction Tree Algorithm (HybridJT) ................. 70
A.1 Minimal Width Heuristic Triangulation (MW) [47] ......... 113
A.2 Minimal Induced Width Heuristic Triangulation (MIW) [37] 114
A.3 Minimal Fill-in Heuristic Triangulation (MF) [47] ............ 115
A.4 Minimal Induced Fill-in Heuristic Triangulation (MIF) [37] 116
A.5 Lexicographic BFS Heuristic Triangulation, variant Perfect (LEXP) [51] .......................................................... 117
A.6 Lexicographic BFS Heuristic Triangulation, variant Minimal (LEXM) [51] .......................................................... 118
A.7 Maximum Cardinality Heuristic Triangulation (MC) [56] .. 119
I would like to thank my supervisor, Professor Alan Mackworth, for his guidance and support in this work. I would not be here without his inspiration and encouragement. Many thanks also to Professor David Poole and Professor Kevin Leyton-Brown for their valuable comments. I would also like to thank Ben D’Andrea for his hard work on my thesis proof-reading. Especially, I would like to give my special thanks to my wife Ming Yan whose patient love enabled me to complete this work.
Chapter 1

Introduction

1.1 Motivation

Constraint-Based Inference (CBI) is a general term covering many different problems in several research communities. It consists of discovering new constraints from a set of given constraints over individual entities. These new constraints reveal previously undiscovered properties of those entities.

Practical problems from many different fields can be seen as constraint-based inference problems, including probabilistic inference, decision-making under uncertainty, constraint satisfaction problems, propositional satisfiability, decoding problems, and possibility inference. A constraint here can be seen as a function that maps possible value assignments to a specific value domain.

Along with the development of inference approaches for concrete CBI problems, researchers are increasingly aware that these problems share common abstract representation features. Many inference algorithms, described differently, implicitly have essentially identical ideas underlying them. Understanding the common features and characteristics of CBI representations helps research communities exchange ideas and design more efficient inference algorithms.

The purpose of this thesis is to use the algebraic semiring concept to generalize a wide range of CBI problems and different exact and approximate inference algorithms into a single formal framework based on the synthesis of existing generalized representation and algorithmic framework from different fields. We aim to analyze the common characteristics of inference algorithms based on the abstract problem representation, then apply the abstract knowledge learnt from the unified framework to improve the concrete inference algorithm design in specific application domains.

To demonstrate the representation power of the proposed semiring-based unified framework, a toolkit named Generalized Constraint-Based Inference Toolkit in Java (GCBIJ) is implemented. It can be used to verify, test, compare, and analyze generalized approaches for CBI problems. The flexible architecture of GCBIJ enables implemented generalized inference algorithms to be applied to solve concrete problems simply through specifying different semirings. GCBIJ's extensibility enables users to design their own task-specific semirings and use available generalized inference algorithms. This not only demonstrates the feasibility of using semirings to unify the CBI problem representations and the various inference algorithms, but also shows that GCBIJ is a suitable platform for both research and practical problem-solving.
1.2 Related Work

Generalized representation and inference algorithms for CBI problems have been studied for the past ten years. All of these studies are based on the following observation: there are two essential operations in constraint-based inference: (1) combination, which corresponds to an aggregation of constraints, and (2) marginalization, which corresponds to the projection of a specified constraint onto a narrower domain. Different generalized representations express these two operations through various tools or notions. Given the distributivity property of the two operations, generalized inference algorithms can be abstracted that exploit that property. The semiring-based unified framework proposed in this thesis is developed based on the synthesis of these generalized representation and algorithmic frameworks.

1.2.1 Generalized CBI Representations

Semiring-based CSP (SCSP) [6] is a generalized representation framework for soft constraint programming, which includes max CSPs, fuzzy CSPs, weighted CSPs, and probabilistic CSPs. It generalizes these soft constraint programming problems, as well as classic CSPs, into a single abstract framework. The c-semiring, a commutative semiring with the additive idempotency property, is the key notion in SCSP that represents both the combination and marginalization operations. The additive idempotency property excludes general CBI problems like probabilistic inference from the SCSP framework. Given that the CSP is an instance of the generalized CBI problem, the SCSP framework inspired us to propose the semiring-based unified framework in this thesis. In addition to the generalized representation of hard and soft CSPs, SCSP also generalizes the soft local consistency approaches. Extended from local consistency techniques for classic CSPs, generalized soft local consistency can be instantiated to solve a wide range of soft constraint programming problems. The success of the SCSP framework [5] demonstrates that a generalized problem representation provides opportunities to migrate an existing approach for one problem to solve another problem, if the two problems have the same abstract representation. That is our motivation for proposing a semiring-based unified framework to represent CBI problems from other disciplines as well as constraint programming.

The valuation algebra framework [35] is an abstract framework for describing probabilistic inference problems in expert systems. It is inspired by the formulation of simple axioms that solve global problems through local computing. Knowledge or information in the framework is called valuation. Inferences are made by computing the marginal of a combination of several valuations. The framework of the valuation algebra is abstracted to include many different formalisms. Bayesian networks, Dempster-Shafer belief function models, Spohn's epistemic belief models, and possibility theory are shown [35] to fit into the valuation algebra framework. The authors also suggested that constraint satisfaction problems, propositional logic, and discrete optimization problems could be expressed in the valuation algebra framework.
1.2.2 Generalized Inference Algorithms

Many algorithms have been proposed for probabilistic inference. The variable elimination algorithm [64] and the junction tree algorithms [55] [39] [29] were among the first inference algorithms for probability inference in Bayesian networks. Originating in dynamic programming, the variable elimination and junction tree algorithms explicitly use the distributivity of arithmetic additive and multiplicative operations. The generalization of the bucket elimination algorithm [14], a variant of variable elimination, can be used in belief assessment, most probable explanation, maximum a posteriori hypothesis, and maximum expected utility problems. The bucket elimination approach was also applied in constraint satisfaction problems. It was proved that the generalized bucket elimination algorithm is applicable to both probabilistic and deterministic inference [15].

The Generalized Distributive Law (GDL) [1] is a general message-passing algorithm, a synthesis of work in the information theory, signal processing, statistics, and artificial intelligence communities. GDL generalizes the computational problems as "Marginalize a Product Function" (MPF) problems through using commutative semirings to represent the combination and marginalization operations. GDL can be seen, in the AI community, as a junction tree algorithm. As a semiring-based generalized inference algorithm, GDL can represent a wide range of inference algorithms.

More recently, a unified algorithmic framework was proposed [32] to represent tree-decomposition algorithms as a generalized bucket-tree elimination algorithm for automated reasoning tasks. Different from GDL, the generalized bucket-tree elimination does not rely on semirings. Abstract concepts are used to represent combination and marginalization operations, though the distributivity is also the key idea behind the algorithm. Using those concepts, the message-passing within the tree structure can be abstractly represented. The authors also suggested using their generalized algorithmic framework for constraint satisfaction problems.

1.3 Thesis Outline

This thesis is organized as follows: related definitions and background for CBI problems, graphical models, and abstract algebra are introduced in Chapter 2. In Chapter 3 we present a semiring-based unified framework for CBI problems. We also show that a series of problems from different fields can be seen as instances of our proposed framework. In Chapter 4, two popular exact inference algorithms, variable elimination algorithm and junction tree algorithm, are incorporated into the semiring-based unified framework. Instances of these algorithms are discussed. Theoretical and empirical complexities of these algorithms are also discussed. It is well known that complexities of exact inference algorithms are exponential in the parameters of their underlying graphical representations. For many practical CBI problems with intractable parameters, we
will resort to approximate techniques. We discuss approximate inference algo-
rithms within our abstract framework in Chapter 5. To put the ideas of the
semiring-based unified framework into concrete form, a Generalized Constraint-
Based Inference Toolkit in Java (GCBIJ) is designed and implemented. We
present the design specification of GCBIJ in Chapter 6. We also discuss the
use and limitations of GCBIJ in this chapter. In Chapter 7, a series of exper-
iments are conducted based on GCBIJ platform, which include problems from
probability inference, CSP, SAT, and MaxSAT. Although our experimental re-
sults are not totally new for research communities, as a whole they verify the
feasibility of using the semiring to generalize the representations of various CBI
problems. Generalized exact and approximate inference algorithms, based on
the proposed semiring-based unified framework, are proven suitable for solving
concrete applications. Finally, conclusions and future work appear in Chapter
8.
Chapter 2

Background

2.1 Graphical Models for Constraint-Based Inference

Generally speaking, inference means discovering previously unknown facts from a set of given facts (or knowledge). In an inference problem, knowledge can be represented by a set of (hard or soft) constraints on individual entities. Usually individual entities are represented by variables. In this thesis, a constraint-based inference problem is characterized in terms of a set of variables with values in a finite domain and a set of constraints on these variables. The inference task for a CBI problem is then to make explicit new constraints over some variables, based on the given constraints. Here the new constraints could specify the property of one variable (unary), several variables, or the whole problem. An example of CBI problem is given in Example 2.1. The formal definitions of a CBI problem and related tasks appear in Chapter 3.

Example 2.1 Consider a constraint-based inference problem with 5 variables $V_1, \ldots, V_5$, $V_i \in \{0,1\}$. There are 3 constraints defined over these variables: $f_1(V_1, V_2, V_3)$, $f_2(V_2, V_3, V_4)$ and $f_3(V_5, V_3)$, which specify the set of tuples permitted by these constraints respectively. An inference task in this example is to discover tuples permitted by the derived constraint over $V_2$ and $V_3$.

In the following chapters of this thesis, we use uppercase letters to denote variables and lowercase letters to denote values in the domain of a variable. Constraints over variables, usually represented as functions, are denoted by lowercase letters as well. Given a function $f$, $\text{Scope}(f)$ denotes the subset of variables that appear in the domain of $f$. For convenience, we also use uppercase letters to denote subsets of variables. The meaning of uppercase letters can be easily distinguished given the context.

2.1.1 Hypergraph Representations

In many cases, we can use graphs to represent CBI problems. The most straightforward graphical representation is the hypergraph representation, where nodes represent variables and hyperarcs represent constraints.

Given a CBI problem as described in Example 2.1, the corresponding hypergraph representation is shown in Figure 2.1(a).
2.1.2 Primal Graph Representations

Although the hyper-graph representation is straightforward, it is hard to represent practical CBI problems using hyperarcs in graphical representations. In a primal graph the hyperarcs in a hypergraph are replaced by cliques. The primal graph representation of a CBI problem is an undirected graph $G = (V, E)$, where $V = \{V_1, \ldots, V_n\}$ is a set of vertices, each vertex $V_i$ corresponding to a variable $X_i$ of the CBI problem; and $E = \{(V_i, V_j) \mid V_i, V_j \in V\}$ is a set of edges between $V_i$ and $V_j$. There exists an edge $(V_i, V_j)$ if and only if corresponding variables $X_i$ and $X_j$ appear in the scope of the same constraint. A moralized graph of the Bayesian Network (BN), which is obtained by adding edges among vertices with the common child vertex in the corresponding BN, is an example of a primal graph representation of probabilistic inference problems. A constraint graph of binary CSP is another example of a primal graph representation. In this thesis, we use $Neighbors(V_i)$ to denote the set of vertices that have edges to $V_i$ and $Family(V_i) = Neighbors(V_i) \cup \{V_i\}$.

Given a CBI problem as described in Example 2.1, the corresponding primal graph is shown in Figure 2.1(b).

2.1.3 Junction Tree Representations

Sometimes the primal graph of a CBI problem is re-organized as a secondary structure to achieve better computational efficiency. The junction tree is a widely used secondary structure in graphical models. A junction tree is an undirected graph $T = (C, S)$. $C = \{C_1, \ldots, C_n\}$ is a set of clusters, where each cluster $C_i$ corresponds to an aggregation of a subset of vertices $V_{C_i} \subseteq V$ in the primal graph $G = (V, E)$. $S = \{S_{ij}, \ldots, S_{im}\}$ is a set of separators between clusters, where $S_{ij}$ is the separator of clusters $C_i$ and $C_j$, corresponding to the vertices of $V_{C_i} \cap V_{C_j}$. In addition, the following junction tree properties have to be satisfied:

1. **Singly connected property:** $T = (C, S)$ is a tree;

2. **Running intersection property:** $\forall C_i, C_j \in C$, $V_{C_i} \cap V_{C_j} \subseteq V_{C_k}$ holds for any cluster $C_k$ on the path between $C_i$ and $C_j$;

3. **Constraint allocation property:** For any constraint $f$ of the CBI problem, $\exists C_i \in C$ s.t. $Scope(f) \subseteq C_i$.

Typically a junction tree is undirected. In some computational schemes, we can pick one cluster as the root of the tree and assign directions to all separators. A separator $S_{ij} = (C_i, C_j)$ has a direction from $C_i$ to $C_j$ if $C_i$ is in the path from the root to $C_j$. For each cluster $C_i$, $Parent(C_i)$ denotes the cluster that points to $C_i$; $Children(C_i)$ denotes the set of clusters that $C_i$ points to. Also, we use $Root(T)$ to denote the root of junction tree $T$.

Given a CBI problem as described in Example 2.1, a corresponding junction tree representation is shown in Figure 2.1(c).
2.1.4 Factor Graph Representations

The factor graph [38] is another graphical representation for CBI problems, widely used in information theory research. A factor graph is a bipartite graph that expresses the structure of the factorization. A factor graph has a variable node for each variable, a factor node for each constraint, and an edge connecting a variable node to a factor node if and only if the variable is in the scope of the constraint.

Given a CBI problem as described in Example 2.1, the corresponding factor graph representation is shown in Figure 2.1(d).

Figure 2.1: Graphical Representations of an Example CBI Problem. (a) Hypergraph (b) Primal graph (c) Junction tree (d) Factor graph
2.2 Topological Parameters to Characterize Graphs

In the following chapters we will find that the success of different inference algorithms for the inference task for a CBI problem largely depends on the characteristics of its corresponding graphical representation. In this section, related topological parameters to characterize a graph are introduced.

2.2.1 Treewidth and Induced Width

Treewidth, or equivalently, induced width \([22]\), has been proved to be an important notion in computational complexity theory. Many \(NP\)-hard graph problems, such as the maximum independent set problem and the Hamiltonian cycle problem, can be solved in polynomial time for graphs with treewidth bounded by a constant. We will show in the following chapters that the time and space complexities of generalized inference algorithms for CBI problems are polynomial in the size of graphical representations with bounded treewidth and have a constant factor which is exponential in the treewidth of primal graphs.

**Induced Width**

**Definition 2.2 (Induced Width)**

Given an undirected graph \(G = (V, E)\) and an ordering \(\sigma = (V_1, \ldots, V_n)\) of vertices in \(V\), the ordered graph is a pair \((G, \sigma)\).

The width of a vertex in an ordered graph \((G, \sigma)\) is the number of its lower-indexed neighbors in \(\sigma\). The width of an ordered graph, denoted by \(w(\sigma)\), is the maximal width of all vertices.

The induced graph of an ordered graph \((G, \sigma)\), denoted by \((G^*, \sigma)\), is obtained by processing the vertices in \(V\) recursively, from the vertex with the largest index to the vertex with the smallest index; when vertex \(V_i\) is processed, all its lower-indexed neighbors are connected to each other.

The induced width \(w^*(\sigma)\) of an ordered graph \((G, \sigma)\) is the width of the induced graph \((G^*, \sigma)\).

The induced width of a graph \(G\), denoted by \(w^*(G)\), is the minimum induced width for all possible orderings of \(G\).

According to the definition, computing the induced width of a graph \(G\) is to find an ordering \(\sigma\) with a minimum induced width of the ordered graph \((G, \sigma)\). In this thesis, we also call such an ordering an elimination ordering, because it corresponds to eliminating a vertex after connecting all its lower-indexed neighbors.

**Treewidth**

Treewidth \([50]\) is the measure for the study of tree decomposition in the context of graph minor research. It is proved to be an equivalent concept of induced width \([22]\).
Chapter 2. Background

Definition 2.3 (Treewidth, Robertson and Seymour [50])

A tree decomposition of graph $G = (V, E)$ is a pair $(T, \mathcal{X})$, where $T = (C, S)$ is a tree with vertex set $C$ and edge set $S$, and $\mathcal{X} = \{X_c, c \in C\}$ is a family of subsets of $V$, one for each vertex of $T$, such that

1. $\bigcup_{c \in C} X_c = V$,
2. for every edge $(v_i, v_j) \in E$, there is a $c \in C$ with $v_i \in X_c$ and $v_j \in X_c$, and
3. for all $i, j, k \in C$, if $k$ is on the path from $i$ to $j$ in $T$, then $X_i \cap X_j \subseteq X_k$.

The width of a tree decomposition is $\max_{c \in C}|X_c| - 1$. The treewidth of a graph $G = (V, E)$, denoted by $\omega^*(G)$, is the minimum width over all possible tree decompositions of $G$.

Once these definitions are compared, it is obvious that the junction tree representation is a tree decomposition of the primal graph of a given CBI problem.

2.2.2 Triangulated Graph and Treewidth Computing

A graph is triangulated if every cycle of length at least four contains a chord, that is, two non-consecutive vertices on the cycle are adjacent. Triangulated graphs are also called chordal graphs due to the existence of a chord in every cycle. The triangulated graph of graph $G = (V, E)$ is obtained by adding edges $E_t$ to $G$ such that $G_t = (V, E \cup E_t)$ is a triangulated graph.

There exists an equivalent relation between treewidth of a graph and maximum clique size in its triangulated graph [8]:

Theorem 2.4 (Bodlaender [8])

Let $G = (V, E)$ be a graph, and let $k \geq 0$. The following statements are equivalent:

1. The treewidth of $G$ is at most $k$.
2. The maximum clique size of an triangulated graph of $G$ is at least $k + 1$.

Corollary 2.5 Computing the treewidth of graph $G$ is equivalent to finding an optimal triangulated graph of $G$, with minimum max-clique size.

Computing the treewidth or the induced width of a graph, as well as finding an optimal triangulated graph, is $NP$-hard [2] in general. For fixed $k$, there exists some linear time algorithms [7] to determine whether a graph has treewidth $k$. However, these algorithms are very limited because the running time contains a large constant that is exponential in $k$. There also exist some approximate algorithms with factors $4, 4\frac{1}{2}$ and $O(\log(n))$ [21] which can be calculated in polynomial time, even though they contain a large constant which is exponential in $k$ as well.

The maximum clique size (minus one) of any optimal triangulated graph of $G$ provides an upper bound of the treewidth of $G$. Although having no guarantee of
Algorithm 2.1 Triangulating a Graph, Given an arbitrary Ordering (RAND)

Input: A connected graph $G = (V, E)$, an ordering $\sigma = (v_1, \cdots, v_n)$

Output: Triangulated graph $G_t = (V, E \cup E_t)$ with $w^*(G) \leq w^*(G_t) = w$

1: for each $v \in V$ do
2: $M(v) := \text{Neighbor}(v)$
3: end for
4: $E_t := \emptyset$, $w := 0$
5: for $i = 1$ to $i = |V|$ do
6: if $|M(v_i)| > w$ then
7: $w := |M(v_i)|$
8: end if
9: for each $x, y \in M(v_i)$ and $(x, y) \notin E \cup E_t$ do
10: $E_t := E_t \cup (x, y)$
11: $M(x) := M(x) \cup y$ and $M(y) := M(y) \cup x$
12: end for
13: for each $u \in M(v_i)$ do
14: $M(u) := M(u) \setminus v_i$
15: end for
16: end for

returning an optimal triangulation, we can use heuristic triangulation algorithms (which are linear and do not contain exponentially large constants) to get the upper bounds of the treewidth.

Given an arbitrary elimination ordering $\sigma = (v_1, \cdots, v_n)$, a graph $G = (V, E)$ can be triangulated according to the procedure in Algorithm 2.1 [47]:

Appendix A lists several heuristic triangulation algorithms. Basically they use various heuristics to find an elimination ordering, then triangulate the graph according to Algorithm 2.1. The time complexities of these algorithms are compared in Table 2.1. $n = |V|$ is the number of vertices and $m' = |E| + |E_t|$ is the number of edges in the triangulated graph. Chapter 7 presents empirical comparison of different heuristic triangulation algorithms. Details of heuristic triangulation algorithms and their empirical evaluations can be found in [33].

2.2.3 Small World and Scale-Free Topology

Small world topology [59] exists widely in graphical representations of many real world problems. In a graph with small world topology, vertices are highly clustered yet the path lengths between them are small. By comparison, random graphs with a similar number of vertices and edges have short path lengths but little clustering, while regular graphs like lattices tend to have high clustering but large path lengths.

Watts and Strogatz [59] shows that small world topologies exist widely in social graphs (e.g., collaboration graph of actors in feature files), biological graphs (e.g., the neural network of the nematode worm C.elegans), and man-made graphs (e.g., the electrical power grid of the western United States). A
small world topology may have a significant impact on the behavior of dynamical systems. Walsh [58] shows that the cost of solving search problems with small world topologies has a heavy-tailed distribution. In this thesis, we are interested in how the small world topology affects the cost of inference in different CBI problems.

Given an undirected graph \( G = (V, E) \), the parameters to characterize the small world topology include:

- \( L \): The averaged path length over all pairs of vertices;
- \( L_{\text{rand}} \): The averaged path length over all pairs of vertices in a random graph \( G_{\text{rand}} = (V_{\text{rand}}, E_{\text{rand}}) \) with \( |V| = |V_{\text{rand}}| \) and \( |E| = |E_{\text{rand}}| \);
- \( C \): The averaged clustering coefficient of all vertices, where the clustering coefficient of a vertex is defined as the fraction of the number of existing edges among the neighbors of the vertex, and maximum possible edges among them (a vertex with \( k \) neighbors will at most have \( k(k-1)/2 \) edges among its neighbors);
- \( C_{\text{rand}} \): The averaged clustering coefficient of all vertices in a random graph \( G_{\text{rand}} = (V_{\text{rand}}, E_{\text{rand}}) \) with \( |V| = |V_{\text{rand}}| \) and \( |E| = |E_{\text{rand}}| \);
- \( \mu \): ratio of \( C/L \) normalized by \( C_{\text{rand}}/L_{\text{rand}} \).

Watts and Strogatz [59] propose a procedure to generate graphs with small world topologies. It can be described as Algorithm 2.2. Chapter 7 presents empirical results of small world topology parameters of different CBI problems and their impacts on inference algorithms.

Another typical real world topology of the graphical representation of CBI problems is the scale-free network [3], where the degree distribution has a power-law form. The distribution is not related to the scale of the network. In other words, scale-free networks have a large number of vertices with a few connecting edges and a small number of vertices with many connecting edges. It is shown [3] that scale-free networks exists widely in the real world. The Internet, WWW, and metabolic networks are all examples of scale-free networks.
Chapter 2. Background

Algorithm 2.2 Small World Modelling [59]

**Input:** Number of vertices $n$, average degree $k$, rewritten probability $p$

**Output:** A graph with Small World topology

1. Generate vertices $V_1, \cdots, V_n$;
2. for $i = 1$ to $n$ do
3. for $j = 1$ to $k$ do
4. Uniformly generate a random number $p_0 \in [0, 1)$;
5. if $p_0 < p$ then
6. $k := i + \lfloor j/2 \rfloor (mod n)$
7. else
8. Uniformly generate a random number $k \in \{1, \cdots, n\} \setminus \{i\}$
9. end if
10. Add an edge between $V_i$ and $V_k$;
11. end for
12. end for

Algorithm 2.3 generates a graph with scale-free topology. This algorithm is based on the definition of scale-free topology in [3].

Empirical results in Chapter 7 show that many scale-free networks are also in small world topology. In other words, scale-free networks usually exhibit high clustered vertices and small path lengths. In contrary, a graph with the small world topology does not necessarily have a power-law form degree distribution. For example, the degree distribution of a small world graph generated by Algorithm 2.3 is uniform.

2.3 The Basis of Abstract Algebra

As mentioned in Chapter 1, there are two essential operations in real world CBI problems: (1) combination, which corresponds to an aggregation of con-

Algorithm 2.3 Scale-Free Modelling

**Input:** Number of vertices $n$, number of edges $m$

**Output:** A graph with Scale-Free topology

1. Generate vertices $V_1, \cdots, V_n$;
2. Let $d_i$ be degree of the vertex $V_i$;
3. for $i = 2$ to $n$ do
4. Connect $V_i$ to $V_j$ according probability $p_j = d_j/\sum_k d_k$, $k = \{1, \cdots, i-1\}$;
5. end for
6. for $i = 1$ to $m - n + 1$ do
7. Uniformly choose $i \in \{1, \cdots, n\}$
8. Connect $V_i$ to $V_j$ according probability $p_j = d_j/\sum_k d_k$, $k = \{1, \cdots, n\} \setminus \{i\}$;
9. end for
Chapter 2. Background

2.3.1 Set, Group, and Semigroup

Definition 2.6 (Set) A set is defined as an arbitrary collection of objects or elements. There are no predefined operations between elements in a set. The number of elements in the set is referred to as the cardinality of the set.

Definition 2.7 (Group) Let \( G \) be a set and \( \cdot \) be a binary operation defined on \( G \). \((G, \cdot)\) is a group if the operation satisfies the following four conditions:

- Closure: \( a \cdot b = c \in G, \forall a, b \in G \);
- Associativity: \((a \cdot b) \cdot c = a \cdot (b \cdot c), \forall a, b, c \in G \);
- Identity: \( \exists 1 \in G \ s.t. \ a \cdot 1 = 1 \cdot a = a, \forall a \in G \);
- Inverses: \( \forall a \in G, \exists a^{-1} \in G, s.t. \ a \cdot a^{-1} = 1 \)

A group \((G, \cdot)\) is said to be commutative, or Abelian, if the operation satisfies one more condition:

- Commutativity: \( a \cdot b = b \cdot a, \forall a, b \in G \);

A semigroup is a group that does not need an identity element and whose elements do not need inverses within the semigroup. Only closure and associativity hold for a semigroup. A semigroup with an identity is called a monoid. A semigroup is commutative if the associated binary operation is commutative.

Figure 2.2 shows relations among axioms in group and related notions.

2.3.2 Ring, Semiring, and k-Semiring

Definition 2.8 (Ring) Let \( R \) be a set. Let \( \oplus \) and \( \otimes \) be two closed binary operations defined on \( R \). Here we assume operation \( \otimes \) has precedence over operation \( \oplus \). \((R, \oplus, \otimes)\) is a ring if the operations satisfy the following axioms:

- Additive associativity: \( \forall a, b, c \in R, (a \oplus b) \oplus c = a \oplus (b \oplus c) \);
- Additive commutativity: \( \forall a, b \in R, a \oplus b = b \oplus a \);
- Additive Identity: \( \exists 0 \in R, s.t. \forall a \in R, a \oplus 0 = 0 \oplus a = a \);
- Additive Inverses: \( \forall a \in R, \exists -a \in R, s.t. a \oplus -a = -a \oplus a = 0 \);
- Multiplicative associativity: \( \forall a, b, c \in R, (a \otimes b) \otimes c = a \otimes (b \otimes c) \);
Figure 2.2: Axioms in the Definitions of Group and Related Concepts

- **Left and right distributivity:** \( \forall a, b, c \in R, a \circ (b \odot c) = a \circ b \odot a \otimes c \) and 
  \( (b \odot c) \circ a = b \odot a \odot c \odot a. \)

Therefore, a ring is a commutative group under addition and a semigroup under multiplication. \( 0 \) is called the additive identity element (of operation \( \circ \)), and \( 1 \) is called the multiplicative identity element (of operation \( \odot \)).

**Definition 2.9 (Semiring)** Let \( S \) be a set. Let \( \oplus \) and \( \otimes \) be two closed binary operations defined on \( S \). \((S, \oplus, \otimes)\) is a **semiring** if the operations satisfy the following axioms:

- **Additive associativity:** \( \forall a, b, c \in S, (a \oplus b) \oplus c = a \oplus (b \oplus c) \);
- **Additive commutativity:** \( \forall a, b \in S, a \oplus b = b \oplus a \);
- **Multiplicative associativity:** \( \forall a, b, c \in S, (a \otimes b) \otimes c = a \otimes (b \otimes c) \);
- **Left and right distributivity:** \( \forall a, b, c \in S, a \otimes (b \oplus c) = a \otimes b \oplus a \otimes c \) and 
  \( (b \oplus c) \otimes a = b \otimes a \oplus c \otimes a. \)

A semiring \((S, \oplus, \otimes)\) is a ring that need not have either additive \((\oplus)\) inverses or an additive identity element. In other words, a semiring is a commutative semigroup under addition and a semigroup under multiplication.
Chapter 2. Background

Figure 2.3: Axioms in the Definitions of Ring, Semiring, and Commutative Semiring

Definition 2.10 (Commutative Semiring) A commutative semiring is a semiring that satisfies the following additional conditions:

- Multiplicative commutativity: \( \forall a, b \in S, a \otimes b = b \otimes a \);
- Multiplicative identity: there exists a multiplicative identity element \( 1 \in S \), such that \( a \otimes 1 = 1 \otimes a = a \) for any \( a \in S \);
- Additive identity: there exists an additive identity element \( 0 \in S \), such that \( a \oplus 0 = 0 \oplus a = a \) for any \( a \in S \);

A commutative semiring \((S, \oplus, \otimes)\) is a commutative monoid under both addition and multiplication.

Figure 2.3 shows the relations among axioms of ring, semiring, and commutative semiring.

Here we define \textit{b-semiring} as a generalization of semiring with a \( I \)-semiring corresponding to the definition of a semiring.
Chapter 2. Background

Table 2.2: Summary of Different Commutative Semirings

<table>
<thead>
<tr>
<th>No.</th>
<th>( R )</th>
<th>( \oplus )</th>
<th>Add. Identity</th>
<th>( \otimes )</th>
<th>Multi. Identity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{true, false}</td>
<td>( \lor )</td>
<td>false</td>
<td>( \land )</td>
<td>true</td>
</tr>
<tr>
<td>2</td>
<td>[0,1]</td>
<td>max</td>
<td>0</td>
<td>min</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>[0,1]</td>
<td>max</td>
<td>0</td>
<td>( \times )</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>((-\infty,0])</td>
<td>max</td>
<td>(-\infty)</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>[0,\infty)</td>
<td>max</td>
<td>0</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>[0,\infty)</td>
<td>+</td>
<td>0</td>
<td>( \times )</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>[0,\infty)</td>
<td>max</td>
<td>0</td>
<td>( \times )</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>((-\infty,\infty))</td>
<td>min</td>
<td>( \infty )</td>
<td>+</td>
<td>0</td>
</tr>
</tbody>
</table>

Definition 2.11 ((Commutative) k-Semiring) A (commutative) k-semiring is a tuple \((S, op_0, op_1, \cdots, op_k)\) such that for each \( i \in \{1, \cdots, k\} \), \((S, op_j, op_i)\) is a (commutative) semiring for every \( j < i \).

The definition of (commutative) k-semiring is sometimes too strong in practice. For example, we may need only \((S, op_i, op_i)\) to satisfy the (commutative) semiring properties. We do not necessarily need \((S, op_j, op_i)\) to be a (commutative) semiring for \( \forall j < i \) always, although the strong definition provides computational flexibility in algorithm designs.

\((\[0, \infty), +, \times\) and \((\{0,1\}, \lor, \land)\) are all examples of commutative semirings. Table 2.2 shows further examples of commutative semirings. \((\[0, \infty), \max, +, \times\) is an example of a commutative 3-semiring.

Furthermore, we say that 0 is a multiplicative absorbing element if \(a \otimes 0 = 0 \otimes a = 0\), and 1 is an additive absorbing element if \(a \oplus 1 = 1 \oplus a = 1\), for any \(a \in S\). We say that \(\oplus\) is idempotent if \(a \oplus a = a\), and \(\otimes\) is idempotent if \(a \otimes a = a\) for any \(a \in S\). We notice that the idempotency of \(\oplus\) defines a partial ordering \(\leq\) over the set \(S\), i.e., \(a \leq b \iff a \oplus b = b\) for any \(a, b \in S\).

Semiring and k-semiring are enough to generalize the representation of a wide range of CBI problems. However, to characterize inference algorithms for these problems, we will use commutative semiring and commutative k-semiring in the following chapters. The commutativity and identity elements provide many computational benefits for solving these problems.

2.4 Summary

This chapter introduced the basic definitions and notions used in this thesis. Constraint-based inference problems can be graphically represented as primal graphs. Primal graphs can be reorganized as secondary structures, to achieve computational efficiency.

The success of inference algorithms for specific problems is largely decided by the parameters of the underlying graphs. One of these graphical parameters is treewidth, or induced width. Computing treewidth, or finding the optimal triangulated graph, is known as \(\mathcal{NP}\)-hard [2]. Though some polynomial time...
approximation algorithms have been designed to compute treewidth with constant factors, they are not practical since the time complexities always contain some constant that is exponentially large. For practical scenarios, heuristic searches are used to estimate the upper bound of treewidth. In this chapter we briefly discussed several heuristic triangulation algorithms. Two other important notions to characterize the graphical representation of CBI problems are the small world topology and scale-free networks. This chapter also defined and explained the parameters used to characterize the small world topology and scale-free networks.

As powerful tools to generalize CBI problems, this chapter introduced algebraic structures, and more specifically, semirings and commutative semirings. Basic definitions and axioms of abstract algebra were given. We also discussed the relations among these definitions.
Chapter 3

A Semiring-Based Generalized Framework for Constraint-Based Inference

3.1 A Semiring-Based Generalized Framework

A constraint-based inference problem is defined in terms of a set of variables with values in finite domains and a set of constraints on these variables. We use commutative semirings to unify the representation of constraint-based inference problems from various disciplines. Formally:

**Definition 3.1 (Constraint-Based Inference (CBI) Problem)** A constraint-based inference (CBI) problem \( \mathbf{P} \) is a tuple \((X, D, R, F)\) where:

- \( X = \{X_1, \ldots, X_n\} \) is a set of variables;
- \( D = \{D_1, \ldots, D_n\} \) is a collection of finite domains, one for each variable;
- \( R = (S, \otimes, \oplus) \) is a commutative semiring;
- \( F = \{f_1, \ldots, f_r\} \) is a set of constraints. Each constraint is a function that maps value assignments of a subset of variables, its scope, to values in \( S \).

Before defining various tasks for CBI problems, we define the two basic constraint operations as follows, using the two binary operation \( \otimes \) and \( \oplus \) of the commutative semiring \( R \). Please note that we use the same two symbols \( \otimes \) and \( \oplus \) to represent the two constraint level operations. The meaning of them can be easily distinguished given the context.

**Definition 3.2 (The Combination of Two Constraints)** The combination of two constraints \( f_1 \) and \( f_2 \) is a new constraint \( g = f_1 \otimes f_2 \), where \( \text{Scope}(g) = \text{Scope}(f_1) \cup \text{Scope}(f_2) \) and \( g(w) = f_1(w_{\text{Scope}(f_1)}) \otimes f_2(w_{\text{Scope}(f_2)}) \) for every value assignment \( w \) of variables in the scope of the constraint \( g \).

**Definition 3.3 (The Marginalization of a Constraint)** The marginalization of \( X_i \) from a constraint \( f \), where \( X_i \in \text{Scope}(f) \), is a new constraint \( g = \bigoplus_{X_i} f \), where \( \text{Scope}(g) = \text{Scope}(f) \setminus X_i \) and \( g(w) = \bigoplus_{x_i \in \text{Domain}(X_i)} f(x_i, w) \) for every value assignment \( w \) of variables in the scope of the constraint \( g \).
Definition 3.4 (The Inference Task for a CBI problem) Given a variable subset of interest \( Z \subseteq X \), let \( Y = X \setminus Z \), then the inference task for a CBI problem \( P = (X, D, R, F) \) is defined as computing:

\[
g_{\text{CBI}}(Z) = \bigoplus_{Y \in F} f
\]  

(3.1)

Given a CBI problem \( P = (X, D, R, F) \), if \( \oplus \) is idempotent, we can define the assignment task for a CBI problem.

Definition 3.5 (The Assignment Task for a CBI problem) Given a variable subset of interest \( Z \subseteq X \), let \( Y = X \setminus Z \), the assignment task for a CBI problem \( P = (X, D, R, F) \) is to find a value assignment for the marginalized variables \( Y \), which gives the result of the corresponding inference task \( g_{\text{CBI}}(Z) \). Formally, a solution to the assignment task is:

\[
y = \arg \bigoplus_{Y \in F} f
\]  

(3.2)

where \( \arg \) is a prefix of operation \( \oplus \). In other words, \( \arg \oplus_x f(x) \) returns \( x \) s.t. \( f(x) = \oplus_x f(x) \).

If a total ordering of \( S \) exists, we can define the optimization task for a CBI problem as maximizing (or minimizing) the computed result of the corresponding CBI inference task by finding a value assignment to variables \( Z \). Formally

Definition 3.6 (The Optimization Task for a CBI problem) Given a variable subset of interest \( Z \subseteq X \), let \( Y = X \setminus Z \) and \( R = (S, \max, \oplus, \otimes) \) be a commutative 2-semiring, then the optimization task for a CBI problem \( P = (X, D, R, F) \) is defined as computing:

\[
g_{\text{OPT}} = \max_Z \left( \bigoplus_{Y \in F} f \right)
\]  

(3.3)

The assignment task for an optimization task is then to compute:

\[
z = \arg \max_Z \left( \bigoplus_{Y \in F} f \right)
\]  

(3.4)

In general, \( \otimes \) is a combination operation for CBI problems that combines a set of constraints into a constraint with a larger scope; \( \oplus_Y = \oplus_{X \setminus Z} \) is a marginalization operation that projects a constraint over the scope \( X \) onto its subset \( Z \), through enumerating all possible value assignments of \( Y = X \setminus Z \).

As discussed in Chapter 2, a CBI problem can be graphically represented by a primal graph, which is an undirected graph with variables as its vertices,
and there exists an edge between two vertices if the two corresponding variables appear in the scope of a same constraint. Also it can be re-organized as a junction tree or a factor graph to achieve computational benefits.

Practically, sometimes there are observations of, or evidence about, the values of some variables. In such cases, constraints with observed variables in their scopes can be modified through instantiation before performing the inference. Although there are many observation-handling techniques to accelerate the inference process, we will not incorporate them into our framework here. Another issue concerning practical CBI tasks is the normalization after the computation. In this thesis, we omit normalization constants since they are simple to define and compute in specific application scenarios.

3.2 Instances of the Semiring-Based Generalized Framework

Many CBI problems from different disciplines can be embedded into the proposed generalized CBI framework. In this section we will briefly introduce these problems as instances of the framework.

3.2.1 Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) is defined over a set of variables $X = \{X_1, \cdots, X_n\}$, a collection of domains $D = \{D_1, \cdots, D_n\}$ for variables, and a set of constraints or evaluation functions $F = \{f_1, \cdots, f_r\}$, where each $f_i$ is defined on a subset of variables $S_i = \text{Scope}(f_i) \subseteq X$.

The inference task for a constraint satisfaction problem is to find if there exists a value assignment that satisfies all the constraints. The assignment task for a constraint satisfaction problem is to find such a value assignment or all assignments according to some criteria.

Classic CSPs

Classic CSP [40] is a class of constraint satisfaction problems such that all constraints have to be satisfied by a valid value assignment. Constraint $f_i$ of classic CSP maps possible value assignments of variables in $\text{Scope}(f_i)$ into logical $\text{true}$ if the assignment satisfies the constraint, or into logical $\text{false}$ if not. In terms of our semiring-based framework, a classic CSP is a tuple $(X, D, R, F)$, where $X$, $D$, and $F$ are defined before, and $R = (\{\text{false}, \text{true}\}, \vee, \wedge)$ is a commutative semiring, where $\vee$ is the binary OR operation with $\text{false}$ as the identity element and $\wedge$ is the binary AND operation with $\text{true}$ as the identity element.

Given $Z = \emptyset$, the inference task (decision problem) of a classic CSP is to compute:

$$g_{CSP} = \bigvee_{X} \bigwedge_{f \in F} f$$
Chapter 3. A Semiring-Based Generalized Framework for CBI

$g_{\text{CSP}} = true$ means there exists a value assignment that satisfies all constraints. In such a case, the assignment task for classic CSP is to compute:

$$x = \arg \bigvee_{x} \bigwedge_{f \in F} f$$

Soft CSPs

Constraints in classic CSPs are sometimes called hard constraints, which means that all constraints have to be satisfied simultaneously. However, in practice many problems are over-constrained. Several frameworks have been proposed to handle over-constrained problems, mostly by introducing soft constraints that may be (partially) violated. These frameworks include: (1) the Max CSP framework that tries to maximize the number of satisfied constraints, which is equivalent to minimizing the number of violations; (2) the Weighted CSP framework that associates a degree of violation to each constraint and minimizing the sum of all weighted violations; and (3) the Fuzzy CSP framework [52] that associates a preference to each value assignment of a constraint. We generalize these soft CSPs into our semiring-based unified framework using various semirings, following the results of the Semiring CSP [6] and Valued CSP [53] frameworks.

Max CSPs and Weighted CSPs

Sometimes we do not need all the constraints to be satisfied at the same time. In Max CSPs, the assignment task is to find a value assignment that satisfies the maximum number of constraints. A Max CSP is same as a Classic CSP except that it is defined on a different commutative semiring $R = ([0, \infty), \max, +)$. The inference task for a Max CSP is to compute:

$$g_{\text{MaxCSP}} = \max_{X} \sum_{f \in F} f$$

And the assignment task for a Max CSP is to compute:

$$x = \arg \max_{X} \sum_{f \in F} f$$

A Weighted CSP is slightly different from a Max CSP in that satisfying different constraints has different weights, whereas weights are always the same for all constraints in a Max CSP. The same semiring used in the Max CSP applies in the representation of the Weighted CSP in our semiring-based unified framework.

Fuzzy CSPs

Fuzzy CSP [52] extends Classic CSP by mapping all possible value assignments of each constraint into preference levels. The levels of preference are defined on some discrete numbers from 0 to 1, where 0 stands for not allowing such an assignment. A solution of Fuzzy CSP is a value assignment that has the maximum value of the minimum preferences of all constraints (max-min
value). In terms of the semiring-based unified framework, a fuzzy CSP is a tuple 
\((X, D, R, F)\), where \(X\), \(D\) and \(F\) are defined beforehand, and \(R = ([0, 1], \max, \min)\) 
is a commutative semiring with 0 as the identity element of \(\max\), and 1 as the 
identity element of \(\min\).

Given \(Z = \emptyset\), the inference task for a Fuzzy CSP is to compute:

\[ g_{\text{FuzzyCSP}} = \max_X \min_{f \in F} f \]

And the assignment task for a Fuzzy CSP is to compute:

\[ x = \arg \max_X \min_{f \in F} f \]

### 3.2.2 Propositional Satisfiability Problems

The propositional satisfiability problem (SAT) is a central problem in logic 
and artificial intelligence, which consists of a logical propositional formula in \(n\) 
variables. For \(k\)-SAT, the formula consists of a conjunction of clauses, and each 
clause is a disjunction of at most \(k\) variables, any of which may be negated. For 
\(k \geq 3\), these problems are NP-complete.

**SAT**

Like the constraint satisfaction problem, SAT can be abstracted in the semiring-
based unified framework as a tuple \((X, D, R, F)\):

- \(X = \{X_1, \ldots, X_n\}\) is a set of \(n\) variables;
- \(D = \{\text{true, false}\}\) is the domain for each variable;
- \(R = (\{\text{false, true}\}, \lor, \land)\) is a commutative semiring, \(\lor\) is the binary OR 
operation with \(\text{false}\) as its identity element ; \(\land\) is the binary AND oper­
ation with \(\text{true}\) as its identity element.
- \(F = \{f_1, \ldots, f_r\}\) is a set of \(r\) clauses, \(f_i : D^k \rightarrow \{\text{false, true}\}\).

Given \(Z = \emptyset\), the inference task for SAT is to compute:

\[ g_{\text{SAT}} = \bigvee_X \bigwedge_{f \in F} f \]

And the assignment task for SAT is to compute:

\[ x = \arg \bigvee_X \bigwedge_{f \in F} f \]
MaxSAT

Similarly, MaxSAT is defined as finding a value assignment for each variable that allows the maximum number of clauses to be true. A MaxSAT is the same with SAT except the definitions of $R$ and $F$:

- $R = [0, \infty), \text{max}, +)$ is a commutative semiring, 0 is the identity elements of both $\text{max}$ and $+$.
- $F = \{f_1, \ldots, f_r\}$ is a set of $r$ clauses, $f_i : \{\text{true, false}\}^k \rightarrow \{0, 1\}$, $k = |\text{Scope}(f_i)|$.

Given $Z = \emptyset$, the inference task for MaxSAT is to compute:

$$g_{\text{MaxSAT}} = \max_X \sum_{f \in F} f$$

And the assignment task for MaxSAT is to compute:

$$x = \arg \max_X \sum_{f \in F} f$$

### 3.2.3 Probability Inference Problems

Probability inference problems can be seen as constraint-based inference by treating conditional probability distributions (CPDs) as soft constraints over variables. A belief network or a Bayesian network (BN) [49] is a graphical representation for probability inference under conditions of uncertainty. BN is defined as a directed acyclic graph (DAG) where vertices $X = \{X_1, \ldots, X_n\}$ denote $n$ random variables and directed edges denote causal influences between variables. $D = \{D_1, \ldots, D_n\}$ is a collection of finite domains for variables. A set of conditional probability distributions $F = \{f_1, \ldots, f_n\}$, where $f_i = P(X_i|\text{Parents}(X_i))$ is attached to each variable (vertex) $X_i$. Then the probability distribution over $X$ is given by $P(X) = \prod_{i=1}^n P(X_i|\text{Parents}(X_i))$. The moralized graph of a BN is obtained by adding edges among the vertices with a common child vertex and removing directions of all edges. The moralized graph of a BN is the primal graph of the corresponding probability inference problem. Generally, there are three types of tasks in belief inference: probability assessment, most probable explanation, and maximum a posteriori hypothesis (maximum likelihood). All of these tasks can be generalized into our semiring-based framework.

#### Probability Assessment

Probability assessment is a CBI problem over a BN, which computes the posterior marginal probability of a subset of variables, given values for some variables as known evidence. Formally, a probability assessment problem is a tuple $(X, D, R, F)$, where $X$, $D$, and $F$ are defined in BN, and $R = ([0, \infty), +, \times)$ is
Chapter 3. A Semiring-Based Generalized Framework for CBI

A commutative semiring, where \((\oplus, 0) = (+, 0)\) and \((\otimes, 1) = (\times, 1)\). Given \(Z\) as the variable subset of interests, the inference task for a probability assessment problem is to compute:

\[
g_{BA}(Z) = \alpha \sum_{X \setminus Z} \prod_{f \in F} f
\]

where \(\alpha\) is a normalization constant. Obviously \(((0, \infty), \times)\) has the invertible property, making normalization feasible.

**Most Probable Explanation (MPE)**

Most probable explanation is a CBI problem over a BN, which is to find a complete value assignment of variables that agrees with the available evidence and has highest probability among all such assignments. Formally, the most probable explanation problem is a tuple \((X, D, R, F)\), where \(X, D,\) and \(F\) are defined in the corresponding belief network, and \(R = ([0, \infty), \max, \times)\) is a commutative semiring, where \((\oplus, 0) = (\max, 0)\) and \((\otimes, 1) = (\times, 1)\).

Given \(Z = \emptyset\), the inference task for a MPE is to compute:

\[
g_{MPE} = \max_{X} \prod_{f \in F} f
\]

The assignment task for a MPE is to compute:

\[
x = \arg \max_{X} \prod_{f \in F} f
\]

**Maximum A Posteriori Hypothesis (MAP)**

A maximum a posteriori hypothesis problem is an optimization task over a BN, which finds a value assignment of a set of hypothesized variables to agree with the available evidence and has highest probability among all such assignments. Formally, a maximum a posteriori hypothesis problem is a tuple \((X, D, R, F)\), where \(X, D,\) and \(F\) are defined in BN, and \(R = ([0, \infty), \max, +, \times)\) is a commutative \(\mathbb{Z}\)-semiring, where \((\max, 0)\) is defined over non-negative real values, \((\oplus, 0) = (+, 0)\), and \((\otimes, 1) = (\times, 1)\).

Given \(Z = \{Z_1, \cdots, Z_t\} \subseteq X\) as a subset of hypothesized variables, the inference task for a MAP is to compute:

\[
g_{MAP} = \max_{Z} (\sum_{X \setminus Z} \prod_{f \in F} f)
\]

The assignment task for a MAP is to compute:

\[
z = \arg \max_{Z} (\sum_{X \setminus Z} \prod_{f \in F} f)
\]
3.2.4 Dynamic Bayesian Networks

Many discrete-time stochastic processes can be graphically represented as dynamic Bayesian networks (DBN) [13]. DBN is a powerful tool to model dynamic systems with \( N \) random variables \( X = \{X_1, \ldots, X_n\} \). Let \( X^t \) be the state vector of \( X \) at time \( t \). As an extension of Bayesian networks (BN), inference tasks can be performed over a DBN.

Formally, a dynamic Bayesian network is defined [45] to be a pair \( (B_1, B_-) \), where \( B_1 \) is a BN which defines the prior \( P(X^1) \), and \( B_- \) is a two-slice temporal Bayesian net (2TBN) which defines \( P(X^t|X^{t-1}) \) as

\[
P(X^t|X^{t-1}) = \prod_{i=1}^n P(X^t_i|\text{Parents}(X^t_i))
\]

Like representing BN, a DBN can be abstracted in the semiring-based framework as a tuple \( (X, D, R, F) \):

- \( X^t = \{X^t_1, \ldots, X^t_n\} \) is a set of random variables at time \( t = 1, \ldots, T \);
- \( D = \{D_1, \ldots, D_n\} \) is a collection of finite domains for each item in \( X \);
- \( R = ([0, \infty), +, \times) \) is a commutative semiring, \((\oplus, 0) = (+, 0)\) and \((\otimes, 1) = (\times, 1)\).
- \( F^t = \{f^t_1, \ldots, f^t_n\} \) where \( f^t_i = P(X^t_i|\text{Parents}(X^t_i)) \)

Given \( Z^T = \{Z^T_1, \ldots, Z^T_k\} \) as a subset of random variables of interests at time \( t = T \), the inference task in a DBN is to compute the marginal of joint distribution:

\[
\mathcal{g}_{DBN} = \sum_{X^1, \ldots, X^{T-1}, X^T \setminus Z^T} \prod_{t=1}^T \prod_{f^t \in F_t} f^t
\]

3.2.5 Decision-Making Problems

The Decision Network (DN), or the influence diagram [27], is a graphical representation for decision-making problems. A decision network is a directed acyclic graph with three types of nodes: random nodes, decision nodes, and value nodes. Each random node represents a random variable, associating with a conditional probability table; each decision node represents a decision variable with a finite set of possible actions (or decisions); each value node associates with a utility function. The edge from a random node to a decision node is called an information edge, and so a decision node \( A_i \) can be viewed as a random node. Given a value assignment of its parents nodes, \( A_i \) has a conditional probability \( P(A_i = a|\text{Parents}(A_i)) = 1 \) for an decision \( a \in \text{domain}(A_i) \), and 0 for other decisions.

The goal of the one-off decision-making problem in decision networks is to find such policies for each decision node that maximize the summation of expected utilities. We do not integrate sequential decision-making problems [62] in decision networks into our framework and leave it in our future work.
In terms of our semiring-based framework, a one-off decision-making problem is a tuple \((X, D, R, F)\):

- \(X = C \cup A\), where \(C = \{X_1, \ldots, X_k\}\) is a set of random (chance) variables and \(A = \{A_{k+1}, \ldots, A_n\} = \{X_{k+1}, \ldots, X_n\}\) is a set of decision (action) variables;
- \(D = \{D_1, \ldots, D_n\}\) is a collection of finite domains for each item in \(X\);
- \(R = ((-\infty, \infty), \max, +, \times)\) is a commutative \(2\)-semiring, where \(-\infty\) is the identity element of \(\max\), \(0\) is the identity element of \(+\), and \(1\) is the identity element of \(\times\);
- \(F = \{u\}\) where \(u = \sum_{i=1}^{t} u_i\) is the summation of \(t\) utility functions, each \(u_i\) associating with a value node, \(u : X^n \rightarrow \mathbb{R}\).

Given \(Z = A = \{A_{k+1}, \ldots, A_n\}\) as a subset of variables of interest, the inference task for a one-off decision-making problem is to compute:

\[
g_{DN} = \max_{A} \sum_{C} \prod_{i=1}^{n} P(X_i|Parents(X_i)) \cdot u(A) \tag{3.5}
\]

The assignment task for a one-off decision making problem is to compute:

\[
a = \arg \max_{A} \sum_{C} \prod_{i=1}^{n} P(X_i|Parents(X_i)) \cdot u(A)
\]

### 3.2.6 Possibility Inference Problems

Possibility theory [61] is another approach to characterizing uncertainties. A possibility inference problem is defined over a set of variables \(X = \{X_1, \ldots, X_n\}\), a collection of discrete domains \(D = \{D_1, \ldots, D_n\}\) for variables, and a set of constraints or possibility distribution functions \(F = \{f_1, \ldots, f_r\}\). Each \(f_i\) specifies the degree of possibility of the configuration of variables in \(\text{Scope}(f_i)\). The degree of possibility, or possibility potential, is a value in [0,1], representing non-possibility to full certainty.

In possibility theory, a binary operation \(T\) is a function \(T : [0,1] \times [0,1] \rightarrow [0,1]\), which satisfies the following conditions:

- Boundary: \(\forall a \in [0,1], 1Ta = a\) and \(0Ta = 0\);
- Monotonicity: \(\forall a_1, a_2, b_1, b_2 \in [0,1] \text{ s.t. } a_1 \leq a_2 \text{ and } b_1 \leq b_2, a_1 Tb_1 \leq a_2 Tb_2\) holds;
- Commutativity: \(\forall a, b, c \in [0,1], aTb = bTa\) and \((aTb)Tc = aT(bTc)\).

Such a binary operation is also called a \(t\)-norm. There are many \(t\)-norms in possibility theory research. The most frequently used \(t\)-norms are:
Chapter 3. A Semiring-Based Generalized Framework for CBI

- Product t-norm: \( a \times b \);
- Gödel's t-norm: \( a \land b \);
- Lukaszewicz's t-norm: \( a \lor b = \max(0, a + b - 1) \).

The inference task for possibility inference is to find the maximum possibility of some variables and the corresponding configuration of these variables. In terms of the semiring-based unified framework, a possibility inference problem is a tuple \((X, D, R, F)\) if the corresponding t-norm satisfies the commutative semiring properties:

- \( X = \{X_1, \cdots, X_n\} \) is a set of random variables;
- \( D = \{D_1, \cdots, D_n\} \) is a collection of finite domains for each item in \( X \);
- \( R = ([0,1], \max, T) \) is a commutative semiring, \((\emptyset, 0) = (\max, 0)\) and \((\emptyset, 1) = (T, 1)\) where \( T \) is a t-norm and \( 1 \) is the identity element for the t-norm;
- \( F = \{f_1, \cdots, f_r\} \) is a set of possibility distribution functions.

Given \( Z \subseteq X \) as a subset of variables of interests and let \( Y = X \setminus Z \), the inference task for a possibility inference problem \((X, D, R, F)\) is to compute the marginal of joint possibility distribution:

\[
g_{poss} = \max_{Y} T_{f \in F} \prod_{i=1}^{n} P(X_i|\text{Parents}(X_i)) \cdot \left( \sum_{j=1}^{t} u_j(A_j) \right)
\]

The assignment task for a possibility inference problem is to compute:

\[
y = \arg \max_{Y} T_{f \in F} \prod_{i=1}^{n} P(X_i|\text{Parents}(X_i)) \cdot \left( \sum_{j=1}^{t} u_j(A_j) \right)
\]

3.2.7 Coordination Graph

In decision-making problems, we sometimes assume that utility functions are additive, i.e., \( u(A) = \sum_{i=1}^{n} u_i(A) \), then Eq. 3.5 can be re-written as:

\[
g_{DN} = \max_{A} \sum_{C} \prod_{i=1}^{n} P(X_i|\text{Parents}(X_i)) \cdot \left( \sum_{j=1}^{t} u_j(A_j) \right)
\]

\[
= \max_{A} \sum_{j=1}^{t} \sum_{C} \prod_{i=1}^{n} P(X_i|\text{Parents}(X_i)) \cdot u_j(A_j)
\]

\[
= \max_{A} \sum_{j=1}^{t} \sum_{C} \prod_{i=1}^{n} P(X_i|\text{Parents}(X_i)) \cdot Q_j(A_j)
\]

\[
= \max_{A} \sum_{j=1}^{t} Q_j(A_j)
\] (3.6)
where \( Q_j(A_j) = \sum_C \prod_{i=1}^n P(X_i|\text{Parents}(X_i)) \cdot u_j \) is a utility function that depends only on a subset actions \( A_j \subseteq A \), but does not depend on random variables. After computing \( Q_j(A_j) \), the task is transformed to find an equilibrium of a coordination game, which requires finding a decision or action from a finite set of possible actions for each player, to maximize the summation of players' utilities. Such a reduced problem is formalized by Guestrin et al. [25] and represented by a Coordination Graph.

**Definition 3.7** A Coordination Graph for a set of players with local utilities \( \{Q_1, \ldots, Q_n\} \) is a directed graph whose nodes are the actions for the players \( \{A_1, \ldots, A_n\} \), and there exists an edge from \( A_i \) to \( A_j \) if and only if \( A_i \in \text{Scope}(Q_j) \).

The task for finding an optimal joint action in a coordination graph can be easily embedded in our semiring-based framework as an assignment task for a CBI problem \((X, D, R, F)\) where:

- \( X = A = \{A_1, \ldots, A_t\} \) is a set of decision (or action) variables;
- \( D = \{D_1, \ldots, D_t\} \) is a collection of finite domains, each for a variable;
- \( R = ([0, \infty), \max, +) \) is a commutative semiring, \((\oplus, 0) = (\max, 0)\) and \((\otimes, 1) = (+, 0)\);
- \( F = \{Q_1, \ldots, Q_t\} \) is a set of utility functions, \( Q_i : D_{i_1} \times \cdots \times D_{i_k} \rightarrow S \), where \( \text{Scope}(Q_i) = \{A_{i_1}, \ldots, A_{i_k}\} \).

The inference task for computing optimal joint action in coordination graph is just to compute Eq. (3.6). The assignment task for a coordination graph is then to compute:

\[
\mathbf{a} = \arg \max_A \sum_{j=1}^t Q_j(A_j)
\]

### 3.2.8 Maximum Likelihood Decoding

In digital communication, decoding is an important step in recovering the original information transmitted through a discrete memoryless channel. How to decode information efficiently and exactly is always a central problem in coding theory. Maximum Likelihood Decoding (MLD) is one widely used approach. Semiring-based general massage-passing approach for MLD were first proposed in [1].

Suppose a codeword \( Z = \{Z_1, \ldots, Z_n\} \) is transmitted over a discrete memoryless channel. A vector \( Y = \{Y_1, \ldots, Y_n\} \) is received. The likelihood of the codeword \( Z \) is then:

\[
P(Z|Y) = \prod_{i=1}^n P(Z_i|Y_i)
\]
where $P(Z_i|Y_i)$ is the transition probability of the channel. For computational convenience, sometimes we compute the log-likelihood:

$$-\log(P(Z|Y)) = \sum_{i=1}^{n} -\log(P(Z_i|Y_i))$$

It is equivalent to minimizing the log-likelihood decoding. We use $h_i$ to denote $-\log(P(Z_i|Y_i))$.

The MLD problem is to find the codeword $Z$ that maximizes the likelihood. In addition, different linear block codes define a set of indicator functions $\chi = \{\chi_1, \cdots, \chi_k\}$ to specify if a subset of $Z$ is a valid part of the codeword. An indicator function $\chi_i$ is defined as:

$$\chi_i(Z_{i1}, \cdots, Z_{ik}) = \begin{cases} 0 & \text{if } (Z_{i1}, \cdots, Z_{ik}) \text{ is a part of the codeword;} \\ \infty & \text{otherwise} \end{cases}$$

In terms of the semiring-based unified framework, a maximum likelihood decoding problem is a tuple $(X, D, R, F)$, where:

- $X = Z \cup Y$ is a set of $2n$ variables;
- $D = \{D_1, \cdots, D_n\}$ is the domains for each $Z_i$ and $Y_i$;
- $R = ([0, \infty), \min, +)$ is a commutative semiring, $(\emptyset, 0) = (\min, \infty)$ and $(\emptyset, 1) = (+, 0)$;
- $F = H \cup \chi$ is a set of constraints where $H = \{h_1, \cdots, h_n\}$ and $\chi = \{\chi_1, \cdots, \chi_k\}$.

The inference task for a MLD is to compute:

$$g_{\text{MLD}} = \min_Z \sum_{h_i \in H} h_i \sum_{\chi_j \in \chi} \chi_j$$

And the assignment task for a MLD is to compute:

$$z = \arg \min_Z \sum_{h_i \in H} h_i \sum_{\chi_j \in \chi} \chi_j$$

### 3.3 Summary

This chapter presented a semiring-based unified framework to generalize the representation of CBI problems from different fields. The general multiplication operation $\otimes$ denotes the combination of constraints over local variables. The general addition operation $\oplus$ denotes the marginalization of a large constraint into a smaller scope. Through specifying variables and local constraints, CBI problems are generalized as computing a general sum-of-products in semirings.
We discussed many concrete CBI problems in this chapter as instances of the proposed semiring-based unified framework. These problems cover a wide range of research fields like probabilistic inference, decision-making under uncertainty, constraint satisfaction problems, propositional satisfiability, decoding problems, and possibility inference. We will show in following chapters that the abstract representation provides a powerful tool for studying and analyzing inference algorithms for these problems. The generalized representation, together with abstract level algorithm analyses, will eventually improve the concrete inference approach design for specified CBI problems.
Chapter 4

Exact Inference Algorithms

The distributivity and the commutativity properties of a commutative semiring \( R = (S, \oplus, \otimes) \) provide computational opportunities for solving constraint-based inference problems more efficiently. Comparing the computation costs (in terms of the number of operations) of the two tasks listed in Table 4.1, we will find that the two computation tasks are equivalent if distributivity holds. Both tasks need \( n - 1 \oplus \) operations, however, Task 2 only needs one \( \otimes \) operation, compared to \( n \otimes \) operations in computing Task 1.

<table>
<thead>
<tr>
<th>Task</th>
<th>Formula To Compute</th>
<th># of ( \oplus ) Operations</th>
<th># of ( \otimes ) Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((a \otimes b_1) \oplus \cdots \oplus (a \otimes b_n))</td>
<td>(n - 1)</td>
<td>(n)</td>
</tr>
<tr>
<td>2</td>
<td>(a \otimes (b_1 \oplus \cdots \oplus b_n))</td>
<td>(n - 1)</td>
<td>1</td>
</tr>
</tbody>
</table>

The basic idea behind these two computation tasks is: different computation sequences lead to different costs, given the same computation task defined by a commutative semiring. The distributivity of the two operations of a semiring provides possibilities to change the computation sequences equivalently, which makes finding a lower computation cost for the task possible.

Many inference algorithms have been developed independently, explicitly or implicitly exploiting the distributivity of commutative semirings. The key idea of these algorithms is to rearrange the computation sequences of the task solutions. In Chapter 3 we generalize these algorithms into two categories: variable elimination algorithms and junction tree algorithms.

4.1 Variable Elimination Algorithm

4.1.1 Generalized Variable Elimination Algorithm

The basic idea behind variable elimination (VE) algorithms, or bucket elimination algorithms, comes from non-serial dynamic programming [4]. Since dynamic programming algorithms work by eliminating variables one by one while computing the effect of each eliminated variable on the remaining problem, they can be viewed as variable elimination algorithms.

Formally, given a CBI problem \((X, D, R, F)\) and a variable subset of interest \(Z \subseteq X\), let \(Y = X \setminus Z\) and \(\sigma = \langle Y_1, \cdots, Y_k \rangle\) be a permutation of elements in \(Y\). The ordering \(\sigma\) is decided by the variable elimination algorithm. Let \(F_{Y_k}\)
be the subset of constraints in $F$ whose scopes contain $Y_k$. Let $F_{Y_k}$ be $F \setminus F_{Y_k}$. Then Eq. 3.1 can be re-written as:

$$g_{CBI}(Z) = \bigoplus_{X \setminus Z} \bigotimes_{f \in F} f$$
$$= \bigoplus_{Y} \bigotimes_{f \in F} f$$
$$= \bigoplus_{Y \setminus \{Y_k\}} \bigotimes_{\{Y_k\}} \bigotimes_{f \in F_{Y_k}} f$$
$$= \bigoplus_{Y \setminus \{Y_k\}} f' \bigotimes_{f \in F_{Y_k}} f$$

Eq. (4.1)

where $f' = \bigoplus_{Y_k} \bigotimes_{f \in F_{Y_k}} f$ is a new constraint that does not contain variable $Y_k$ in its scope. In Eq. 4.1, variable $Y_k$ is eliminated. In another words, now the inference task for the CBI problem does not depend on $Y_k$.

The generalized variable elimination algorithm follows the same procedure, eliminating variables recursively one by one, according to some ordering of these variables. See Fig. 4.1 for the description of our generalized variable elimination algorithm for the CBI inference task, abstracted from the variable elimination algorithm [64] in probability inference. Another revision of the generalized-VE algorithm is the generalized bucket elimination algorithm, which is abstracted from the bucket elimination algorithm [14] in the probabilistic inference field. See Algorithm 4.2 for details. The difference of these two algorithms is in the implementation, where the bucket elimination algorithm requires sorting constraints before the elimination of variables.

A common concern of applying VE algorithms is how to find an optimal elimination ordering to minimize the computation cost. Finding an optimal elimination ordering, which is equivalent to finding an optimal triangulated graph or finding the treewidth, the minimum width of all tree decompositions, is $NP$-hard [2]. Several heuristic search methods discussed in Chapter 2 can be used to generate elimination orderings. We will show their empirical effectiveness in Chapter 7.

For the assignment task for a CBI problem, backtracking approaches can be applied. All intermediate constraints during the elimination procedure are cached. The final computation value of $g_{CBI}(Z)$ is used as an index to track valid value assignments in the intermediate constraints.

To apply VE algorithms, commutativity of $\bigotimes$ is required; for the CBI inference task that enables us to rearrange the combination sequence of local functions. The commutativity of $\bigotimes$ is a desired property as well; that enables
Algorithm 4.1 Generalized Variable Elimination Algorithm (GVE) for a CBI Inference Task

Input: A CBI problem \((X, D, R, F)\) and a variable subset of interest \(Z\)

Output: \(g_{CBI}(Z) = \bigoplus_{X-Z} \bigotimes_{f \in F} f\)

1. Let \(Y = X \setminus Z\)
2. Choose an elimination ordering \(\sigma = \langle Y_1, \ldots, Y_k \rangle\) of \(Y\)
3. for \(i = k\) to 1 do
   4. \(F' := \emptyset\)
   5. for each \(f \in F\) do
      6. if \(Y_i \in \text{Scope}(f)\) then
         7. \(F' := F' \cup \{f\}\)
         8. \(F := F \setminus \{f\}\)
      end if
   end for
   9. \(f' := \bigoplus_{Y_i} \bigotimes_{f' \in F'} f'\)
   10. \(F := F \cup \{f'\}\)
end for
11. Return \(g_{CBI}(Z) := \bigotimes_{f \in F} f\)

Algorithm 4.2 Generalized Bucket Elimination Algorithm (GBE) for a CBI Inference Task

Input: A CBI problem \((X, D, R, F)\) and a variable subset of interest \(Z\)

Output: \(g_{CBI}(Z) = \bigoplus_{X-Z} \bigotimes_{f \in F} f\)

1. Choose an elimination ordering \(Y = \langle Y_1, \ldots, Y_k \rangle\), where \(Y = X \setminus Z\)
2. Initialize \(k + 1\) empty buckets \(B = \{B_0, B_1, \ldots, B_k\}\), \(B_i = \emptyset\)
3. for \(i = 1\) to \(r\) do
   4. if \(Y \cap \text{Scope}(f_i) = \emptyset\) then
      5. \(B_0 := B_0 \cup \{f_i\}\)
   else
      7. Find the variable \(Y_t \in Y \cap \text{Scope}(f_i)\) with maximum index \(t\);
      8. \(B_t := B_t \cup \{f_i\}\)
   end if
end for
11. for \(i = k\) to 1 do
12. \(f' := \bigoplus_{Y_i} \bigotimes_{f' \in B_i} f'\)
13. if \(Y \cap \text{Scope}(f') = \emptyset\) then
14. \(B_0 := B_0 \cup \{f'\}\)
else
16. Find a variable \(Y_t \in Y \cap \text{Scope}(f')\) with highest order;
17. \(B_t := B_t \cup \{f'\}\)
end if
19. end for
20. Return \(g_{CBI}(Z) := \bigotimes_{f \in B_0} f\)
us to exploit different elimination orderings. Identity elements for the two operations are not required, so we can relax the requirement of commutative semiring here. Then we can solve the problem using the concrete VE algorithm through instantiating our generalized algorithms.

### 4.1.2 Generalized VE Algorithm for k-Semiring

An extension of the generalized-VE algorithm works with CBI problems defined on a commutative k-semiring \((S, \text{op}_0, \text{op}_1, \cdots, \text{op}_k)\). In such cases, we repeatedly use the generalized-VE for the last two operations, then replace \(F\) by the new marginalized constraints\(^1\). Details of the generalized VE algorithm for CBI problems defined on a commutative k-semiring are shown in Algorithm 4.3.

#### Algorithm 4.3 Generalized VE Algorithm for k-Semiring (kGVE) for a CBI Inference Task

**Input:** A k-semiring-based CBI problem \((X, D, R, F)\) where \(R = (S, \text{op}_1, \text{op}_2, \cdots, \text{op}_{k+1})\) is a k-semiring, and \(Z = \{Z_1, \ldots, Z_k\}\), \(Z_1 \subseteq \cdots \subseteq Z_k\) is a set of variable subsets of interest

**Output:** \(g_{\text{CBI}}(Z) = \text{op}_{1}X_{-Z_1}\text{op}_{2}X_{-Z_2}\cdots\text{op}_{k}X_{-Z_k}\text{op}_{k+1}f_{\in F}f\)

1. for \(m = k\) to 1 do
2. \(F := \text{GVE}(X, D, Z_m, (S, \text{op}_m, \text{op}_{k+1}), F)\)
3. end for

4. \(g_{\text{AR}}(Z) := \text{op}_{k+1}f_{\in F}f\)

### 4.1.3 Instances of Variable Elimination Algorithm

Many concrete algorithms from different fields can be seen as instances of variable elimination.

For classic CSPs, Seidel [54] proposed a variable elimination algorithm in the early 80s. Later, the Adaptive Consistency (AC) algorithm [16] was proposed. It works by eliminating variables one by one, while deducing the effect of the eliminated variable on the rest of the problem. Adaptive Consistency algorithm can be easily understood as a VE algorithm through instantiating the semiring \((S, \oplus, \otimes)\) by \(\{\text{false, true}\}, \lor, \land\) in our generalized VE algorithm.

For propositional SAT problems, directional resolution is the core of the well-known Davis-Putnam (DP) algorithm [18]. The basic idea of directional resolution is exactly variable elimination. We can get the concrete DP algorithm through instantiating the generalized VE algorithm through using semiring \(\{\text{false, true}\}, \lor, \land\) as well.

For probabilistic inference, the variable elimination algorithm [64] and the bucket elimination algorithm [14] are the first two VE algorithms that are widely studied in tackling inference tasks for probability assessment, MPE, and

---

\(^1\)Interestingly, another variable elimination process eliminates the second last operation repeatedly.
MAP problems. The concrete algorithms can be obtained through instantiating the generalized VE algorithm through using semiring \((0, \infty), +, \times\) or \((0, \infty), \max, \times\).

For decision-making problems, Zhang [63] reduced the influence diagrams, the graphical representations of the decision-making problems to Bayesian networks; the variable elimination algorithm for probabilistic inference in BN is then applied to solve decision-making problems.

For maximum likelihood decoding of coding theory, Viterbi decoding [57] is an instance of variable elimination, which is defined on the semiring \((0, \infty), \max, \times\).

In [25], a variable elimination algorithm was proposed to find an optimal joint action in the coordination graph. The algorithm was applied in RoboSoccer successfully [36], incorporating other techniques like discretizing the continuous domain and introducing role assignments.

We can design other concrete VE algorithms for problems that can be abstractly represented by the semiring-based unified framework, through instantiating different semirings. In general, variable elimination is a variant of dynamic programming, which is the key idea of these algorithms.

### 4.1.4 Space and Time Complexity

Given a CBI problem \((X, D, R, F)\), the space and time complexities of the generalized variable elimination algorithm are discussed in terms of the following:

- \(G = (\mathcal{V}, \mathcal{E})\): primal graph representation of CBI problem;
- \(d = \max(|D_i|), D_i \in D\): maximum domain size of variables;
- \(w\): width of the tree decomposition of \(G\) along the elimination ordering;
- \(r = |F|\): number of constraints;
- \(r_i\): number of constraints with \(V_i\) in their scopes;

**Theorem 4.1** (Space Complexity of VE) The space complexity of the generalized variable elimination algorithm is \(O(|V| + d^{w+1})\).

**Proof:** To eliminate a variable \(V_i\), all constraints with \(V_i\) in their scopes are combined. Because \(|\text{Neighbors}(V_i)| \leq w\), the size of the combined constraint is at most \(d^{w+1}\). We eliminate variables sequentially and reuse the space. So the total size required of the generalized VE is \(O(|V| + d^{w+1})\). \(\square\)

**Theorem 4.2** (Time Complexity of VE) The time complexity of the generalized variable elimination algorithm is \(O(r \cdot d^{w+1})\).

**Proof:** To eliminate a variable \(Y_j \in Y = X \setminus Z\), we need at most \(t_i(\otimes) = (r_i - 1) \cdot d^{w+1} \otimes\) operations, and at most \(t_i(\ominus) = d^{w+1} \ominus\) operations. So in total we need at most \(T(\otimes) = \sum_{i=1}^{|V|} t_i(\otimes) = (r - |V|) \cdot d^{w+1}\), and \(T(\ominus) = \sum_{i=1}^{|V|} t_i(\ominus) = |V| \cdot d^{w+1}\). If in the semiring \(R, \oplus\) and \(\otimes\) have the same processing times, the total time of the generalized VE is \(O(r \cdot d^{w+1})\). \(\square\)
4.1.5 Discussion of Applying the VE Algorithm

The variable elimination algorithm is essentially a dynamic programming approach, which exploits the semiring structure of a constraint-based inference problem. The inference task is finished through repeatedly combining local constraints with a common variable in their scopes, caching the result of the marginalization as a new constraint, and posting the new constraint to refine the solution of the original problem. In theory, all CBI problems with semiring-based structures can apply the scheme of the generalized variable elimination algorithm. The only difference between one problem and another is in the different implementations of combination and marginalization.

In the application of VE algorithms, the identity elements of combination and marginalization operations are not necessary. The commutativity of combination is not a necessary property but desired. If commutativity does not hold, the only possible elimination ordering is decided by the representation of the problem. Otherwise, we could use heuristic or approximate techniques to find a near-optimal elimination ordering to reduce computation costs.

According to the complexity analysis, both the space and time complexities of the generalized VE algorithm are linear in the size of the task, but exponential in the width of a tree decomposition of the corresponding primal graph representation. For a large scale CBI problem with a complex graphical representation, treewidth (the lower bound of widths of all tree decompositions of the problem) is often intractable, which makes the direct application of the VE algorithm infeasible. There are basically two ways to overcome it: the first one is to seek approximation inference solutions in terms of some criteria; the second is to incorporate value properties of the variables and exploit the structure properties of the problem. The approximation inference will be discussed in Chapter 5. The improvement of VE algorithms from the value point of view will be discussed briefly in Section 8.2.

4.2 Junction Tree Algorithm

4.2.1 Generalized JT Algorithm

The major motivation that prompts researchers to use junction tree algorithms to solve CBI problems is handling multiple queries. The structure of a given CBI problem always remains unchanged, whereas the variables of interest may be frequently changed according to the requests of users. Junction tree algorithms can share intermediate computational results among different queries, which is an advantage relative to variable elimination algorithms. In fact, junction tree algorithms can be seen as memorized dynamic programming [9], where solutions for subproblems are memorized for later use. A junction tree is a structure that efficiently divides the original problem into subproblems.

Given a CBI problem $P = (X, D, R, F)$, the primal graph $G = (V, E)$ of $P$ can be re-organized as a secondary structure, junction tree $T = (C, S)$, by triangulating the primal graph and identifying the maximal cliques. A junction
tree is a tree decomposition of the original graph $G$. A junction tree is optimal if the maximum clique size (minus 1) is equal to the treewidth of $G$. Finding an optimal junction tree of the given graph, equivalent to finding the treewidth, is $\mathcal{NP}$-hard in general. Various heuristics search algorithms or approximation algorithms can be applied to find junction tree decompositions with near-optimal width.

Let $\mathcal{C} = \{C_1, \ldots, C_p\}$ be a set of clusters, where $C_i \subseteq V$, and $\mathcal{S} = \{S_1, \ldots, S_q\}$ is a set of separators. Here we also use $C_i$ and $S_j$ to denote a subset of variables $X$ or vertices $V$. $S_i = C_i \cap C_j$ if $C_i$ and $C_j$ are connected by separators $S_i$. Then we say $T = (\mathcal{C}, \mathcal{S})$ is a junction tree for a CBI problem $P = (X, D, R, F)$ if the following three properties hold:

- **Singly connected property:** $T$ is a tree;
- **Constraint allocation property:** For any constraint $f_i \in F$, there are some clusters $C_j$ such that $\text{Scope}(f_i) \subseteq C_j$;
- **Running intersection property:** If a vertex (or variable) appears in two clusters $C_i$ and $C_j$, then it also appears in all clusters on the unique path between $C_i$ and $C_j$.

In general, junction tree algorithms assign constraints to clusters and combine constraints in the same cluster. The combined constraint is marginalized and passed as a message between clusters. Following a specified message-passing scheme, the junction tree reaches consistency and any variable of interest can be queried through marginalizing out other variables in the cluster that contains that variable. Formally, the generalized junction tree algorithm for a CBI problem is shown in Algorithm 4.4, which is abstracted from the message-passing scheme in the Shenoy-Shafer architecture [55] of probabilistic inference.

One interesting case of applying the JT algorithm is that the variable subset of interest $Z$ in query is not contained in a single cluster. One naive way to overcome it is adding a clique of $Z$ before constructing the junction tree. However, such an approach hurts the flexibility of the JT algorithm to handle multiple queries. A more elegant solution is to find a subtree $T_Z = (C_Z, S_Z)$ of $T = (\mathcal{C}, \mathcal{S})$, where $Z \subseteq C = \bigcup_{C_i \in \mathcal{C}} C_i$. The answer to the query is computed as the marginal of the combination of all the local constraints together with all the incoming messages of the subtree $T_Z$. Formally:

$$g_{CBI}(Z) := \bigoplus_{C_i \in \mathcal{C}_Z} \bigotimes_{C_j \in \text{Neighbors}(C_i), C_j \in \mathcal{C}_Z} m(C_j, C_i)$$  \hspace{1cm} (4.2)$$

The basic idea of Eq. 4.2 is to treat the subtree $T_Z$ as a virtual cluster and compute the marginal as treating normal concrete clusters.

We can modify the generalized JT algorithm to solve the assignment task for a CBI problem. After computing $g_{CBI}(Z)$ in some cluster $C_i$, backtracking is applied to variables contained in $C_i$. After a value assignment for these variables is decided, the assignment is passed to all neighboring clusters of $C_i$. The
Chapter 4. Exact Inference Algorithms

Algorithm 4.4 Generalized JT Algorithm (GJT) for a CBI Inference Task

Input: A junction tree $T = (C, S)$ of a CBI problem $(X, D, R, F)$ and a variable subset of interest $Z$

Output: $g_{\text{CBI}}(Z) = \bigoplus_{X \setminus Z} \bigotimes_{f \in F} f$

1: Attach to each cluster $C_i$ a constraint $\phi_{C_i} = 1$
2: for each $f \in F$ do
3: Find a cluster $C_i$ such that $\text{Scope}(f) \subseteq C_i$
4: $\phi_{C_i} := \phi_{C_i} \otimes f$
5: end for
6: for each Edge $S_{ij}$ which is from clusters $C_i$ to $C_j$ do
7: if $C_i$ has received messages from all its neighbors other than $C_j$ then
8: $N_{i,j} := \text{Neighbor}(C_i) \setminus \{C_j\}$
9: $m(C_i, C_j)$ is the message sent from $C_i$ to $C_j$;
10: $m(C_i, C_j) := \bigoplus_{C_i \setminus S_{ij}} (\phi_{C_i} \otimes \bigotimes_{C_l \in N_{i,j}} m(C_l, C_i))$
11: end if
12: end for
13: for each $C_i \in C$ do
14: if $Z \subseteq C_i$ then
15: $\phi_{C_i} := \phi_{C_i} \otimes \bigotimes_{C_l \in \text{Neighbor}(C_i)} m(C_l, C_i)$
16: Return $g_{\text{CBI}}(Z) := \bigoplus_{C_i \setminus Z} \phi_{C_i}$
17: end if
18: end for

procedure recursively instantiates values of variables according to the messages passing from the instantiated clusters to the un-instantiated ones.

Following the result of [55], commutativity of both $\oplus$ and $\otimes$ is required, which ensures the correctness and completeness of the JT algorithm. The message-passing scheme in the generalized junction tree algorithm can be seen as consisting of two passes. By choosing any cluster as the root of the tree, the first pass is from the root to leaves and the second pass is from leaves to the root. We can use several message-passing scheduling techniques, such as parallel or hybrid computing, to improve the efficiency of message-passing. The message-passing scheme in Algorithm 4.4 is sometimes called Shenoy-Shafer architecture, following the authors' names of [55].

To apply junction tree algorithms, the commutativity of either $\otimes$ or $\oplus$ is required, which ensures the correctness and completeness of the JT algorithm. The identity elements of the two operations are required as well. Some revised versions of the junction tree algorithm require $\otimes$ has the combination invertible property, which makes the message-passing scheme more efficient, though in our generalized JT algorithm it is not mandatory.

In practice, many problems are graphically represented as disconnected junction trees. We can add an arbitrary edge between two clusters in different subtrees and assign an empty separator set to the new edge. The empty separator set between subtrees separates the message passing, as well as satisfies...
the junction tree properties. Now the problem can be solved by the proposed generalized junction tree algorithm. Another class of problems can not be efficiently re-organized as junction trees due to the intractable maximum cluster sizes. One revision is to use junction graphs. In such a case, the generalized JT algorithm does not work directly since cycles in the junction graph prohibit message-passing being terminated. However, some experimental results [1] show that we can get a high quality approximation through using the iteratively (loopy) message-passing scheme.

4.2.2 Generalized JT Algorithm for $k$-Semiring

Given a CBI problem defined on the commutative $k$-semiring $(S, \circ_k, \circ_1, \cdots, \circ_k)$, we can solve the problem by attaching $k$ constraints $\phi^{(j)}_{C_i}$ to each cluster $C_i$, where $1 \leq j \leq k$. Therefore, we have $k$ messages to pass to neighboring clusters of $C_i$, each corresponds to marginalize $\phi^{(j)}_{C_i}$ by $\circ_{j-1}$. The constraints are updated by combining all received messages through using $\circ_j$. Each constraint for a cluster is initialized as the identity element of the corresponding operation. At the beginning of the algorithm, $\phi^{(0)}_{C_i} = f^{(1)} \circ_j \phi^{(j+1)}_{C_i}$, where $f^{(j)}$ denotes the combination of constraints attached to the constraint $\phi^{(j)}_{C_i}$ of $C_i$. It is actually the basic idea behind the junction tree algorithm for solving decision-making problems in influence diagrams by Jensen et al. Details of the generalized JT algorithm for CBI problems defined on $k$-semirings are given in Algorithm 4.5.

4.2.3 Generalized JT Algorithm for Idempotent Semirings

The setup of the generalized junction tree algorithm in Algorithm 4.4 requires semirings of CBI problems be to commutative under both the combination and marginalization. The identity element of the combination is required though the identity element of marginalization is not necessary. According to these observations, any CBI problem with commutative semiring representation can be solved by instantiating the generalized JT algorithm.

In some CBI problems, the corresponding semirings have idempotency properties under combination operations, which provide possibilities for improving computational efficiencies of the generalized JT algorithm. A commutative semiring $(R, \circ, \otimes)$ is idempotent under combination if $\forall a \in R$, $a \circ a = a$. We call such a semiring an idempotent semiring, without explicitly mentioning that the idempotent property is actually for the combination operation.

Since combination corresponds to information (or constraint) gathering, the idempotency of combination implies that repeatedly combining the same information will not produce new information. Furthermore, considering the marginalization of a set of information $m = \bigoplus_i m_i$ in an idempotent semiring, we get $m = m \otimes m = m \otimes \bigoplus_i m_i = \bigoplus_i (m \otimes m_i)$. This induction implies $m = \bigoplus_i m_i = \bigoplus_i (m \otimes m_i)$, which means that combining the marginalization with original information will not produce new information after another
Algorithm 4.5 Generalized JT Algorithm for k-Semiring (kGJT) for a CBI Inference Task

Input: A junction tree $T = (C, S)$ of k-semiring-based CBI problem $(X, D, R, F)$, where $R = (S, op_1, op_2, \ldots, op_{k+1})$ is a k-semiring. $Z = \{Z_1, \cdots, Z_k\}$, $Z_1 \subset \cdots \subset Z_k$ are variable subsets of interest and $Z_k$ is contained in at least one cluster of $T$.

Output: $g_{k\text{CBI}}(Z) = op_1 X \setminus Z_i op_2 X \setminus Z_2 \cdots op_k X \setminus Z_k op_{k+1} f_j$

1: Build $k$ empty function buckets $F_i$, each corresponds to $op_{i+1}$
2: for each $f \in F$ do
3: if $f$ is a factor of $op_{i+1}$ then
4: Allocate $f$ to $F_i$
5: end if
6: end for
7: for each $C_i \in C$ do
8: Attach $C_i$ with a constraint $\phi_{C_i}$
9: Initialize $\phi_{C_i}$ to identity element of $op_{k+1}$
10: end for
11: for $m = k$ to 1 do
12: for each $f_i \in F_m$ do
13: Find a cluster $C_j$ such that $\text{Scope}(f_i) \subseteq C_j$
14: $\phi_{C_j} := \phi_{C_j} op_{k+1} f_i$
15: end for
16: for each Edge $S_{ij}$ which is between clusters $C_i$ and $C_j$ do
17: if $C_i$ has received messages from all its neighbors other than $C_j$ then
18: $N_{i \setminus j} := \text{Neighbor}(C_i) - \{C_j\}$
19: $m(C_i, C_j) = op_{m|C_i \setminus S_{ij}} op_{k+1} C_i \in N_{i \setminus j} m(C_i, C_i) op_{k+1} \phi_{C_i}$
20: end if
21: end for
22: for each $C_i \in C$ do
23: $\phi_{C_i} := op_{k+1} C_i \in N_{i \setminus j} m(C_i, C_i) op_{k+1} \phi_{C_i}$
24: end for
25: end for
26: Find a cluster $C_i$ such that $Z_i \subseteq C_i$
27: Return $g_{k\text{CBI}}(Z_i) := op_{1} X \setminus Z_i \phi_{C_i}$
marginalization.

The observation above shows that for an idempotent semiring, we can ignore the double-counted messages in the generalized junction tree algorithm, without loss of correctness. In Algorithm 4.6, the generalized JT is revised to cope with idempotent semirings.

Algorithm 4.6 Generalized JT Algorithm for Idempotent Semirings (GJT-Idemp) for a CBI Inference Task

Input: A junction tree $T = (C, S)$ of a CBI problem $(X, D, R, F)$ and a variable subset of interest $Z$

Output: $g_{CBI}(Z) = \bigoplus_{X \setminus Z} \bigotimes_{f \in F} f$

1. Attach each cluster $C_i$ with a constraint $\phi_{C_i} = 1$
2. for each $f \in F$ do
3. Find a cluster $C_i$ such that $\text{Scope}(f) \subseteq C_i$
4. $\phi_{C_i} := \phi_{C_i} \otimes f$
5. end for
6. Choose an arbitrary cluster $R \in C$ as the root of $T$
7. for each Cluster $C_i$ {Upward Phase} do
8. if All messages from $C_i$'s children clusters are ready then
9. $\phi_{C_i} := \phi_{C_i} \otimes (\bigotimes_{C_k \in \text{Children}(C_i)} m(C_i, C_k))$
10. if $C_i \neq R$ then
11. $C_j := \text{Parent}(C_i)$
12. $m(C_i, C_j) := \bigoplus_{C_i \setminus S_j} \phi_{C_i}$
13. end if
14. end if
15. end for
16. for each Non-leaf cluster $C_i$ {Downward Phase} do
17. if message from $C_i$'s parent clusters is ready then
18. if $C_i \neq R$ then
19. $\phi_{C_i} := \phi_{C_i} \otimes m(C_i, \text{Parent}(C_i))$
20. end if
21. for each $C_j \in \text{Children}(C_i)$ do
22. $m(C_i, C_j) := \bigoplus_{C_i \setminus S_j} \phi_{C_i}$
23. end for
24. end if
25. end for
26. Find a cluster $C_i$ such that $Z \subseteq C_i$
27. Return $g_{CBI}(Z) := \bigoplus_{C_i \setminus Z} \phi_{C_i}$

4.2.4 Generalized JT Algorithm for Invertible Semirings

Some CBI problems are defined on semirings with invertible properties under combination operations, which provide possibilities for improving the generalized JT algorithm. We call such a semiring an invertible semiring, with-
out explicitly mentioning that the invertible property under the combination operation. A commutative semiring \((R, \oplus, \otimes)\) is an invertible semiring if \(\forall a \in R, \exists a^{-1} \in R\), s.t. \(a \otimes a^{-1} = a^{-1} \otimes a = 1\). The binary operation \(\otimes\) of \(R\) is then defined as:

\[\forall a, b \in R, a \otimes b \equiv a \otimes b^{-1}\]

**Generalized JT Algorithm for Invertible Semirings**

The generalized JT algorithm can be modified to use the combination invertible property, as listed in Algorithm 4.7. The basic idea here is caching the combination of all incoming messages and dividing the specific incoming message to get the outgoing message for that separator. Dividing here eliminates duplicated information from the combined message.

**Lauritzen-Spiegelhalter (LS) architecture and HUGIN architecture**

There are another two message-passing schemes for semirings with combination invertible properties: the Lauritzen-Spiegelhalter (LS) architecture [39] and the HUGIN architecture [29]. We generalize the JT algorithm for LS architecture in Algorithm 4.8 and the JT algorithm for HUGIN architecture in Algorithm 4.9. The major differences between LS and HUGIN architectures are: (1) The reverses operation is done in the domain of separators in HUGIN but in the domain of clusters in LS. A small separator domain size enables HUGIN to achieve more time-efficiency than LS architecture; (2) HUGIN architecture requires additional storage for the constrains attached to separators, whereas LS architecture passes messages on the fly (do not need to be stored). A detailed discussion of the time and space complexities of Shenoy-Shafer architecture, Lauritzen-Spiegelhalter (LS) architecture, and HUGIN architecture can be found in the following sections.

**4.2.5 Instances of Junction Tree Algorithm**

Many concrete algorithms from different fields can be seen as instances of the generalized junction tree algorithm.

Junction tree algorithms have been widely used and studied in the probabilistic inference community since the late 1980s. Lauritzen and Spiegelhalter [39] developed a message-passing scheme in probabilistic inference based on clustered tree structures, which exploits the combination invertible property of semirings \(([0, \infty), +, \times)\) and \(([0, \infty), \max, \times)\). Shenoy and Shafer [55] independently developed another general message-passing scheme that does not rely on the combination reverse property. Such a scheme is general enough to be applied to CBI problems with any type of commutative semirings. Lately, Jensen et al. [29] revised the message-passing scheme and developed HUGIN architecture for probabilistic inference problems.
Algorithm 4.7 Generalized JT Algorithm for Invertible Semirings (GJT-Inv) for a CBI Inference Task

Input: A junction tree $T = (C, S)$ of a CBI problem $(X, D, R, F)$ and a variable subset of interest $Z$

Output: $g_{CBI}(Z) = \bigoplus_{X \setminus Z} \bigotimes_{f \in F} f$

1: Attach each cluster $C_i$ with a constraint $\phi_{C_i} = 1$
2: for each $f \in F$ do
3:    Find a cluster $C_i$ such that $\text{Scope}(f) \subseteq C_i$
4:    $\phi_{C_i} := \phi_{C_i} \otimes f$
5: end for
6: Choose an arbitrary cluster $R \in C$ as the root of $T$
7: for each Cluster $C_i$ {Upward Phase} do
8:    if All messages from $C_i$’s children clusters are ready then
9:        $\phi_{C_i} := \phi_{C_i} \otimes (\bigotimes_{C_k \in \text{Children}(C_i)} m(C_i, C_k))$
10:       if $C_i \neq R$ then
11:           $C_j := \text{Parent}(C_i)$
12:           $m(C_i, C_j) := \bigoplus_{C_i \setminus S_j} \phi_{C_i}$
13:       end if
14:    end if
15: end for
16: for each Non-leaf cluster $C_i$ {Downward Phase} do
17:    if message from $C_i$’s parent clusters is ready then
18:        if $C_i \neq R$ then
19:            $\phi_{C_i} := \phi_{C_i} \otimes m(C_i, \text{Parent}(C_i))$
20:        end if
21:        for each $C_j \in \text{Children}(C_i)$ do
22:            $m(C_i, C_j) := \bigoplus_{C_i \setminus S_j} (\phi_{C_i} \otimes m(C_j, C_i))$
23:        end for
24:    end if
25: end for
26: Find a cluster $C_i$ such that $Z \subseteq C_i$
27: Return $g_{CBI}(Z) := \bigoplus_{C_i \setminus Z} \phi_{C_i}$
Algorithm 4.8 Generalized JT Algorithm for LS Architecture (GJT-LS)

**Input:** A junction tree $T = (C, S)$ of a CBI problem $(X, D, R, F)$ and a variable subset of interest $Z$

**Output:** $g_{AR}(Z) = \bigoplus_{X \setminus Z} \bigotimes_{f \in F} f$

1. Attach each cluster $C_i$ with a constraint $\phi_{C_i} = 1$
2. for each $f \in F$ do
3.   Find a cluster $C_i$ such that $\text{Scope}(f) \subseteq C_i$
4.   $\phi_{C_i} := \phi_{C_i} \otimes f$
5. end for
6. Choose an arbitrary cluster $R \in C$ as the root of $T$
7. for each Cluster $C_i \neq R$ {Upward Phase} do
8.   if $C_i$ has received messages from all its children clusters then
9.     $C_j := \text{Parent}(C_i)$
10.    $m(C_i, C_j) := \bigoplus_{C_i \setminus S_{C_j}} \phi_{C_i}$
11.    $\phi_{C_j} := \phi_{C_j} \otimes m(C_i, C_j)$
12.    $\phi_{C_i} := \phi_{C_i} \otimes m(C_i, C_j)$
13. end if
14. end for
15. for each Non-leaf cluster $C_i$ {Downward Phase} do
16.   if $C_i$ has received messages from its parent cluster then
17.     for each $C_j \in \text{Children}(C_i)$ do
18.       $m(C_i, C_j) := \bigoplus_{C_i \setminus S_{C_j}} \phi_{C_i}$
19.       $\phi_{C_j} := \phi_{C_j} \otimes m(C_i, C_j)$
20.     end for
21. end if
22. end for
23. Find a cluster $C_i$ such that $Z \subseteq C_i$
24. Return $g_{CBI}(Z) := \bigoplus_{C_i \setminus Z} \phi_{C_i}$
Algorithm 4.9 Generalized JT Algorithm for HUGIN Architecture (GJT-HUGIN)

**Input:** A junction tree $T = (C, S)$ of a CBI problem $(X, D, R, F)$ and a variable subset of interest $Z$

**Output:** $g_{AR}(Z) = \bigoplus_{X \setminus Z} \otimes_{f \in F} f$

1. Attach each cluster $C_i$ with a constraint $\phi_{C_i} = 1$
2. **for each** $f \in F$ **do**
   3. Find a cluster $C_i$ such that $\text{Scope}(f) \subseteq C_i$
   4. $\phi_{C_i} := \phi_{C_i} \otimes f$
   5. **end for**

6. Choose an arbitrary cluster $R \in C$ as the root of $T$
7. **for each** Cluster $C_i \neq R$ **{Upward Phase} do**
   8. **if** $C_i$ has received messages from all its children clusters **then**
      9. $C_j := \text{Parent}(C_i)$
     10. $\phi_{S_{ij}} = m(C_i, C_j) := \bigoplus_{C_i \setminus S_{ij}} \phi_{C_i}$
     11. $\phi_{C_j} := \phi_{C_j} \otimes m(C_i, C_j)$
   12. **end if**
   13. **end for**

14. **for each** Non-leaf cluster $C_i$ **{Downward Phase} do**
15. **if** $C_i$ has received messages from its parent cluster **then**
   16. **for each** $C_j \in \text{Children}(C_i)$ **do**
      17. $m(C_i, C_j) := \bigoplus_{C_i \setminus S_{ij}} \phi_{C_i}$
     18. $\phi_{C_j} := \phi_{C_j} \otimes (m(C_i, C_j) \otimes \phi_{S_{ij}})$
     19. $\phi_{S_{ij}} := m(C_i, C_j)$
   20. **end for**
   21. **end if**
   22. **end for**

23. Find a cluster $C_i$ such that $Z \subseteq C_i$
24. Return $g_{CBI}(Z) := \bigoplus_{C_i \setminus Z} \phi_{C_i}$
The application of junction tree algorithms in constraint satisfaction problems can be traced back to [17]. A message-passing algorithm on the tree clustering structures of constraint networks were developed. The algorithm is essentially the same scheme as Shenoy-Shafer architecture in probabilistic inference, except that the join and project operations are used as relation combination and marginalization operations. As a special case of CSP, propositional SAT problems can be solved by the same algorithm on the semiring \( (\{false, true\}, \lor, \land) \).

In [65], Zhang and Mackworth generalized the Arc Consistency (AC) algorithm for constraint networks and designed TAC, a AC algorithm for the rooted join tree, to solve CSP. In the TAC algorithm, the marginalization operation corresponds to relational projection and the combination operation corresponds to relational join. As a result, the marginal of combining two relations is represented as relational semi-join. The contribution of [65] is proposing and analyzing parallel and distributed message-passing algorithms (PTAC and DTAC) that can be generalized to apply to junction tree algorithms from other disciplines.

The junction tree algorithm was explicitly generalized in decoding applications as Generalized Distributive Law (GDL) by Aji and McEliece [1]. Many concrete decoding algorithms, including the Baum-Welch algorithm, the Gallager-Tanner-Wiberg decoding algorithm, and the BCJR algorithm are all special cases of the GDL algorithm.

In decision-making problems, Jensen et al. [30] proposed an algorithm that attaches two types of potentials to the junction tree converted from the influence diagram, and solves the corresponding decision-making problem by passing messages in the junction tree.

We could design other concrete JT algorithms for problems that can be abstractly represented in the semiring-based unified framework by instantiating different semirings. Specifically, when a CBI problem is defined on the semiring with the combination invertible property, the generalized LS architecture and HUGIN architecture are both suitable to solve the inference task.

### 4.2.6 Space and Time Complexity

Given a CBI problem \( (X, D, R, F) \), the space and time complexities of the generalized junction tree algorithm are discussed in terms of the following notions:

- \( T = (C, S) \): a junction tree of the CBI problem;
- \( d = \max(|D_i|), D_i \in D \): the maximum domain size of variables;
- \( w \): the width of the junction tree \( T \);
- \( sep \): the maximum separator size of the junction tree \( T \);
- \( r = |F| \): the number of constraints;
- \( r_i \): the number of constraints allocated to cluster \( C_i \);
- \( deg_i \): degree of cluster \( C_i \);
Chapter 4. Exact Inference Algorithms

- \( k \): ratio of \( r \) and \( n \), etc. \( r = k \cdot n \).

**Theorem 4.3 (Space Complexity of JT in Shenoy-Shafer (SS) Architecture)**

The space complexity of the generalized junction tree algorithm in Shenoy-Shafer architecture is \( O(|C| \cdot d^{w+1}) \).

**Proof:** To pass messages in SS architecture, each cluster needs one storage for the combination of constraints allocated to this cluster, and another storage for the marginalized constraint after absorbing all the messages passing to it. So the needed storage for all clusters is at most \( 2 \cdot |C| \cdot d^{w+1} \). In each separator, both the upward and downward messages are stored, and the needed storage is then \( 2 \cdot |S| \cdot d^{sep} \) at most. The total storage needed for the JT algorithm in SS architecture is \( 2 \cdot |C| \cdot d^{w+1} + 2 \cdot |S| \cdot d^{sep} \). Since \( sep \leq w \) and \( |C| = |S| + 1 \) in trees, the space complexity is \( O(|C| \cdot d^{w+1}) \). \( \Box \)

**Theorem 4.4 (Space Complexity of JT in SS Architecture with Idempotent Semiring (SS-Idemp))**

The space complexity of the generalized junction tree algorithm in Shenoy-Shafer architecture with an idempotent semiring is \( O(|C| \cdot d^{w+1}) \).

**Proof:** To pass messages in SS-Idemp, each cluster needs only one storage for the combination of constraints initially allocated to this cluster. Lately, it is used to store the marginalized constraint after absorbing all the incoming messages. So the needed storage for all clusters is at most \( |C| \cdot d^{w+1} \). All messages do not need to be stored, so the space complexity is \( O(|C| \cdot d^{w+1}) \). \( \Box \)

**Theorem 4.5 (Space Complexity of JT in SS Architecture with Invertible Semiring (SS-Inv))**

The space complexity of the generalized junction tree algorithm in Shenoy-Shafer architecture with the combination invertible property is \( O(|C| \cdot d^{w+1}) \).

**Proof:** To pass messages in SS-Inv, each cluster needs only one storage for absorbing all the messages passing to it. So the needed storage for all clusters is at most \( |C| \cdot d^{w+1} \). In each separator, the upward and downward messages are stored in the same place, so the needed storage is then \( |S| \cdot d^{sep} \) at most. The total storage needed for the JT algorithm in SS-Inv architecture is \( |C| \cdot d^{w+1} + |S| \cdot d^{sep} \). Since \( sep \leq w \) and \( |C| = |S| + 1 \) in trees, the space complexity is \( O(|C| \cdot d^{w+1}) \). \( \Box \)

**Theorem 4.6 (Space Complexity of JT in Lauritzen-Spiegelhalter (LS) Architecture)**

The space complexity of the generalized junction tree algorithm in Lauritzen-Spiegelhalter architecture is \( O(|C| \cdot d^{w+1}) \).

**Proof:** To pass messages in LS, each cluster requires only one storage for the combination of constraints allocated to this cluster. After passing messages, the combined constraint is revised to be the marginal of the total constraints combination. So the needed storage for all clusters is at most \( |C| \cdot d^{w+1} \). In each separator, no message is stored since all messages are absorbed by the cluster on the fly. Then the total storage needed for JT algorithm in the LS architecture is at most \( |C| \cdot d^{w+1} \), which means the space complexity is \( O(|C| \cdot d^{w+1}) \). \( \Box \)
Theorem 4.7 (Space Complexity of JT in HUGIN Architecture) The space complexity of the generalized junction tree algorithm in HUGIN architecture is \(O(|C| \cdot d^{w+1})\).

**Proof:** To pass messages in HUGIN, each cluster requires only one storage for the combination of constraints allocated to this cluster. After passing messages, the combined constraint is revised to be the marginal of the total constraints combination. So the needed storage for all the clusters is at most \(|C| \cdot d^{w+1}\). In each separator, one storage is required to save the intermediate marginal constraint, which requires \(|S| \cdot d^{sep}\) storage at most. The total storage needed for the JT algorithm in HUGIN architecture is \(|C| \cdot d^{w+1} + |S| \cdot d^{sep}\), which is half of the storage needed for the JT algorithm in SS architecture. Since \(sep \leq w\) and \(|C| = |S| + 1\) in trees, the space complexity is \(O(|C| \cdot d^{w+1})\).

Then we discuss the time complexities of the generalized JT algorithm and its variants.

Theorem 4.8 (Time Complexity of JT in Shenoy-Shafer (SS) Architecture) The time complexity of the generalized junction tree algorithm in Shenoy-Shafer architecture is \(O(|C|^2 \cdot d^{w+1})\).

**Proof:** To combine the initial constraints allocated to clusters, \(\sum_{i=1}^{|C|} (r_i - 1) \cdot d^{w+1}\) \(\otimes\) operations are needed at most. To compute a message from \(C_i\) to \(C_j\), \((deg_i - 1) \cdot d^{w+1}\) \(\otimes\) operations and \((d^{w+1} / d^{sep}) \cdot d^{sep}\) \(\oplus\) operations are required at most. For each cluster \(C_i\), there are \(deg_i\) messages to compute, so in total \(\sum_{i=1}^{|C|} deg_i \cdot (deg_i - 1) \cdot d^{w+1}\) \(\otimes\) operations and \(\sum_{i=1}^{|C|} deg_i \cdot (d^{w+1} / d^{sep}) \cdot d^{sep}\) \(\oplus\) operations. Finally, to combine all the incoming messages, each cluster requires at most \(deg_i \cdot d^{w+1}\) \(\otimes\) operations. Combining all of them, the upper bound of \(\otimes\) operations \(T(\otimes)\) of the JT algorithm in SS architecture is:

\[
T(\otimes) = \sum_{i=1}^{|C|} (r_i - 1 + deg_i^2 - deg_i + deg_i) \cdot d^{w+1}
\]

\[
= \left( \sum_{i=1}^{|C|} deg_i^2 + r - |C| \right) \cdot d^{w+1},
\]

\[
\leq \left( \sum_{i=1}^{|C|} deg_i + r - |C| \right) \cdot d^{w+1}
\]

\[
= (2 \cdot |S| + r - |C|) \cdot d^{w+1}
\]

\[
\leq (4 \cdot |C|^2 + r - |C|) \cdot d^{w+1}
\]

And the upper bound of \(\oplus\) operations \(T(\oplus)\) of the JT algorithm in SS architecture is:
Chapter 4. Exact Inference Algorithms

\[ T(\oplus) = \sum_{i=1}^{|C|} deg_i \cdot (d^{w+1} - d^{sep}) \]
\[ \leq 2 \cdot |S| \cdot (d^{w+1} - d^{sep}) \]
\[ \leq 2 \cdot |C| \cdot d^{w+1} \]

If in the semiring \( R \), \( \oplus \) and \( \otimes \) have the same processing time, the total time of the generalized JT in SS architecture is \( O(|C|^2 \cdot d^{w+1}) \).

Theorem 4.9 (Time Complexity of JT in SS Architecture with Idempotent Semiring (SS-Idemp)) The time complexity of the generalized junction tree algorithm in Shenoy-Shafer architecture with an idempotent semiring is \( O(|C| \cdot d^{w+1}) \).

Proof: To combine the initial constraints allocated to clusters, \( \sum_{i=1}^{|C|} (r_i - 1) \cdot d^{w+1} \)
\( \otimes \) operations are needed at most. In the upward phase, SS-Idemp architecture needs at most \( \sum_{i=1}^{|C|} (deg_i - 1) \cdot d^{w+1} \otimes \) operations and \( \sum_{i=1}^{|C|} (d^{w+1} - d^{sep}) \oplus \).

In the downward phase, \( \sum_{i=1}^{|C|} d^{w+1} \otimes \) operations and \( \sum_{i=1}^{|C|} ((deg_i - 1) \cdot (d^{w+1} - d^{sep})) \oplus \) operations are required.

Combining all of them, the upper bound of \( \otimes \) operations \( T(\otimes) \) of the JT algorithm in SS-Idemp architecture is:

\[ T(\otimes) = \sum_{i=1}^{|C|} (r_i - 1 + deg_i - 1 + 1) \cdot d^{w+1} \]
\[ = (r - |C| + 2 \cdot |S|) \cdot d^{w+1} \]
\[ \leq (|C| + r) \cdot d^{w+1} \]

The upper bound of \( \oplus \) operations \( T(\oplus) \) of the JT algorithm in SS-Idemp architecture is:

\[ T(\oplus) = \sum_{i=1}^{|C|} (deg_i - 1 + 1) \cdot (d^{w+1} - d^{sep}) \]
\[ = 2 \cdot |S| \cdot (d^{w+1} - d^{sep}) \]
\[ \leq 2 \cdot |C| \cdot d^{w+1}. \]

If in the semiring \( R \), \( \oplus \) and \( \otimes \) have the same processing time, the total time of the generalized JT in SS-Inv architecture is \( O(|C| \cdot d^{w+1}) \).

If semiring \( R \) has the combination invertible property, message-computing in SS architecture can be improved by caching the combination of all the incoming messages and dividing the specific incoming message to get the outgoing message for that separator. We call it SS-Inv architecture.
Chapter 4. Exact Inference Algorithms

Theorem 4.10 (Time Complexity of JT in Shenoy-Shafer Architecture with Combination Invertible (SS-Inv)) The time complexity of the generalized junction tree algorithm in Shenoy-Shafer architecture with the combination invertible property is \( O(|C| \cdot d^{w+1}) \).

**Proof:** To combine the initial constraints allocated to clusters, \( \sum_{i=1}^{|C|} (r_i - 1) \cdot d^{w+1} \otimes \) operations are needed at most. In the upward phase, SS-Inv architecture needs at most \( \sum_{i=1}^{|C|} (deg_i - 1) \cdot d^{w+1} \otimes \) operations and \( \sum_{i=1}^{|C|} (d^{w+1} - d^{sep}) \oplus \) operations. In the downward phase, \( \sum_{i=1}^{|C|} d^{w+1} \otimes \) operations, \( \sum_{i=1}^{|C|} ((deg_i - 1) \cdot (d^{w+1} - d^{sep})) \oplus \) operations, and \( \sum_{i=1}^{|C|} ((deg_i - 1) \cdot d^{w+1}) \otimes \) operations are required.

Combining all of them, the upper bound of \( \otimes \) operations \( T(\otimes) \) of the JT algorithm in SS-Inv architecture is:

\[
T(\otimes) = \sum_{i=1}^{|C|} (r_i - 1 + deg_i - 1 + 1) \cdot d^{w+1} \\
= (r - |C| + 2 \cdot |S|) \cdot d^{w+1} \\
\leq (|C| + r) \cdot d^{w+1}
\]

The upper bound of \( \oplus \) operations \( T(\oplus) \) of the JT algorithm in SS-Inv architecture is:

\[
T(\oplus) = \sum_{i=1}^{|C|} (deg_i - 1 + 1) \cdot (d^{w+1} - d^{sep}) \\
= 2 \cdot |S| \cdot (d^{w+1} - d^{sep}) \\
\leq 2 \cdot |C| \cdot d^{w+1}
\]

And the upper bound of \( \ominus \) operations \( T(\ominus) \) of JT algorithm in SS-Inv architecture is:

\[
T(\ominus) = \sum_{i=1}^{|C|} (deg_i - 1) \cdot d^{w+1} \\
= (2 \cdot |S| - |C|) \cdot d^{w+1} \\
\leq |C| \cdot d^{w+1}
\]

If in the semiring \( R, \oplus, \otimes \) and \( \ominus \) have the same processing time, the total time of the generalized JT in SS-Inv architecture is \( O(|C| \cdot d^{w+1}) \). \( \Box \)

Theorem 4.11 (Time Complexity of JT in Lauritzen-Spiegelhalter (LS) Architecture) The time complexity of the generalized junction tree algorithm in Lauritzen-Spiegelhalter architecture is \( O(|C| \cdot d^{w+1}) \).
Proof: To combine the initial constraints allocated to clusters, \( \sum_{i=1}^{\lvert C \rvert} (r_i - 1) \cdot d^{w+1} \) \( \otimes \) operations are needed at most. In the upward phase, LS architecture needs at most \( \sum_{i=1}^{\lvert C \rvert} (\text{deg}_i - 1) \cdot d^{w+1} \otimes \) operations, \( \sum_{i=1}^{\lvert C \rvert} (d^{w+1} - d^{sep}) \oplus \) operations and \( \sum_{i=1}^{\lvert C \rvert} d^{w+1} \otimes \) operations. In the downward phase, \( \sum_{i=1}^{\lvert C \rvert} d^{w+1} \otimes \) operations and \( \sum_{i=1}^{\lvert C \rvert} (\text{deg}_i - 1) \cdot (d^{w+1} - d^{sep}) \oplus \) operations are required.

Combining all of them, the upper bound of \( \otimes \) operations \( T(\otimes) \) of the JT algorithm in LS architecture is:

\[
T(\otimes) = \sum_{i=1}^{\lvert C \rvert} (r_i - 1 + \text{deg}_i - 1 + 1) \cdot d^{w+1} \\
= (r - \lvert C \rvert + 2 \cdot \lvert S \rvert) \cdot d^{w+1} \\
\leq (\lvert C \rvert + r) \cdot d^{w+1}
\]

The upper bound of \( \oplus \) operations \( T(\oplus) \) of the JT algorithm in LS architecture is:

\[
T(\oplus) = \sum_{i=1}^{\lvert C \rvert} (\text{deg}_i - 1 + 1) \cdot (d^{w+1} - d^{sep}) \\
= 2 \cdot \lvert S \rvert \cdot (d^{w+1} - d^{sep}) \\
\leq 2 \cdot \lvert C \rvert \cdot d^{w+1}
\]

And the upper bound of \( \otimes \) operations \( T(\otimes) \) of the JT algorithm in LS architecture is:

\[
T(\otimes) = \sum_{i=1}^{\lvert C \rvert} d^{w+1} \\
= 2 \cdot \lvert S \rvert \cdot d^{w+1} \\
\leq 2 \cdot \lvert C \rvert \cdot d^{w+1}
\]

If in the semiring \( R, \oplus, \otimes, \) and \( \oplus \) have the same processing time, the total time of the generalized JT in LS architecture is \( O(\lvert C \rvert \cdot d^{w+1}) \). □

**Theorem 4.12 (Time Complexity of JT in HUGIN Architecture)** The time complexity of the generalized junction tree algorithm in HUGIN architecture is \( O(\lvert C \rvert \cdot d^{w+1}) \).

Proof: To combine the initial constraints allocated to clusters, \( \sum_{i=1}^{\lvert C \rvert} (r_i - 1) \cdot d^{w+1} \) \( \otimes \) operations are needed at most. In the upward phase, HUGIN architecture
needs at most \( \sum_{i=1}^{\lvert C \rvert} (\text{deg}_i - 1) \cdot d^{w+1} \otimes \) operations and \( \sum_{i=1}^{\lvert C \rvert} (d^{w+1} - d^{\text{sep}}) \otimes \) operations. In the downward phase, \( \sum_{i=1}^{\lvert C \rvert} d^{w+1} \otimes \) operations, \( \sum_{i=1}^{\lvert C \rvert} (\text{deg}_i - 1) \cdot (d^{w+1} - d^{\text{sep}}) \) \( \otimes \) operations and \( \sum_{i=1}^{\lvert C \rvert} d^{\text{sep}} \otimes \) operations are required.

Combining all of them, the upper bound of \( \otimes \) operations \( T(\otimes) \) of the JT algorithm in HUGIN architecture is:

\[
T(\otimes) = \sum_{i=1}^{\lvert C \rvert} (r_i - 1 + \text{deg}_i - 1 + 1) \cdot d^{w+1}
= (r - \lvert C \rvert + 2 \cdot \lvert S \rvert) \cdot d^{w+1}
\leq (\lvert C \rvert + r) \cdot d^{w+1}
\]

The upper bound of \( \otimes \) operations \( T(\otimes) \) of the JT algorithm in HUGIN architecture is:

\[
T(\otimes) = \sum_{i=1}^{\lvert C \rvert} (\text{deg}_i - 1 + 1) \cdot (d^{w+1} - d^{\text{sep}})
= 2 \cdot \lvert S \rvert \cdot (d^{w+1} - d^{\text{sep}})
\leq 2 \cdot \lvert C \rvert \cdot d^{w+1}.
\]

And the upper bound of \( \otimes \) operations \( T(\otimes) \) of the JT algorithm in HUGIN architecture is:

\[
T(\otimes) = \sum_{i=1}^{\lvert C \rvert} d^{\text{sep}}
= 2 \cdot \lvert S \rvert \cdot d^{\text{sep}}
\leq 2 \cdot \lvert C \rvert \cdot d^{w+1}.
\]

If in the semiring \( R, \otimes, \otimes, \) and \( \otimes \) have the same processing time, the total time of the generalized JT in HUGIN architecture is \( O(\lvert C \rvert \cdot d^{w+1}) \). \( \Box \)

4.2.7 Discussion of Applying the JT Algorithm

In general, junction tree algorithms are memorized dynamic programming, which cache solutions for subproblems to answer multiple queries. Both the time and space complexities of junction tree algorithms are polynomial in the size of the junction tree, with a constant factor exponential in the maximum subproblem size (width of the tree plus 1). So the key to applying JT algorithms is to divide the original CBI problem into subproblems with tractable sizes. However,
many practical CBI problems have large treewidth. As the lower bound of the width of any tree decomposition, a large treewidth makes exact inference in the junction tree infeasible. There are two ways to perform approximate inference in such a situation: (1) Splitting the oversized clusters, which corresponds to removing some edges from the primal graph representation, or retracting some constraints in the original CBI problem; (2) Using junction graphs to perform inference, instead of junction trees. In a junction graph, messages can pass in loops and may not terminate, which means that information may be counted repeatedly. Some criteria are used to terminate the message-passing after reaching preset thresholds. Both of these approaches will be discussed in Chapter 5. Another interesting approach for performing exact inference in junction trees is to consider the effect of variable values and their impacts on the structure of the tree. We will briefly discuss this in Section 8.2.

All concrete junction tree algorithms discussed in this chapter use serial message-passing schemes. In practical applications, there are more efficient implementations. Both paralleled and hybrid message-passing schemes can be applied to achieve computational efficiency.

The time and space complexities of the generalized junction tree algorithm, as well as the generalized variable elimination algorithm, are compared in Table 4.2. Here $G = (V, E)$ is the primal graph of a CBI problem $(X, D, R, F)$. $T = (C, S)$ is a junction tree transformed from $G$. Both the VE and the JT algorithms are exponential in the induced width of a given elimination ordering, or the width of the junction tree representation. For the VE and the JT algorithms with combination invertible properties or combination idempotency properties, the time complexity is linear in the size of the problem, whereas in the generalized JT, it is quadratic in the size of the problem.

<table>
<thead>
<tr>
<th>Algor.</th>
<th>Space Complexity</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>GVE</td>
<td>$O(</td>
<td>V</td>
</tr>
<tr>
<td>GJT</td>
<td>$O(</td>
<td>C</td>
</tr>
<tr>
<td>GJT-Idemp</td>
<td>$O(</td>
<td>C</td>
</tr>
<tr>
<td>GJT-Inv</td>
<td>$O(</td>
<td>C</td>
</tr>
<tr>
<td>GJT-LS</td>
<td>$O(</td>
<td>C</td>
</tr>
<tr>
<td>GJT-HUGIN</td>
<td>$O(</td>
<td>C</td>
</tr>
</tbody>
</table>

The big $O$ expressions in Table 4.2 do not convey sufficient information about the running times of these algorithms. The upper bounds of different operations in these algorithms are listed in Table 4.3.

According to the running time of each concrete operation in Java (c.f. Table 4.4), users can compare the running times of different junction tree algorithms and instantiate these algorithms for their own purposes. For a CBI problem with the combination invertible property, More specifically, for a CBI problem defined on a semiring with the combination invertible property, generally we conclude: (1) GJT-Inv uses the least time, followed by GJT-HUGIN and then
Table 4.3: Upper Bounds of VE and JT Running Times

<table>
<thead>
<tr>
<th>Algor.</th>
<th>$\otimes$</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GVE</td>
<td>$(r -</td>
<td>V</td>
<td>) \cdot d^{w+1}$</td>
</tr>
<tr>
<td>GJT</td>
<td>$(4 \cdot</td>
<td>C</td>
<td>+ r) \cdot d^{w+1}$</td>
</tr>
<tr>
<td>GJT-Idemp</td>
<td>$(</td>
<td>C</td>
<td>+ r) \cdot d^{w+1}$</td>
</tr>
<tr>
<td>GJT-Inv</td>
<td>$(</td>
<td>C</td>
<td>+ r) \cdot d^{w+1}$</td>
</tr>
<tr>
<td>GJT-LS</td>
<td>$(</td>
<td>C</td>
<td>+ r) \cdot d^{w+1}$</td>
</tr>
<tr>
<td>GJT-HUGIN</td>
<td>$(</td>
<td>C</td>
<td>+ r) \cdot d^{w+1}$</td>
</tr>
</tbody>
</table>

GJT-LS; (2) GJT-LS uses the least space while GJT-HUGIN and GJT-Inv use about the same space. This conclusion is supported by the experimental results in Chapter 7.

Table 4.4: Running Times (per $10^6$ operation) for Different Operations in Java on a PIII833MHz PC, Running Windows XP and JDK1.4

<table>
<thead>
<tr>
<th>Operations</th>
<th>Running Time (ms)</th>
<th>Objects in Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\times$</td>
<td>1570</td>
<td>Double, Float, Integer</td>
</tr>
<tr>
<td>$+$</td>
<td>1534</td>
<td>Double, Float, Integer</td>
</tr>
<tr>
<td>$\div$</td>
<td>1584</td>
<td>Double, Float, Integer</td>
</tr>
<tr>
<td>$\max$</td>
<td>1687</td>
<td>Double, Float, Integer</td>
</tr>
<tr>
<td>$\min$</td>
<td>1688</td>
<td>Double, Float, Integer</td>
</tr>
<tr>
<td>$\lor$</td>
<td>128</td>
<td>Boolean</td>
</tr>
<tr>
<td>$\land$</td>
<td>128</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

4.3 Summary

In this chapter, two popular exact inference algorithms for CBI problems are generalized based on our semiring-based unified framework. Both variable elimination algorithms and junction tree algorithms are dynamic programming, where junction tree algorithms cache solutions of subproblems for answering multiple queries. In some semirings with the combination invertible property or the combination idempotency property, the JT algorithms can be improved to achieve more computational efficiencies. Generalized versions of these improved algorithms are discussed as well.

In this chapter, we conclude that many concrete algorithms for CBI problems from different disciplines are actually special cases of the generalized variable elimination algorithms and the generalized junction tree algorithms. These algorithms are listed in Table 4.5 for comparison.

The complexity analyses indicate that both the time and space complexities of VE and JT algorithms are polynomial in the size of the CBI problem, but have a constant factor which is exponential in the width of the tree decomposition.
Therefore, finding an optimal elimination ordering to minimize the induced width, which is equivalent to finding a tree decomposition with the minimum width, is the key of applying these two exact inference algorithms. For many practical problems with intractable treewidth (lower bound of the widths of all tree decompositions), exact VE and JT algorithms are infeasible. In such cases, we can either use approximate inference approaches if permitted, or exploit the value of variables, as well as the structures of the CBI problems to reduce the scale of the problems.
Table 4.5: Concrete VE and JT Algorithms in Different Fields.

<table>
<thead>
<tr>
<th>Fields</th>
<th>VE</th>
<th>JT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint Satisfaction</td>
<td>VE for CSP (Seidel [54]); Adaptive Consistency (AC) (Dechter and Pearl [16])</td>
<td>Bucket-Tree Elimination (Dechter and Pearl [17]); Arc Consistency for JT (TAC) (Zhang and Mackworth [65])</td>
</tr>
<tr>
<td>Satisfiability Problems</td>
<td>Davis-Putnam (DP) (Dechter and Rish [18])</td>
<td>Bucket-Tree Elimination (Dechter and Pearl [17])</td>
</tr>
<tr>
<td>Probability Inference in BN</td>
<td>Variable Elimination Algorithm [64]; Bucket Elimination (Dechter [14])</td>
<td>SS (Shenoy and Shafer [55]); LS (Lauritzen and Spiegelhalter [39]); and HUGIN (Jensen et al. [29])</td>
</tr>
<tr>
<td>Decision Making</td>
<td>Transform to VE of BN (Zhang [63])</td>
<td>JT for ID (Jensen et al. [30])</td>
</tr>
<tr>
<td>Dynamic Bayesian Network</td>
<td>-</td>
<td>Belief Propagation (BP)</td>
</tr>
<tr>
<td>Possibility Inference</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Coordination Games</td>
<td>VE for Coordination Graph (Guestrin et al. [25])</td>
<td>-</td>
</tr>
<tr>
<td>Max-Likelihood Decoding</td>
<td>Viterbi Decoding (Viterbi [57])</td>
<td>Generalized Distributive Law (GDL) (Aji and McEliece [1])</td>
</tr>
</tbody>
</table>
Chapter 5

Approximate Inference Algorithms

5.1 Motivation for the Design of Approximate Algorithms

The analyses in previous chapters show that both the variable elimination algorithm and the junction tree algorithm can perform exact inference for a constraint-based inference problem in polynomial time and space. However, there always exist constant factors in the time and space complexities that are exponential in the maximum cluster size of the problem's tree decomposition. This means that the generalized exact inference algorithms would be infeasible when the treewidth of the corresponding problem, a lower bound of width for all possible tree decompositions, is intractable. This practical challenge encourages researchers to develop approximate inference algorithms for CBI problems from different disciplines, if approximate inference results with some quality guarantee are acceptable in their application domains.

The key idea of these approximate inference algorithms is to restrict the size of the maximum subproblem, or equivalently, the maximum cluster size of a tree decomposition or the induced width given an elimination ordering, to an acceptable level. In general, there are at least two possible ways to design an algorithm to achieve this purpose. The first approach is to revise the original CBI problems by removing some less important constraints, which makes the structure of the problem's graphical representation much simpler; another approach does not touch the original CBI problem but re-organizes it into more complex graphical representations, e.g., a junction graph with loops. Inference procedures are carefully re-designed to cope with these graphical representations.

5.2 Algebraic Approximation for VE and JT

The algebraic foundation of approximate algorithms is based on Eq. 5.1. Here $F$ is a set of constraints and $F_i$ is a subset of constraints for $i = 1, \ldots, b$, where $F_1 \cup \cdots \cup F_b = F$ and $F_i \cap F_j = \emptyset$ for any $i, j \in \{1, \ldots, b\}, i \neq j$. The basic idea here is breaking the original CBI problem into $b$ (overlapped) subproblems, solving them individually, then joining them again to get the solution.
Algorithm 5.1 describes how to compute the approximate marginal of combination.

\[
\bigoplus_{Y \in F} \bigotimes_{i=1}^{b} \bigoplus_{f \in F_i} f \approx \bigotimes_{Y \in F} \bigoplus_{f \in F_i} f
\]  

Algorithm 5.1: The Approximate Marginal of Combination Algorithm (ApproxMC)

**Input:** \( F \): set of constraints with variable \( Y \) in their scopes, \( b \): number of approximate sets, \( w_{\text{max}} \): threshold width

**Output:** \( F' = \{f_1, \cdots, f_b\} \) where

\[
\bigoplus_{f' \in F'} f' \approx \bigotimes_Y \bigoplus_{f \in F} f
\]

1: \( S = \emptyset \)
2: for each \( f \in F \) do
3: \( S := S \cup \text{Scope}(f) \)
4: end for
5: if \(|S| - 1 \leq w_{\text{max}}\) then
6: \( f' := \bigotimes_Y \bigoplus_{f \in F} f \)
7: \( F' := \{f'\} \)
8: else
9: \( F' := \emptyset \)
10: for \( j = 1 \) to \( b \) do
11: \( F_j := \emptyset \)
12: end for
13: for each \( f \in F \) do
14: Choose \( j \in \{1, \cdots, b\} \)
15: \( F_j := F_j \cup \{f\} \)
16: end for
17: for \( j = 1 \) to \( b \) do
18: \( F'_{j} := \text{ApproxMC}(F_j, Y, b, w_{\text{max}}) \)
19: \( F' := F' \cup F'_{j} \)
20: end for
21: end if
22: Return \( F' \)

In the following sections we will present the approximate VE and JT algorithms based on the subroutine \( \text{ApproxMC} \).

### 5.2.1 Approximate Variable Elimination Algorithm

According to the generalized exact variable elimination algorithm in Chapter 4, the bottleneck of computation occurs when we eliminate a variable \( X_i \), too many constraints with \( X_i \) in their scopes have to be combined, which implies a large constraint after combining and marginalizing. For example, in Figure 5.1(a), \( Y_1, \cdots, Y_6 \) are fully connected after eliminating \( X_1 \), which means that the size of the marginal constraint is exponential in \( 6 \).
One way to overcome this bottleneck is to clone $X_i$ with many identical copies $X_i^{(1)}, \ldots, X_i^{(b)}$. Constraints with $X_i$ in their scopes are revised by replacing $X_i$ with $X_i^{(j)}, j \in \{1, \ldots, b\}$ according to specified rules. Then the VE algorithm can be applied. In Figure 5.1(b), applying $b = 2$ leads to two marginal constraints with cubic sizes. Of course, introducing identical copies $X_i^{(1)}, \ldots, X_i^{(b)}$ of $X_i$ will introduce conflicts of the $X_i$ values and errors in the marginalization. Experimental analyses appear in Chapter 8.

![Figure 5.1: Graphical Interpretation of Approximate VE.](image)

Min-Buckets [19] is the first algorithm proposed for applying this idea to solve probability inference problems approximately. In this section, we generalize the idea of Min-Buckets as the approximate VE algorithm in Algorithm 5.2.

**Algorithm 5.2 Generalized Approximate VE Algorithm (GVE-Approx)**

**Input:** A CBI problem $(X, D, R, F)$, a variable subset of interest $Z$, $b$, $w_{\text{max}}$

**Output:** $\tilde{g}_{\text{CBI}}(Z) = \bigotimes_{f \in F} f \approx \bigotimes_{X - Z} \bigotimes_{f \in F} f$

1. Choose an elimination ordering $\langle Y_1, \ldots, Y_k \rangle$, where $\{Y_1, \ldots, Y_k\} = X - Z$

2. for $i = k$ to 1 do
3.     for each $f \in F$ do
4.         $F_b := \emptyset$
5.         if $Y_i \in \text{Scope}(f)$ then
6.             $F := F \setminus \{f\}$
7.             $F_b := F_b \cup \{f\}$
8.         end if
9.     end for
10. $F' := \text{ApproxMC}(F_b, Y_i, b, w_{\text{max}})$
11. $F := F \cup F'$
12. end for
13. Return $\tilde{g}_{\text{CBI}}(Z) = \bigotimes_{f \in F} f$
5.2.2 Approximate Junction Tree Algorithm

The generalized junction tree algorithm can be modified to perform approximate inference based on Eq. 5.1. The basic idea of the approximate JT algorithm is to pass a set of messages from one cluster to another, instead of passing a single message. The combination of these messages is an approximation of the message passed in the exact JT algorithm. For example, in Figure 5.2(a), $m_{ij}$ is marginalized from the combination of $m_1$ to $m_6$, together with the local constraint of $C_i$; in Figure 5.2(b), $m_{ij}$ is approximated by the combination of $m_{ij}^{(1)}$ and $m_{ij}^{(2)}$, each of which is generated from the marginalization of three messages and a part of local constraints.

![Graphical Interpretation of Approximate JT](image)

The Min-Clustering Tree [43] is an approximate probability inference algorithm following this idea. In this section, we generalize the idea of The Min-Clustering Tree as the approximate junction tree algorithm in Algorithm 5.3.

For semirings with combination invertible properties, the reverse of a message is similarly approximated by the combination of a set of the messages' inverses. The approximation can be formalized as Eq. 5.2.

$$1 \otimes \left( \bigoplus_{Y} \bigotimes_{f \in F} f \right) \approx \bigotimes_{i=1}^{b} \left( 1 \otimes \left( \bigoplus_{Y} \bigotimes_{f \in F_i} f \right) \right) \quad (5.2)$$

After applying Eq. 5.2, we can revise any one of the three exact junction tree algorithms for invertible semirings to cope with approximate inference tasks.

5.2.3 Discussion

Algebraic approximation is the foundation of many approximate inference approaches for CBI problems. On the CBI problems level, it is equivalent to retracting some given constraints. On the primal graph representation level, it is equivalent to removing some edges from the graph. On the junction tree representation level, it is equivalent to splitting some clusters in the junction.
Algorithm 5.3 Generalized Approximate JT Algorithm (GJT-Approx)

Input: A junction tree $T = (C, S)$ of a CBI problem $(X, D, R, F)$, a variable subset of interest $Z$, $b$ and $w_{\text{max}}$

Output: \{\Phi_C\} s.t. $\exists C_i \in C$, $\tilde{\gamma}_{CBI}(Z) = \bigotimes_{\Phi \in \Phi_C} (\bigoplus_{C_i \in \mathcal{C} \setminus Z} \Phi) \approx \bigotimes_{X-Z} \bigotimes_{J \in F} f$

1. Attach each cluster $C_i$ with a constraint set $\Phi_{C_i} = \emptyset$
2. for $i = 1$ to $r$ do
3. Find a cluster $C_j$ such that $\text{Scope}(f_i) \subseteq C_j$
4. $\Phi_{C_i} := \Phi_{C_i} \cup \{f_i\}$
5. end for
6. for each Edge $S_{ij}$ which is from cluster $C_i$ to $C_j$ do
7. $M(C_i, C_j) := \emptyset$
8. if $C_i$ has received messages from all its neighbors other than $C_j$ then
9. $Q := \Phi_{C_i}$
10. for each $C_i \in (\text{Neighbor}(C_i) \setminus \{C_j\})$ do
11. $Q := Q \cup M(C_i, C_i)$
12. end for
13. $M(C_i, C_j) := \text{Approx}MC(Q, C_i \setminus S_{ij}, b, w_{\text{max}})$
14. end if
15. end for
16. for each $C_i \in C$ do
17. for each $C_j \in \text{Neighbor}(C_i)$ do
18. $\Phi_{C_i} := \Phi_{C_i} \cup M(C_j, C_i)$
19. end for
20. $\Phi_{C_i} := \text{Approx}MC(\Phi_{C_i}, \emptyset, b, w_{\text{max}})$
21. end for

The purpose of these approximations is to restrict the size of the maximum subproblem to a tractable level. Though the idea of algebraic approximation is straightforward, it is hard to analyze the error bounds of these approaches, especially when the combination and marginalization operations are abstract. So far there are few theoretical guidelines for choosing which constraints should be released (which edges should be moved, how to split a cluster); only empirical analyses currently apply in such cases.

5.3 Thin Junction Tree Algorithm

Recently, Paskin [48] introduced the thin junction tree algorithm to solve Simultaneous Localization and Mapping (SLAM) problems. A thin junction tree is a tractable approximation of the original junction tree representation that imposes an upper bound of the cluster size. Generally, it can be seen as a special case of the generalized approximate JT algorithm in Section 5.2.2. Given a cluster $C$ with intractable size $|C|$, the thin junction tree algorithm splits it into two clusters. One cluster $C_1$ consists of only one variable $v$ and another cluster
Chapter 5. Approximate Inference Algorithms

$C_2$ consists of $|C| - 1$ variables, where $v \notin C_2$. If the cluster $C$ has only one neighbor cluster $C'$ with $v \in C'$, $C_1$ can be safely absorbed by $C'$. All of the incoming messages to $C$, other than the one from $C'$, can be redirected to $C_2$. Variable $v$ is not contained in these messages. Then we can connect $C_2$ to $C'$ and remove $C$ from the junction tree. The size of $C$ is reduced to $|C_2| = |C| - 1$. The procedure can be performed repeatedly to restrict the cluster size under an imposed level.

5.3.1 Thinning through Variable Contraction

According to the description in [48], the basic operation for making a thin junction tree is variable contraction.

**Definition 5.1 (Variable Contraction [48])** Let $x$ be a variable that appears in more than one cluster of the junction tree $T$. Let $C$ be a leaf of $T_x$ (the subtree of $T$ induced by clusters containing $x$). Let $S$ be the separator joining $C$ to $T_x$. A variable contraction of $x$ from $C$ removes $x$ from $C$ and $S$, then marginalizes $x$ out of $\phi_C$ (and out of $\phi_S$ if using HUGIN architecture). If $C$ becomes non-maximum, it is merged into its subsuming neighbor.

Let $\hat{T}$ be a thin junction tree after performing a variable contraction of $T$. It is shown [48] that $\hat{T}$ is still a tree with the running intersection property. The procedure of variable contraction is formalized in Algorithm 5.4, according to [48].

**Algorithm 5.4 Variable Contraction (Contract($v$, $C_i$))**

**Input:** Contract variable $v$ from cluster $C_i$, $\Phi_{C_i}$ is a set of constraints allocated to $C_i$

**Output:** $\hat{\Phi}_{C_i}$ after contraction

1: Let $C_j$ be the parent cluster of $C_i$ in $T_v$
2: $\hat{\Phi}_{C_i} := \emptyset$
3: $\phi_v := 1$
4: for each $f \in \Phi_{C_i}$ do
5:  if $\text{Scope}(f) \subseteq C_j$ then
6:   $\Phi_{C_j} := \Phi_{C_j} \cup \{f\}$
7:  else if $v \in \text{Scope}(f)$ then
8:   $\phi_v := \phi_v \otimes f$
9:  else
10:   $\Phi_{C_i} := \Phi_{C_i} \cup \{f\}$
11: end if
12: end for
13: $C_i := C_i \setminus \{v\}$
14: $S_{ij} := S_{ij} \setminus \{v\}$ where $C_j$ i
15: $f := \bigoplus \phi_v$
16: $\hat{\Phi}_{C_i} := \Phi_{C_i} \cup \{f\}$
There are two special cases for performing variable contractions. The first case is that \( v \) appears only in one cluster \( C_i \) of the tree. In this case, we can clone \( C_i \) to obtain an identical cluster \( C_j \), add an edge between \( C_i \) and \( C_j \), contract \( v \) from \( C_i \) to \( C_j \), and finally contract some other variable \( u \) from \( C_j \) to \( C_i \). The second case is that \( v \) appears elsewhere, but \( v \) is not a leaf of \( T_v \). In this case, we can contract \( v \) from the other clusters until \( C_i \) is a leaf of \( T_v \), then contract \( v \) from \( C_i \).

### 5.3.2 Generalized Thin Junction Tree Algorithm

The basic procedure of the thin junction tree algorithm is to (1) build a junction tree, (2) limit the maximum cluster size by a given upper bound by performing variable contraction, and (3) pass messages within the thin junction tree. The generalized thin junction tree algorithm within the semiring-based CBI framework is shown in Algorithm 5.5.

**Algorithm 5.5 Generalized Thin Junction Tree Algorithm (GThinJT)**

**Input:** A junction tree \( T = (C, S) \) of a CBI problem \( P = (X, D, R, F); Z \): an variable set of interests; \( w_{\text{max}} \): the upper bound of cluster size

**Output:** \( \tilde{g}_{\text{CBI}}(Z) \approx \bigoplus X \subset Z \otimes_{f \in F} f \)

1. Attach each cluster \( C_i \) with a constraint set \( \Phi_{C_i} := \emptyset \)
2. For each \( f \in F \) do
3. Find a cluster \( C_i \) such that \( \text{Scope}(f) \subseteq C_i \)
4. \( \Phi_{C_i} := \Phi_{C_i} \cup \{f\} \)
5. End for
6. For each \( C_i \in C \) s.t. \( |C_i| > w_{\text{max}} \) do
7. \( v = \text{ToBeContracted}(C_i) \)
8. If \( v \) only appears in \( C_i \) then
9. Clone \( C_j = C_i \)
10. \( C := C \cup C_j \) and \( S := S \cup S_j \)
11. Contract \( (v, C_i) \)
12. \( u = \text{ToBeContracted}(C_j) \)
13. Contract \( (u, C_j) \)
14. Else if \( v \) is the leaf of \( T_v \) then
15. Contract \( (v, C_i) \)
16. Else
17. Recursively contract \( v \) from leaf of \( T_v \) until \( C_i \) is the leaf of \( T_v \)
18. Contract \( (v, C_i) \)
19. End if
20. End for
21. Perform message-passing of the generalized JT algorithm in the thin JT \( \tilde{T} = (\tilde{C}, \tilde{S}) \)
22. Find a cluster \( \tilde{C} \in \tilde{C} \) s.t. \( Z \subseteq \tilde{C} \)
23. Return \( \tilde{g}_{\text{CBI}}(Z) \approx \bigoplus \tilde{C} \in Z \Phi_{\tilde{C}} \otimes \bigoplus_{C_i \in \text{Neighbors}(\tilde{C})} m(C_j, \tilde{C}) \)
5.3.3 Discussion

The key of the thinning junction tree algorithm is to implement procedure $v = \text{ToBeContracted}(C_i)$, which chooses a variable to contract from the cluster. The chosen variable should minimize the approximation error. In probabilistic inference, the approximation error is often measured as *Kullback Leibler (KL)* divergence.

**Definition 5.2 (Kullback Leibler (KL) Divergence [34])** KL divergence $D(p \parallel q)$ of two discrete probability distributions $p(x)$ and $p(y)$ is defined as:

$$D(p \parallel q) = \sum_x p(x) \log \frac{p(x)}{p(y)}$$

Paskin [48] shows that the error introduced by contracting $x$ from a cluster $C$ can be computed using only the marginal over $C$. Thus the error can be estimated through local computation; also the errors of a series of contractions are additive, providing a way to estimate the approximation of the accumulated error of the thinning junction tree algorithm.

For other application domains without explicit error measurements or error additive properties, heuristic searches can be applied. For example, a variable induced minimum local constraint (within a cluster) can be the one to contract. There are very few studies on this topic. It deserves more attention.

5.4 Loopy Message Propagation

The algebraic approximation is equivalent to relaxing some constraints of the original CBI problems, removing some edges in the primal graph representations, or splitting some clusters in the junction tree representations. These approximations mean that the original CBI problems are revised and simplified to make computation feasible. Loopy message propagation, by contrast, does not revise the original CBI problems.

As already known, in junction tree algorithms both time and space complexities are bounded by the maximum cluster size. To maintain junction tree properties, the maximum cluster size is usually large in practical problems. If junction tree properties are relaxed, in other words, if the secondary structure is not necessarily a tree but a graph with loops, the maximum cluster size can be dramatically reduced.

At the same time, the message-passing may not terminate due to the introduction of loops. Also messages will be repeatedly counted. Both of these bring errors of inference. However, empirical results in the probability inference [46] and Turbo Codes decodings [44] show that the same message-passing schemes in exact junction tree algorithms work well in junction graphs with loops. When the junction graph is singly connected, the exact inference result is produced after one iteration of message-passing. When there exists only a single loop in the junction graph, it is guaranteed to converge to the correct result under some conditions [60].
5.4.1 Building a Junction Graph

Definition 5.3 (Junction Graph) Given the primal graph $G = (V, E)$ of a CBI problem $P = (X, D, R, F)$, a junction graph is $J = (C, S)$, where $C = \{C_1, \ldots, C_n\}$ is a set of clusters, each cluster $C_i$ corresponds to an aggregation of a subset vertices (correspondingly, variables) $V_{C_i} \subseteq V$; and $S = \{(C_i, C_j) | C_i, C_j \in C\}$ is a set of separators between $C_i$ and $C_j$. In addition, a junction graph satisfies that for any constraint $f \in F$, there exists a cluster $C_i \in C$ s.t. $\text{Scope}(f) \subseteq C_i$.

A junction tree is a junction graph without loops. In a junction graph, both the constraint allocation property and the running intersection property still hold.

Given a CBI problem $P = (X, D, R, F)$, there are many ways to generate a junction graph. The most straightforward way is shown in Algorithm 5.6. Following this algorithm, the maximum cluster size of the generated junction graph is the size of a constraint with the maximum number of variables in its scope, which is relatively small to cope with. The junction graph generated by Algorithm 5.6 is sometimes called the dual graph.

Algorithm 5.6 Building a Junction Graph for a CBI Problem

Input: A CBI problem $P = (X, D, R, F)$ and a variable subset of interest $Z$
Output: A junction graph $J = (C, S)$
1: $C := \{Z\}$
2: $S := \emptyset$
3: for each $f \in F$ do
4:   if $\text{Scope}(f) \subseteq C_i$ for some $C_i \in C$ then
5:     Continue
6:   else if $C_i \subset \text{Scope}(f)$ for some $C_i \in C$ then
7:     Replace $C_i$ with $\text{Scope}(f)$
8:   else
9:     Cluster $C_i := \text{Scope}(f)$
10:    $C := C \cup C_i$
11: end if
12: end for
13: for each $C_i, C_j \in C$ do
14:   if $C_i \cap C_j \neq \emptyset$ then
15:      $S_{ij} := C_i \cap C_j$
16:      $S := S \cup S_{ij}$
17: end if
18: end for

A dual graph can be revised to be a junction graph with less clusters and a larger maximum cluster size. The basic operation of the revision is the merging of neighboring clusters. Given two clusters $C_1$ and $C_2$ in neighbors, merging is defined as the following sequence of actions: (1) introducing a new cluster
Chapter 5. Approximate Inference Algorithms

C = C_1 \cup C_2, (2) connecting all neighbors of C_1 and C_2 to C, and (3) removing C_1 and C_2 from the graph. The size of the merged cluster obviously increases. It is a tradeoff between the number of clusters and the maximum cluster size.

5.4.2 Generalized Loopy Message Propagation Algorithm

The generalized loopy message propagation for a junction graph J = (C, S) of a CBI problem P = (X, D, R, F) is shown in Algorithm 5.7. The basic idea of the generalized loopy message propagation is the same as message-passing in the generalized junction tree algorithm, except that messages at iteration \( t \) are updated by incoming messages at iteration \( t - 1 \). The initial messages are produced based on local constraints and do not depend on the incoming messages from neighboring clusters.

**Algorithm 5.7 Generalized Loopy Message Propagation Algorithm (GLoopy)**

**Input:** A junction graph J = (C, S) of a CBI problem P = (X, D, R, F); Z:
- an variable set of interests; \( t_{max} \): maximum iteration times

**Output:** \( \hat{g}_{CBI}(Z) \approx \bigoplus_{X \subseteq Z} \bigotimes_{f \in F} f \)

1. Attach each cluster C_i with a constraint \( \phi_{C_i} = 1 \)
2. for each \( f \in F \) do
3. Find a cluster C_j such that \( \text{Scope}(f) \subseteq C_j \)
4. \( \phi_{C_i} := \phi_{C_i} \otimes f \)
5. end for
6. for each \( C_i \in C \) do
7. for each \( C_j \in \text{Neighbors}(C_i) \) do
8. \( m^{(t)}(C_i, C_j) := \bigoplus_{C_i \backslash S_j} \phi_{C_i} \otimes \bigotimes_{C_i \in \text{Neighbors}(C_i), j \neq 1} m^{(t-1)}(C_i, C_i) \)
9. end for
10. end for
11. for \( t = 1 \) to \( t_{max} \) do
12. for each \( C_i \in C \) do
13. for each \( C_j \in \text{Neighbors}(C_i) \) do
14. \( m^{(t)}(C_i, C_j) := \bigoplus_{C_i \backslash S_j} \phi_{C_i} \otimes \bigotimes_{C_i \in \text{Neighbors}(C_i), j \neq 1} m^{(t-1)}(C_i, C_i) \)
15. end for
16. end for
17. end for
18. Find a cluster C \( \in C \) s.t. \( Z \subseteq C \)
19. Return \( \hat{g}_{CBI}(Z) \approx \bigoplus_{C \subseteq X} \phi_{C} \otimes \bigotimes_{C_j \in \text{Neighbors}(C_j)} m^{(t)}(C_j, C_i) \)

If a CBI problem is defined on a semiring with the combination invertible property, the algorithm can be revised slightly to save computational cost. Details of this revised version of the generalized loopy message propagation algorithm for semirings with the combination invertible property are shown in Algorithm 5.8.
Algorithm 5.8 Generalized Loopy Message Propagation Algorithm with Inverses (GLoopy-Inv)

Input: A junction graph \( J = (C, S) \) of a CBI problem \( P = (X, D, R, F) \); \( Z \): an variable set of interests; \( t_{\text{max}} \): maximum iteration times

Output: \( \hat{g}_{\text{CBI}}(Z) \approx \bigoplus_{X,Z} \otimes_{f \in F} f \)

1: Attach each cluster \( C_i \) with a constraint \( \phi_{C_i} = 1 \)
2: for each \( f \in F \) do
3: Find a cluster \( C_j \) such that \( \text{Scope}(f) \subseteq C_j \)
4: \( \phi_{C_i} := \phi_{C_i} \otimes f \)
5: end for
6: for each \( C_i \in C \) do
7: for each \( C_j \in \text{Neighbors}(C_i) \) do
8: \( m^{(t)}(C_i, C_j) := \bigoplus_{C_i \setminus S_{ij}} \phi_{C_i} \)
9: end for
10: end for
11: for \( t = 1 \) to \( t_{\text{max}} \) do
12: for each \( C_i \in C \) do
13: \( \varphi^{(t)}_{C_i} := \phi_{C_i} \otimes \bigoplus_{C_j \in \text{Neighbors}(C_i)} m^{(t-1)}(C_j, C_i) \)
14: for each \( C_j \in \text{Neighbors}(C_i) \) do
15: \( m^{(t)}(C_i, C_j) := \varphi^{(t)}_{C_i} \otimes m^{(t-1)}(C_j, C_i) \)
16: end for
17: end for
18: end for
19: Find a cluster \( C \in C \) s.t. \( Z \subseteq C \)
20: Return \( \hat{g}_{\text{CBI}}(Z) \approx \bigoplus_{C \setminus Z} \phi_{C} \otimes \bigoplus_{C_j \in \text{Neighbors}(C)} m^{(t)}(C_j, C) \)

5.4.3 Loopy Message Propagation As an Exact Inference

A special case of applying loopy message propagation is performing inference on CBI problems that are defined on semirings with the combination idempotency property. Since the idempotency of combination implies that repeatedly counted messages will not introduce errors, loopy message propagation returns exact answers after finite iterations.

Theorem 5.4 (Loopy Message Propagation As an Exact Inference) Loopy message propagation algorithm (Algorithm 5.7) runs as an exact inference algorithm in CBI problems with combination idempotency property. Maximum iterations to be converged are the diameter \( d \) of the graph, which is defined as the number of edges in the longest path (with backtrack, detour, or loop excluded from consideration) between any two vertices of the graph.

Proof: Consider any cluster \( C_1 \) in the junction graph \( J = (C, S) \) of a CBI problem. It will receive messages from another cluster \( C_2 \) in \( d \) iterations at most. Since passing through other clusters will not filter the message from \( C_2 \) and repeatedly counting a message will not introduce errors, \( C_1 \) will receive messages
from all of the other $|C| - 1$ clusters after $d$ iterations at most. Combining these messages with local constraints in $C_1$ will result in the marginal of the joint combination of total constraints. □

Classic CSPs can be embedded into our semiring-based unified framework using the semiring $R = (\{false, true\}, \lor, \land)$. It is easy to show that the combination operation $\land$, logical AND, is idempotent. According to the theorem, loopy message propagation should be an exact inference algorithm for classic CSPs. In practice, arc-consistency [42], the key technique for CSPs, can be seen as a special case of the generalized loopy message propagation for CBI problems with idempotent semirings. In arc-consistency, the messages are the possible values of the variable in a separator. Combining messages from other clusters will remove illegal values, in other words, revise the outgoing message. The algorithm terminates when all the messages remain the same.

5.4.4 Discussion

In the probability inference field, the convergence of loopy message propagation in a junction graph with a single loop is proven. For more complex junction graphs of probability inference, little theoretic work has been done. For classic CSPs, loopy message propagation is proven as an exact inference algorithm. Chapter 7 will present the empirical results of applying loopy message propagation in classic CSPs.

Loopy message propagation is natively a distributed and parallel algorithm. At the iteration $t$, each cluster can compute its outgoing messages by collecting all the incoming messages from other neighbor clusters at iteration $t - 1$ in parallel. The number of needed processors is the same as the number of clusters. We may resort to junction graph building approaches to reduce the number of clusters, while increasing the maximal cluster size at the same time.

5.5 Hybrid Junction Tree Algorithm

The last approach for approximate inference of CBI problems is the hybrid junction tree (HybridJT) algorithm. Generally, it passes messages exactly to/from clusters with tractable sizes. When a cluster with intractable size is encountered, the hybrid junction tree algorithm builds a local junction graph based on local constraints and incoming messages. Loopy message propagation is used in this local junction graph. After several iterations of loopy message propagation, the approximate messages of such a cluster are generated and ready to pass to its neighbors.

Formally, the hybrid junction tree algorithm is shown in Algorithm 5.9. The hybrid junction tree algorithm here is analogous to the generalized approximate junction tree algorithm, except in the method of dealing with large clusters. In ApproxJT, large clusters are split into small sub-clusters, whereas in HybridJT, large clusters are treated as subproblems, which are approximately solved using loopy message propagation. In addition to the local constraints of a subprob-
lem, all the incoming messages are seen as constraints of the subproblem. All outgoing messages are initially seen as unitary constraints. After performing loopy message propagation over the junction graph of the subproblem, we can approximately produce outgoing messages.

5.6 Summary

The exponential complexities of exact inference algorithms for CBI problems make approximate approaches not only desirable but mandatory in practical scenarios. All generalized approximate inference algorithms presented in this chapter aim to restrict the maximum cluster size or induced width and to reduce the exponentially large factor in the complexities expression. The cluster sizes actually represent sizes of subproblems. We can achieve this purpose through either relaxing some constraints or re-organizing subproblems in a more complex way. With a bounded subproblem size, all exact inference approaches can be modified to cope approximately with scale problems.

So far, approximate inference approaches are mainly studied empirically. Little theoretical work has been done to analyze the bounds of approximation errors or give directions for choosing approximate parameters. These are definitely interesting topics for future study.
Algorithm 5.9 Hybrid Junction Tree Algorithm (HybridJT).

**Input:** A junction tree $T = (C, S)$ of a CBI problem $P = (X, D, R, F)$; $Z$: an variable set of interests; $w_{\text{max}}$: upper bound of cluster size; $b$: number of partial messages

**Output:** (Approximate) Consistent junction tree that is ready to answer queries

1: Attach each cluster $C_i$ with a constraint set $\Phi_{C_i} = \emptyset$
2: for each $f \in F$ do
3:   Find a cluster $C_i$ such that $\text{Scope}(f) \subseteq C_i$
4:   $\Phi_{C_i} := \Phi_{C_i} \cup \{f\}$
5: end for
6: for each Edge $S_{ij}$ which is from clusters $C_i$ to $C_j$ do
7:   $M(C_i, C_j) := \emptyset$
8:   if $C_i$ has received messages from all its neighbors other than $C_j$ then
9:     $Q := \emptyset_{C_i}$
10:    for each $C_i \in (\text{Neighbor}(C_i) \setminus \{C_j\})$ do
11:         $Q := Q \cup M(C_i, C_i)$
12:     end for
13:     if $|Q| \leq w_{\text{max}}$ then
14:         $M(C_i, C_j) := \{\bigoplus_{C_i \setminus S_{ij}} \otimes_{f \in Q} f\}$
15:     else
16:         if $|S_{ij}| \leq w_{\text{max}}$ then
17:             Define $q^{(0)} : S_{ij} \rightarrow 1$
18:             $Q := Q \cup \{q^{(0)}\}$
19:         else
20:             Split $S_{ij}$ into $b$ parts, $\bigcup_{k=1}^{b} S_{ij}^{(k)} = S_{ij}$
21:             for $k = 0$ to $b - 1$ do
22:                 Define $q^{(k)} : S_{ij}^{(k)} \rightarrow 1$
23:                 $Q := Q \cup \{q^{(k)}\}$
24:             end for
25:         end if
26:     end if
27:     Building junction graph $J$ of constraints set $Q$ and do loopy message propagation over $J$
28:     let $Q_k$ be the cluster in $J$ to represent $q^{(k)}$
29:     for $k' = 0$ to $b - 1$ do
30:         $M(C_i, C_j) := M(C_i, C_j) \cup \bigotimes_{Q_i \in \text{Neighbor}(Q_k)} m(Q_i, Q_k)$
31:     end for
32: end if
33: end for
Chapter 6

Generalized CBI Toolkit in Java - GCBIJ

6.1 GCBIJ Motivation

In a constraint-based inference problem, the two basic constraint handling operations, combination and marginalization, are expressive enough to generalize various inference approaches. The semiring-based unified framework for CBI problems introduced in Chapter 3 is based on this observation. The proposed framework defines the generalized representation of CBI problems in terms of discrete variables and constraints on local variables. It also uses the semiring, an importation concept in abstract algebra, to generalize operations of the constraint combination and marginalization. Any CBI problem, then, can be theoretically represented by our semiring-based framework. Any discrete inference approaches based on combination and marginalization operations can be theoretically interpreted by the framework as well. In previous chapters, we have embedded into this framework many concrete CBI problems, such as probabilistic inference, decision-making under uncertainty, constraint satisfaction problems, propositional satisfiability, decoding problems, and possibility inference. We also show that many concrete exact and approximate inference algorithms can be generalized within the framework. In other words, we show that the semiring-based unified framework is powerful enough to represent both CBI problems and various concrete inference approaches.

To prove the representation power of our proposed semiring-based unified framework in practice, we implement a software toolkit, named Generalized Constraint-Based Inference Toolkit in Java (GCBIJ). It is a concrete system for implementing the ideas and concepts of our semiring-based unified framework. Besides demonstrating the representation power of the framework, there are also two reasons for implementing the framework: (1) GCBIJ provides a series of generalized inference approaches, either exact or approximate, to solve practical CBI problems. Users can tackle problems in their own domains simply through calling procedures provided by this toolkit. They only need to specify or design their task-specific semirings. (2) Researchers can design, verify, compare, and analyze different inference approaches for CBI problems on GCBIJ's platform. Discovering these general ideas for performing inference is more straightforward by dealing with abstract versions of inference algorithms. The process of abstracting inference approaches from other disciplines provides
a good opportunity to improve the algorithm design in one discipline.

6.2 GCBIJ System Architecture

The main body of GCBIJ consists of 7 packages. Figure 6.1 shows the relations among these packages. We will discuss the details of GCBIJ implementations in the following sections.

1. **Semiring**: Various semirings are defined and implemented in this package. Public interfaces are shared among semirings, which provide access to combination and marginalization operations, as well as to the identity and absorbing elements of each operation;

2. **CBITask**: This package provides the objects (variable, domain, constraint) to specify a CBI problem. Together with an instance of a specified semiring, a CBI problem can be fully characterized;

3. **Graph**: Two sub-packages are included in the package **Graph**:
   - **Components**: This package provides the primal graph, junction tree, junction graph, and related graphical components. The basic graphical elements, such as vertices and edges, are implemented through wrapping *OpenJGraph* [31], an open source Java Graph and Graph Drawing library under GNU Lesser General Public License;
   - **Algorithms**: This package provides the basic algorithms of the graph theory, including triangulation, cliques identification, and spanning tree finding;

4. **Inference**: This package provides various inference algorithms. These algorithms are categorized into exact inference and approximate inference. All of these algorithms solve specified CBI tasks by accessing the operations of the abstract **Semiring** interface;

5. **Parser**: To solve concrete CBI problems, different parsers are implemented to translate various types of practical problem representations into the internal format of our framework;

6. **Utilities**: For research purposes, we need to collect information about inference approaches, either in terms of CPU time or in terms of the number of operations. Some statistic utilities are implemented in this package to assist with our algorithm analyses;

7. **GCBIJ_Main**: This example package includes routines on how to use GCBIJ integrally. Example CBI problems and inference algorithms are tested. Statistic utilities are used here to analyze various inference algorithms. There are also some test units included in this package.
Figure 6.1: System Architecture of GCBIJ
Table 6.1: Public Methods of the Class Semiring

<table>
<thead>
<tr>
<th>Return</th>
<th>Method Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Semiring(int)</td>
<td>Constructor with specified semiring type ID.</td>
</tr>
<tr>
<td>Object</td>
<td>product(Object, Object)</td>
<td>Binary combination operation.</td>
</tr>
<tr>
<td>Object</td>
<td>sum(Object, Object)</td>
<td>Binary marginalization operation.</td>
</tr>
<tr>
<td>Object</td>
<td>getSumIdentity()</td>
<td>Get the identity element of marginalization, if exists.</td>
</tr>
<tr>
<td>Object</td>
<td>getProdIdentity()</td>
<td>Get the identity element of combination, if exists.</td>
</tr>
<tr>
<td>Object</td>
<td>getSumAbsorb()</td>
<td>Get the absorption element of marginalization, if exists.</td>
</tr>
<tr>
<td>Object</td>
<td>getProdAbsorb()</td>
<td>Get the absorption element of combination, if exists.</td>
</tr>
<tr>
<td>boolean</td>
<td>isCommutative()</td>
<td>Determine if it is a commutative semiring.</td>
</tr>
</tbody>
</table>

6.3 GCBIJ Implementation and Interfaces

GCBIJ is capable of representing a set of CBI problems with discrete variables, which include probabilistic inference, decision making under uncertainty, constraint satisfaction, propositional satisfiability, decoding problems, and possibility inference. All the constraints are represented in the table form. Several generalized inference algorithms, either exact or approximate, are implemented. By manipulating various semirings, these generalized algorithms can solve concrete CBI problems within the semiring-based unified framework.

6.3.1 Semiring

Semiring is the key concept in the proposed framework. The implementation of semirings should capture their common computational properties. The abstract class Semiring is the base class for all concrete semirings, which provides basic operations (combination and marginalization), as well as provides access to other semiring properties.

The public methods provided by class Semiring are listed in Table 6.1. Different semirings are derived from it, which implement concrete methods. In addition, there are two public interfaces, IReversible and IIdempotent, which specify the behaviors of semirings with combination idempotency and invertible properties. The methods of these two interfaces are listed in Table 6.2 and Table 6.3, respectively. Concrete semirings with these two properties can implement these interfaces, besides inheriting the base class Semiring.

Implemented concrete semirings in GCBIJ include:

- SumProdSemiring;
### Table 6.2: Public Methods of the Interface IIdempotent

<table>
<thead>
<tr>
<th>Return</th>
<th>Method Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>compares(Object, Object)</td>
<td>Return a partial ordering of the two given objects.</td>
</tr>
<tr>
<td>boolean</td>
<td>isSumIdempotent()</td>
<td>Determine if marginalization operation is idempotent.</td>
</tr>
<tr>
<td>boolean</td>
<td>isProdIdempotent()</td>
<td>Determine if combination operation is idempotent.</td>
</tr>
</tbody>
</table>

### Table 6.3: Public Methods of the Interface IReversible

<table>
<thead>
<tr>
<th>Return</th>
<th>Method Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>reverses(Object)</td>
<td>Return the reverse of a given object.</td>
</tr>
<tr>
<td>Collection</td>
<td>normalize(Collection)</td>
<td>Normalize a collection of objects.</td>
</tr>
</tbody>
</table>

- MaxSumSemiring;
- OrAndSemiring;
- MaxMinSemiring;
- MinMaxSemiring;
- MaxProdSemiring;
- MinSumSemiring;
- MinProdSemiring.

Provided with the base class and common interfaces, users can easily design and implement their own task-specific semirings.

### 6.3.2 Parser

To tackle practical CBI problems from different disciplines, GCBIJ provides several parser classes to construct CBI problems from input files. All parsers are derived from the base abstract class `Parser`. The public methods provided by the base class are listed in Table 6.4.

Implemented concrete parsers in GCBIJ include:

- `XmlbifParser`: This parser corresponds to parse XML files in the Interchange Format for Bayesian Networks [11].
- `BifParser`: This parser corresponds to parse BIF files in older versions of the Interchange Format for Bayesian Networks [10]. Together with `XmlbifParser` and a 3rd-party Bayesian Networks Formats Convertor [28], GCBIJ can parse and perform inference on most Bayesian networks in the Bayesian Network Repository [23].
Table 6.4: Public Methods of the Class Parser

<table>
<thead>
<tr>
<th>Return</th>
<th>Method Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Parser(Semiring)</td>
<td>Constructor with specified semiring.</td>
</tr>
<tr>
<td>CBITask</td>
<td>parse(String)</td>
<td>Given a filename, parse it into CBITask, the internal CBI problem representation of GCBIJ.</td>
</tr>
<tr>
<td>Semiring</td>
<td>setSemiring(Semiring)</td>
<td>Specify the semiring used during parsing a file.</td>
</tr>
</tbody>
</table>

- **CspifParser**: This parser corresponds to parse CSPs in CSPIF format that is used in the CSP representation of CISpace \[41\]. Currently, the parser can parse scheduling problems with custom constraints, n-queens problems, and crossword problems. Given more CSPIF specifications, we can extend this parser to incorporate GCBIJ into Clspace in the future.

- **DimacsParser**: This parser corresponds to parse SAT problems in DIMACS CNF format. All benchmark problems in SATLIB \[26\] are in this format.

- **DimacsColorParser**: This parser corresponds to parse coloring problems in COL format. COL format is an older version of the DIMACS CNF format used to characterize clique and coloring problems in graphs.

### 6.3.3 CBITask

This package consists of several elements used to characterize a CBI problem. Class **Domain** wraps a *List* in Java as domain values. Class **Variable** specifies the variable name. It contains an instance of class **Domain** as the domain of the variable. Class **Constraint** is the implementation of the constraint. It consists of a list of variables and a (logically) multi-dimensional table to specify the constraint value (in **Object** in Java) under every value assignment of variables. Internally the table is implemented as a *List*. A series of accessing methods are provided.

The class **CBITask** is the container of variables and constraints, which corresponds to the basic elements of a concrete CBI problem. It also has an instance of **Semiring** as its member. The combination and marginalization operations for both exact and approximate approaches are implemented for the **CBITask** through wrapping corresponding methods of this semiring.

### 6.3.4 Inference

The Inference package is a collection of generalized exact and approximate inference algorithms for CBI inference tasks, which queries a new constraint over a variable, a set of variables, or the whole problem (no particular variable is
Table 6.5: Public Methods of the Class *InferAlg*

<table>
<thead>
<tr>
<th>Return</th>
<th>Method Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>InferAlg</td>
<td>Default constructor, no CBI problem specified.</td>
</tr>
<tr>
<td>CBITask</td>
<td>InferAlg(CBITask)</td>
<td>Constructor with specifying a CBI problem to solve.</td>
</tr>
<tr>
<td>void</td>
<td>setCBITask(CBITask)</td>
<td>Specify a CBI problem to solve.</td>
</tr>
<tr>
<td>int</td>
<td>setOrderingType(int)</td>
<td>Set the heuristic search type to determine a variables elimination ordering.</td>
</tr>
<tr>
<td>CBITask</td>
<td>getCBITask()</td>
<td>Return a CBI problem to be solved.</td>
</tr>
<tr>
<td>int</td>
<td>getOrderingType()</td>
<td>Return current heuristic search type.</td>
</tr>
<tr>
<td>Constraint</td>
<td>query()</td>
<td>Query the property of the whole CBI problem.</td>
</tr>
<tr>
<td>Constraint</td>
<td>query(Variable)</td>
<td>Query the constraint on specified variable.</td>
</tr>
<tr>
<td>Constraint</td>
<td>query(Collection)</td>
<td>Query the constraint on a set of specified variables.</td>
</tr>
<tr>
<td>Collection</td>
<td>queryOneAssignment()</td>
<td>Find one assignment of all variables.</td>
</tr>
<tr>
<td>Collection</td>
<td>queryAllAssignments()</td>
<td>Find all assignments (collection of Maps) of all variables.</td>
</tr>
</tbody>
</table>

Another task for CBI problems is the assignment task with combination idempotent semirings. Inference algorithms need to find one valid value assignment or all valid value assignments to uninterested variables. The basic methods of inference algorithms are defined in the abstract base class *InferAlg*, as shown in Table 6.5. All concrete inference algorithms inherit it and follow the same interface.

**Generic Inference**

The class *GenericInferAlg* is the only concrete class derived directly from the abstract class *InferAlg*. It directly implements the Eq. 3.1 of CBI problems. Allocation tasks are solved by looking up the value table of the total combined constraint. The generic inference can be classified as an exact inference. However, we decided to implement it independently.

**Exact Inference**

The abstract class *ExactInferAlg* is derived from the abstract class *InferAlg*, without more public methods specified. Based on it, several exact inference algorithms introduced in Chapter 4 are implemented. These algorithms include:

- *VEEExactInferAlg*: The generalized variable elimination algorithm (Algorithm 4.1) with various elimination orderings is implemented in this class. Backtracking is used to answer assignment tasks.
Chapter 6. Generalized CBI Toolkit in Java - GCBIJ

- JTExactInferAlg: The generalized junction tree algorithm (Algorithm 4.4) for Shenoy-Shafer architectures is implemented in this class. Backtracking in the final cluster and constraints-passing are used to answer assignment tasks. There is no special requirement for semirings except commutativity.

Three classes derived from JTExactInferAlg aim to cope with semirings with combination invertible and idempotent properties.

- LSJTExactInferAlg: The generalized JT algorithm for Lauritzen-Spiegelhalter (LS) architectures (Algorithm 4.8) is implemented in this class, which is designed to solve CBI problems with combination invertible semirings.

- HUGINJTExactInferAlg: The generalized JT algorithm for HUGIN architectures (Algorithm 4.9) is implemented in this class, which is designed to solve CBI problems with combination invertible semirings.

- IdempotentJTExactInferAlg: The generalized JT algorithm for CBI problems with combination idempotent properties (Algorithm 4.6) is implemented in this class.

Approximate Inference

The abstract class ApproxInferAlg is derived from the abstract class InferAlg, with additional public methods to specify approximation parameters. These methods are listed in Table 6.6. Based on the abstract class ApproxInferAlg, several approximate inference algorithms introduced in Chapter 5 are implemented. These algorithms include:

- VEApproxInferAlg: The generalized approximate variable elimination algorithm (Algorithm 5.2) with various elimination orderings is implemented in this class.

- LoopyApproxInferAlg: The generalized loopy message propagation algorithm (Algorithm 5.7) is implemented in this class.

- JTApproxInferAlg: The generalized approximate junction tree algorithm (Algorithm 5.3) for Shenoy-Shafer architectures is implemented in this class. There is no special requirement for semirings except commutativity. Two classes derived from JTApproxInferAlg aim to cope with semirings with the combination invertible property. Both of them are implemented based on the parent abstract class JTApproxInferAlg. The implementation of the inverses approximation follows Eq. 5.2.

- LSJTApproxInferAlg: The generalized approximate JT algorithm for Lauritzen-Spiegelhalter (LS) architectures.

- HUGINJTApproxInferAlg: The generalized approximate JT algorithm for HUGIN architectures.
Table 6.6: Public Methods of the Class \textit{ApproxInferAlg}

<table>
<thead>
<tr>
<th>Return</th>
<th>Method Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>setMaxWidth(int)</td>
<td>Specify the upper bound of the size of combined constraint (the maximum cluster size in a junction tree/graph).</td>
</tr>
<tr>
<td>void</td>
<td>setMaxBucket(int)</td>
<td>Specify the number of buckets to divide constraints.</td>
</tr>
<tr>
<td>void</td>
<td>setLoopNum(int)</td>
<td>Specify the maximum loops in loopy message propagation.</td>
</tr>
<tr>
<td>int</td>
<td>getMaxWidth()</td>
<td>Get the upper bound of the size of combined constraint (the maximum cluster size in a junction tree/graph).</td>
</tr>
<tr>
<td>int</td>
<td>getMaxBucket()</td>
<td>Get the number of buckets to divide constraints.</td>
</tr>
<tr>
<td>int</td>
<td>getLoopNum()</td>
<td>Get the maximum loops in loopy message propagation.</td>
</tr>
</tbody>
</table>

The other two approximation inference algorithms derived from \textit{JTApproxInferAlg} are:

- \textit{ThinJTApproxInferAlg}: The generalized thin junction tree algorithm (Algorithm 5.5).
- \textit{HybridJTApproxInferAlg}: The generalized hybrid inference algorithm (Algorithm 5.9).

6.3.5 Graph

This package includes two sub-packages: \textbf{Component} and \textbf{Algorithm}.

\textbf{Graph.Component}

The basic graph element classes \textit{Vertex} and \textit{Edge} are implemented through wrapping classes \textit{VertexImpl} and \textit{EdgeImpl} in OpenJGraph [31]. The primal graph class \textit{Graph} wraps OpenJGraph's class \textit{GraphImpl}. More specifically, the class \textit{Vertex} has a \texttt{Variable} instance as its member.

Class \textit{Cluster} is derived from \textit{Vertex} with a collection of vertices as its member. Additional public methods (c.f. Table 6.7) for \textit{Cluster} are implemented. Class \textit{Separator} is derived from \textit{Edge} with a cluster as its member. The separator set and messages are stored in that cluster. Accessing methods for the class \textit{Separator} are implemented as well.

Class \textit{JGraph} is derived from \textit{Graph}, which consists of clusters and separators. Class \textit{JTree} is derived from \textit{JGraph} with the tree property enforcement.
Graph Algorithm

A collection of graph algorithms are implemented in this package.

Class TriangulationAlg is the implementation of Algorithm 2.1 by feeding in the adjacent matrix of a given primal graph. Also, it provides a method for getting a variable ordering if specifying a heuristic search algorithm. The abstract class HeuristicsSearch provides a common interface for various heuristic search algorithms in Appendix A. These concrete heuristic search algorithms include:

- RandomSearch;
- MinWidthSearch;
- MinFillinSearch;
- MinInducedFillinSearch;
- MaxCardinalitySearch;
- LexMinimalSearch;
- LexPerfectSearch;

Another class SmallWorldParam specifies the methods for computing the graphical parameters of small world models. Class SmallWorldUtilities acts as the helper to parameter computing procedures.

6.3.6 Utilities

Several utility classes are implemented in this package, which include StatisticsUtility, SetUtility and CSPGenerator.

The class StatisticsUtility implements a collection of static methods for statistical purposes, which includes a start/stop CPU timer, counting numbers of operations, and basic data post-processing methods.

The class SetUtility implements set operations such as interaction, difference, and union. Basically, it wraps methods of Collection in Java.

The class CSPGenerator is an abstract class that specifies the interface of generating binary CSPs in CSPIF format. The CSPIF format can be parsed either by GCBIJ or by CIspace. In GCBIJ we implement three concrete binary CSP generators, inherited from CSPGenerator:

- The class RandomCSPGenerator generates random binary CSPs;
- The class SmallWorldCSPGenerator generates binary CSPs with Small World topologies, which implements Algorithm 2.2;
- The class ScaleFreeCSPGenerator generates binary CSPs with Scale-Free topologies, which implements Algorithm 2.3.
Chapter 6. Generalized CBI Toolkit in Java - GCBIJ

To generate binary CSPs, we need to specify the number of variables (vertices), the number of constraints (edges), the size of the variable domain, the forbidden rate of constraints, and the number of generated instances. Furthermore, we need to specify the re-written probability of _SmallWorldCSPGenerator_. Generated binary CSP instances with various topologies are used in our experiments (c.f. Chapter 7).

6.3.7 GCBIJ_Main

The package GCBIJ_Main contains a main class _GCBIJ_ as the entry point and an example for using GCBIJ. Also, several test unit classes are included in this package. For test purposes, a command line mode is implemented.

6.4 Summary

The proposed semiring-based unified framework for CBI problems in Chapter 3 is implemented as the Generalized Constraint-Based Inference Toolkit in Java (GCBIJ). The toolkit provides a way to represent various concrete CBI problems and a series of generalized inference algorithms, from exact to approximate inference. By specifying various semirings, these generalized inference algorithms can be instantiated as concrete algorithms that solve CBI problems from different disciplines. GCBIJ also provides a collection of parsers to translate concrete problems from various fields with domain-specified formats.

The architecture of GCBIJ is flexible, making it easy to extend. Users can implement their own task-specific semirings to fulfill their purposes. All the implemented inference algorithms use the abstract class _Semiring_ to access basic operations, without relating to the properties of concrete semirings. On the contrary, to design a new inference algorithm, users do not need knowledge of specific semirings, which ensures the generality of the algorithm. Given the common parser interface, users can also design a new parser to translate their domain problems into our internal CBI problem representation. The graph package is flexible as well. Users can later add more graphical features and algorithms.

GCBIJ is a proof of the representation power of the proposed semiring-based unified framework. It is a concrete system for implementing the ideas and concepts of the framework. The flexibility of its open architecture makes it a suitable toolkit for both research and application purposes.
### Table 6.7: Public Methods of the Class `Cluster`

<table>
<thead>
<tr>
<th>Return Type</th>
<th>Method Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td><code>getSize()</code></td>
<td>Return the number of variables in this cluster.</td>
</tr>
<tr>
<td>Collection</td>
<td><code>getVertexSet()</code></td>
<td>Return vertices in this cluster.</td>
</tr>
<tr>
<td>void</td>
<td><code>addVertex(Vertex)</code></td>
<td>Add a vertex to this cluster.</td>
</tr>
<tr>
<td>void</td>
<td><code>addVertices(Collection)</code></td>
<td>Add a collection of vertices to this cluster.</td>
</tr>
<tr>
<td>void</td>
<td><code>removeVertex(Vertex)</code></td>
<td>Remove a vertex from this cluster.</td>
</tr>
<tr>
<td>void</td>
<td><code>removeVertices(Collection)</code></td>
<td>Remove a collection of vertices (if any) from this cluster.</td>
</tr>
<tr>
<td>boolean</td>
<td><code>containVertex(Vertex)</code></td>
<td>Determine if the vertex is in this cluster.</td>
</tr>
<tr>
<td>boolean</td>
<td><code>containVertices(Collection)</code></td>
<td>Determine if a collection of vertices is a subset of this cluster.</td>
</tr>
<tr>
<td>void</td>
<td><code>addToLocalPotential(Collection)</code></td>
<td>Add a collection of constraints to the local constraint set of this cluster.</td>
</tr>
<tr>
<td>void</td>
<td><code>clearLocalPotential(Collection)</code></td>
<td>Clear the local constraint set of this cluster.</td>
</tr>
<tr>
<td>void</td>
<td><code>addToIncomingMsg(Collection)</code></td>
<td>Add a collection of constraints to the incoming message set of this cluster.</td>
</tr>
<tr>
<td>void</td>
<td><code>clearIncomingMsg(Collection)</code></td>
<td>Clear the incoming message set of this cluster.</td>
</tr>
<tr>
<td>void</td>
<td><code>addToOutgoingMsg(Collection)</code></td>
<td>Add a collection of constraints to the outgoing message set of this cluster.</td>
</tr>
<tr>
<td>void</td>
<td><code>clearOutgoingMsg(Collection)</code></td>
<td>Clear the outgoing message set of this cluster.</td>
</tr>
</tbody>
</table>
Chapter 7

GCBIJ Applications: Case Studies

In this chapter, we use the Generalized Constraint-Based Inference toolkit in Java (GCBIJ), an implementation of the proposed semiring-based unified framework and generalized inference algorithms, to do a series of experiments for CBI problems from different disciplines. Based on these experiments, we show that (1) the proposed semiring-based unified framework and generalized inference algorithms are suitable for representing and solving various concrete CBI problems; (2) GCBIJ is a good platform for studying CBI problems; (3) GCBIJ has the potential to tackle practical applications.

7.1 Test Suites

In this chapter, we study a series of concrete CBI problems based on GCBIJ. These problems belong to probability inference, CSP, and SAT respectively. See Table 7.1 for basic information on these problems.

7.2 Studies of Heuristic Triangulation Algorithms

Analyses in previous chapters show that both the time and space complexities of exact inference algorithms are exponential in the induced width of a given elimination ordering, or equivalently, exponential in the width of a tree decomposition of the original CBI problem. Ideally, the complexity will reach minimum if we can find the treewidth of the primal graph of the given CBI problem. However, computing treewidth is known as $\mathcal{NP}$-hard [2]. To get a near-optimal tree decomposition, we sometimes resort to heuristic triangulation algorithms. Basically, heuristic triangulation algorithms produce an elimination ordering of variables, which leads to an upper bound of the induced width or the treewidth.

In this chapter, we study several heuristic triangulation algorithms, which include Minimum Induced Width (MIW), Minimum Induced Fill-in (MIF), Maximum Cardinality (MC), Perfect Lexicographic BFS (LEXP), and Minimum Lexicographic BFS (LEXM). Details of these algorithms are listed in Appendix
A. To compare the results of different algorithms, we studied the triangulation algorithm with a random ordering. Table 7.2 shows the upper bounds of treewidth for various CBI problems. Here \( w_{\text{min}} \) is the best reported result so far, adopted from [24]. The number in the parentheses is the maximum separator size of the generated junction tree. For heuristic search algorithms (LEXP, LEXM, MC) with a random initial vertex, the upper bound of treewidth is averaged from \( 0.1 \times |V| \) runs, each with a random initial vertex.

For straightforward comparisons, we normalize the upper bounds of treewidth from various heuristics by the best reported results. The normalized upper bounds are shown in Figure 7.1. According to it, the minimum induced fill-in heuristic results in the best approximation of treewidth among all of these heuristics. The second best is the minimum induced width heuristic, which outperforms to the maximum cardinality heuristic in most problems. We also notice that heuristics are sensitive in different problems. Problems No. 3 (Pigs), No. 4 (Link), and No. 7 (zeroin.i) are hard to approximate by many heuristic techniques. Following this experiment, we use the minimum induced fill-in as the default heuristic triangulation algorithm in GCBIJ, though users can specify other heuristics by changing the default parameters.

Further information and discussion about heuristic triangulation algorithms and their empirical evaluations can be found in [33].
<table>
<thead>
<tr>
<th>Problem</th>
<th>$\omega_{\text{min}}$</th>
<th>RAND</th>
<th>MIW</th>
<th>LEXP</th>
<th>LEXM</th>
<th>MC</th>
<th>MIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagnosis</td>
<td>5</td>
<td>13(8)</td>
<td>6(4)</td>
<td>5(3)</td>
<td>5(3)</td>
<td>5(3)</td>
<td>5(3)</td>
</tr>
<tr>
<td>Insurance</td>
<td>8</td>
<td>14(13)</td>
<td>8(6)</td>
<td>10(8)</td>
<td>10(8)</td>
<td>9(6)</td>
<td>8(7)</td>
</tr>
<tr>
<td>Pigs</td>
<td>10</td>
<td>137(131)</td>
<td>17(14)</td>
<td>26(26)</td>
<td>31(30)</td>
<td>21(20)</td>
<td>11(9)</td>
</tr>
<tr>
<td>Link</td>
<td>13</td>
<td>254(250)</td>
<td>36(29)</td>
<td>68(66)</td>
<td>71(70)</td>
<td>121(120)</td>
<td>16(14)</td>
</tr>
<tr>
<td>fpsol2.i</td>
<td>66</td>
<td>247(202)</td>
<td>71(66)</td>
<td>72(67)</td>
<td>79(73)</td>
<td>72(68)</td>
<td>67(66)</td>
</tr>
<tr>
<td>mulsol.i</td>
<td>50</td>
<td>130(121)</td>
<td>51(50)</td>
<td>71(64)</td>
<td>77(73)</td>
<td>62(45)</td>
<td>51(50)</td>
</tr>
<tr>
<td>zeroin.i</td>
<td>50</td>
<td>110(106)</td>
<td>51(50)</td>
<td>97(96)</td>
<td>102(99)</td>
<td>63(48)</td>
<td>51(49)</td>
</tr>
<tr>
<td>uf20</td>
<td>17</td>
<td>18(17)</td>
<td>17(16)</td>
<td>17(16)</td>
<td>18(16)</td>
<td>17(16)</td>
<td>17(14)</td>
</tr>
<tr>
<td>flat100</td>
<td>100</td>
<td>175(173)</td>
<td>107(105)</td>
<td>120(119)</td>
<td>126(124)</td>
<td>114(111)</td>
<td>101(89)</td>
</tr>
<tr>
<td>BlocksWorld</td>
<td>49</td>
<td>80(77)</td>
<td>74(72)</td>
<td>50(46)</td>
<td>53(50)</td>
<td>49(48)</td>
<td>52(44)</td>
</tr>
<tr>
<td>Rnd. Binary CSP</td>
<td>9</td>
<td>9(8)</td>
<td>9(8)</td>
<td>9(8)</td>
<td>9(8)</td>
<td>9(8)</td>
<td>9(8)</td>
</tr>
</tbody>
</table>
7.3 Empirical Studies of Exact Inference Algorithms

7.3.1 Exact Inference Algorithms with Invertible Semiring

In this experiment, we solve the probability inference problem in GCBIJ, by specifying the regular sum-product semiring (Semiring No. 6 in Table 2.2). It is obvious that the sum-product semiring has the combination (product) invertible property. So, besides generalized VE and JT algorithms, both JT algorithms in LS and HUGIN architectures can be used to tackle inference tasks.

The probability inference problem to be tested in this experiment is the Diagnosis network (c.f. Figure 7.2, which is taken from CIspace [41]). The Diagnosis problem has 37 variables. The primal graph, the moralized graph of the original Bayesian network, has 65 edges.

To test the performances of various generalized exact inference algorithms (GVE, GJT, GJT-LS and GJT-HUGIN) in GCBIJ, we query the marginal probability for every variable (37 queries in total). Validated by CIspace, all generalized exact inference algorithms return correct results for all queries. The running time and the number of binary operations used are shown in Table 7.3.

Results here empirically support the theoretical analyses conveyed by Table 4.2 and Table 4.3 in Chapter 4. We can conclude that: (1) JT algorithms are
Figure 7.2: Bayesian Network of the Diagnosis Problem
Table 7.3: Running Times of Exact Inference Algorithms with the Combination Invertible Property for the Diagnosis Problem. (37 Independent Queries)

<table>
<thead>
<tr>
<th></th>
<th>GVE</th>
<th>GVE (perquery)</th>
<th>GJT</th>
<th>GJT-LS</th>
<th>GJT-HUGIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run. Time (ms)</td>
<td>60068</td>
<td>1623</td>
<td>7922</td>
<td>4307</td>
<td>4217</td>
</tr>
<tr>
<td>#Operation ×</td>
<td>77946</td>
<td>2107</td>
<td>6768</td>
<td>2881</td>
<td>2881</td>
</tr>
<tr>
<td>#Operation +</td>
<td>43910</td>
<td>1187</td>
<td>3694</td>
<td>3694</td>
<td>3694</td>
</tr>
<tr>
<td>#Operation /</td>
<td>0</td>
<td>0</td>
<td>966</td>
<td>250</td>
<td></td>
</tr>
</tbody>
</table>

ideal for multiple queries and VE algorithms are ideal for single queries; (2) for CBI problems defined on invertible semirings, both LS and HUGIN architectures are superior to the generalized JT algorithm in the Shenoy-Shafter architecture; and (3) the HUGIN architecture is slightly better than the LS architecture in using fewer / operations.

7.3.2 Exact Inference Algorithms with Idempotent Semiring

In this experiment, we solve the inference task (decision problem) of a random binary CSP based on GCBIJ, by specifying the logical or-and semiring (Semiring No. 1 in Table 2.2). It is obvious that the or-and semiring has the combination (\( \land \)) idempotent property. So besides generalized VE and JT algorithms, the idempotent JT algorithm (Algorithm 4.6) can also solve this inference task.

The test problem in this experiment is a random generated binary CSP with 35 variables and 70 constraints. The domain size of each variable is 3. The forbidden rate for each constraint is set at 0.5, in other words, each value assignment of a constraint has a probability 0.5 of not being allowable. In GCBIJ we compose a binary CSP generator to produce binary CSPs in CSpace's CSPIF format.

To test the performances of various generalized exact inference algorithms (GVE, GJT, GJT-Idemp) in GCBIJ, we query the marginal constraint on every variable (35 queries in total). The marginal constraint of each variable indicates valid values for this variable. Validated by CSpace, all generalized exact inference algorithms return correct results for all queries. The running time and the number of binary operations used are shown in Table 7.4.

These results empirically support the theoretical analysis conveyed by Table 4.2 and Table 4.3 in Chapter 4. We can conclude that: (1) JT algorithms are ideal for multiple queries. For CSPs, the query for a variable returns valid values for that variable; (2) for a CBI problem defined on idempotent semirings, the idempotent JT algorithm is superior to the generalized JT algorithm.
## 7.3.3 Exact Inference Algorithms for General CBI Problems

The last problem to test for exact inference algorithms in GCBIJ is the Max CSP for previous random binary CSP (35 variables and 70 constraints). According to the solution of the inference task, the test problem is satisfiable. So the inference result of the corresponding Max CSP should be 70. Both generalized VE and JT algorithms return this correct result. See Table 7.5 for details. We also conclude the maximum number of satisfiable constraints for each variable, given different domain values (c.f. Table 7.6).

From Table 7.5 we can clearly see that the VE algorithm is ideal for single queries. Compared to corresponding columns in Table 7.4, we notice that different semirings cause different running times even when we use the same generalized exact inference algorithm.

### 7.4 Empirical Studies of Approximate Inference Algorithms

#### 7.4.1 Approximate VE and JT for CSPs

In this experiment, we apply both the generalized approximate variable elimination (GVE-Approx, Algorithm 5.2) and the approximate junction tree (GJT-Approx, Algorithm 5.3) to solve a random binary CSP with 35 variables and 70 constraints with GCBIJ.

The exact inference results of the problem was given in previous sections. We query the valid values for each variable. Each valid value corresponds to its

<table>
<thead>
<tr>
<th>Table 7.4: Running Times of Exact Inference Algorithms with the Combination Idempotent Property for a Random Binary CSP. (35 Independent Queries)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GVE</td>
</tr>
<tr>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>Running Time (ms)</td>
</tr>
<tr>
<td>#Operation ∧</td>
</tr>
<tr>
<td>#Operation ∨</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 7.5: Running Times of General Exact Inference Algorithms for a Max CSP (Single Query)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GVE</td>
</tr>
<tr>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>Running Time (ms)</td>
</tr>
<tr>
<td>#Operation +</td>
</tr>
<tr>
<td>#Operation max</td>
</tr>
</tbody>
</table>
Table 7.6: The Maximum Number of Satisfiable Constraints Given Different Domain Values for a Random Binary CSP with 35 Variables and 70 Constraints

<table>
<thead>
<tr>
<th>Variable</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>70</td>
<td>64</td>
<td>63</td>
</tr>
<tr>
<td>V2</td>
<td>70</td>
<td>66</td>
<td>67</td>
</tr>
<tr>
<td>V3</td>
<td>70</td>
<td>66</td>
<td>66</td>
</tr>
<tr>
<td>V4</td>
<td>70</td>
<td>63</td>
<td>69</td>
</tr>
<tr>
<td>V5</td>
<td>70</td>
<td>64</td>
<td>70</td>
</tr>
<tr>
<td>V6</td>
<td>70</td>
<td>67</td>
<td>70</td>
</tr>
<tr>
<td>V7</td>
<td>70</td>
<td>67</td>
<td>70</td>
</tr>
<tr>
<td>V8</td>
<td>70</td>
<td>68</td>
<td>70</td>
</tr>
<tr>
<td>V9</td>
<td>70</td>
<td>64</td>
<td>70</td>
</tr>
<tr>
<td>V10</td>
<td>70</td>
<td>63</td>
<td>70</td>
</tr>
<tr>
<td>V11</td>
<td>70</td>
<td>63</td>
<td>69</td>
</tr>
<tr>
<td>V12</td>
<td>70</td>
<td>67</td>
<td>69</td>
</tr>
<tr>
<td>V13</td>
<td>70</td>
<td>68</td>
<td>70</td>
</tr>
<tr>
<td>V14</td>
<td>70</td>
<td>67</td>
<td>70</td>
</tr>
<tr>
<td>V15</td>
<td>70</td>
<td>64</td>
<td>69</td>
</tr>
<tr>
<td>V16</td>
<td>70</td>
<td>67</td>
<td>69</td>
</tr>
<tr>
<td>V17</td>
<td>70</td>
<td>68</td>
<td>67</td>
</tr>
<tr>
<td>V18</td>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>V19</td>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>V20</td>
<td>70</td>
<td>69</td>
<td>69</td>
</tr>
<tr>
<td>V21</td>
<td>70</td>
<td>68</td>
<td>67</td>
</tr>
<tr>
<td>V22</td>
<td>70</td>
<td>68</td>
<td>68</td>
</tr>
<tr>
<td>V23</td>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>V24</td>
<td>70</td>
<td>68</td>
<td>69</td>
</tr>
<tr>
<td>V25</td>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>V26</td>
<td>70</td>
<td>68</td>
<td>68</td>
</tr>
<tr>
<td>V27</td>
<td>70</td>
<td>68</td>
<td>68</td>
</tr>
<tr>
<td>V28</td>
<td>70</td>
<td>68</td>
<td>70</td>
</tr>
<tr>
<td>V29</td>
<td>70</td>
<td>69</td>
<td>69</td>
</tr>
<tr>
<td>V30</td>
<td>70</td>
<td>69</td>
<td>69</td>
</tr>
<tr>
<td>V31</td>
<td>70</td>
<td>68</td>
<td>69</td>
</tr>
<tr>
<td>V32</td>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>V33</td>
<td>70</td>
<td>68</td>
<td>69</td>
</tr>
<tr>
<td>V34</td>
<td>70</td>
<td>67</td>
<td>70</td>
</tr>
<tr>
<td>V35</td>
<td>70</td>
<td>69</td>
<td>70</td>
</tr>
</tbody>
</table>
appearance in a valid assignment. If there is no valid value for a variable, the problem is not satisfiable.

Both GVE-Approx and GJT-Approx return the approximate marginal of each variable. Compared to exact results, all the errors made by approximate algorithms are false-positive. In other words, only introduced errors falsely report some invalid values as valid. In general, both GVE-Approx and GJT-Approx trade time and accuracy to save the space of the corresponding exact inference algorithms.

Two parameters control the performance of approximate inference algorithms. The maximum width $w$ is the upper bound of the acceptable width during computing, which controls the accuracy of the approximation. A larger acceptable width, at the same time, will take more space. If $w$ is the same as or larger than the treewidth or induced width of the given CBI problem, the approximate VE and JT algorithms return exact results. Another parameter $b$, the number of buckets to partition constraints, controls the speed of the approximation. A larger $b$ will reduce the running time. However, more errors will be introduced at the same time. We can change the settings of $w$ and $b$ to achieve balance between accuracy and speed.

Figure 7.3 shows the accuracy and running time of the approximate VE algorithm for a random binary CSP under different parameter settings. Analogous results for the approximate JT algorithm are shown in Figure 7.4.

Figure 7.3: Accuracy and Running Time of the Approx. VE for a CSP
7.4.2 Approximate VE and JT for Probability Inference

The approximate VE and JT algorithms in GCBIJ can be applied to solve probability inference problems without any modifications. The only thing needed is to specify a regular sum-product semiring to replace the logical or-and semiring for CSPs.

In this experiment, we use approximate VE and JT algorithms to solve the Insurance problem from BN Repository [23]. The Insurance problem contains 27 variables and 27 conditional probability tables (CPTs). Figure 7.5 shows the Bayesian network of the Insurance problem. After moralization, we have 27 vertices and 70 edges in GCBIJ's internal primal graph representation.

In each run of the experiment, we randomly choose 4 variables as observed. Observed values are chose randomly as well. Then we compute the marginal of the rest variables. For both GVE-Approx and GJT-Approx, we fix the number of buckets at \( b = 2 \). The maximum width \( w \) changes from 4 to 7 (treewidth of the Insurance problem is 8). The correspondence between the exact and approximate marginal for the Insurance problem are shown in Figure 7.6 and Figure 7.7, corresponding to approximate VE and JT algorithms respectively. We find that the accuracy of the approximate VE and JT algorithms are increase with the maximum width \( w \), the same as the results for solving CSPs.
Figure 7.5: Bayesian Network of the Insurance Problem
Figure 7.6: Marginal Correspondence of the Approx. VE for a Probability Inference Problem.
7.5 Loopy Message Propagation As Exact Inference Algorithm

As discussed in Chapter 5, loopy message propagation (Algorithm 5.7) is in general an approximate inference algorithm for CBI problems. However, when a CBI problem is defined on semirings with the combination idempotency property, loopy message propagation returns exact results within at most \( d \) iterations, where \( d \) is the diameter of the corresponding dual graph. In this section, we apply loopy message propagation algorithm to solve a random binary CSP.

The random binary CSP is the same as the one used in Section 7.3.2, which has 35 variables with 3 domain values each and 70 constraints with the forbidden rate at 0.5. In the experiment, we examine the different upper bounds of the maximum cluster size in the join graph. We gradually increase the maximum allowable iterations. Within the allowable iterations, the program exits when all messages are unchanged (stable), returning the actual number of iterations.
Figure 7.8: Accuracy (A) and Time Complexity (B) of loopy Message Propagation for a CSP

Figure 7.8(A) and Figure 7.8(B) report the accuracy and time complexity (in terms of binary operations) respectively. Just like using the approximate VE and JT algorithms for CSPs, loopy message propagation returns only false-positives (in other words, reporting some invalid values as valid). Errors are reduced when the iterations of the message propagation increase. Eventually all messages are stable and loopy message propagation returns exact inference results. Roughly speaking, a junction graph with a larger cluster size consumes more binary operations (combination and marginalization) for random CSPs.

7.6 Impact of Small World Topology

7.6.1 Small World Topology of CBI problems

The Small World topology [59] appears widely in social graphs (e.g., the collaboration graph of actors in feature files), biological graphs (e.g., the neural network of the nematode worm *C. elegans*), and man-made graphs (e.g., the electrical power grid of the western United States). In a graph with the Small World topology, vertices are highly clustered, yet the path length between them is small. In this thesis, we are interested in whether or not such a topology exists in primal graph representations of real world CBI problems and how this characteristic impacts the performance of inference algorithms.
The Small World characteristic parameters for various CBI problems, including characteristic path length $L$, clustering coefficient $C$, and normalized scalar $\mu$ (c.f. Chapter 2), are computed and shown in Table 7.7, and compared to some practical networks [59]. For comparison, we also generate three scale-free networks based on Algorithm 2.3. All of them exhibit characteristic of small world topology. That observation shows that the scale-free network is probably a subset of small world topology, though no formal proof is available so far.

In this table, we are particularly interested in the Pigs and the Link networks, which are taken from a real-world application dataset. Figure 7.9 and Figure 7.10 show the characteristic parameter distributions of these two problems respectively.

In both of the Pigs and the Link networks, characteristic path lengths are relatively small, which implies that the message from one cluster will quickly reach another cluster in the junction graph. Although degree distributions show that most vertices in the graph have small degrees, some large degrees, in addition to a large amount of medium clustering coefficients, imply that exact inference algorithms are infeasible. Combining these observations, loopy message propagation is probably an efficient approach to coping with these two CBI problems.
Table 7.7: Parameters of the Small World Topology for Graphs

| Fields      | Problem                  | $|V|$  | $|E|$  | $L$   | $L_{rand}$ | $C$   | $C_{rand}$ | $\mu$ |
|-------------|--------------------------|------|------|------|-----------|------|-----------|------|
| Practical   | Film Actors              | 225,226 | 6,869,393 | 3.65 | 2.99     | 0.79 | 0.00027 | 2396 |
|             | Power Grid               | 4942 | 6819 | 18.7 | 12.4     | 0.08 | 0.005 | 10.61 |
|             | C.elegans                | 282 | 1974 | 2.65 | 2.25     | 0.28 | 0.05 | 4.755 |
| Bayesian    | Diagnosis                | 37 | 65 | 3.82 | 2.92     | 0.79 | 0.56 | 1.08 |
| Networks    | Insurance                | 27 | 70 | 2.23 | 2.12     | 0.70 | 0.48 | 1.37 |
|             | Pigs                      | 441 | 806 | 4.97 | 4.77     | 0.79 | 0.50 | 1.56 |
|             | Link                      | 724 | 1738 | 6.37 | 4.35     | 0.68 | 0.40 | 1.17 |
| Coloring    | fpsol2.i                  | 496 | 11654 | 1.69 | 1.92     | 0.45 | 0.13 | 3.83 |
|             | mulsol.i                  | 197 | 3925 | 1.81 | 1.80     | 0.50 | 0.24 | 2.42 |
|             | zeroin.i                  | 211 | 4100 | 1.66 | 1.82     | 0.48 | 0.23 | 2.33 |
| SAT         | Unf20                     | 20 | 147 | 1.22 | 1.22     | 0.80 | 0.80 | 1.00 |
|             | Flat100                   | 300 | 1017 | 3.72 | 3.18     | 0.33 | 0.31 | 0.92 |
|             | BlocksWorld               | 116 | 777 | 2.56 | 2.07     | 0.52 | 0.25 | 1.72 |
| Scale-Free  | sf100                     | 100 | 400 | 1.92 | 2.42     | 0.88 | 0.08 | 13.87 |
|             | Link-Like                 | 724 | 1738 | 2.56 | 4.37     | 0.388 | 0.049 | 13.66 |
|             | C.elegans-Like            | 282 | 1974 | 1.95 | 2.42     | 0.926 | 0.049 | 23.58 |
7.6.2 Inference in the Small World

To analyze the impacts of the small world topology for CBI problems, we generate a series of binary CSPs with different re-written probabilities, according to [59]. In general, a small re-written probability corresponds to ordered graphs, whereas a large re-written probability corresponds to random graphs. Visually small world graphs are those between ordered and random graphs. Figure 7.11 is adopted from [59]. It illustrates the relations between small world parameters and the re-written probabilities of graphs with 100 vertices and 400 edges.

In this section, we use loopy message propagation to perform inference tasks for binary CSPs with the Small World topology. These CSPs are generated based on Algorithm 2.2. Each of them has 100 binary variables and 400 binary constraints, with a 0.5 forbidden rate. The re-written probabilities range from 0 to 1. We collect the number of binary operations under each re-written probability (with 5 instances). The results are generalized in Figure 7.12.

Compared to Figure 7.11, we find that loopy message propagation works well in CSPs with the small world topology. A relatively small characteristic path length implies quick convergence. A relatively large clustering suggests that we can resort to a large maximum cluster size in loopy message propagation if the computation power permits. The results also suggest that loopy message propagation is a suitable inference approach for CSPs with ordered graphical representations, but not suitable for random CSPs.
Figure 7.11: Characteristic Path Length $L(p)$ and Clustering Coefficient $C(p)$ for Different Re-written Probabilities (Figure is taken from [59]).

Figure 7.12: Time Complexity of loopy MP in the Small World
Chapter 7. GCBIJ Applications: Case Studies

7.7 Summary

Based on the Generalized Constraint-Based Inference toolkit in Java (GCBIJ), this chapter presents a series of experiments of CBI problems from different disciplines. The results of these experiments are not totally new to research communities. However, these results as a whole verify the feasibility of using semirings to generalize representations of various CBI problems. Generalized exact and approximate inference algorithms, based on the proposed semiring-based unified framework, are proven to be suitable to apply in concrete application domains.

Case studies also show that GCBIJ is a flexible toolkit of constraint-based inference research. Various exact and approximate inference algorithms can solve different CBI problems when different semirings within GCBIJ are specified. New algorithms can be added to the toolkit by either generalizing existing concrete approaches or designing new approaches on the abstract level. Although more optimization and implementation work is required, the extendibility and flexibility of GCBIJ make it a suitable toolkit for both CBI research and applications.
Chapter 8

Conclusions and Future Work

8.1 Conclusions

As the first contribution of this paper, we propose a semiring-based unified framework, a single formal representation framework with a broader coverage of the constraint-based inference problem space based on the synthesis of existing generalized frameworks. Our framework explicitly subsumes and unifies many concrete CBI problems, such as probabilistic inference, decision making under uncertainty, constraint satisfaction problems, propositional satisfiability, decoding problems, and possibility inference, in an abstract representation.

The unified framework is also a single formal algorithmic framework that provides a broader coverage of both exact and approximate inference algorithms. This is the second contribution of this paper. We unify various widely used inference algorithms, such as the exact and approximate variable elimination algorithms, the exact and approximate junction tree algorithms, and loopy message propagation, based on the framework. Many of these algorithms depend only on the basic properties of the commutative semirings. Based on other special properties of different commutative semirings, we also generalize the variants of these algorithms that arise in different application scenarios.

Abstract representations of CBI problems, as well as abstract inference algorithms, provide several opportunities for researchers from various fields: (1) they can study the most important common characteristics of various CBI problems without representation barriers; (2) they can analyze and compare different inference approaches; (3) they can borrow design ideas from other fields and improve the inference approaches' efficiency in their own domains; and (4) implementations at the abstract level significantly reduce the amount of work targeted previously at the individual problems. In other words, researchers from different fields may reinterpret many familiar approaches in their domains at a higher level. The algorithm discussions and the complexity analyses in this thesis are examples of applying the abstract knowledge to the concrete application domains.

The unified representation for CBI problems and inference algorithms is not, of course, a totally novel idea. Much research has been conducted in various disciplines through using different tools or notions. Here we have significantly broadened the scope of the problems and the coverage of the algorithms. The
final contribution of this paper is a software toolkit, the Generalized Constraint-Base Inference Toolkit in Java (GCBIJ). GCBIJ is the first concrete toolkit to implement ideas for unifying the representations of CBI problems using semirings and to implement various inference algorithms on the abstract level. The generalization and extensibility of GCBIJ make it a good platform for both CBI research and practical problem solving.

8.2 Possible Improvements to Inference Approaches

Here we briefly discuss some possible improvements to generalized inference algorithms for CBI problems. These preliminary discussions are not complete but suggest some interesting research topics for the future.

8.2.1 From Variables to Values

The inference approaches generalized in Chapter 4 and Chapter 5 do not depend on possible values or the domains of variables in CBI representations. If values are taken into account, we can improve these approaches by manipulating values. As a restriction of our proposed semiring-based unified framework, the domain of a variable is discrete.

Considering Values in Heuristic Search

Heuristic searches are used in both VE and JT algorithms to find a near-optimal elimination ordering or tree decomposition before performing the inference. The most straightforward improvement to heuristic-search-based approaches is to consider the actual domain size of the variables instead of treating them uniformly. For example, we use heuristic searches to find the upper bounds of the treewidth or the induced width. For either Minimum Induced Fill-in or Minimum Induced Width heuristics, we break ties by randomly choosing a vertex. Alternatively we can use the domain size as the weight of a vertex (variable) to break ties with largest weight first.

In approximate inference algorithms, the key idea of the approximation is to restrict the size of the maximum sub-problem to a tractable level. In our framework, the sub-problem size is measured by the cluster size or the number of variables participating in the sub-problem. More specifically, the sub-problem size could be measured as the product of the domain sizes of all participating variables. Using such a measurement as a heuristic function or an upper bound of approximation procedures will bring computational benefits such as giving a more accurate approximation of the treewidth or introducing less approximation errors. At the same time, using values in heuristics will increase computational costs.
Instantiating and Domain Splitting

In general, when a vertex of a graph is removed, the graph's structure will be simplified. The treewidth of a simplified graph will possibly be reduced. As shown in previous chapters, the time and space complexities of exact inference algorithms in this thesis are proven to be exponential in the width of a tree decomposition of the corresponding primal graph representation. One way to remove a vertex is to assign a value to its corresponding variable. These naive observations suggest that instantiating variables will bring computational benefits if carefully designed.

For the completeness of inference, we need to store instantiations of a variable for further use. Once the inference of a conditioning value is finished, we may need to backtrack to other instantiated values. Basically, it is a backtracking process. The DPLL algorithm [12] of SAT is a typical example of implementing such an idea.

Given a normal inference approach, when to start instantiating, which variable to be instantiated, and what value to assign are the key problems of instantiation.

A related improvement is to split the domain of a variable. Instead of assigning one value to a variable, we could assign a subset of the original domain values to the variable. For context-specific constraint representations, a variable with splitting domains will simplify the graph since some edges are not presented in the current context.

Domain Shrinking

The last possible improvement to inference approaches we would like to mention is domain shrinking. This idea comes from the Arc Consistency of CSPs. Given a constraint, the value of a participating variable can be discarded once all the assignments induced by this value are false. After checking out all-false values for a variable, we shrink its domain size. The reduced domain size leads to computational benefits for regular inference approaches.

The same idea can be generalized to abstract CBI problems. In CSPs, false can be used to detect shrinking values since false is the absorbing element of the combination operation (logical AND) in the OR-AND semiring. The absorbing element of the combination operation means that it is a trap for further combinations. At the same time, false is also the identity element of the marginalization operation (logical OR), which means that no more new information is contained.

Based on the previous observations, we can apply domain shrinking to probability inference problems. Since zero is both the absorbing element of the combination (product) and the identity element of the marginalization (plus), we can remove a value from a variable domain if all the assignments in which this value participates are mapped to zero. If approximation is permitted, near zero values with a user-specified threshold can be used to detect the values to remove.
### Table 8.1: Detecting Elements of Some Commutative Semirings

<table>
<thead>
<tr>
<th>No.</th>
<th>$R$</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
<th>Detecting Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${\text{true, false}}$</td>
<td>$\lor$</td>
<td>$\land$</td>
<td>$\text{false}$</td>
</tr>
<tr>
<td>2</td>
<td>$[0,1]$ or $[0, \infty)$</td>
<td>$\max$</td>
<td>$\min$</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>$[0,1]$</td>
<td>$\max$</td>
<td>$\times$</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>$(-\infty,0]$</td>
<td>$\max$</td>
<td>$+$</td>
<td>N/A</td>
</tr>
<tr>
<td>5</td>
<td>$[0,\infty)$</td>
<td>$\max$</td>
<td>$+$</td>
<td>N/A</td>
</tr>
<tr>
<td>6</td>
<td>$[0,\infty)$</td>
<td>$+$</td>
<td>$\times$</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>$[0,\infty)$</td>
<td>$\max$</td>
<td>$\times$</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>$(-\infty,\infty)$</td>
<td>$\min$</td>
<td>$+$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>9</td>
<td>$(-\infty,0]$</td>
<td>$\min$</td>
<td>$\times$</td>
<td>0</td>
</tr>
</tbody>
</table>

In general, given a semiring $R = (S, \oplus, \otimes)$, $a \in R$ can be used to detect values to remove from the domain if the following two statements hold at the same time: (1) $a$ is the absorbing element of $\otimes$; (2) $a$ is the identity element of $\oplus$. We call $a$ the **detecting element** of the semiring $R$. Table 8.1 concludes the detecting elements of the some semirings.

### 8.2.2 From Systematic to Stochastic

Basically, all the inference algorithms generalized in our framework are systematic approaches to CBI problems. In concrete application domains, stochastic approaches, such as sampling techniques in probability inference and stochastic local searches in SAT, are very successful. Given the fact that underlying representations of these problems are highly analogous, an abstract representation of stochastic or hybrid (stochastic and systematic) inference approaches should be a reasonable goal for future research.

### 8.3 Future Work

In Section 8.2, we briefly discussed some possible improvements to existing inference approaches. Some of these improvements have already achieved success in many fields, which suggests generalizing the characteristics of these approaches, adding them into our semiring-based unified framework, and improving the design of other concrete approaches.

One limitation of our framework is that the domain of variables must be discrete. However, both the semiring itself and the combination and marginalization operations do not impose any restriction on variable types. It would definitely exciting to extend our framework to tackle continuous variables in the future.

In this thesis, we propose some naive inference algorithms for CBI problems, which can be abstracted with a $k$-semiring. For $k = 2$, we show that the decision making under uncertainty is one example. For $k > 2$, the application of $k$-
semiring is far from clear. If we have some input from experts in other fields to support the usefulness of $k$-semirings, we would like to investigate more efficient inference algorithms for $k$-semirings.

As an open and extendable toolkit, the implementation and optimization of GCBIJ is far from complete. To maintain generality, we sacrificed some running efficiency in our current implementation. To attract people to using this toolkit, more comprehensive documents and clear interfaces are needed. In general, much coding and documenting work for GCBIJ remains.
Bibliography


Appendix A

Heuristics Search Algorithms for Triangulation

A.1 Minimal Width Heuristic Triangulation

Algorithm A.1 Minimal Width Heuristic Triangulation (MW) [47]

Input: A connected graph $G = (V, E)$

Output: Triangulated graph $G_t = (V, E \cup E_t)$ with $w(G) \leq w(G_t)$ and an elimination ordering $\sigma = (v_1, \ldots, v_n)$

1: $E_t := \emptyset$
2: for $v \in V$ do
3: \hspace{1em} $factor(v) := |\text{Neighbor}(v)|$
4: \hspace{1em} $M(v) := \text{Neighbor}(v)$
5: \hspace{1em} end for
6: $S := V$
7: for $i = |V|$ to 1 do
8: \hspace{1em} $u := \arg\min_{v \in S} factor(v)$
9: \hspace{1em} $\sigma(i) := u$
10: for $x, y \in M(u)$ and $(x, y) \notin E \cup E_t$ do
11: \hspace{1em} \hspace{1em} $E_t := E_t \cup (x, y)$
12: \hspace{1em} \hspace{1em} $M(x) := M(x) \cup y$ and $M(y) := M(y) \cup x$
13: \hspace{1em} end for
14: for $w \in M(u)$ do
15: \hspace{1em} $M(w) := M(w) \setminus u$
16: \hspace{1em} end for
17: $S := S \setminus \{u\}$
18: end for
A.2 Minimal Induced Width Heuristic Triangulation

Algorithm A.2 Minimal Induced Width Heuristic Triangulation (MIW) [37]

Input: A connected graph \( G = (\mathcal{V}, \mathcal{E}) \)

Output: Triangulated graph \( G_t = (\mathcal{V}, \mathcal{E} \cup \mathcal{E}_t) \) with \( w^*(G) \leq w^*(G_t) \) and an elimination ordering \( \sigma = (v_1, \ldots, v_n) \)

1: \( \mathcal{E}_t := \emptyset \)
2: for \( v \in \mathcal{V} \) do
3: \( M(v) := \text{Neighbor}(v) \)
4: end for
5: \( S := \mathcal{V} \)
6: for \( i = |\mathcal{V}| \) to 1 do
7: \( u := \arg \min_{v \in S} |M(v)| \)
8: \( \sigma(i) := u \)
9: for \( x, y \in M(u) \) and \( (x, y) \notin \mathcal{E} \cup \mathcal{E}_t \) do
10: \( \mathcal{E}_t := \mathcal{E}_t \cup (x, y) \)
11: \( M(x) := M(x) \cup y \) and \( M(y) := M(y) \cup x \)
12: end for
13: for \( w \in M(u) \) do
14: \( M(w) := M(w) \setminus u \)
15: end for
16: \( S := S \setminus \{u\} \)
17: end for
A.3 Minimal Fill-in Heuristic Triangulation

**Algorithm A.3 Minimal Fill-in Heuristic Triangulation (MF) [47]**

**Input:** A connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

**Output:** Triangulated graph $\mathcal{G}_t = (\mathcal{V}, \mathcal{E}_t \cup \mathcal{E}_i)$ with $w^*(\mathcal{G}) \leq w^*(\mathcal{G}_t)$ and an elimination ordering $\sigma = (v_1, \cdots, v_n)$

1. $\mathcal{E}_t := \emptyset$
2. for $v \in \mathcal{V}$ do
3. \hspace{1em} fillin($v$) := 0
4. \hspace{1em} $M(v) := \text{Neighbor}(v)$
5. \hspace{1em} for each $x, y \in M(u)$ and $(x, y) \notin \mathcal{E}$ do
6. \hspace{2em} fillin($v$) := fillin($v$) + 1
7. \hspace{1em} end for
8. \hspace{1em} end for
9. $S := \mathcal{V}$
10. for $i = |\mathcal{V}| \text{ to } 1$ do
11. \hspace{1em} $u := \arg\min_{v \in S} \text{fillin}(v)$
12. \hspace{1em} $\sigma(i) := u$
13. \hspace{1em} for $x, y \in M(u)$ and $(x, y) \notin \mathcal{E} \cup \mathcal{E}_t$ do
14. \hspace{2em} $\mathcal{E}_t := \mathcal{E}_t \cup (x, y)$
15. \hspace{2em} $M(x) := M(x) \cup y$ and $M(y) := M(y) \cup x$
16. \hspace{1em} end for
17. \hspace{1em} for $w \in M(u)$ do
18. \hspace{2em} $M(w) := M(w) \setminus u$
19. \hspace{1em} end for
20. $S := S \setminus \{u\}$
21. end for
A.4 Minimal Induced Fill-in Heuristic Triangulation

Algorithm A.4 Minimal Induced Fill-in Heuristic Triangulation (MIF) [37]

Input: A connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

Output: Triangulated graph $\mathcal{G}_t = (\mathcal{V}, \mathcal{E} \cup \mathcal{E}_t)$ with $w^*(\mathcal{G}) \leq w^*(\mathcal{G}_t)$ and an elimination ordering $\sigma = (v_1, \ldots, v_n)$

1: $\mathcal{E}_t := \emptyset$
2: for $v \in \mathcal{V}$ do
3: $\text{fillin}(v) := 0$
4: $M(v) := \text{Neighbor}(v)$
5: for each $x, y \in M(u)$ and $(x, y) \notin \mathcal{E}$ do
6: $\text{fillin}(v) := \text{fillin}(v) + 1$
7: end for
8: end for
9: $S := \mathcal{V}$
10: for $i = |\mathcal{V}|$ to 1 do
11: $u := \arg \min_{v \in S} \text{fillin}(v)$
12: $\sigma(i) := u$
13: for $x, y \in M(u)$ and $(x, y) \notin \mathcal{E} \cup \mathcal{E}_t$ do
14: $\mathcal{E}_t := \mathcal{E}_t \cup (x, y)$
15: $M(x) := M(x) \cup y$ and $M(y) := M(y) \cup x$
16: end for
17: for $w \in M(u)$ do
18: $M(w) := M(w) \setminus u$
19: $\text{fillin}(v) := 0$
20: for each $x, y \in M(w)$ and $(x, y) \notin \mathcal{E} \cup \mathcal{E}_t$ do
21: $\text{fillin}(v) := \text{fillin}(v) + 1$
22: end for
23: end for
24: $S := S \setminus \{u\}$
25: end for
A.5 Lexicographic BFS Heuristic
Triangulation, variant Perfect

Algorithm A.5 Lexicographic BFS Heuristic Triangulation, variant Perfect (LEXP) [51]

Input: A connected graph $G = (V, E)$, arbitrary vertex $v^* \in V$
Output: Triangulated graph $G_t = (V, E \cup E_t)$ with $w^*(G) \leq w^*(G_t)$ and an elimination ordering $\sigma = (v_1, \ldots, v_n)$

1: $E_t := \emptyset$
2: for $v \in V$ do
3: label$(v)$ := 0
4: end for
5: $S := V$
6: for $i = |V|$ to 1 do
7: if $i == |V| \text{ then}$
8: $u := v^*$
9: else
10: $u := \arg \max_{v \in S} \text{ label}(v)$
11: end if
12: $\sigma(i) := u$
13: for $j, k \in \{i + 1, \ldots, |V|\}, j \neq k$ do
14: if $(\alpha(i), \alpha(j)), (\alpha(i), \alpha(k)) \in E$ and $(\alpha(j), \alpha(k)) \notin E \cup E_t$ then
15: $E_t := E_t \cup (\alpha(j), \alpha(k))$
16: end if
17: end for
18: $S := S \setminus \{u\}$
19: for $w \in \text{ Neighbor}(u) \cap S$ do
20: label$(w)$ := label$(w) \cup \{i\}$
21: end for
22: end for

Appendix A. Heuristics Search Algorithms for Triangulation
A.6 Lexicographic BFS Heuristic Triangulation, variant Minimal

Algorithm A.6 Lexicographic BFS Heuristic Triangulation, variant Minimal (LEXM) [51]

**Input:** A connected graph \( G = (V, E) \), arbitrary vertex \( v^* \in V \)

**Output:** Triangulated graph \( G_t = (V, E \cup E_t) \) with \( w^*(G_t) \leq w^*(G_t) \) and an elimination ordering \( \sigma = (v_1, \ldots, v_n) \)

1: \( E_t := \emptyset \)
2: for \( v \in V \) do
3: \( \text{label}(v) := 0 \)
4: end for
5: \( S := V \)
6: if \( i = |V| \) to 1 do
7: \( u := v^* \)
8: else
9: \( u := \text{arg max}_{v \in S} \text{label}(v) \)
10: end if
11: \( \sigma(i) := u \)
12: for \( j, k \in \{i + 1, \ldots, |V|\}, j \neq k \) do
13: if \( (\alpha(i), \alpha(j)), (\alpha(i), \alpha(k)) \in E \) and \( (\alpha(j), \alpha(k)) \notin E \cup E_t \) then
14: \( E_t := E_t \cup (\alpha(j), \alpha(k)) \)
15: end if
16: end for
17: \( S := S \setminus \{u\} \)
18: for \( w \in S \) s.t. \( \exists \) path \( \{u = v_1, \ldots, v_{k+1} = w\} \) in \( G \) with \( v_j \in S \) and \( \text{label}(v_j) < \text{label}(w) \) for \( j = 2, 3, \ldots, k \) do
19: \( \text{label}(w) := \text{label}(w) \cup \{i\} \)
20: end for
21: end for
22: end for
A.7 Maximum Cardinality Heuristic Triangulation

Algorithm A.7 Maximum Cardinality Heuristic Triangulation (MC) [56]

Input: A connected graph $G = (V, E)$

Output: Triangulated graph $G_t = (V, E \cup E_t)$ with $w^*(G) \leq w^*(G_t)$ and an elimination ordering $\sigma = (v_1, \ldots, v_n)$

1: $E_t := \emptyset$
2: for $v \in V$ do
3:   $\text{counter}(v) := 0$
4: end for
5: $S := V$
6: for $i = |V|$ to 1 do
7:   $u := \arg \max_{v \in S} \text{counter}(v)$
8:   $\sigma(i) := u$
9:   for $j, k \in \{i + 1, \ldots, |V|\}, j \neq k$ do
10:     if $(\sigma(i), \sigma(j)), (\sigma(i), \sigma(k)) \in E$ and $(\sigma(j), \sigma(k)) \notin E \cup E_t$ then
11:         $E_t := E_t \cup (\sigma(j), \sigma(k))$
12:     end if
13:   end for
14:   $S := S \setminus \{u\}$
15: for $w \in \text{Neighbor}(u) \cap S$ do
16:   $\text{counter}(w) := \text{counter}(w) + 1$
17: end for
18: end for