

**UTILIZING SEMANTICS IN ITEMSET  
MINING AND JOINTREE PROBABILITY  
PROPAGATION**

A Thesis

Submitted to the Faculty of Graduate Studies and Research

for the Degree of Doctor of Philosophy

in

Computer Science

University of Regina

by

Hong Yao

Regina, Saskatchewan

August, 2006

© Copyright 2006: Hong Yao

# Abstract

The present investigation studies the use of semantics in itemset mining and jointree propagation. By representing more of the semantics of itemsets than previous research, we establish a theoretical basis for the study of itemset mining from the view of utility and we provide a utility based itemset mining approach to find itemsets conforming to user preferences. Our utility based method for itemset mining can find types of itemsets that could not be identified using previous theories and techniques. We also propose a novel algorithm for the discovery of functional dependencies from data. Our approach can reduce the number of functional dependencies to be checked in comparison with previous methods. Then, we show a practical application of discovering functional dependencies from data, namely, constructing a sound Bayesian network efficiently. Finally, by applying more of the semantic information available in a Bayesian network than previous research, we propose the first jointree propagation architecture that labels the probability information passed between jointree nodes in terms of conditional probability tables rather than potentials. We prove the soundness of our architecture. One striking feature of our architecture is that after propagation, each jointree node has a sound, local Bayesian network that preserves all conditional independencies of the original Bayesian network. We show how our architecture is beneficial to three techniques for implementing inference in probabilistic expert systems, namely, direct computation, multiply sectioned Bayesian networks, and LAZY propagation.

# Acknowledgements

I wish to express my deepest gratitude to my co-supervisors, Dr. Cory Butz and Dr. Howard Hamilton, for their instruction, inspiration, continuous encouragement, and support. I will remain indebted to them for leading me into scientific research. As advisors, they taught me practices and skills that I will use in my future career.

I am thankful to Dr. Yiyu Yao for sharing with me his knowledge and experience. Talking with him is enjoyable and inspiring. I would also like to thank Dr. Dianliang Deng who made insightful comments which resulted in a much improved dissertation and Professor Weiyi Liu in Yunnan University for educating and training me.

I am grateful to the Faculty of Graduate Studies and Research, the Department of Computer Science, Nature Sciences and Engineering Research Council of Canada, and of course again my supervisors, for their generous financial support during the course of my Ph.D. study.

The faculty of the Computer Science department have always been very helpful to me. My particular thanks go to Dr. Maguire.

I would also like to thank many people in our department who have helped me on many occasions over the years. I thank my friends for their help. A particular acknowledgement goes to four couples, Yan Zhao and Chi Song, Honglan Zhong and Chunren Lai, Hao Zheng and Dan Wu, and Jidong Liu and Joseph Herbert.

My parents made many sacrifices to give me a better education. Their understanding and encouragement made the tough training process easier. Not least, I wish to express my heartfelt gratitude to my wife who has fully supported my studies. Without them, I would have found it difficult to complete my studies.

# Dedication

I dedicate this thesis to my parents and my wife. The love of my family accompanies me wherever I go. I would like to acknowledge my mother and father who have made many sacrifices for my benefit. When I was young and needed them the most, they were always there to support me. When they are getting old and need me the most, I am away. I feel that they paid too high a price for me to get a better education. I am also grateful to my wife for her patience and encouragement. Without their constant understanding, encouragement, support, and love, it would be very hard for me to undertake and finish my Ph.D. program. I believe that reaching high standards in my current training will prepare me well to meet tough demands in the future. I hope that my achievements will make them happy and proud.

# Post Defense Acknowledgements

I am thankful to my external examiner Dr. Dale Schuurmans, internal examiners Dr. Yiyu Yao and Dr. Dianliang Deng, and chair Dr. David deMontigny. Their constructive criticisms and valuable feedback helped me improve the quality of my dissertation.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgements</b>	<b>ii</b>
<b>Dedication</b>	<b>iii</b>
<b>Post Defense Acknowledgements</b>	<b>iv</b>
<b>List of Figures</b>	<b>x</b>
<b>List of Tables</b>	<b>xii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Utility Based Itemset Mining</b>	<b>10</b>
2.1 Constraint Based Itemset Mining . . . . .	10
2.1.1 Frequent Itemset Mining . . . . .	10
2.1.2 An Example of the Problem . . . . .	15
2.1.3 Constraint Based Approaches . . . . .	17
2.2 Theoretical Foundation . . . . .	21
2.2.1 Definitions . . . . .	21
2.2.2 Mathematical Properties of the Utility Constraint . . . . .	28
2.3 Two Efficient Pruning Strategies . . . . .	34
2.4 Algorithms for Utility Based Itemset Mining . . . . .	39
2.4.1 The UMining Algorithm . . . . .	40
2.4.2 The UMining_H Algorithm . . . . .	46

2.5	Experimental Results . . . . .	48
2.5.1	Experimental Summary . . . . .	49
2.5.2	Algorithm Comparison . . . . .	54
2.5.3	Utility versus Support . . . . .	57
2.6	Summary . . . . .	58
<b>3</b>	<b>Mining Functional Dependencies from Data</b>	<b>60</b>
3.1	Functional Dependencies . . . . .	60
3.2	Theoretical Foundation . . . . .	64
3.2.1	Equivalent Attributes . . . . .	64
3.2.2	Pruning Rules . . . . .	67
3.3	The FD_Mine Algorithm . . . . .	71
3.4	Experimental Results . . . . .	79
3.4.1	Experimental Summary . . . . .	79
3.4.2	Algorithm Comparison . . . . .	80
3.5	Constructing a Bayesian Network . . . . .	83
3.5.1	Bayesian Networks . . . . .	84
3.5.2	The FD2BN Algorithm . . . . .	88
3.5.3	Correctness and Complexity . . . . .	91
3.6	Summary . . . . .	93
<b>4</b>	<b>A Jointree Probability Propagation Architecture for Semantic Mod- eling</b>	<b>94</b>
4.1	LAZY Propagation . . . . .	95
4.2	Modeling Inference Not Involving Evidence . . . . .	99
4.3	Complexity and Correctness . . . . .	109

4.4	Modeling Inference Involving Evidence . . . . .	118
4.5	Local BNs and Practical Applications . . . . .	129
4.6	Summary . . . . .	137
<b>5</b>	<b>Conclusion</b>	<b>138</b>
5.1	Utility based Itemset Mining . . . . .	139
5.2	Learning Functional Dependencies and Bayesian Networks . . . . .	140
5.3	Jointree Probability Propagation for Semantic Modelling . . . . .	141
5.4	Future Research . . . . .	142
	<b>References</b>	<b>145</b>



# List of Figures

2.1	The Apriori algorithm. . . . .	14
2.2	The Apriori-Gen algorithm. . . . .	14
2.3	The UMining algorithm. . . . .	41
2.4	The <i>Scan</i> function for the UMining algorithm. . . . .	41
2.5	The <i>CalculateAndStore</i> function for the UMining algorithm. . . . .	42
2.6	The <i>Discover</i> function for the UMining algorithm. . . . .	42
2.7	The <i>Generate</i> function for the UMining algorithm. . . . .	42
2.8	The <i>Prune</i> function for the UMining algorithm. . . . .	43
2.9	Itemset semi-lattice for the UMining algorithm. . . . .	45
2.10	The <i>CalculateAndStore_H</i> function for the UMining_H algorithm. . . . .	46
2.11	The <i>Prune_H</i> function for the UMining_H algorithm. . . . .	47
2.12	Itemset semi-lattice for the UMining_H algorithm. . . . .	48
2.13	Itemsets ranked by their utility values and support values. . . . .	58
2.14	Graph of itemsets ranked by their utility values. . . . .	58
3.1	All possible nonempty combinations of attributes $A, B, C, D$ , and $E$ . . . . .	68
3.2	The FD_Mine algorithm. . . . .	72
3.3	The <i>InitializeClosure</i> algorithm of FD_Mine. . . . .	73
3.4	The <i>ObtainFDs</i> algorithm of FD_Mine. . . . .	74
3.5	The <i>ObtainEquivalences</i> algorithm of FD_Mine. . . . .	74
3.6	The <i>Prune</i> algorithm of FD_Mine. . . . .	75
3.7	Portion of semi-lattice accessed by FD_Mine. . . . .	77
3.8	Portion of the semi-lattice accessed by TANE. . . . .	81

3.9	A directed acyclic graph (DAG) on $U = \{a, b, c, d, e, f, g, h, i, j, k\}$ [32].	
	87	
3.10	The <i>ObtainOrdering</i> algorithm of FD2BN. . . . .	89
3.11	The FD2BN algorithm. . . . .	90
3.12	The DAG of a Bayesian network for the heart disease dataset. . . . .	91
4.1	One possible jointree for the Bayesian network in Figure 3.9. . . . .	96
4.2	A jointree for the BN in Figure 3.9 with assigned CPTs. . . . .	98
4.3	The probability information being passed between JT nodes. . . . .	101
4.4	The CPT-graph for Example 35. . . . .	103
4.5	The CPT-graph defined by the CPT labels in Example 36. . . . .	103
4.6	The elder-sets $E_1, E_2$ , and $E_3$ for the variables in the child-set $C_i =$ $\{v_1 = j, v_2 = k, v_3 = l\}$ . . . . .	104
4.7	Identification of the CPT messages in Figure 4.3. . . . .	108
4.8	The propagated CPT labels given evidence $b = 0$ . . . . .	119
4.9	[54] A JT with assigned BN CPTs (top) and all identified CPT labels given evidence $a = 0, c = 0, f = 0$ (bottom). . . . .	120
4.10	[54] A BN and a JT with assigned CPTs. . . . .	121
4.11	[54] A BN and a JT with assigned CPTs. . . . .	122
4.12	The irrelevant message from $abc$ to $bcdef$ given evidence $d = 0$ . . . . .	124
4.13	The irrelevant message from $abcd$ to $acde$ . . . . .	124
4.14	An extended CHD BN and a JT with assigned CPTs. . . . .	125
4.15	All empty messages sent by non-leaf nodes. . . . .	125
4.16	The variables that can be eliminated at non-leaf nodes before any messages are received. . . . .	126

4.17 *Local* BNs after propagation not involving evidence for the CHD BN. 131

# List of Tables

2.1	A transaction database. . . . .	12
2.2	The support of all itemsets. . . . .	13
2.3	The profit table for the items. . . . .	16
2.4	The support and the profits of all itemsets. . . . .	16
2.5	A medical dataset obtained from an example in [19]. . . . .	20
2.6	A transaction database. . . . .	39
2.7	Experimental results on synthetic dataset for UMining and UMining_H. . . . .	50
2.8	The accuracy of estimation and percentage of time saved . . . . .	51
2.9	Experimental results on the customer database. . . . .	52
2.10	The accuracy of estimation and percentage of time saved . . . . .	53
2.11	The effect of the size of commercial database on the running time. . . . .	54
2.12	The comparison of UMining and SIP on IBM synthetic dataset. . . . .	56
2.13	Comparison of UMining and SIP on the commercial database. . . . .	57
3.1	A relation $r$ on the relational $U = \{v_1, \dots, v_m\}$ . . . . .	61
3.2	An example relation. . . . .	62
3.3	Effect of removing a redundant attribute $D$ . . . . .	70
3.4	Experimental results for FD_Mine on fifteen datasets. . . . .	80
3.5	Number of FDs checked for the sample relation in Table 3.2. . . . .	82
3.6	The number of FDs checked in the UCI datasets. . . . .	82
3.7	Number of FDs checked on the <i>Imports-85</i> and <i>Hepatitis</i> Datasets. . . . .	83
3.8	The number of FDs found in the UCI datasets UCI Datasets. . . . .	84
3.9	A joint probability distribution $p(v_1, v_2)$ . . . . .	85

3.10	Eleven conditional probability tables (CPTs). . . . .	86
4.1	Five potentials $\phi(b)$ , $\phi(f, g)$ , $\phi(g, h)$ , $\phi(f)$ , and $\phi(g)$ . . . . .	96
4.2	Information about five Bayesian networks. . . . .	127
4.3	Experimental results on five Bayesian networks without evidence. . .	128
4.4	Experimental results on five Bayesian networks with 18% of the variables randomly instantiated as evidence variables. . . . .	129
4.5	The computation needed in DC to process five localized queries in the original CHD BN in Figure 3.9 versus the local BNs in Figure 4.17.	136

# Chapter 1

## Introduction

Two primary research areas of Artificial Intelligence (AI) are knowledge discovery from databases and uncertain reasoning. Doyle et al. [25] explicitly stated that one of the principal objectives of AI is to formalize knowledge and mechanize reasoning. Reasoning involves knowledge, which can be learned from experience or acquired from data. For example, doctors often use clinical reasoning to select and apply an appropriate treatment based on their knowledge of medical science. Financial companies identify suspicious transactions and detect credit card fraud by analyzing transaction data. Thus, reasoning and knowledge discovery are used for solving complex problems. The work in this thesis spans these two important areas. The main objective in this thesis is to apply semantics to knowledge discovery and reasoning.

*Knowledge discovery in databases* (KDD), also known as *data mining*, refers to the finding of valid, novel, potentially useful, and ultimately understandable patterns in data [27]. Many kinds of knowledge can be discovered from data. For instance,

decision trees [71] can be learned for solving classification problems, such as deciding whether an unknown email is a useful message or a spam message. Clusters [34, 83] can be identified to group similar items, such as documents. Outliers [34, 83] can be detected for security management. For instance, credit card companies can identify suspicious charges if a purchase does not fit a pattern of spending that a customer has established. Frequent itemsets [2, 3, 57] can be discovered from market basket data and used to derive association rules [2, 3, 57] for predicting the conditional probability of the purchase of certain items, given the purchase of other items. In the portion of this thesis concerning knowledge discovery, we concentrate on itemset mining.

Frequent itemset mining plays an essential role in the theory and practice of many important data mining tasks, such as the mining of association rules, long patterns [8], emerging patterns [24], and dependency rules [81]. For example, an association rule  $beer \xrightarrow{0.8} diapers$  indicates that diapers occur in 80% of the transactions that include beer. Frequent itemset mining has been applied in fields, such as telecommunications [5], census analysis [9], and text analysis [81]. An *itemset* is a set of items. The goal of frequent itemset mining is to identify all *frequent itemsets*, i.e., itemsets that have at least a specified minimum *support*, which is the percentage of transactions containing the itemset. Frequent itemset mining is based on the assumption that only itemsets with high support are of interest to users.

The usefulness of itemset mining with the support measure is restricted by problems with the quantity and quality of the mined results. A huge number of frequent itemsets that are not interesting to the user are often generated when the minimum support is set to a low value. For example, thousands of combinations of products may occur in 1% of the transactions. If too many uninteresting frequent itemsets

are found, the user is forced to do additional work to select the association rules that are interesting. The quality problem is that support, as defined based on the frequency of itemsets, is not necessarily an adequate measure of a typical user's interest. A sales manager may not be interested in frequent itemsets that do not generate significant profit. More generally, an itemset that is of interest to one user may not be of interest to another user, since users have different levels of interest in itemsets. Thus, by performing itemset mining using previous approaches, a user may incur a high computational cost that is disproportionate to what the user wants and receives [62].

One solution proposed to address the problems with the quantity and quality of the discovered results is constraint based itemset mining [2, 7, 17, 19, 49, 53, 57, 70, 92, 94]. The goal of constraint based itemset mining is to ensure the usefulness of itemsets by means of constraints. Two kinds of constraints have been considered.

One kind of constraint is based on interestingness measures [30, 38, 82, 96]. An *interestingness measure* is a metric that captures the dependencies of interest among items [82]. For example, metrics such as support, confidence, and correlation have been used extensively to evaluate the interestingness of association rules [2, 81]. A comprehensive study of twenty-one measures that were originally developed in diverse fields such as statistics, social science, machine learning, and data mining is presented by Tan et al. [82]. Hilderman and Hamilton [38] theoretically and empirically evaluated twelve diversity measures used as heuristic measures of interestingness for ranking summaries generated from a database. Yao et al. [96] presented a simple and unified framework for the study of quantitative measures associated with rules. Recent research on interestingness measures has focused on using a statistical or mathematical method to evaluate the usefulness of rules [30]. It is not trivial for



a human expert to understand or choose one of these measures. Even data mining specialists or practitioners may not be familiar with all available measures. More importantly, these interestingness measures may not express the semantics of applications, such as cost, profit, or aesthetic value. A natural way of allowing a user to specify an interesting measure is to allow the user to express his or her beliefs about the usefulness of potential results since only the user knows his or her information needs. That is, it is appropriate to consider user-specified constraints, which bring more of the semantics of the applications into itemset mining.

Another kind of constraint is based on functional dependency, which obey sound and complete axioms. A *functional dependency* [56] is a constraint between two sets of attributes in a relation from a database. Functional dependency has a sound and complete axiomatization, called *Armstrong's Axioms* [56], which are a set of axioms used to infer all the functional dependencies on a relational database. The discovery of functional dependencies from data has been extensively studied [41, 58, 64, 65, 91]. Research related to constraints with Armstrong's Axioms attempts to develop an efficient pruning strategy by incorporating these axioms as deeply as possible into the mining process. Previous research has reduced the number of functional dependencies to be checked by using pruning rules, but other efficient pruning rules can be identified and incorporated into mining process. Moreover, other practical applications of discovering functional dependencies from data need to be identified. It is appropriate to consider more efficient pruning rules and other potential applications of discovering functional dependencies from data.

A problem closely related to the issue of discovering knowledge is that of using knowledge to support reasoning. Before reasoning can commence, a specification of a model is required. Bayesian networks are widely used for uncertain reasoning

using probability [18, 22, 32, 43, 61, 68, 87]. A *Bayesian network* is a graphical model that encodes probabilistic relationships among variables of interest. Bayesian networks have been applied to medical diagnosis by the Heart Disease Program at the Massachusetts Institute of Technology [52] and the Pathfinder Project for lymph-node diseases at the Stanford University [35]. A Bayesian network is used in the intelligent assistant in the Microsoft Office software that offers users help based on other user previous experience [36, 40]. Another Bayesian network system is used by NASA for ground controllers at Space Shuttle Mission Control in Houston. [39]. At Nokia, Bayesian networks are used to predict and monitor network problems [63].

The central task when performing probabilistic inference using a Bayesian network is to determine the posterior probability of one or more variables given some observations. Three approaches have been developed to perform probabilistic inference in a Bayesian network. One approach, called *direct computation* [23, 48, 97], answers queries directly in the original Bayesian network. A second approach, called *multiply sectioned Bayesian networks* [86–89], performs inference in sections of a Bayesian network. A third approach, called *jointree propagation*, performs inference in a jointree [68, 77] constructed from the original Bayesian network. Shafer, a pioneer of Bayesian networks, explicitly stated that jointree probability propagation is central to the theory and practice of probabilistic expert systems [76]. In this thesis, we emphasize the third approach, but we also describe some results relevant to the other approaches.

Jointree propagation has been extensively studied [42, 45, 54, 55, 78]. Three classical methods for propagating probabilities in a jointree were proposed by Lauritzen and Spiegelhalter [45], Shafer and Shenoy [76], and Jensen et al. [42]. A recent jointree propagation algorithm suggested by Madsen and Jensen [54], called *LAZY*

*propagation*, appears to be the most efficient probabilistic inference algorithm, according to the experimental results presented in [54]. LAZY propagation uses lazy evaluation of the messages passed between a jointree node and a separator in a jointree tree [32, 43]. Unlike traditional approaches that multiply together all the potentials to form a single potential at each jointree node and jointree separator, LAZY propagation maintains a multiplicative factorization of the potentials at each jointree node and jointree separator. As a result, LAZY propagation can remove *irrelevant potentials*, i.e., potentials irrelevant to the computation of an outgoing message, from the multiplicative factorization. However, the independencies holding in the remaining relevant potentials are ignored in LAZY propagation. It is appropriate to improve the inference performance of jointree propagation by identifying and applying independence information that remains unnoticed in previous architectures.

Based on the preceding analysis, the three problems addressed in this thesis are as follows:

- (1) Given a transaction database, find all itemsets that satisfy a user defined constraint on their utility.
- (2) Given a transaction database, efficiently find functional dependencies encoded in this database and organize the resulting functional dependencies to support reasoning.
- (3) Improve the performance of probabilistic inference in a Bayesian network by identifying and applying independence information that remains unnoticed in previous jointree architectures.

To solve the above three problems, we propose the following approaches by utilizing semantics in itemset mining and jointree propagation.

(1) We present a utility based itemset mining approach by applying the semantics of user-specified constraints, in the form of a utility constraint.

(2) We propose an efficient method to discover functional dependencies, and construct a sound Bayesian network using resulting functional dependencies by exploiting the semantics of functional dependency and conditional independence.

(3) We propose a jointree propagation architecture for semantic modelling by utilizing semantic information concerning Bayesian networks.

All of these approaches utilize more semantic information than previous approaches. In the first approach, the semantics of usefulness of an itemset is defined as the utility value of the itemset. In the second approach, the semantic relationship between functional dependency and conditional independence is exploited. In the third approach, the semantics of the messages passed between jointree nodes are identified in terms of conditional probability tables.

In addition, the first approach relates to KDD, the third approach relates to uncertainty reasoning, and the second relates to both. More specifically, the second approach requires establishing a connection between KDD and uncertainty reasoning.

The remainder of this thesis is organized in four chapters. Each of Chapter 2, 3, and 4 describes and evaluates one of the approaches, and Chapter 5 presents our conclusions. We now describe these chapters in some detail.

In Chapter 2, we propose a utility based itemset mining approach to discover itemsets matching a user's interest [92–94]. We use a utility function as a quantitative representation of user preference. In particular, the usefulness of an itemset is quantified in terms of its utility value. Based on the utility value of an itemset, a utility constraint is defined. By analyzing the mathematical properties of utility

constraints, the upper bound of the utility value of an itemset is characterized. A pruning strategy is proposed to reduce the search space by exploiting the upper bound property of the utility constraint. Moreover, a heuristic pruning strategy to further reduce search cost is proposed by analyzing the relationship between the upper bound of the utility value of an itemset and the support of the itemset. Based on these two pruning strategies, we propose two algorithms, called UMining and UMining\_H. UMining finds all itemsets with utility values of at least a given value. UMining\_H is a heuristic algorithm that finds some and perhaps all itemsets with utility values of at least a given value. We prove the correctness of UMining. The effectiveness of the algorithms is demonstrated by applying them to synthetic and real-world datasets.

In Chapter 3, we propose the FD\_Mine algorithm [91], for discovering functional dependencies from data. Based on Armstrong's Axioms, we identify equivalences among attributes. Next, we summarize four pruning rules to avoid searching for functional dependencies that are logically implied by the functional dependencies already discovered. By applying these pruning rules, the FD\_Mine algorithm is developed. We report the results of a series of experiments on synthetic and real-world datasets. Our study shows that FD\_Mine can prune more candidates than previous methods [41,58,64,65] without eliminating any valid candidates. Furthermore, using the implication relationship that functional dependency logically implies conditional independence [15], we propose a new algorithm, called FD2BN [95], to construct a sound Bayesian network from functional dependencies discovered by FD\_Mine. We prove the correctness of FD\_Mine and FD2BN.

In Chapter 4, we propose the first jointree propagation architecture for modeling two inference tasks, one involving evidence and the other not. By modeling jointree

inference involving evidence, our architecture [16] precisely labels the probability information propagated in terms of conditional probability tables by identifying independencies that are not utilized in previous architectures. We propose a novel algorithm, called *IdentifyCPTMessages*, to determine the labels of conditional probability table corresponding to the probability information to be sent from a jointree node to a neighbour. We prove the correctness of our architecture and also show that each jointree node can identify its labels of conditional probability table in polynomial time. Screen shots of our implemented system demonstrate the improvements in the semantic information. We make use of this semantic information to generate three work schedules for LAZY propagation. Then we show that our architecture is also useful for modeling inference not involving evidence. After the conditional probability tables identified by our architecture have been physically constructed, each jointree node has a sound, local Bayesian network preserving all conditional independencies of the original Bayesian network involving variables in this node. We show two practical applications of local Bayesian networks. First, we propose an automated multiply sectioned Bayesian network modeling procedure, and secondly, we show how direct computation approaches can exploit localized queries.

Chapter 5 draws conclusions concerning our study of utilizing semantics in item-set mining and jointree propagation. We summarize the results of this study and discuss future research topics.

# Chapter 2

## Utility Based Itemset Mining

In this chapter, we briefly review constraint based itemset mining in Section 2.1. In Section 2.2, the theoretical foundations of utility constraints are analyzed. Two pruning strategies are presented in Section 2.3. Section 2.4 describes algorithms for utility based itemset mining. In Section 2.5, our experimental results are presented. Finally, in Section 2.6, we summarize the chapter.

### 2.1 Constraint Based Itemset Mining

In this section, we first review frequent itemset mining, a special case of constraint based itemset mining. Then, an example of the problem is given. Finally, we survey several previous approaches relevant to constraint based itemset mining.

#### 2.1.1 Frequent Itemset Mining

Adapting from the notations used in the descriptions of other itemset mining approaches [2, 17, 70], we use the following notation.  $I = \{i_1, \dots, i_m\}$  is a set of items.

Each item is an object associated with an attribute of a database. A transaction database  $T$  is a set of variable length transactions, denoted  $T = \{t_1, \dots, t_n\}$ . Each  $t_q$  in  $T$  is called a *transaction* and contains a set of items  $i_1, \dots, i_k \in I$ . A transaction is also assigned a unique transaction identifier, called *TID*.  $|T|$  denotes the number of transactions in  $T$ . An *itemset*  $S$  is a subset of  $I$ , i.e.,  $S \subseteq I$ . A  $k$ -itemset is an itemset containing exactly  $k$  items. To simplify notation, we sometimes write an itemset  $\{i_1, \dots, i_k\}$  as  $i_1 \dots i_k$ ; e.g.,  $ABCD$  represents itemset  $\{A, B, C, D\}$ . The union  $X \cup Y$  of two itemsets  $X$  and  $Y$  is sometimes simply denoted as  $XY$ .

**Definition 1** The *transaction set of an itemset*  $S$ , denoted  $T_S$ , is the set of transactions that contain itemset  $S$ , i.e.,

$$T_S = \{t_q \mid S \subseteq t_q, t_q \in T\}.$$

**Definition 2** The *support of an itemset*  $S$ , denoted  $sup(S)$ , in a transaction database  $T$ , is the fraction of transactions in  $T$  containing  $S$ , i.e.,

$$sup(S) = \frac{|T_S|}{|T|}.$$

**Definition 3** An itemset  $S$  is a *frequent itemset* if  $sup(S) \geq min\_sup$ , where  $min\_sup$  is a support threshold defined by the user. Otherwise,  $S$  is an *infrequent itemset*.

**Example 1** Consider the small transaction database shown in Table 2.1. Each



nonzero value in the transaction database indicates the quantity sold of an item and a zero value indicates absence of an item in a transaction. By definition,  $sup(AD) = |T_{AD}|/|T| = 7/10 = 70\%$ . The supports for the other itemsets are shown in Table 2.2. Suppose  $min\_sup = 40\%$ . By definition, the itemsets  $A$ ,  $C$ ,  $D$ , and  $AD$  are frequent itemsets.

Table 2.1: A transaction database.

TID	A	B	C	D
$t_1$	4	0	1	0
$t_2$	2	0	0	6
$t_3$	0	0	1	30
$t_4$	3	0	0	5
$t_5$	1	0	0	6
$t_6$	4	0	2	10
$t_7$	2	0	0	8
$t_8$	1	1	1	1
$t_9$	0	1	0	10
$t_{10}$	5	0	0	9

An important property of the support constraint  $sup(S) \geq min\_sup$  is that it satisfies the Apriori property.

**Theorem 1** (Apriori property). *Let  $C$  be the support constraint  $sup(S) \geq min\_sup$ . Whenever an itemset  $S$  violates constraint  $C$ , so does any superset of  $S$ .*

Theorem 1 is presented in [2] without proof. A straightforward proof of this theorem is given as follows.

*Proof:* Let  $S \subset S'$ . By Definition 1,  $T_{S'} \leq T_S$ . By Definition 2,  $sup(S') \leq sup(S)$ . Thus, if  $sup(S) < min\_sup$ , then  $sup(S') < min\_sup$ .  $\square$

The Apriori property indicates that if an itemset is frequent, then all its non-empty subsets are also frequent. Thus, only those itemsets that consist of frequent

Table 2.2: The support of all itemsets.

Itemsets	Support (%)
<i>A</i>	80
<i>B</i>	20
<i>C</i>	40
<i>D</i>	90
<i>AB</i>	10
<i>AC</i>	30
<i>AD</i>	70
<i>BC</i>	10
<i>BD</i>	20
<i>CD</i>	30
<i>ABC</i>	10
<i>ABD</i>	10
<i>ACD</i>	20
<i>BCD</i>	10
<i>ABCD</i>	10

subsets can potentially be frequent. In other words, if any subset of an itemset  $S$  is an infrequent itemset, then  $S$  must be an infrequent itemset.

Based on the Apriori property, the Apriori algorithm was developed. Our version of the Apriori algorithm is presented in Figure 2.1. In this algorithm, the Apriori property is applied by the *Apriori-Gen* algorithm, which is shown in Figure 2.2. *Apriori-Gen* generates all possible candidate  $k$ -itemsets from the  $(k - 1)$ -itemsets in  $L_{k-1}$ . For details on the Apriori algorithm, the reader is referred to work by Agrawal et al. [3].

**Apriori**( $T, min\_sup$ )

Input: Transaction database  $T$ , support threshold  $min\_sup$ .

Output: A set of frequent itemsets  $F$ .

```
1.   $I = Scan(T)$ ;  
2.   $C_1 = I$ ;  
3.   $k = 1$ ;  
4.   $C_k = CalculateAndStore(C_k, T)$ ;  
5.   $L_k = Discover(C_k, min\_sup)$ ;  
6.  while (  $|L_k| > 0$  ) do  
7.  {  
8.     $k = k + 1$ ;  
9.     $C_k = Apriori-Gen(L_{k-1})$ ;  
10.    $C_k = Prune(C_k, L_{k-1})$ ;  
11.    $C_k = CalculateAndStore(C_k, T)$ ;  
12.    $L_k = Discover(C_k, min\_sup)$ ;  
13. }  
14.  $F = \bigcup_{j=1}^{k-1} L_j$ ;  
15. return( $F$ );
```

Figure 2.1: The Apriori algorithm.

**Apriori-Gen**( $L_{k-1}$ )

Input:  $L_{k-1}$ , a set of frequent  $(k - 1)$ -itemsets.

Output:  $C_k$ , a set of candidate  $k$ -itemsets.

```
1.  insert into  $C_k$   
2.    select  $p.i_1, \dots, p.i_{k-1}, q.i_{k-1}$   
3.    from  $L_{k-1}.p, L_{k-1}.q$   
4.    where  $p.i_1 = q.i_1, \dots, p.i_{k-2} = q.i_{k-2}$ , and  $p.i_{k-1} \prec q.i_{k-1}$ ;  
5.  return  $C_k$ ;
```

Figure 2.2: The Apriori-Gen algorithm.

### 2.1.2 An Example of the Problem

The practical usefulness of frequent itemsets is limited by the significance of the discovered itemsets. Though selecting itemsets based on frequency alone is valuable, this approach treats all items and transactions in a transaction database uniformly. Identifying an itemset as frequent reflects the frequency of combinations of items, but it does not reflect the semantic significance of the items. The support constraint may not adequately measure a typical user's interest. The following example shows that support based itemset mining may lead to some itemsets that are significant to the user not being discovered due to their low supports.

**Example 2** Consider the small transaction database shown in Table 2.1 and the unit profit for the items shown in Table 2.3. Suppose that the goal of a sales manager is to find the itemsets that can generate a profit greater than or equal to a threshold. Using Table 2.1 and 2.3, the support and profit for all itemsets can be calculated (see Table 2.4). For example, since for the 10 transactions in Table 2.1, only two transactions,  $t_8$  and  $t_9$ , include both items  $B$  and  $D$ , the support of the itemset  $BD$  is  $2/10 = 20\%$ . Since  $t_8$  includes one  $B$  and one  $D$ , and  $t_9$  includes one  $B$  and ten  $D$ s, a total of two  $B$ s and eleven  $D$ s appear in transactions containing the itemset  $BD$ . Using the Table 2.3, the profit for each item  $B$  is 100 and the profit for each item  $D$  is 1. Thus, the profit of the itemsets  $BD$  can be considered to be  $2 \times 100 + 11 \times 1 = 211$ . The profit of the other itemsets in Table 2.4 can be obtained in a similar fashion. Supposing that the minimum support is 40%, the frequent itemsets in Table 2.4 are  $D$ ,  $A$ ,  $AD$ , and  $C$ , but the four most profitable itemsets are  $BD$ ,  $B$ ,  $AC$ , and  $CD$ , all of which are infrequent itemsets.

Table 2.3: The profit table for the items.

Item Name	Profit (\$)
Item A	5
Item B	100
Item C	38
Item D	1

Table 2.4: The support and the profits of all itemsets.

Itemsets	Support (%)	Profit (\$)
<i>A</i>	<b>80</b>	110
<i>B</i>	20	<b>200</b>
<i>C</i>	<b>40</b>	190
<i>D</i>	<b>90</b>	85
<i>AB</i>	10	105
<i>AC</i>	30	<b>197</b>
<i>AD</i>	<b>70</b>	135
<i>BC</i>	10	138
<i>BD</i>	20	<b>211</b>
<i>CD</i>	30	<b>193</b>
<i>ABC</i>	10	143
<i>ABD</i>	10	106
<i>ACD</i>	20	150
<i>BCD</i>	10	139
<i>ABCD</i>	10	144

Example 2 shows that itemset frequency may not measure how useful an itemset is in accordance with a user's preferences, such as profit. It is appropriate to be able to model a variety of types of semantic significance for itemsets. More precisely, more general constraints are required to express different types of semantic significance for itemsets. This requirement motivated researchers [7, 17, 19, 49, 53, 70, 92] to develop constraint based itemset mining approaches.

### 2.1.3 Constraint Based Approaches

Recent work [7,17,19,49,53,70,92] has highlighted the importance of constraint based itemset mining. Various aspects of semantics, such as the significance of items [17,53] or the significance of transactions [19], are expressed as constraints. Five constraint based approaches are Convertible Constraints [70], Weighted Items [17,49,53], Value Added Mining [49], High Utility Mining [19], and Itemset Share [7,92]. The main differences among these approaches are: (1) different levels of granularity are used to specify the semantic significance of itemsets, and (2) different pruning strategies are developed according to the identified constraints on the itemsets.

Two important mathematical properties of constraints, namely, the anti-monotone (or monotone) property and the convertible property, have been identified and used by existing constraint based itemset mining approaches [2,3,17,19,53,57,70].

**Definition 4** [69]. A constraint  $C$  is *anti-monotone* iff whenever an itemset  $S$  violates a constraint  $C$ , so does any superset of  $S$ . A constraint  $C$  is *monotone* iff whenever an itemset  $S$  satisfies a constraint  $C$ , so does any superset of  $S$ .

Let the constraint  $C$  be the support constraint  $sup(S) \geq min\_sup$ . If an itemset  $S$  satisfies the Apriori property, by Definition 4, it also satisfies the anti-monotone property. Thus, the Apriori property is a special case of the anti-monotone property.

**Definition 5** [69]. An itemset  $S_1 = i_1 \dots, i_m$  is a *prefix itemset* of itemset  $S_2 = i_1 \dots, i_n$  if the items in  $S_1$  and  $S_2$  are listed in the same order and  $m \leq n$ .

For example, given an itemset  $ABCD$ . By Definition 5, itemsets  $A$ ,  $AB$ , and  $ABC$  are prefix itemsets of  $ABCD$  with respect to the order  $\langle A, B, C, D \rangle$ .

Based on prefix itemsets of an itemset, the convertible property of the itemset is defined as follows.

**Definition 6** [69]. A constraint  $C$  is *convertible anti-monotone* w.r.t. an order  $\mathcal{O}$  on items such that whenever an itemset  $S$  satisfies property  $P$ , so do any prefix itemsets of  $S$ . A constraint  $C$  is *convertible monotone* w.r.t. an  $\mathcal{O}$  on items such that whenever an itemset  $S$  violates property  $P$ , so do any prefix itemsets of  $S$ . A constraint  $C$  is *convertible* w.r.t. an order  $\mathcal{O}$  if whenever it is convertible anti-monotone or convertible monotone w.r.t. the order  $\mathcal{O}$ .

The Convertible Constraints (CC) approach suggested by Pei et al. [70] provided a significant advance in the study of constraint based mining. In this approach, the notation of convertible constraints are introduced. The mathematical properties of convertible constraints are systematically analyzed and characterized. That is, an itemset w.r.t. an order is downward closed in the lattice of all its prefix itemsets defined w.r.t. this order. In other word, if an itemset is convertible w.r.t. an order, then all of its prefix itemsets defined w.r.t. this order do. This closure property has permitted the development of efficient algorithms that traverse only a portion of the itemset lattice, yet find all possible itemsets. Pei et al. [70] explicitly stated that convertible constraints cannot be literally incorporated into an Apriori algorithm. As a result, two new algorithms, namely,  $FIC^A$  algorithm [70] for convertible anti-monotone constraints and  $FIC^M$  algorithm [70] for convertible monotone constraints, are developed.

The Weighted Items (WI) approach [17,53] and the Value Added Mining (VAM) approach [49] capture the semantic significance of itemsets at the item level. Unlike Apriori algorithm treats all items uniformly, both of these approaches assume that

items in a transaction database (columns in the table) have different weights to reflect their importance to the user. For example, the weights may correspond the profitability of different items such as a computer (item A) may be more important than a phone (item B) in terms of profit. Since there is always a decreasing order based on weights of all items, their convertible property can be defined as  $\sum_{i_p \in S} f(i_p)$  for itemset  $S$  w.r.t. the descending order on the weight of items, where  $f(i_p)$  is the weight of the item  $i_p$ . As a result, the semantic of weight is a measure of the importance of an itemsets. Two new algorithms are proposed to find weighted itemsets. In WI approach, the algorithm, called *MINWAL*, is developed. The pruning strategy of *MINWAL* is designed by using convertible constraints w.r.t. the order obtained by sorting the items in decreasing order based on their weights. An exhaustive search algorithm is suggested in the VAM approach, since it fails to identify the convertible property of the constraints.

The High Utility Mining (HUM) approach [19] captures the semantic significance of itemsets at the transaction level. Similar to the WI and VAM approaches, the HUM approach assumes that transactions in a database (rows in the table) have different utility values to reflect their importance to the user. For instance, the same medical treatment for different patients (different transactions) will have different levels of effectiveness. In more detail, considering a simplified dataset on medical treatments for a certain disease shown in Table 2.5, which comes from [19]. Transactions  $t_2$ ,  $t_3$ ,  $t_4$ , and  $t_5$  all use the same treatment and medicine but obtained different effectiveness and side-effects for different patients. As a result, in this example, the utility value of a transaction could be its corresponding values of attribute *Effectiveness* minus values of attribute *Side-effect*. E.g. the utility value of  $t_2$  is  $4 - 2 = 2$ . In fact, the utility values used for each transaction by HUM is the



weight of transactions. Since there is always a decreasing order based on weights of all transactions, their convertible property is defined as  $\sum_{t_q \in T_S} f(t_q)$  for itemset  $S$  w.r.t. the descending order on the weight of transactions, where  $f(t_q)$  is the weight of the transactions  $t_q$ . As a result, the semantic of weight on transactions is a measure of the importance of an itemsets. A pruning strategy is developed in this approach by using a convertible constraint w.r.t. the order obtained by sorting the transactions in decreasing order based on their weights and an algorithm to mine top  $K$  high utility closed itemsets is proposed.

Table 2.5: A medical dataset obtained from an example in [19].

TID	Treatment	Medicine	Effectiveness	Side-effect
$t_1$	1	1	2	4
$t_2$	2	1	4	2
$t_3$	2	1	4	2
$t_4$	2	1	2	3
$t_5$	2	1	1	3
$t_6$	3	1	4	2
$t_7$	3	2	4	2
$t_8$	3	2	1	4

The Itemset Share (IS) approach [7, 92] captures the semantic significance of numerical values that are typically associated with the individual items in a transaction database (cells in the table). The precise impact of the purchase of an itemset can be measured by the item *share*, the fraction of some overall numerical value, such as the total quantity of items sold. For example, five computers sold in one transaction may be considered to be more important than only two computers sold in another transactions. Thus, the domain of the table can be explicit quantities, such as the number of items sold as shown in Table 2.1, rather than the binary domain  $\{0, 1\}$ , where 1 indicates the presence of an item in a transaction, and 0 in-

dicates its absence. Five algorithms are developed in [7] to find itemsets with share values above the minimum share threshold. The *Zero Pruning* (ZP) algorithm is defined as the removal of any itemset  $S$  for which  $|T_S| = 0$ , before candidate itemset generation is performed. The *Zero Subset Pruning* (ZSP) algorithm is defined to prevent a generated itemset from having a subset that was found to have a zero transaction count. The *Share Infrequency Pruning* (SIP) algorithm is defined as the removal of any itemset  $S$  in the candidate set  $C_{k-1}$  whose actual share of  $S$  is less than the share threshold before candidate set  $C_k$  is generated. The *Combine All Counted* (CAC) algorithm defined as to allow all information, which is collected in the  $(k - 1)$  pass to generate  $k$ -itemsets, to add to  $C_k$  by delaying the infrequency pruning of the itemsets in  $C_{k-1}$ , until after the itemsets in  $C_k$  have been generated. The *Item Add-Back* (IAB) algorithm is defined as each single item found in the first pass is added to the share of itemset from the  $(k - 1)$  pass in the  $k$ th pass. More detail of these algorithms is referred to [7]. Among these five algorithms, ZP and ZSP are deterministic algorithms. The ZIP, SIP, and IAB are heuristic algorithms that may fail to find some high share itemsets.

## 2.2 Theoretical Foundation

In this section, we present the theoretical foundation for our utility based itemset mining approach. We begin by introducing some formal definitions of key terms.

### 2.2.1 Definitions

We denote the utility value of itemset  $S$  as  $u(S)$ , which will be described in more detail shortly.

**Definition 7** The *utility constraint* is a constraint of the form  $u(S) \geq \text{minutil}$ .

**Definition 8** An itemset  $S$  is a *high utility itemset* if  $u(S) \geq \text{minutil}$ , where *minutil* is the threshold defined by the user. Otherwise,  $S$  is a *low utility itemset*.

Based on the utility constraint, the utility based itemset mining problem is defined as follows.

**Definition 9** The *utility based itemset mining problem* is to discover the set  $H$  of all high utility itemsets, i.e.,

$$H = \{S \mid S \subseteq I, u(S) \geq \text{minutil}\}. \quad (2.1)$$

For example, consider the itemsets in Table 2.4. If  $u(S)$  is the profit of an itemset  $S$  and  $\text{minutil} = 150$ , then  $H = \{B, C, AC, BD, CD, ACD\}$ .

According to Definition 9,  $u(S)$  plays a key role in specifying utility based itemset mining problems. Next, we show how to define  $u(S)$  in terms of a user defined utility function  $f$ . In Example 2, the profit of an itemset reflects a store manager's goal of discovering itemsets producing significant profit (e.g.,  $\text{minutil} = 150$ ). A user judges  $BD$  to be useful, since the profit of itemset  $BD$  is greater than  $\text{minutil}$ . Here, we observe that the semantic significance of profit can be captured by a function  $f(x, y)$ , where  $x$  is the quantity sold of an item and  $y$  is the unit profit of an item. The usefulness of an itemset is quantified as the product of  $x$  and  $y$ , namely,  $f(x, y) = x \cdot y$ . The value of  $x$  can be obtained from the transaction database and depends only on the underlying database used in the data mining process [80]. On the other hand,  $y$  is often not available in a transaction database and may depend on the user who

examines the itemset [80]. Thus, in this case, the significance of an item is measured by two parts. One is the statistical significance of the item measured by parameter  $x$ , which is an objective term independent of its intended application. The other part is the semantic significance of the item measured by parameter  $y$ , which is a subjective term dependent on the application and the user. As a result,  $f(x, y)$  combines objective and subjective measures of an item together. The combination captures the significance of the itemset for this application, which reflects not only the statistical significance but also the semantic significance of the itemset. To define  $f(x, y)$  as such a utility function for utility based itemset mining, we start by defining the parameters  $x$  and  $y$ .

**Definition 10** The *objective value of an item*, denoted  $x_{pq}$ , is the value of an attribute associated with an item  $i_p$  in a transaction  $t_q$ .

For example, in Table 2.1, the quantity sold values in the transactions are the objective values. If  $i_4 = D$ , then  $x_{43} = 30$  is the objective value of item  $D$  in transaction  $t_3$ .

**Definition 11** The *subjective value of an item*, denoted  $y_p$ , is a real number assigned by the user such that for any two items  $i_p$  and  $i_q$ ,  $y_p$  is greater than  $y_q$  iff the user prefers item  $i_p$  to item  $i_q$ .

The definition indicates that a subjective value is associated with a specific value in a domain to express user preference. In practice, the value of  $y_p$  is assigned by the user according to his interpretation of domain specific knowledge measured by a utility such as cost, profit, or aesthetic value. For example, let  $i_1 = A$  and  $i_2 = B$ . Using the Table 2.3, we have  $y_1 = 5$  and  $y_2 = 100$ . The inequality  $y_2 > y_1$  reveals

that the store manager prefers item  $B$  to item  $A$ , since each item  $B$  earns more profit than each item  $A$ .

By obtaining the objective value  $x_{pq}$  from a transaction database and the subjective value  $y_p$  from the user, a utility function to express the significance of an itemset can be defined as a two dimensional function  $f(x, y)$ . We restrict attention to nonnegative utility functions

**Definition 12** A *nonnegative utility function*  $f$  is a function  $f(x, y) : (R, R) \rightarrow R^+$ , where  $R$  is the set of real numbers, and  $R^+$  is the set of nonnegative real numbers.

Many utility functions of interest are nonnegative and other functions can be transformed into nonnegative utility functions. Nonnegative utility functions can be monotone, non monotone, convertible, or inconvertible. A function  $f_1(x, y)$  with range  $[-n, m]$ , where  $n, m \geq 0$ , can be transformed to a nonnegative function by adding  $n$  to all values, i.e.,  $f'_1(x, y) = f_1(x, y) + n$ . Also a nonpositive function  $f_2(x, y) \leq 0$  can be transformed to its absolute value, namely  $|f_2(x, y)|$ . Thus, all results obtained for nonnegative utility function in this chapter can also be applied to function  $f_1$  or  $f_2$ . We acknowledge that this transformation will reduce the accuracy of the transformed function  $f'_1$  in comparison with the original utility function  $f_1$  for expressing user preferences. However, this approach ensures that a one to one correspondence between these two utility functions exists. The transformed function  $f'_1$  also ensures that if a user prefers item  $A$  to item  $B$ , then the utility value of  $A$  is higher than  $B$ .

Nonnegative utility functions are useful in practice. For example, the *TF-IDF* function, which is widely used in information retrieval to score similarity between a query and a document [73], can be used as a utility function. The *TF-IDF* function

is defined as  $f(tf_{ij}, idf_i) = tf_{ij} * idf_i$ , where  $tf_{ij}$  denotes the frequency of a term  $i$  appearing in a document  $j$ , and  $idf_i$  is the the inverse document frequency for term  $i$  such that  $idf_i = \log \frac{M}{df_i}$ , where  $M$  is the number of documents in the dataset and  $df_i$  is the number of documents containing item  $i$ .

**Example 3** Consider the transaction database in Table 2.1 and the corresponding profit in Table 2.3. Let items  $i_1, i_2, i_3$ , and  $i_4$  be items  $A, B, C$ , and  $D$ , respectively. Suppose that the user defines utility function  $f(x_{pq}, y_p)$  as  $f(x_{pq}, y_p) = x_{pq} \cdot y_p$ , where  $x_{pq}$  is the quantity sold of an item  $i_p$  in transaction  $t_q$ , and  $y_p$  is the unit price of the item  $i_p$ . Then  $f(x_{11}, y_1) = 4 \times 5 = 20$ , which indicates that the supermarket earns \$20 by selling four of item  $A$  in transaction  $t_1$ . Similarly,  $f(x_{21}, y_2) = 0$ ,  $f(x_{31}, y_3) = 1 \times 38 = 38$ , and  $f(x_{41}, y_4) = 0$ .

In this thesis, we consider a generalized additive independence model [6, 13], which is a natural but flexible and fully expressive generalization of an additive utility model. That is, we assume that the utility value of an item is the sum of the values of the utility function for each transaction. We also assume that the utility value of an itemset is represented by the sum of the utility values of every item in the itemset.

**Definition 13** The *utility value of an item  $i_p$  in an itemset  $S$* , denoted  $l(i_p, S)$ , is the sum of the values of the utility function  $f(x_{pq}, y_p)$  for each transaction  $t_q$  in  $T_S$ , i.e.,

$$l(i_p, S) = \sum_{t_q \in T_S} f(x_{pq}, y_p). \quad (2.2)$$

For example, consider the transaction database in Table 2.1 with the corresponding profit in Table 2.3. Let  $S = ACD$ ,  $T_S = \{t_6, t_8\}$ , thus  $l(A, S) = 4 \times 5 + 1 \times 5 = 25$ .

**Definition 14** The *utility value of an itemset  $S$* , denoted  $u(S)$ , is the sum of the utility value of each item in  $S$ , i.e.,

$$u(S) = \sum_{i_p \in S} l(i_p, S). \quad (2.3)$$

By substituting Equation (2.2) into Equation (2.3), we obtain

$$u(S) = \sum_{i_p \in S} \sum_{t_q \in T_S} f(x_{pq}, y_p). \quad (2.4)$$

For example, given  $f(x_{pq}, y_p) = x_{pq} \cdot y_p$ , for itemset  $S = ACD$ , we have  $T_S = \{t_6, t_8\}$ , then  $u(S) = l(A, S) + l(C, S) + l(D, S) = 5 \times 5 + 3 \times 38 + 11 \times 1 = 150$ .

Equation (2.4) indicates that user plays an important role in utility based itemset mining process since a user can measure the semantic significance of the itemset by using his own utility function  $f(x, y)$ . Therefore, an itemset that is of interest to one user, may be of no interest to another user, since users have different levels of interest in itemsets, as expressed by their utility functions. In other words, different itemsets may be discovered for two users according to their interests, as expressed by their utility functions.

As previously mentioned, the two main pruning strategies used in itemset mining are based on the Apriori property for frequent itemset mining [2, 3, 57] and the convertible property for convertible constraint based itemset mining [17, 19, 53, 70]. Thus, it is worthwhile determining whether these two pruning strategies can be applied to the utility based itemset mining.

The following example shows that the utility constraint may be neither anti monotone nor monotone.

**Example 4** Consider the itemsets shown in Table 2.4. Let  $u(S)$  be the profit of an itemset  $S$ . We have  $u(AD) = 135$ ,  $u(ABD) = 106$ , and  $u(ACD) = 150$ . Since  $u(AD) > u(ABD)$  and  $u(AD) < u(ACD)$ , the profit of a superset of  $AD$  can be lower or higher than the profit of  $AD$ .

The next following example shows that a utility constraint is not convertible w.r.t. a decreasing or an increasing order on the profit of each item.

**Example 5** Consider the itemsets shown in Table 2.4. Let  $u(S)$  be the profit of an itemset  $S$ . In Table 3,  $u(A) = 110$ ,  $u(B) = 200$ ,  $u(C) = 190$ , and  $u(D) = 85$ . Thus, the profit decreasing order is  $\langle B, C, A, D \rangle$ . Itemsets  $B$  and  $BC$  are prefix itemsets of  $BCAD$ . Since  $u(B) = 200$ ,  $u(BC) = 138$ , and  $u(BCAD) = 144$ , then  $u(B) > u(BCAD)$  and  $u(BC) < u(BCAD)$ , which indicates that the profit of prefix itemsets of  $BCAD$  can be higher or lower than the profit of  $BCAD$ . Similarly, the profit increasing order is  $\langle D, A, C, B \rangle$ . The itemsets  $DAC$  and  $DA$  are prefix itemsets of  $DACB$ . Since  $u(DAC) = 150$  and  $u(DA) = 135$ , then  $u(DA) < u(DACB)$  and  $u(DAC) > u(DACB)$ , which indicates that the profit of the prefix itemsets of  $DACB$  can be higher or lower than the profit of  $DACB$ . Since neither decreasing nor increasing orders can guarantee that the profit of itemset  $ABCD$  is always higher (or lower) than the profit of any prefix itemsets, we conclude that  $u(S)$  is not convertible w.r.t. a decreasing or an increasing order.

Theorem 2 shows that the pruning strategies used in existing approaches for frequent itemset mining and convertible constraint based itemset mining cannot be



applied to utility based itemset mining.

**Theorem 2** *A utility constraint  $u(S) \geq \text{minutil}$  is not necessarily anti monotone, monotone, or convertible w.r.t. a decreasing or increasing order.*

*Proof:* The counterexamples are shown in Examples 4 and 5.  $\square$

## 2.2.2 Mathematical Properties of the Utility Constraint

In this subsection, we analyze the mathematical properties of utility constraints. Furthermore, a theorem, which provides the theoretical foundation for the pruning strategies proposed in next subsection, is formally established.

**Definition 15** *A  $k$ -itemset, denoted as  $S^k$ , is an itemset of  $k$  distinct items.*

**Definition 16** *The set of all  $(k - 1)$ -itemsets of  $S^k$ , denoted  $L^{k-1}$ , is the set  $\{S^{k-1} \mid S^{k-1} \subset S^k\}$*

**Definition 17** *The set of  $(k - 1)$ -itemsets including item  $i_p$ , denoted  $L_{i_p}^{k-1}$ , is the set  $\{S^{k-1} \mid i_p \in S^{k-1}, S^{k-1} \in L^{k-1}\}$ .*

**Example 6** *Let  $S^4$  be the 4-itemset ABCD. Then, by Definition 16, we have  $L^3 = \{ACD, ABD, ABC, BCD\}$ . Since  $A \notin BCD$ , by Definition 17,  $L_A^3 = \{ACD, ABD, ABC\}$ .*

**Lemma 1** *Let  $\min_{S^{k-1} \in L_{i_p}^{k-1}} \{l(i_p, S^{k-1})\}$  be the minimum of the utility value of item  $i_p$  in any itemset of  $L_{i_p}^{k-1}$ . Then the following property holds.*

$$l(i_p, S^k) \leq \min_{S^{k-1} \in L_{i_p}^{k-1}} \{l(i_p, S^{k-1})\} \leq \frac{\sum_{S^{k-1} \in L_{i_p}^{k-1}} l(i_p, S^{k-1})}{k - 1} \quad (2.5)$$

*Proof:* By Equation (2.2), for any  $S^{k-1} \in L_{i_p}^{k-1}$ , we have Equation (2.6).

$$l(i_p, S^{k-1}) = \sum_{t_q \in T_{S^{k-1}}} f(x_{pq}, y_p) \quad (2.6)$$

Since  $S^{k-1} \subset S^k$ , then for each  $t_q \in T_{S^k}$ , by Definition 1,  $t_q$  must satisfy  $t_q \in T_{S^{k-1}}$ .

Namely,  $T_{S^k} \subseteq T_{S^{k-1}}$ . Thus, Equation (2.6) can be rewritten as follows.

$$\begin{aligned} l(i_p, S^{k-1}) &= \sum_{t_q \in T_{S^k}} f(x_{pq}, y_p) + \sum_{t_q \in (T_{S^{k-1}} - T_{S^k})} f(x_{pq}, y_p) \\ &= l(i_p, S^k) + \sum_{t_q \in (T_{S^{k-1}} - T_{S^k})} f(x_{pq}, y_p) \end{aligned} \quad (2.7)$$

By Equation (2.7), we have Equation (2.8).

$$l(i_p, S^{k-1}) - l(i_p, S^k) = \sum_{t_q \in (T_{S^{k-1}} - T_{S^k})} f(x_{pq}, y_p) \quad (2.8)$$

By Definition 12,  $f(x_{pq}, y_p) \geq 0$  for all  $t_q \in T_{S^k}$ , so  $\sum_{t_q \in (T_{S^{k-1}} - T_{S^k})} f(x_{pq}, y_p) \geq 0$ .

Thus, by Equation (2.8), we have

$$l(i_p, S^{k-1}) - l(i_p, S^k) \geq 0, \quad (2.9)$$

which satisfies  $l(i_p, S^k) \leq l(i_p, S^{k-1})$  for any  $S^{k-1} \in L_{i_p}^{k-1}$ . Thus, the first inequality holds.

Since the number of  $(k-1)$ -itemsets of any  $k$ -itemset is  $k$ , thus the number of  $S^{k-1}$  in  $L^{k-1}$  is  $k$ . Since for all  $S^{k-1}$  in  $L^{k-1}$ , there is only one  $S^{k-1}$  that satisfies  $i_p \notin S^{k-1}$ , it follows that the number of  $S^{k-1}$  in  $L_{i_p}^{k-1}$  is  $k-1$ . Thus, the following

$(k - 1)$  inequalities hold for all  $S^{k-1}$  in  $L_{i_p}^{k-1}$ .

$$\min_{S^{k-1} \in L_{i_p}^{k-1}} \{l(i_p, S^{k-1})\} \leq l(i_p, S^{k-1})$$

By adding all these  $(k - 1)$  inequalities together, we have

$$(k - 1) \cdot \min_{S^{k-1} \in L_{i_p}^{k-1}} \{l(i_p, S^{k-1})\} \leq \sum_{S^{k-1} \in L_{i_p}^{k-1}} l(i_p, S^{k-1}) \quad (2.10)$$

By dividing by  $(k - 1)$  on both sides of Inequality (2.10), we obtain the second inequality in Inequality (2.5).  $\square$

**Example 7** Consider itemset  $I = \{A, B, C, D\}$  in 2.1 with the corresponding profit in Table 2.3. Suppose  $S^3 = ACD$ , and  $S^2 = AD$ . By Definition 1,  $T_{S^3} = \{t_6, t_8\}$ , and  $T_{S^2} = \{t_2, t_4, t_5, t_6, t_7, t_8, t_{10}\}$ . Thus,  $T_{S^3} \subseteq T_{S^2}$ . By Definition 13,  $l(A, ACD) = (4 + 1) \times 5 = 25$ ,  $l(A, AD) = (2 + 3 + 1 + 4 + 2 + 1 + 5) \times 5 = 18 \times 5 = 90$ . We have  $l(A, ACD) \leq l(A, AD)$ . By Definition 17, we have  $L_A^3 = \{ACD, ABD, ABC\}$ . By Lemma 1, we have

$$\begin{aligned} l(A, ABCD) &\leq \min\{l(A, ABC), l(A, ABD), l(A, ACD)\} \\ &\leq \frac{l(A, ABC) + l(A, ABD) + l(A, ACD)}{3} \end{aligned}$$

Lemma 1 indicates that the utility value of an item  $i_p$  in a  $k$ -itemset  $S^k$  is limited by the utility value of the item  $i_p$  in all  $(k - 1)$ -itemsets of  $S^k$ . The reason is that the utility value of an item  $i_p$  in  $S$  must be less than the utility value of  $i_p$  in any

subset of  $S$  that contains  $i_p$ . In other words, the utility value of any single  $i_p$  in  $S$  monotonically decreases as the size of  $S$  increases. Thus, by considering all  $k$  items in  $S^k$ , the upper bound of the utility value of the itemset  $S^k$  can be obtained as follows.

**Theorem 3** (Utility Upper Bound Property). *Let  $u(S^k)$  be the utility value of a  $k$ -itemset  $S^k$ . Then the following property holds.*

$$u(S^k) \leq \frac{\sum_{S^{k-1} \in L^{k-1}} u(S^{k-1})}{k-1} \quad (2.11)$$

*Proof:* By Lemma 1,

$$l(i_p, S^k) \leq \frac{\sum_{S^{k-1} \in L_{i_p}^{k-1}} l(i_p, S^{k-1})}{k-1} \quad (2.12)$$

holds for each  $i_p \in S^k$ . Since itemset  $S^k$  contains  $k$  items, by adding the  $k$  inequalities for all  $i_p \in S^k$ , we have the following.

$$\sum_{i_p \in S^k} l(i_p, S^k) \leq \sum_{i_p \in S^k} \frac{\sum_{S^{k-1} \in L_{i_p}^{k-1}} l(i_p, S^{k-1})}{k-1} \quad (2.13)$$

By Definition 13, the left side of Inequality (2.13) satisfies

$$u(S^k) = \sum_{i_p \in S^k} l(i_p, S^k) \quad (2.14)$$

The right side of Inequality (2.13) can be rewritten as

$$\begin{aligned}
& \sum_{i_p \in S^k} \frac{\sum_{S^{k-1} \in L_{i_p}^{k-1}} l(i_p, S^{k-1})}{k-1} \\
&= \frac{\sum_{i_p \in S^k} \sum_{S^{k-1} \in L_{i_p}^{k-1}} l(i_p, S^{k-1})}{k-1}
\end{aligned} \tag{2.15}$$

For a given  $i_p$ , there is only one  $S^{k-1}$  in  $L^{k-1}$  that satisfies  $i_p \notin S^{k-1}$ . Let  $S'$  be this  $(k-1)$ -itemset. Thus, Equation (2.15) can be rewritten as follows.

$$\begin{aligned}
& \sum_{i_p \in S^k} \frac{\sum_{S^{k-1} \in L_{i_p}^{k-1}} l(i_p, S^{k-1})}{k-1} \\
&= \frac{\sum_{i_p \in S^k} \sum_{S^{k-1} \in (L^{k-1} - \{S'\})} l(i_p, S^{k-1})}{k-1} \\
&= \frac{\sum_{i_p \in S^k} (\sum_{S^{k-1} \in L^{k-1}} l(i_p, S^{k-1}) - l(i_p, S'))}{k-1}
\end{aligned} \tag{2.16}$$

Since  $i_p \notin S'$ , by Definition 1,  $T_{S'} = 0$ . By Definition 13,  $l(i_p, S') = 0$ . Thus, Equation (2.16) can be rewritten as follows.

$$\begin{aligned}
& \sum_{i_p \in S^k} \frac{\sum_{S^{k-1} \in L_{i_p}^{k-1}} l(i_p, S^{k-1})}{k-1} \\
&= \frac{\sum_{i_p \in S^k} \sum_{S^{k-1} \in L^{k-1}} l(i_p, S^{k-1})}{k-1} \\
&= \frac{\sum_{i_p \in (S^{k-1} \cup (S^k - S^{k-1}))} \sum_{S^{k-1} \in L^{k-1}} l(i_p, S^{k-1})}{k-1} \\
&= \frac{\sum_{i_p \in S^{k-1}} \sum_{S^{k-1} \in L^{k-1}} l(i_p, S^{k-1}) + \sum_{i_q \in (S^k - S^{k-1})} \sum_{S^{k-1} \in L^{k-1}} l(i_q, S^{k-1})}{k-1}
\end{aligned} \tag{2.17}$$

For  $\sum_{i_q \in (S^k - S^{k-1})} \sum_{S^{k-1} \in L^{k-1}} l(i_q, S^{k-1})$  in Equation (2.17), since  $i_q \in (S^k - S^{k-1})$ , then  $i_q \notin S^{k-1}$ . Thus, by Definition 1 and 13,  $l(i_q, S^{k-1}) = 0$ . As a result, Equation (2.17) can be rewritten as follows.

$$\begin{aligned}
& \sum_{i_p \in S^k} \frac{\sum_{S^{k-1} \in L_{i_p}^{k-1}} l(i_p, S^{k-1})}{k-1} \\
= & \frac{\sum_{i_p \in S^{k-1}} \sum_{S^{k-1} \in L^{k-1}} l(i_p, S^{k-1}) + \sum_{i_q \in (S^k - S^{k-1})} \sum_{S^{k-1} \in L^{k-1}} 0}{k-1} \\
= & \frac{\sum_{i_p \in S^{k-1}} \sum_{S^{k-1} \in L^{k-1}} l(i_p, S^{k-1})}{k-1} \\
= & \frac{\sum_{S^{k-1} \in L^{k-1}} \sum_{i_p \in S^{k-1}} l(i_p, S^{k-1})}{k-1} \\
= & \frac{\sum_{S^{k-1} \in L^{k-1}} u(S^{k-1})}{k-1} \quad // \text{by Definition 13} \tag{2.18}
\end{aligned}$$

Therefore, by substituting Equation (2.14) and Equation (2.18) into Inequality (2.13), we obtain Inequality (2.19).

$$u(S^k) \leq \frac{\sum_{S^{k-1} \in L^{k-1}} u(S^{k-1})}{k-1}, \tag{2.19}$$

which is the same as Inequality (2.11).  $\square$

**Example 8** For a 4-itemset  $S^4 = ABCD$ , by Definition 16, we obtain  $L^3 = \{ACD, ABD, ABC, BCD\}$ . Thus, by Theorem 3, we have

$$u(ABCD) \leq \frac{u(ABC) + u(ACD) + u(ABD) + u(BCD)}{3}.$$

It is important to realize that Theorem 3 indicates that the utility value of itemset  $S^k$  is limited by the utilities of all its subset itemsets of size  $(k - 1)$ .

Based on the utility upper bound property, two pruning strategies are provided in next section.

## 2.3 Two Efficient Pruning Strategies

In this section, we design two efficient pruning strategies to mine high utility itemsets using the utility upper bound property.

**Definition 18** The *candidate itemsets* of  $S^k$ , denoted  $C^{k-1}$ , is a set of some  $(k - 1)$ -subsets of  $S^k$ , i.e.,  $C^{k-1} \subseteq L^{k-1}$ .

**Definition 19** The *utility upper bound* of  $S^k$ , denoted  $b(S^k)$ , is defined as follows.

$$b(S^k) = \left\lceil \frac{\sum_{S^{k-1} \in C^{k-1}} u(S^{k-1})}{|C^{k-1}| - 1} \right\rceil, \quad (2.20)$$

where  $|C^{k-1}|$  is the cardinality of  $C^{k-1}$ .

**Theorem 4 (Pruning Strategy 1).** *If every  $(k - 1)$ -itemset  $S^{k-1} \in (L^{k-1} - C^{k-1})$  is a low utility itemset and  $b(S^k) < \text{minutil}$ , then  $k$ -itemset  $S^k$  satisfies  $u(S^k) < \text{minutil}$ .*

*Proof:* Since each  $S^{k-1} \in (L^{k-1} - C^{k-1})$  is a low utility itemset, then

$$\sum_{S^{k-1} \in (L^{k-1} - C^{k-1})} u(S^{k-1}) < \sum_{S^{k-1} \in (L^{k-1} - C^{k-1})} \text{minutil} \quad (2.21)$$

By Theorem 3, we have

$$\begin{aligned}
u(S^k) &\leq \frac{\sum_{S^{k-1} \in L^{k-1}} u(S^{k-1})}{k-1} \\
&= \frac{\sum_{S^{k-1} \in (C^{k-1} \cup (L^{k-1} - C^{k-1}))} u(S^{k-1})}{k-1} \\
&= \frac{\sum_{S^{k-1} \in C^{k-1}} u(S^{k-1}) + \sum_{S^{k-1} \in (L^{k-1} - C^{k-1})} u(S^{k-1})}{k-1} \\
&\leq \frac{\sum_{S^{k-1} \in C^{k-1}} u(S^{k-1}) + \sum_{S^{k-1} \in (L^{k-1} - C^{k-1})} \cdot \text{minutil}}{k-1} \quad // \text{by Equation (2.21)} \\
&\leq \frac{\sum_{S^{k-1} \in C^{k-1}} u(S^{k-1}) + |L^{k-1} - C^{k-1}| \cdot \text{minutil}}{k-1} \\
&\leq \frac{(|C^{k-1}| - 1) \cdot b(S^k) + |L^{k-1} - C^{k-1}| \cdot \text{minutil}}{k-1} \quad // \text{by Equation (2.20)} \\
&= \frac{(|C^{k-1}| - 1) \cdot b(S^k) + (k - |C^{k-1}|) \cdot \text{minutil}}{k-1} \\
&\leq \frac{(|C^{k-1}| - 1) \cdot \text{minutil} + (k - |C^{k-1}|) \cdot \text{minutil}}{k-1} \quad // \text{by } b(S^k) < \text{minutil} \\
&\leq \frac{(k-1) \cdot \text{minutil}}{k-1} \\
&= \text{minutil}
\end{aligned}$$

Thus, we have  $u(S^k) < \text{minutil}$ .  $\square$

**Example 9** For the 3-itemset  $S^3 = BCD$  in Table 2.4, we have

$$b(BCD) = \frac{u(BC) + u(BD) + u(CD)}{3-1} = \frac{138 + 211 + 193}{2} = 271.$$

This result indicates that the utility value of  $BCD$  cannot be greater than 271, based on the known utility values of itemsets  $BC$ ,  $BD$ , and  $CD$ . The actual utility value of  $BCD$ , which is shown in Table 2.4, is  $u(BCD) = 139$ .



Theorem 4 indicates that we can determine whether or not a  $k$ -itemset is a low utility itemset from the utility values of its high utility  $(k - 1)$ -size itemsets. Therefore, a level-wise method [59] can be used to prune the low utility  $k$ -itemset  $S^k$  if the utility upper bound  $b(S^k)$  calculated by Equation (2.20) is less than the *minutil* threshold.

The difference between Theorem 3 and Theorem 4 is that Theorem 3 needs to consider *all* the  $(k - 1)$ -size itemsets of  $S^k$ , whereas Theorem 4 does not. More precisely, no  $(k - 1)$ -size itemsets can be pruned in Theorem 3. However, Theorem 4 only considers some of the  $(k - 1)$ -size itemsets of  $S$ , i.e., the candidate itemsets  $C^{k-1}$ . The other  $(k - 1)$ -size itemsets of  $S$  in  $L^{k-1} - C^{k-1}$  have already been pruned as low utility itemsets at some previous levels.

Next, we will introduce a heuristic pruning strategy to further reduce the utility upper bound of an itemset.

Although the utility upper bound property limits the utility values of the  $k$ -itemset  $S^k$  to the sum of the utility value of candidate itemsets of size  $(k - 1)$ , the utility upper bound may be further reduced by considering their support. As the size of an itemset increases, its support decreases.

**Theorem 5** (Apriori Property [2]). *Let  $s(S^k)$  be the support of itemset  $S^k$ . The following property holds.*

$$s(S^k) \leq \min_{S^{k-1} \in L^{k-1}} \{s(S^{k-1})\} \quad (2.22)$$

Theorem 5 indicates that the support of  $S^k$  is at most the minimum of the support of any itemset of size  $(k - 1)$  of  $S^k$ .

For example, for the 4-itemset  $S^4 = ABCD$  in Table 2.1, we have

$$s(ABCD) \leq \min\{s(ABC), s(ABD), s(ACD), s(BCD)\}.$$

By combining the utility upper bound property of an itemset with the Apriori property, a heuristic pruning strategy is designed as follows.

**Definition 20** Let  $S^k$  be a  $k$ -itemset, and let  $C^{k-1}$  be a set of candidate itemsets of  $S^k$ . The *expected utility upper bound* of  $S^k$ , denoted  $b'(S^k)$ , is defined as follows.

$$b'(S^k) = \left[ \frac{s_{min}}{|C^{k-1}| - 1} \sum_{S^{k-1} \in C^{k-1}} \frac{u(S^{k-1})}{s(S^{k-1})} \right] \quad (2.23)$$

where

$$s_{min} = \min_{S^{k-1} \in C^{k-1}} \{s(S^{k-1})\}. \quad (2.24)$$

**Pruning Strategy 2.** Let  $b'(S^k)$  be the expected utility upper bound of  $S^k$ . If  $b'(S^k) < minutil$ , then itemset  $S^k$  can be pruned as a low utility itemset.

**Example 10** Consider the 3-itemset  $S^3 = BCD$  in Table 2.4.

$$s_{min} = \min\{s(BC), s(BD), s(CD)\} = \min\{0.1, 0.2, 0.3\} = 0.1.$$

Thus,

$$\begin{aligned}
 b'(BCD) &= \frac{s_{min}}{3-1} \times \left[ \frac{u(BC)}{s(BC)} + \frac{u(BD)}{s(BD)} + \frac{u(CD)}{s(CD)} \right] \\
 &= \frac{0.1}{2} \times \left[ \frac{138}{0.1} + \frac{211}{0.2} + \frac{193}{0.3} \right] = 153.91
 \end{aligned}$$

The estimated utility upper bound of itemset  $BCD$  is only 153.91, as compared to the utility upper bound of 271, which was determined in Example 9.

The reason that Pruning Strategy 2 is a heuristic strategy is that when the size of the itemset is increased, more items are included, but the support for the itemset may be decreased due to fewer transactions being included. It is possible that  $u(S) \geq \text{minutil}$  when  $b'(S) < \text{minutil}$ . As a result, some high utility itemsets may be erroneously pruned by the heuristic strategy.

**Example 11** Using Table 2.6, we have  $u(A) = 109$ ,  $u(B) = 1$ ,  $s(A) = 100\%$ ,  $s(B) = 10\%$ , and  $u(AB) = 101$ . By Definition 20,  $b'(AB) = 0.1 * (109/1 + 1/0.10) = 11.9$ , which is less than the actual  $u(AB)$ . As a result, if  $\text{minutil} \geq 12$ , then the high utility itemset  $AB$  is erroneously pruned.

This example indicates that erroneous pruning may happen when a high utility itemset  $S$  has a low utility subset with a support that is much lower than the support of other subsets of  $S$ .

Therefore, it is important that the proposed heuristic method provides a reasonable balance between accuracy and efficiency. In Section 2.5, a series of experiments are reported that provide evidence that this balance has been achieved.

Table 2.6: A transaction database.

Transaction ID	Item A	Item B
$t_1$	100	1
$t_2$	1	0
$t_3$	1	0
$t_4$	1	0
$t_5$	1	0
$t_6$	1	0
$t_7$	1	0
$t_8$	1	0
$t_9$	1	0
$t_{10}$	1	0

## 2.4 Algorithms for Utility Based Itemset Mining

In this section, we describe how to mine high utility itemsets efficiently by using pruning strategies. The main idea is to incorporate the pruning strategies given in Section 2.3 into the mining process, thereby pruning the search space.

Recall that in the two pruning strategies described in Section 2.3, both Equation (2.20) and Equation (2.23) use the utility values of  $(k - 1)$ -itemsets to predict the utility upper bound of a  $k$ -itemset. Thus, the utility values of the  $(k - 1)$ -itemsets at level  $k - 1$  can be used to constrain the utility values of the  $k$ -itemsets at level  $k$ .

A level based approach is taken to finding  $H$ , the set of high utility itemsets. At level 1, the utility values of all items (i.e., 1-itemsets) are calculated by Equation (2.4), and the set of all 1-itemsets with their utility values is denoted  $C_1$ . Then the high utility itemsets in  $C_1$  are placed in  $H$ . At level 2,  $C_1$  is used to generate the candidate 2-itemsets for level 2, denoted as  $C_2$ . The utility upper bound of each itemset in  $C_2$  is calculated using Equation (2.20). By pruning the 2-itemsets that

have utility upper bounds less than the threshold, the size of  $C_2$  is reduced. The actual utility values of the remaining 2-itemsets in  $C_2$  are calculated using Equation (2.4), and the high utility itemsets in  $C_2$  are added to  $H$ . Similarly,  $C_2$  is used to generate the candidate 3-itemsets for level 3. And so on until the candidate itemsets at level  $K$  have been checked or no candidate itemsets remain, i.e.,  $C_k = \phi$  for ( $k \leq K - 1$ ), where  $K$  is the maximum size of an itemset of interest. The purpose of  $K$  is to provide the user with an opportunity to control the algorithm. The running time for the algorithm may be reduced by setting  $K$  to a low value. If  $K$  is set high enough, then the algorithm runs until no more itemsets can be found. In general,  $C_k$  is the set of candidate  $k$ -itemsets for the  $k^{th}$  pass. After the  $k^{th}$  pass, information is available about the utility value of each high utility  $k$ -itemset. Based on the above reasoning, two algorithms for utility based itemset mining are presented.

### 2.4.1 The UMining Algorithm

We first develop an algorithm, called UMining, for mining all high utility itemsets using pruning strategy 1, which is guaranteed to never prune a high utility itemset. The framework of the UMining algorithm is shown in Figure 2.3.

**Algorithm UMining**( $T, f, \text{minutil}, K$ )Input: Transaction database  $T$ , Utility function  $f$ ,Utility value threshold  $\text{minutil}$ , Maximum size of itemset  $K$ .Output: A set of high utility itemsets  $H$ .

1.  $I = \text{Scan}(T)$ ;
2.  $C_1 = I$ ;
3.  $k = 1$ ;
4.  $C_k = \text{CalculateAndStore}(C_k, T, f)$ ;
5.  $H = \text{Discover}(C_k, \text{minutil})$ ;
6. **while** ( $|C_k| > 0$  and  $k \leq K$ )
7. {
8.      $k = k + 1$ ;
9.      $C_k = \text{Generate}(C_{k-1}, I)$ ;
10.      $C_k = \text{Prune}(C_k, C_{k-1}, \text{minutil})$ ;
11.      $C_k = \text{CalculateAndStore}(C_k, T, f)$ ;
12.      $H = H \cup \text{Discover}(C_k, \text{minutil})$ ;
13. }
14. **return**  $H$ ;

Figure 2.3: The UMining algorithm.

The functions called by the UMining algorithm are *Scan*, *CalculateAndStore*, *Discover*, *Generate*, and *Prune*, which are given in Figures 2.4 to 2.8.

The *Scan* function finds the set of all items in the transaction database  $T$ .

**Scan**( $T$ )

1.  $I = \phi$ ;
2. **for** each item  $i_p$  in  $T$
3.      $I = I \cup \{i_p\}$ ;
4. **return**  $I$ ;

Figure 2.4: The *Scan* function for the UMining algorithm.

The *CalculateAndStore* function accesses transaction database  $T$  to calculate the actual utility value of each  $k$ -itemset in  $C_k$  by Equation (2.4). It is assumed that each itemset  $S$  in  $C_k$  has associated with it a  $u$  field, denoted  $u(S)$ , for storing its

utility value.

**CalculateAndStore**( $C_k, T, f$ )

1. **for** each itemset  $S$  in  $C_k$
2.     obtain  $u(S)$  using  $T_S$  and  $f$  by Equation (2.4);
3. **return** ( $C_k$ );

Figure 2.5: The *CalculateAndStore* function for the UMining algorithm.

The *Discover* function selects all high utility itemsets in candidate set  $C_k$ .

**Discover**( $C_k, minutil$ )

1.  $C' = \phi$ ;
2. **for** each itemset  $S$  in  $C_k$
3.     **if** ( $u(S) \geq minutil$ )
4.          $C' = C' \cup \{S\}$ ;
5. **return** ( $C'$ );

Figure 2.6: The *Discover* function for the UMining algorithm.

The *Generate* function generates all possible candidate  $k$ -itemsets from the  $(k - 1)$ -itemsets in  $C_{k-1}$ .

**Generate**( $C_{k-1}, I$ )

1.  $C_k = \phi$ ;
2. **for** each itemset  $S \in C_{k-1}$
3.     **for** each item  $i_p \in (I - S)$
4.          $C_k = C_k \cup \{S \cup \{i_p\}\}$ ;
5. **return** ( $C_k$ );

Figure 2.7: The *Generate* function for the UMining algorithm.

The *Prune* function calculates the utility upper bound of each itemset in  $C_k$  based on the utility values of the candidate itemsets in  $C_{k-1}$ , and then removes any itemset with a utility upper bound less than *minutil* from  $C_k$ . It is assumed that

each itemset  $S$  has an associated  $b$  field, denoted  $b(S)$ , for storing the utility upper bound of  $S$ .

```

Prune( $C_k, C_{k-1}, minutil$ )
1.  for each itemset  $S$  in  $C_k$ 
2.  {
3.    obtain  $b(S)$  using  $C_{k-1}$  by Equation (2.20);
4.    if ( $b(S) < minutil$ )
5.       $C_k = C_k - \{S\}$ ;
6.  }
7.  return ( $C_k$ );

```

Figure 2.8: The *Prune* function for the UMining algorithm.

The UMining algorithm follows the basic framework of the Apriori algorithm [3], but there are significant differences in three subfunctions (*Prune*, *CalculateAndStore*, and *Generate* functions).

Firstly, for the function *Prune*, a different pruning strategy is used in UMining. Itemset  $S$  is pruned based on its utility upper bound  $b(S)$ . The calculation of  $b(S)$  is encapsulated in Equation (2.20). Secondly, for the function *CalculateAndStore*, the utility value of each candidate itemset in  $C_k$  is calculated using utility function  $f$ . This calculation is encapsulated in Equation (2.4). Finally, for the function *Generate*, since the utility values of itemsets do not satisfy the Apriori property, we cannot generate candidate  $k$ -itemsets simply using the Apriori Gen algorithm [3]. As a result, for each  $(k - 1)$ -itemset in  $C_{k-1}$ , all its supersets of size  $k$  are generated at level  $k$  and then the  $k$ -itemsets that have a utility upper bound less than the threshold are pruned from  $C_k$ .

**Example 12** We use the sample database in Table 2.1 together with the corresponding profit in Table 2.3 to provide an example of how the algorithm works.



We assume  $f(x, y) = x \times y$ ,  $minutil = 196$ , and  $K = 4$ . Figure 2.9(a) shows the complete itemset semi-lattice of candidate itemsets for  $ABCD$  and Figure 2.9(b) shows the portion of the semi-lattice used in the UMining algorithm. Each edge between two nodes indicates that the lower level node contributed to the calculation of the utility upper bound of the upper level node. Each node in the semi-lattice is labelled with an itemset name. Below each itemset name is the utility value. In the figure, the high utility itemsets are shaded. The figure indicates that for this example the high utility itemsets were found with slightly less work than would have been required to examine all possible itemsets.

The UMining algorithm is traced as follows. At level 1,  $I = \{A, B, C, D\}$  is returned by the *Scan* function and assigned to  $C_1$ . Using the *CalculateAndStore* function in line 4, the utility value of each itemset in  $C_1$  is calculated, i.e.,  $u(A) = 22 \times 5 = 110$ ,  $u(B) = 200$ ,  $u(C) = 190$ , and  $u(D) = 85$ . Using the *Discover* function in line 5,  $H = \{B\}$ . At level 2,  $C_2 = \{AB, AC, AD, BC, BD, CD\}$  is returned by the *Generate* function. Using the *Prune* function, the utility upper bound of itemsets in  $C_2$  are calculated as  $b(AB) = 310$ ,  $b(AC) = 300$ ,  $b(AD) = 195$ ,  $b(BC) = 390$ ,  $b(BD) = 285$ , and  $b(CD) = 275$ . Since  $b(AD) < minutil$ ,  $AD$  is removed from  $C_2$ , i.e.,  $C_2 = \{AB, AC, BC, BD, CD\}$ . Next, the utility value of each itemset in  $C_2$  is calculated. i.e.,  $u(AB) = 105$ ,  $u(AC) = 197$ ,  $u(BC) = 138$ ,  $u(BD) = 211$ , and  $u(CD) = 193$ . Using the *Discover* function in line 12, the 2-itemsets  $AC$  and  $BD$  are added to  $H$ . At level 3,  $C_3 = \{ABC, ABD, ACD, BCD\}$  is returned by the *Generate* function. Using the *Prune* function, the utility upper bounds of itemsets in  $C_3$  are  $b(ABC) = 220$ ,  $b(ABD) = 316$ ,  $b(ACD) = 390$ , and  $b(BCD) = 271$ . No itemset is removed from  $C_3$ , since all these utility upper bounds are higher than  $minutil$ . Next, the utility value of each itemset in  $C_3$  is calculated,

i.e.,  $u(ABC) = 143$ ,  $u(ABD) = 106$ ,  $u(ACD) = 150$ , and  $u(BCD) = 139$ . When the *Discover* function is called in line 12, no 3-itemset is added to  $H$ , since all itemsets in  $C_3$  are low utility itemsets. At level 4,  $C_4 = \{ABCD\}$  is obtained by the *Generate* function. Using the *Prune* function, we obtain  $b(ABCD) = 179.3$ , and thus itemset  $ABCD$  is removed from  $C_4$ , i.e.,  $C_4 = \phi$ . At this point, the algorithm terminates.

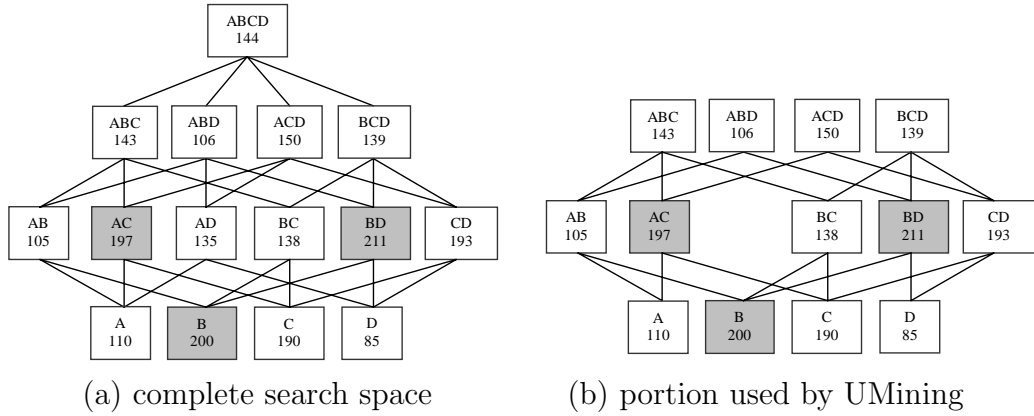


Figure 2.9: Itemset semi-lattice for the UMining algorithm.

The complexity of the UMining algorithm depends on the number of transactions  $n$  in the transaction database  $T$ , the total number  $m$  of distinct items that appear in  $T$ , the complexity of the utility function  $f$ , and the degree of correlation among the items. If more itemsets are identified as low utility itemsets, then more candidates can be pruned, which reduces the running cost of the algorithm. The worst case occurs when all itemsets are high utility itemsets, and thus all combinations of the items are tested. The time required to compute the utility value for an itemset is  $(n \cdot k \cdot C(f))$ , where  $C(f)$  is the complexity of utility function  $f$ . Since the number of combinations of all items is  $C_1^m + C_2^m + \dots + C_m^m = 2^m$ , the worst case time

complexity is  $O(n \cdot k \cdot C(f) \cdot 2^m)$ .

## 2.4.2 The UMining\_H Algorithm

UMining\_H, a heuristic method for mining most high utility itemsets using pruning strategy 2, is presented in this section. The framework for the UMining\_H algorithm is the same as that for the UMining algorithm shown in Figure 2.3, except the *CalculateAndStore* and *Prune* functions are replaced by the *CalculateAndStore\_H* and *Prune\_H* functions shown in Figures 2.10 and 2.11.

**CalculateAndStore\_H**( $C_k, T, f$ )

1. **for** each itemset  $S$  in  $C_k$
2.   {
3.     obtain  $u(S)$  of  $C_k$  by Equation (2.4);
4.     obtain  $s(S)$  of  $C_k$ ;
5.   }
6. **return** ( $C_k$ );

Figure 2.10: The *CalculateAndStore\_H* function for the UMining\_H algorithm.

The *CalculateAndStore\_H* function is similar to the *CalculateAndStore* function given in Figure 2.5, but a line has been added (line 4) to obtain the support  $s(S)$  of itemset  $S$ , as required by Definition 20. Each  $S$  in  $C_k$  has associated with it an  $s$  field, denoted  $s(S)$ , for storing its support value.

In the *Prune\_H* function, pruning strategy 2 is applied by calculating the expected utility upper bound  $b'(S)$  of itemset  $S$  instead of calculating the utility upper bound  $u(S)$  as is done in the *Prune* function. Each itemset has associated with it a  $b'$  field, denoted  $b'(S)$ , for storing the expected utility upper bound of  $S$ .

```

Prune_H( $C_k, C_{k-1}, minutil$ )
1.  for each itemset  $S$  in  $C_k$ 
2.  {
3.    obtain  $b'(S)$  by Equation (2.23);
4.    if ( $b'(S) < minutil$ )
5.       $C_k = C_k - \{S\}$ ;
6.  }
7.  return ( $C_k$ );

```

Figure 2.11: The *Prune\_H* function for the UMining\_H algorithm.

**Example 13** Continuing from Example 12, we show how the heuristic algorithm works. At level 1, the UMining\_H algorithm works in the same manner as the UMining algorithm, except it also obtains  $s(A) = 80$ ,  $s(B) = 20$ ,  $s(C) = 40$ , and  $s(D) = 90$ . At level 2,  $C_2 = \{AB, AC, AD, BC, BD, CD\}$  is obtained by the *Generate* function. Using the *Prune\_H* function, the expected upper bounds of itemsets in  $C_2$  are  $b'(AB) = 227.5$ ,  $b'(AC) = 245$ ,  $b'(AD) = 185.56$ ,  $b'(BC) = 295$ ,  $b'(BD) = 218.89$ , and  $b'(CD) = 227.77$ . Since  $b'(AD) < minutil$ ,  $AD$  is removed from  $C_2$ , i.e.,  $C_2 = \{AB, AC, BC, BD, CD\}$ . Next, the utility value and support of each itemset in  $C_2$  is calculated, i.e.,  $u(AB) = 105$ ,  $s(AB) = 10$ ,  $u(AC) = 197$ ,  $s(AC) = 30$ ,  $u(BC) = 138$ ,  $s(BC) = 10$ ,  $u(BD) = 211$ ,  $s(BD) = 20$ ,  $u(CD) = 193$ , and  $s(CD) = 30$ . As with the UMining algorithm, 2-itemsets  $AC$  and  $BD$  are added to  $H$ . At level 3,  $C_3 = \{ABC, ABD, ACD, BCD\}$  is generated. The expected utility upper bounds of the itemsets in  $C_3$  are  $b'(ABC) = 154.33$ ,  $b'(ABD) = 210.5$ ,  $b'(ACD) = 390$ , and  $b'(BCD) = 153.92$ . Thus,  $ABC$  and  $BCD$  are removed from  $C_3$ , i.e.,  $C_3 = \{ABD, ACD\}$ . Next, the utility value and support of each itemset in  $C_3$  is calculated. i.e.,  $u(ABD) = 106$ ,  $s(ABD) = 10$ ,  $u(ACD) = 150$ , and  $s(ACD) = 20$ . As with the UMining algorithm, no 3-itemsets are added to  $H$ . At

level 4,  $C_4 = \{ABCD\}$  is obtained, and  $b'(ABCD) = 181$  is calculated. Thus,  $C_4 = \phi$  after removing  $ABCD$ . At this point, the algorithm terminates.

The `UMining_H` heuristic method may miss some high utility itemsets, but it only reports correct ones. Figure 2.12(a) shows the complete itemset semi-lattice of candidate itemsets, and Figure 2.12(b) shows the portion of the semi-lattice searched by the `UMining_H` algorithm. Since `UMining_H` searches 11 nodes along 14 arcs, it searches less of the space than `UMining`, which searches 13 nodes along 21 arcs. In this case, `UMining_H` does not miss any high utility itemsets.

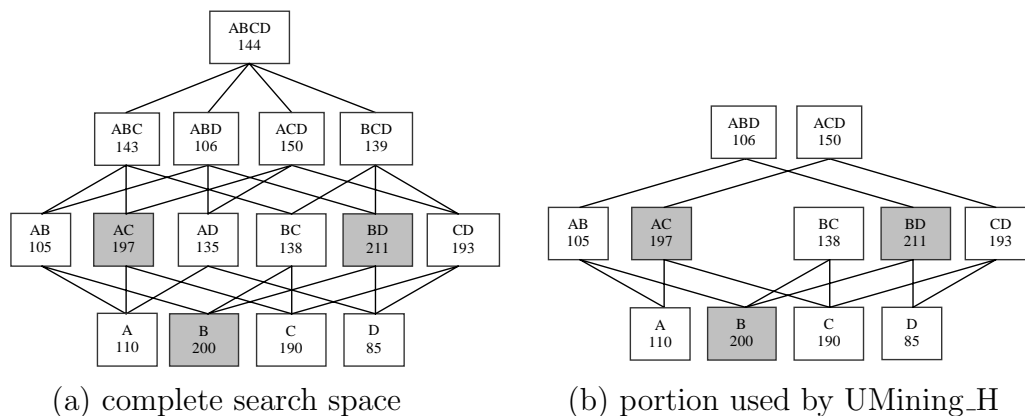


Figure 2.12: Itemset semi-lattice for the `UMining_H` algorithm.

## 2.5 Experimental Results

In this section, experimental results on synthetic datasets and real world databases are summarized. Then, a comparison between our proposed approach and other related approaches is made. Next, the differences between utility based itemset mining and frequent itemset mining are highlighted.

All experiments were performed on a 1 GHz Pentium III PC with 256 MB RAM running Microsoft Windows XP. The UMining and UMining\_H algorithm were implemented in Microsoft/Visual C++ 6.0. For simplicity, the utility threshold *minutil* for all experiments was defined as  $minutil = \alpha \times total\ utility\ value$ , where the *total utility value* is a constant calculated from the transaction database after the first pass of the algorithm. More precisely,  $total\ utility\ value = \sum_{i_p \in I} u(i_p)$ , where  $I$  is the set of all items included in the transaction database. The coefficient  $\alpha$  represents the minimum acceptable ratio of the utility value of an itemset to the total utility value in the database. For example, if the profit of an itemset is regarded as a utility value, then  $\alpha = 10\%$  indicates that the algorithm should select any itemset with a profit of at least ten percent of the total profits. We report our results using  $\alpha$  rather than *minutil*. The maximum size  $K$  of the itemsets was set to a value high enough so that it had no effect on the experiments.

### 2.5.1 Experimental Summary

To evaluate the efficiency of the UMining and UMining\_H algorithms, we performed experiments on both synthetic and real datasets.

First, we report the experimental results found using a synthetic dataset, which was generated by the IBM synthetic data generator [47]. This dataset contains 983,011 transactions and 7,352 unique items. The size of the dataset is approximately 331 megabytes. This dataset was chosen since it is typical of those used in previous data mining performance studies. Each row of the dataset contains a *customer ID*, a *transaction ID* and an *item ID*. We defined the utility function  $f$  as  $f = item\ ID \% 100$ , where  $\%$  is a modulus operator, and the domain of *item ID*

is from 1 to 7,352. The modulus operator fulfills the requirement that the utility function  $f$  be nonnegative but neither non monotone nor convertible. To compare the efficiency of the algorithms, we used a fixed function  $f$  to keep this factor constant. We studied the effect of different values for coefficient  $\alpha$  on the processing time and the number of transactions that contributed to the calculation of the high utility itemsets for the UMining and UMining\_H algorithms. The values of coefficient  $\alpha$  varied from 10% down to 0.25% for each experiment. Lower values for  $\alpha$  typically represent more challenging cases, because the number of itemsets ordinarily increases as  $\alpha$  decreases for itemset algorithms.

The experimental results are summarized in Table 2.7, where the *HUI* column represents the number of high utility itemsets discovered by the algorithm, and the *# of Trans.* column represents the number of transactions that contributed to the calculation of the high utility itemsets. The results show that the execution time decreases when coefficient  $\alpha$  increases. Both algorithms obtain the same number of high utility itemsets when  $\alpha$  is greater than or equal to 2%, because all discovered high utility itemsets are 1-itemsets. Table 2.8 further shows that UMining\_H is 0% to 27% faster than UMining on the same IBM synthetic data with an accuracy of 93.97% to 100%, as measured by the percentage of high utility itemsets found.

Table 2.7: Experimental results on synthetic dataset for UMining and UMining\_H.

Minutil $\alpha$	UMining			UMining_H		
	# of HUI	# of Trans.	Time	# of HUI	# of Trans.	Time
10.00%	21	6,604,369	43 min.	21	6,604,369	43 min.
5.00%	21	6,604,369	43 min.	21	6,604,369	43 min.
2.00%	21	6,604,369	43 min.	21	6,604,369	43 min.
1.00%	39	11,160,188	70 min.	39	9,486,180	62 min.
0.50%	394	18,845,636	125 min.	381	13,568,451	91 min.
0.25%	3,371	26,136,835	152 min.	3,168	23,523,151	134 min.

Table 2.8: The accuracy of estimation and percentage of time saved .

Threshold ( $\alpha$ )	10%	5%	2%	1%	0.50%	0.25%
Accuracy	100%	100%	100%	100%	96.70%	93.97%
Time Saved	0%	0%	0%	12%	27%	12%

In theory, an exhaustive search could also be applied to the same dataset by using the UMining algorithm with the *Prune* function commented out. Unfortunately, in practice, the resulting program ran out of memory on our computer. To show the benefit of the pruning step, we ran our experiments on 3.6 GHZ PC with 2 GB RAM. The UMining algorithm took 36.9 minutes to find all itemsets when  $\alpha$  was set to 0.25%. The exhaustive search generated 3,363,385 candidate 2-itemsets, and then ran out of memory while processing 3-itemsets. As a result, we gradually reduced the number of items in the synthetic dataset from original 7,352 to 500 unique items. Unfortunately, the exhaustive search ran out of memory in all these experiments. Fortunately, when the number of items is reduced to 100 unique items, it takes 160 minutes for exhaustive search. However, our UMining algorithm only needs 23 minutes to obtain all desired results. These experimental results clearly show the benefit of the pruning step. The pruning strategies are effective in reducing the space requirements.

Secondly, we tested the UMining and UMining\_H algorithms on an 8-million-record transaction database provided by a commercial partner, which represents the purchase of 2,238 unique items by 428,665 customers. Each record of the database consists of the account number of a customer, an item code identifying the equipment or service purchased by the customer, and a numerical value indicating the quantity purchased. The utility function  $f(x, y)$  is defined as  $f(x, y) = x \cdot y$ , where  $x$  is the



quantity purchased of an item in a transaction and  $y$  is the unit price of the item stored separately in another database table. The coefficient  $\alpha$  was varied from 10% down to 0.25%. We investigated the effect of different values for coefficient  $\alpha$  and different sizes of datasets on the processing time and the number of passes.

Table 2.9 shows that for both UMining and UMining.H, the execution time and the number of passes decrease as coefficient  $\alpha$  increases. In the table, the  $H$  column represents the number of high utility itemsets discovered by UMining, and the  $H'$  column represents the number discovered by UMining.H. The  $k$ th row of the table shows the number of itemsets discovered during pass  $k$  over the database. Displaying this information while the algorithm is running provides a user with a chance to monitor the progress of the algorithm. In this experiment, UMining.H found all itemsets with  $\alpha = 10\%$  and  $5\%$ , but it found only 87 out of 88 (98.86%) when  $\alpha = 2\%$ , and 5,563 out of 5,694 (97.70%) when  $\alpha = 0.25\%$ . As  $\alpha$  increases, the number of high utility itemsets discovered decreases for both UMining and UMining.H. Table 2.9 also shows that for a fixed value of  $\alpha$ , the number of high utility itemsets does not increase proportionally with the pass number  $k$ .

Table 2.9: Experimental results on the customer database.

Threshold $\alpha$	$\alpha = 10\%$		$\alpha = 5\%$		$\alpha = 2\%$		$\alpha = 1\%$		$\alpha = 0.5\%$		$\alpha = 0.25\%$	
	H	H'	H	H'	H	H'	H	H'	H	H'	H	H'
Pass 1	1	1	2	2	6	6	10	10	22	22	54	54
Pass 2	2	2	7	7	19	19	54	54	213	213	454	454
Pass 3	0	0	9	9	31	31	117	117	432	427	1,309	1,297
Pass 4	0	0	4	4	24	24	123	118	668	642	2,017	1,967
Pass 5	0	0	0	0	8	7	33	32	467	453	1,860	1,791
Total	3	3	22	22	88	87	337	331	1,802	1,757	5,694	5,563
Time	28 m.	25m.	42m.	39m.	65m.	58m.	87m.	76m.	2.7h.	2.4h.	10.5h.	8.4 h.

By comparing the number of high utility itemsets in Table 2.9, Table 2.10 was obtained to show the accuracy at each pass and percentage of time saved by

UMining\_H. Table 2.10 indicates that UMining\_H is from 7.14% to 20.00% faster than UMining on the customer database with accuracy ranging from 97.50% to 100% for the thresholds tested. In this experiment, with  $\alpha = 5\%$  and  $\alpha = 10\%$ , both accuracies are 100%. In other words, no high utility itemsets are missed by UMining\_H. For  $\alpha = 0.25\%$ , the accuracy of the UMining\_H algorithm is 97.70%, with 131 of 5,694 high utility itemsets missed. However, about 20% of the time is saved. Therefore, UMining\_H provides a reasonable balance between accuracy and performance.

Table 2.10: The accuracy of estimation and percentage of time saved

Threshold $\alpha$	10%	5%	2%	1%	0.5%	0.25%
Accuracy	Pass 1	100%	100%	100%	100%	100%
	Pass 2	100%	100%	100%	100%	100%
	Pass 3		100%	99.08%	100%	98.84%
	Pass 4		100%	97.52%	95.93%	96.11%
	Pass 5			96.29%	96.97%	97.00%
	Overall	100%	100%	98.86%	98.21%	97.50%
Time Saved	10.71%	7.14%	10.77%	12.64%	16.05%	20.00%

To examine the scalability of the algorithm, we generated three additional databases from the commercial database with sizes of 10%, 25%, and 50% of the original database. The effect of the database size on the running time is shown in Table 2.11. As the size of the database increases, more execution time is saved. Given the same database, more time is saved as the coefficient  $\alpha$  decreases. The time saved by using UMining\_H instead of UMining ranges from 9.25% to 20% when  $\alpha = 0.25\%$ . The heuristic approach appears to be more suitable for huge datasets, but generally, the extra time UMining requires may be a good investment, because it guarantees complete results.

Table 2.11: The effect of the size of commercial database on the running time.

Size	100% Dataset		50% Dataset		25% Dataset		10% Dataset	
$\alpha$	UMining	UMining_H	UMining	UMining_H	UMining	UMining_H	UMining	UMining_H
10.00%	28 min.	25 min.	9 min.	8 min.	4 min.	4 min.	4 min.	4 min.
5.00%	42 min.	39 min.	15 min.	13 min.	7 min.	6 min.	4 min.	4 min.
2.00%	65 min.	58 min.	21 min.	18 min.	10 min.	9 min.	5 min.	4 min.
1.00%	87 min.	76 min.	31 min.	26 min.	15 min.	14 min.	8 min.	7 min.
0.50%	2.7 hour	2.4 hour	67 min.	58 min.	22 min.	20 min.	13 min.	12 min.
0.25%	10.5 hour	8.4 hour	187 min.	167 min.	54 min.	49 min.	19 min.	17 min.

## 2.5.2 Algorithm Comparison

Since utility constraints may not always be anti monotone or convertible, previous algorithms cannot handle the problems that UMining was designed to address. Thus, we cannot directly compare our algorithm with previous algorithms. However, we can use our algorithms to perform the tasks of frequent itemset mining [3] or convertible constraint based itemset mining [17, 19, 53, 70], if we treat them as special cases of utility constraints. To do so, we need to design a utility constraint that is anti monotone for frequent itemset mining or convertible for constraint based itemset mining. In more detail, for frequent itemset mining,  $u(S)$  could be defined as an anti monotone function  $u(S) = s(S)$ , where  $s(S)$  is the support of the itemset  $S$ . The Apriori algorithm [3] is more efficient than UMining since the Apriori property allows more itemsets to be pruned than our upper bound property allows. For convertible constraint based mining,  $u(S)$  could be defined as  $u(S) = \sum_{i_p \in S} f(i_p)$  w.r.t. a decreasing order on the weight  $f(i_p)$  of item  $i_p$ . Although UMining can solve these problems, its performance may be poorer than that of the CC, WI, and HUM algorithms, since the original algorithms have more efficient and powerful pruning strategies based on the convertible property. Since the VAM algorithm uses exhaus-

tive search, UMining is more efficient than VAM. Although several of the original algorithms are more effective at reducing the the search space than UMining, our algorithm can handle more complex semantics for itemsets in applications than these algorithms. More precisely, our proposed algorithm can handle utility constraints but the others cannot. Therefore, the relationship between our approach and related approaches is complementary rather than competitive.

The Itemset Share approach [7] is the only previous approach that can consider the numerical value on the cell of transaction datasets. The numerical value for an item in a transaction can be regarded as the utility value for that item. Since the Itemset Share approach does not consider any subjective user preferences concerning the utility of items, it treats all users uniformly and will thus discover the same itemsets for any user. Therefore the Itemset Share approach can be viewed as a special case of utility based itemset mining where all user preferences are equal. To make the Itemset Share approach solve a utility based itemset mining problem, we transformed the input data by replacing each numerical value of an item in a transaction by the utility value of the item in the transaction. By running an Itemset Share program on the transformed data and UMining on the original data, we created a fair comparison.

We compare the effect of different values for coefficient  $\alpha$  and different sizes of datasets on the processing time between the UMining algorithm used in the utility based itemset mining approach and the SIP heuristic algorithm proposed for the Itemset Share approach [7]. Like other Itemset Share approaches, SIP is not guaranteed to find all share frequent itemsets. We used Barber's original implementation of the SIP algorithm.

First, an empirical comparison between UMining and SIP on the IBM synthetic

data is presented. The results are shown in Table 2.12, where  $\% \textit{found}$  represents percentage of the high utility itemsets found. The experimental results indicate that UMining is slightly faster than SIP, since UMining examines a smaller part of the search space than SIP. Most importantly, UMining is guaranteed to find *all* high utility itemsets, while SIP is a heuristic approach that may miss many high utility itemsets. For example, when  $\alpha = 0.25\%$ , the accuracy of UMining is 100%, and the accuracy of UMining\_H is 93.97%, but the accuracy of SIP is 28.18% with 2,421 of 3,371 high utility itemsets missed.

Table 2.12: The comparison of UMining and SIP on IBM synthetic dataset.

Minutil $\alpha$	UMining				SIP			
	# of HUI	% found	# of Trans.	Time	# of HUI	% found	# of Trans.	Time
10.00%	21	100%	6,604,369	43 min.	21	100%	6,604,369	43 min.
5.00%	21	100%	6,604,369	43 min.	21	100%	6,604,369	43 min.
2.00%	21	100%	6,604,369	43 min.	21	100%	6,604,369	43 min.
1.00%	39	100%	11,160,188	70 min.	39	100%	11,260,944	79 min.
0.50%	394	100%	18,845,636	125 min.	297	75.38%	19,346,322	129 min.
0.25%	3,371	100%	26,136,835	152 min.	950	28.18%	27,559,764	155 min.

Next, the UMining and SIP algorithms are compared on the commercial database described in Subsection 2.5.1. The effect of the database size on running time is shown in Table 2.13. UMining is faster than SIP on all different sizes, due to its more effective pruning. UMining appears to be slightly faster than SIP for large database sizes. The percentages of time saved for an  $\alpha$  value of 0.25% are 14%, 3%, 14% and 14% on the 100%, 50%, 25%, and 10% datasets, respectively. Although the time differences are not great, it must be emphasized that UMining finds all desired itemsets while SIP does not.

Table 2.13: Comparison of UMining and SIP on the commercial database.

Size	100% Dataset		50% Dataset		25% Dataset		10% Dataset	
$\alpha$	UMining	SIP	UMining	SIP	UMining	SIP	UMining	SIP
10.00%	28 min.	35 min.	9 min.	14 min.	4 min.	9 min.	4 min.	4 min.
5.00%	42 min.	77 min.	15 min.	18 min.	7 min.	14 min.	4 min.	5 min.
2.00%	65 min.	92 min.	21 min.	25 min.	10 min.	17 min.	5 min.	6 min.
1.00%	87 min.	119 min.	31 min.	38 min.	15 min.	22 min.	8 min.	9 min.
0.50%	2.7 hour	3.3 hour	67 min.	76 min.	22 min.	25 min.	13 min.	14 min.
0.25%	10.5 hour	11.4 hour	187 min.	193 min.	54 min.	63 min.	19 min.	22 min.

### 2.5.3 Utility versus Support

To end this section, we use the experimental results to emphasize the differences between utility based itemset mining and frequent itemset mining. Let us consider 16 representative itemsets selected from the commercial database. Figure 2.13(a) shows the utility values of the itemsets and their corresponding support values. In Figure 2.13(b) and (c), the itemsets are ranked in terms of utility value and support value, respectively. The difference between ranking by utility values and ranking by support values is apparent. The itemset ranked 1<sup>st</sup> in terms of utility value was ranked 3<sup>rd</sup> in terms of support value. On the other hand, the itemset ranked 1<sup>st</sup> in terms of support value was only ranked 7<sup>th</sup> in terms of utility value.

In Figure 2.14, the rear series represents the utility value of each itemset, and the front series represents the support values. UMining can find more itemsets of interest to a user by considering the user's preferences.

Itemset ID	Utility (%)	Support (%)
1	54.21	4.28
2	16.62	7.51
3	15.68	1.04
4	11.37	0.86
5	14.72	0.97
6	25.22	1.67
7	11.68	0.76
8	10.74	0.70
9	14.45	0.95
10	13.40	1.00
11	47.44	5.32
12	36.69	3.70
13	34.63	2.47
14	14.71	0.97
15	23.58	1.77
16	11.60	0.76

Itemset ID	Utility Ranking	Support Ranking
1	1	3
11	2	2
12	3	4
13	4	5
6	5	7
15	6	6
2	7	1
3	8	8
5	9	10
14	10	11
9	11	12
10	12	9
7	13	14
16	14	15
4	15	13
8	16	16

Itemset ID	Support Ranking	Utility Ranking
2	1	7
11	2	2
1	3	1
12	4	3
13	5	4
15	6	6
6	7	5
3	8	8
10	9	12
5	10	9
14	11	10
9	12	11
4	13	15
7	14	13
16	15	14
8	16	16

(a) 16 Itemsets                      (b) Ranked by Utility                      (c) Ranked by Support

Figure 2.13: Itemsets ranked by their utility values and support values.

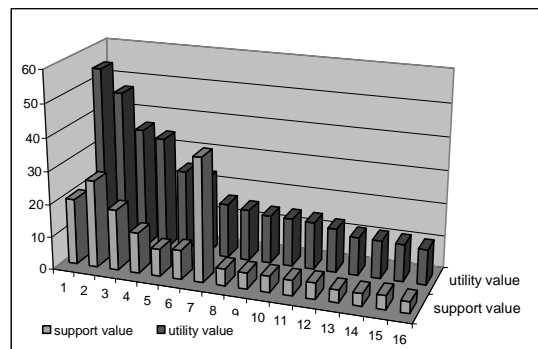


Figure 2.14: Graph of itemsets ranked by their utility values.

## 2.6 Summary

The problem of utility based itemset mining is to discover the itemsets that are significant according to their utility values. In this chapter, we first showed that

the Apriori property and convertible constraint property are not directly applicable to the problem of utility based itemset mining. As a result, the mathematical properties of the utility value of an itemset were analyzed. Next, two novel pruning strategies were presented to reduce the cost of finding high utility itemsets. With these pruning strategies, a  $k$ -itemset with a utility upper bound less than *minutil* can be pruned immediately without accessing the database to calculate its actual utility value. By exploiting these pruning strategies, the UMining and UMining\_H algorithms were developed to provide efficient solutions to the utility based itemset mining problem. The effectiveness of the algorithms was demonstrated by applying them to synthetic and real world databases. The experimental results indicate that the proposed algorithms can discover high utility itemsets efficiently. UMining may be preferable to UMining\_H, because it guarantees the discovery of all high utility itemsets.

Utility based itemset mining was compared to frequent itemset mining, convertible constraint based mining, and share based mining. Utility constraints are capable of expressing more complex semantics than the support measure, convertible constraints, or the share measure. Previously developed algorithms are more efficient for support based or convertible constraint based mining problems. For utility based mining, the proposed UMining and UMining\_H algorithms are more efficient than any previous algorithms. As well, UMining guarantees that all high utility itemsets are found, while the heuristic share based methods may miss many relevant itemsets. Overall, when a utility constraint is required to describe more complex user preferences, the UMining algorithm can efficiently find all useful itemsets from a database, while methods for frequent itemset mining, convertible constraint based mining, and share based mining cannot.



## Chapter 3

# Mining Functional Dependencies from Data

In this chapter, we consider the problem of mining functional dependencies from data. Background knowledge concerning functional dependencies is presented in Section 3.1. In Section 3.2, the theoretical foundations of functional dependencies are analyzed and four pruning rules are presented. Section 3.3 describes algorithms for finding functional dependencies from data. In Section 3.4, our experimental results are presented. We show a practical application of discovering functional dependencies from data, namely, constructing a sound Bayesian network, in Section 3.5. Finally, in Section 3.6, we summarize the chapter.

### 3.1 Functional Dependencies

Let  $U = \{v_1, v_2, \dots, v_m\}$  be a finite set of *attributes* [56] (or variables). Each variable  $v_i$  has a finite domain, denoted  $dom(v_i)$ , representing the values that  $v_i$  can take on.

For a subset  $X = \{v_i, \dots, v_j\}$  of  $U$ , we write  $dom(X)$  for the Cartesian product of the domains of the individual variables in  $X$ , namely,  $dom(X) = dom(v_i) \times \dots \times dom(v_j)$ . Each element  $x \in dom(X)$  is called a *configuration* of  $X$ . A *relation*  $r$  [56] on  $U$ , written  $r(U)$ , is a finite set of mappings  $\{t_1, \dots, t_n\}$  from  $U$  to  $dom(U)$  with the restriction that for each mapping  $t \in r(U)$ ,  $t(v_i)$  must be in  $dom(v_i)$ ,  $1 \leq i \leq m$ , where  $t(v_i)$  denotes the value obtained by restricting the mapping  $t$  to  $v_i$ . A relation  $r$  on  $U = \{v_1, v_2, \dots, v_m\}$  is depicted in Table 3.1. Each mapping  $t$  is called a *tuple* [56] and  $t(v_i)$  is called the  $v_i$ -value of  $t$ . To simplify notation, we may write the singleton set  $\{v_i\}$  as the single attribute  $v_i$  and the set of attributes  $\{v_1, \dots, v_j\}$  as  $v_1 \dots v_j$ . The union  $X \cup Y$  of two attribute sets  $X$  and  $Y$  is sometimes simply denoted as  $XY$ . We use the terms relation  $r(U)$  and transaction dataset  $T$  interchangeably.

Table 3.1: A relation  $r$  on the relational  $U = \{v_1, \dots, v_m\}$ .

$$r = \begin{array}{|c|c|c|c|} \hline v_1 & v_2 & \dots & v_m \\ \hline t_1(v_1) & t_1(v_2) & \dots & t_1(v_m) \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline t_n(v_1) & t_n(v_2) & \dots & t_n(v_m) \\ \hline \end{array}$$

**Definition 21** [41]. Let  $r(U)$  be a relation and  $X, Y \subseteq U$ . A *functional dependency* (*FD*) is a constraint, denoted  $X \rightarrow Y$ . The FD  $X \rightarrow Y$  is satisfied by  $r(U)$  if every two tuples  $t_i, t_j \in r(U)$  that have  $t_i(X) = t_j(X)$  also have  $t_i(Y) = t_j(Y)$ . In an FD  $X \rightarrow Y$ , we refer to  $X$  as the *antecedent* and  $Y$  as the *consequent*.

**Example 14** Consider an example relation  $r(U)$  shown in Table 3.2. By definition, the FDs  $A \rightarrow D$ ,  $D \rightarrow A$ ,  $AB \rightarrow E$ ,  $BD \rightarrow E$ ,  $BE \rightarrow A$ ,  $BE \rightarrow D$ ,  $CE \rightarrow A$ , and  $CE \rightarrow D$  are satisfied by  $r(U)$ . However, the FD  $A \rightarrow B$  is not satisfied by  $r(U)$ ,

since for tuples  $t_1, t_2 \in r(U)$ ,  $t_1(A) = t_2(A)$  but  $t_1(B) \neq t_2(B)$ .

Table 3.2: An example relation.

TID	A	B	C	D	E
$t_1$	0	0	0	2	0
$t_2$	0	1	0	2	0
$t_3$	0	2	0	2	2
$t_4$	0	3	1	2	0
$t_5$	4	1	1	1	4
$t_6$	4	3	1	1	2
$t_7$	0	0	1	2	0

Armstrong's Axioms [56] are the following three inference axioms for FDs defined on sets of attributes  $X$ ,  $Y$ , and  $Z$ .

**F1. (Reflexivity)** If  $Y \subseteq X$ , then  $X \rightarrow Y$ .

**F2. (Augmentation)** If  $X \rightarrow Y$ , then  $XZ \rightarrow YZ$ .

**F3. (Transitivity)** If  $X \rightarrow Y$  and  $Y \rightarrow Z$ , then  $X \rightarrow Z$ .

**Theorem 6** [56]. *The inference axioms F1, F2, and F3 are sound and complete.*

Soundness indicates that given a set  $F$  of FDs satisfied by a relation  $r(U)$ , any FD inferred from  $F$  using F1-F3 is satisfied by  $r(U)$  too. Completeness indicates that the three axioms F1-F3 can be applied repeatedly to infer all FDs logically implied by a set  $F$  of FDs [56]. The following two inference axioms can be derived from Armstrong's Axioms and are introduced for convenience [72].

**Union** If  $X \rightarrow Y$  and  $X \rightarrow Z$ , then  $X \rightarrow YZ$ .

**Decomposition** If  $X \rightarrow YZ$ , then  $X \rightarrow Y$  and  $X \rightarrow Z$ .

By Decomposition,  $X \rightarrow YZ$  is sometimes written as  $X \rightarrow v_i, v_i \in YZ$ .

**Definition 22** [72]. Let  $X, Y \subseteq U$  and  $F$  be a set of FDs. The *closure* of  $X$  ( $X \neq \emptyset$ ) w.r.t.  $F$ , denoted  $X^+$ , is defined as  $\{Y \mid X \rightarrow Y \text{ can be deduced from } F \text{ using Armstrong's Axioms}\}$ . The closure of  $F$ , denoted  $F^+$ , is a set of all FDs that can be deduced from  $F$  using Armstrong's Axioms.

Definition 22 indicates that the FD  $X \rightarrow Y$  holds if and only if  $Y \in X^+$ . For  $X, Y \subseteq U$ , we use  $(XY)^+$  to denote the closure of  $X \cup Y$ .

**Example 15** Let  $F = \{A \rightarrow C, BD \rightarrow AC\}$ . By Definition 22,  $A^+ = \{A, C\}$  and  $(BD)^+ = \{A, B, C, D\}$ .

Now, we introduce partitions of  $X$  and  $XY$ . These partitions can be used to check whether or not an FD  $X \rightarrow Y$  is satisfied by a relation.

**Definition 23** Let  $X \subseteq U$  and let  $t_1, \dots, t_n$  be all the tuples in a relation  $r(U)$ . The *partition* over  $X$ , denoted  $\prod_X$ , is a set of the groups such that  $t_i$  and  $t_j$ ,  $1 \leq i, j \leq n, i \neq j$ , are in the same group if and only if  $t_i[X] = t_j[X]$ . The number of the groups in the partition is called the *cardinality* of the partition, denoted  $|\prod_X|$ .

For a single attribute  $v_i$ , we use  $\prod_{Xv_i}$  to denote the partition of the set of attributes  $X \cup \{v_i\}$ .

**Example 16** In Table 3.2, by Definition 23,  $\prod_A = \{\{t_1, t_2, t_3, t_4, t_7\}, \{t_5, t_6\}\}$ , and  $\prod_{CE} = \{\{t_1, t_2\}, \{t_3\}, \{t_4, t_7\}, \{t_5\}, \{t_6\}\}$ . By Definition 23,  $|\prod_A| = 2$  and  $|\prod_{CE}| = 5$ .

Using the cardinality of the partition, we can check whether or not an FD  $X \rightarrow Y$  is satisfied by a relation using Theorem 7, which was suggested by Huhtala et al. [41].

**Theorem 7** [41]. *An FD  $X \rightarrow Y$  is satisfied by a relation  $r(U)$  if and only if  $|\Pi_X| = |\Pi_{XY}|$ .*

For example, in Table 3.2, FD  $CE \rightarrow A$  is satisfied by  $r(U)$ , since  $|\Pi_{CE}| = |\Pi_{ACE}| = 5$ .

## 3.2 Theoretical Foundation

In this section, we give the theoretical foundation for our approach to finding functional dependencies in a relation. We introduce the concepts of equivalent attributes and nontrivial closure and provide proofs of related lemmas and theorems.

### 3.2.1 Equivalent Attributes

**Definition 24** Let  $X, Y \subseteq U$ . If  $X \rightarrow Y$  and  $Y \rightarrow X$ , then  $X$  and  $Y$  are said to be *equivalent sets of attributes*, denoted by the *equivalence*  $X \leftrightarrow Y$ .

The following theorem shows that we can check whether or not the equivalence  $X \leftrightarrow Y$  is satisfied in a relation  $U$  by comparing the closures of  $X$  and  $Y$ .

**Theorem 8** *Let  $X, Y \subseteq U$ . If  $Y \subseteq X^+$  and  $X \subseteq Y^+$ , then  $X \leftrightarrow Y$ .*

*Proof:* Since  $X \rightarrow X^+$  and  $Y \subseteq X^+$ , then by Decomposition,  $X \rightarrow Y$  holds. By a similar argument,  $Y \rightarrow X$  holds. As  $X \rightarrow Y$  and  $Y \rightarrow X$ , we have  $X \leftrightarrow Y$ .  $\square$

**Example 17** Let  $U = \{A, B, C, D\}$  and  $F = \{BD \rightarrow AC, CD \rightarrow B\}$ . By Definition 22,  $(BD)^+ = \{A, B, C, D\}$  and  $(CD)^+ = \{A, B, C, D\}$ . Since  $CD \subseteq (BD)^+$  and  $BD \subseteq (CD)^+$ , we have  $BD \leftrightarrow CD$ .

Using equivalence  $X \leftrightarrow Y$  and Armstrong's Axioms, Lemmas 2 and 3 are obtained.

**Lemma 2** *Let  $W, X, Y, Y', Z \subseteq U$  and  $Y \subset Y'$ . If  $X \leftrightarrow Y$  and  $XW \rightarrow Z$ , then  $Y'W \rightarrow Z$ .*

*Proof:* By  $X \leftrightarrow Y$ , we have  $Y \rightarrow X$ . By Augmentation,  $YW \rightarrow XW$ . By Transitivity,  $YW \rightarrow XW$  and  $XW \rightarrow Z$  give  $YW \rightarrow Z$ . By Augmentation,  $Y' - Y$  can be added to both sides of  $YW \rightarrow Z$  giving  $YW(Y' - Y) \rightarrow Z(Y' - Y)$ . By  $Y \subset Y'$ ,  $Y'W \rightarrow Z(Y' - Y)$ . By Decomposition,  $Y'W \rightarrow Z$ .  $\square$

**Lemma 3** *Let  $W, X, Y, Z \subseteq U$ . If  $X \leftrightarrow Y$  and  $WZ \rightarrow X$ , then  $WZ \rightarrow Y$ .*

*Proof:* By  $X \leftrightarrow Y$ , we have  $X \rightarrow Y$ . By Transitivity,  $WZ \rightarrow X$  and  $X \rightarrow Y$  give  $WZ \rightarrow Y$ .  $\square$

**Definition 25** Let  $F$  be a set of FDs and  $X^+$  be the closure of  $X$  w.r.t.  $F$ . The *nontrivial closure* of  $X$  w.r.t.  $F$ , denoted  $X^*$ , is defined as  $X^* = X^+ - \{X\}$ .

For  $X, Y \subseteq U$ , we use  $(XY)^*$  to denote the nontrivial closure of the set of attributes  $X \cup Y$ , similarly to how we use  $(XY)^+$  to denote the closure of attributes  $X \cup Y$ . We have  $(XY)^+ = (XY)^* \cup XY$ .

**Example 18** Recalling Example 15, we have  $A^+ = \{A, C\}$  and  $(BD)^+ = \{A, B, C, D\}$ . By Definition 25,  $A^* = \{C\}$  and  $(BD)^* = \{A, C\}$ .

**Theorem 9** *Let  $X, Y, Z \subseteq U$  such that  $Y^+ \subseteq X$ . Then  $X - Y^* \rightarrow Z$  if and only if  $X \rightarrow Z$ .*

*Proof:* We first prove that if  $X - Y^* \rightarrow Z$  then  $X \rightarrow Z$ . By Augmentation,  $Y^*$  can be added to both sides of  $X - Y^* \rightarrow Z$  giving  $X \rightarrow Y^*Z$ . By Decomposition,  $X \rightarrow Z$ . Now we prove that if  $X \rightarrow Z$  then  $X - Y^* \rightarrow Z$ . Since  $Y = Y^+ - Y^*$  and  $Y^+ \subseteq X$ , then  $Y \subseteq X - Y^*$ . By Reflexivity,  $X - Y^* \rightarrow Y$ . By Transitivity,  $X - Y^* \rightarrow Y$  and  $Y \rightarrow Y^*$  give  $X - Y^* \rightarrow Y^*$ . By Union,  $X - Y^* \rightarrow Y^*$  and  $X - Y^* \rightarrow X - Y^*$  give  $X - Y^* \rightarrow X$ . Thus, by Transitivity,  $X - Y^* \rightarrow X$  and  $X \rightarrow Z$  give  $X - Y^* \rightarrow Z$ .  $\square$

**Example 19** Let  $U = \{A, B, C, D, E\}$  such that  $C \rightarrow A$  is satisfied by  $r(U)$ . Suppose  $X = ACE$ ,  $Y = C$ , and  $Z = BD$ . Since  $C \rightarrow A$ ,  $C^* = A$ . Then  $Y^* = A$  and  $Y^+ = C \cup C^* = AC$ . Thus,  $Y^+ \subseteq X$  and  $X - Y^* = CE$ . By Theorem 9, if  $CE \rightarrow BD$  is satisfied by  $r(U)$ , then  $ACE \rightarrow BD$  is also satisfied by  $r(U)$ ; if  $CE \rightarrow BD$  is not satisfied, then  $ACE \rightarrow BD$  is not satisfied either.

**Theorem 10** Let  $X, Y \subseteq U$ . Then  $X^* \cup Y^* \subseteq (XY)^+$ .

*Proof:* By Definition 22,  $X \rightarrow X^*$  is satisfied by  $r(U)$ . By Augmentation,  $Y$  can be added to both sides of  $X \rightarrow X^*$  giving  $XY \rightarrow X^*Y$ . By Decomposition,  $XY \rightarrow X^*$ , which indicates that  $X^* \subseteq (XY)^+$ . It can be similarly shown that  $Y^* \subseteq (XY)^+$ . Thus,  $X^* \cup Y^* \subseteq (XY)^+$ .  $\square$

**Example 20** Let  $U = \{v_1, v_2, v_3, v_4, v_5\}$ ,  $v_1^* = \{v_3\}$  and  $v_2^* = \{v_4\}$ . By Theorem 10,  $\{v_3, v_4\} \subseteq (v_1v_2)^+$ . That is,  $v_1v_2 \rightarrow v_3v_4$ . By Decomposition,  $v_1v_2 \rightarrow v_3$  and  $v_1v_2 \rightarrow v_4$  can also be inferred.

Theorem 10 indicates that when checking all FDs of the form  $XY \rightarrow v_i$ , where  $v_i \in U - XY$ , only the FDs of the form  $XY \rightarrow v_i$ , where  $v_i \in U - X^+Y^+$ , need to be checked, since  $X^* \cup Y^* \subseteq (XY)^+$ . As a result, the right side  $U - XY$  of

FD  $XY \rightarrow v_i$ , where  $v_i \in U - XY$ , can be reduced to  $U - X^+Y^+$ . That is, the attributes  $X^+Y^+$  can be pruned from the right side  $U - XY$  of the FD  $XY \rightarrow v_i$ , where  $v_i \in U - XY$ .

**Lemma 4** Let  $X \subset S \subseteq U$ . If  $X \rightarrow U - X$ , then  $S \rightarrow U - S$ .

*Proof:* By Augmentation,  $S - X$  can be added to both side of  $X \rightarrow U - X$ , giving  $X(S - X) \rightarrow (U - X)(S - X)$ . As  $X \subset S \subseteq U$ , we have  $S \rightarrow U - X$  and  $U - S \subset U - X$ . By Decomposition,  $S \rightarrow U - S$ .  $\square$

Lemma 4 indicates that if  $X$  functionally determines all attributes in  $U$  other than  $X$ , then all supersets of  $X$  also functionally determine all attributes in  $U$  other than  $X$ .

### 3.2.2 Pruning Rules

A simple approach to finding all functional dependencies that are satisfied by a relation is to propose the set of possible candidates for the antecedent and check them one by one with possible consequents. A *candidate*  $X \subset U$  is a nonempty set of attributes being evaluated for functional dependency.

The semi-lattice illustrated in Figure 3.1, excluding the top level, shows the search space of an exhaustive algorithm for finding FDs for five attributes. Figure 3.1 shows all possible nonempty combinations of the five attributes  $A, B, C, D$ , and  $E$ . For these attributes, there are  $2^n = 2^5 = 32$  possible subsets of attributes, of which the  $2^n - 2 = 30$  nonempty, proper subsets are the candidates. The levels of the semi-lattice are numbered from the bottom to the top. The set  $U = \{A, B, C, D, E\}$  at level 5 is not a candidate, because for any FD with the form  $U \rightarrow v_i$ , we have  $v_i = U - U = \phi$ . There are  $n2^{n-1}$  edges in a full lattice for  $n$  attributes [31]. Since



the semi-lattice of the total search space of FDs starts from level 1, rather than the empty set, there are  $n2^{n-1} - n$  edges in the semi-lattice of the complete search space for FDs.

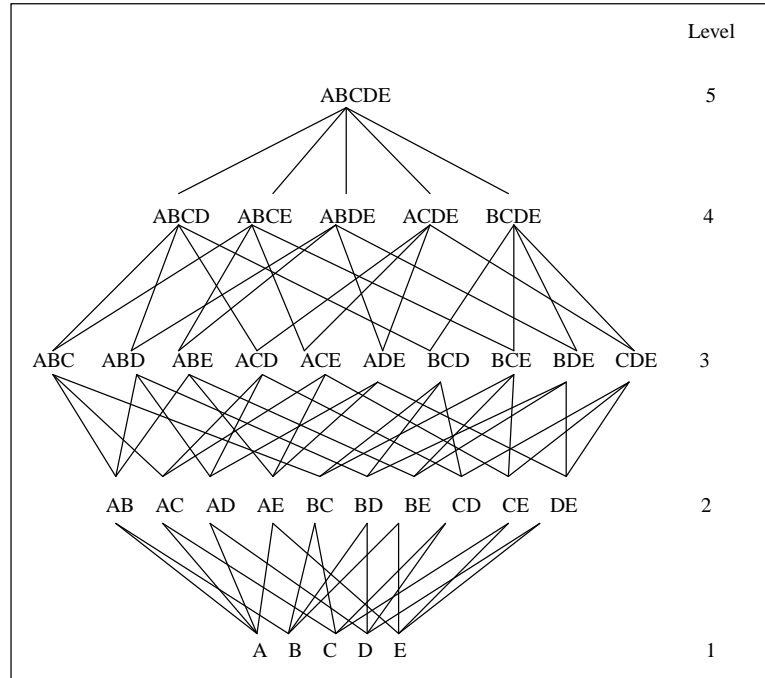


Figure 3.1: All possible nonempty combinations of attributes  $A, B, C, D,$  and  $E$ .

To find the candidates, we generate all possible candidates at level  $k$  from the candidates at level  $k - 1$ . For instance, the candidate  $AB$  at level 2 is generated from the candidates  $A$  and  $B$  at level 1. Overall, given a candidate  $X$ , to find the FDs with  $X$  as antecedent that are satisfied by  $r(U)$ , we will check whether or not  $X$  functionally determines each of the remaining attributes in  $U$ , i.e., we will check whether or not  $X \rightarrow v_i$  is satisfied by  $r(U)$  for each  $v_i \in U - X$  using Theorem 7. For example, let  $U = \{A, B, C, D, E\}$  and  $AE$  be a candidate. Then,  $U - \{A, E\} = \{B, C, D\}$ , which indicates that for candidate  $AE$  only the FDs  $AE \rightarrow B, AE \rightarrow C,$

and  $AE \rightarrow D$  need to be checked. For the semi-lattice shown in Figure 3.1, the number of FDs that could be checked is 75, because  $n2^{n-1} = 5 \cdot 2^{5-1} - 5 = 75$ . However, by designing useful pruning rules, the number of FDs to be checked can be reduced since some FDs can be inferred from discovered FDs without checking them on data.

In general, four kinds of FDs can be inferred. First, using equivalence  $X \leftrightarrow Y$ , any FD  $S \rightarrow v_i$ , where  $Y \subset S$ , can be inferred from  $X \rightarrow v_i$  by Lemmas 2 and 3. Second, given  $Y^+ \subseteq X$ , using nontrivial closure  $Y^*$ , any FD  $X \rightarrow v_i$ , where  $v_i \in U - X$  can be inferred from  $X - Y^* \rightarrow v_i$  by Theorem 9. Third, given  $X^+$ , any FD  $S \rightarrow v_i$ , where  $v_i \in X$  and  $X \subset S$ , can be inferred from  $X \rightarrow v_i$  by Theorem 10. Fourth, given  $X \subset S$ , any FD  $S \rightarrow v_i$ , where  $v_i \in U - S$ , can be inferred from  $X \rightarrow v_i$  by Lemma 4. Thus, all these kinds of FDs can be discovered without checking whether or not they are satisfied by  $r(U)$ .

To simplify our discussion, when considering equivalent attributes  $X$  and  $Y$ , i.e.,  $X \leftrightarrow Y$ , we assume that the equivalence is written such that the set of attributes  $X$  is generated earlier than the set of attributes  $Y$ .

Lemmas 2 and 3 indicate that after an equivalence  $X \leftrightarrow Y$  has been found, no further sets of attributes containing  $Y$  need to be checked.

**Example 21** Suppose that we are given the relation  $r(U)$  shown in Table 3.3(a). By examining all entries for attributes  $A$  and  $D$ , we determine that FDs  $A \rightarrow D$  and  $D \rightarrow A$  are satisfied by  $r(U)$ . According to Definition 24,  $A \leftrightarrow D$  is satisfied by  $r(U)$ . By checking attributes  $A$ ,  $B$ , and  $C$  in all tuples in  $r(U)$ , we determine that  $AB \rightarrow C$  and  $BC \rightarrow A$  are satisfied by  $r(U)$ . Without Lemmas 2 and 3, we would then need to check all tuples in Table 3.3(a) again to determine whether

or not  $BD \rightarrow C$  and  $BC \rightarrow D$  are satisfied by  $r(U)$ . However, with Lemma 2,  $BD \rightarrow C$  can be inferred from  $A \leftrightarrow D$  and  $AB \rightarrow C$ . By Lemma 3,  $BC \rightarrow D$  can also be inferred from  $A \leftrightarrow D$  and  $BC \rightarrow A$ . Thus, using Lemmas 2 and 3, no further candidate FDs containing  $D$  need to be checked. If relation  $r(U)$  is only used for discovering functional dependencies, it may make sense to remove attribute  $D$  from the relation  $r(U)$ . The resulting relation, after removing attribute  $D$ , is shown in Table 3.3(b). Even if the attribute is not removed from the relation, it can be ignored in all subsequent processing.

Table 3.3: Effect of removing a redundant attribute  $D$ .

TID	A	B	C	D
$t_1$	0	0	0	1
$t_2$	0	1	0	1
$t_3$	0	2	0	1
$t_4$	0	3	1	1
$t_5$	4	1	1	2
$t_6$	4	2	1	2
$t_7$	0	0	0	1

(a) Original relation

TID	A	B	C
$t_1$	0	0	0
$t_2$	0	1	0
$t_3$	0	2	0
$t_4$	0	3	1
$t_5$	4	1	1
$t_6$	4	2	1
$t_7$	0	0	0

(b) Relation without attribute  $D$

In the context of finding all FDs satisfied by a relation  $r(U)$ , the following four pruning rules are defined for candidates  $X$  and  $Y$ , where  $X \neq \phi, Y \neq \phi$ , and  $X, Y \subset U$ .

**Pruning rule 1.** If  $X \leftrightarrow Y$  is satisfied by  $r(U)$ , then candidate  $Y$  can be deleted.

**Pruning rule 2.** If  $Y^+ \subseteq X$ , then candidate  $X$  can be deleted.

**Pruning rule 3.** Given  $X^*$  and  $Y^*$ , then when attempting to determine whether or not the set of FDs  $XY \rightarrow v_i$ , where  $v_i \in U - XY$ , is satisfied by  $r(U)$ , only the set of FDs  $XY \rightarrow v_i$ , where  $v_i \in U - X^+Y^+$ , needs to be checked in  $r(U)$ .

**Pruning rule 4.** If  $\forall v_i \in U - X$ ,  $X \rightarrow v_i$  is satisfied by  $r(U)$ , then candidate  $X$  can be deleted.

These four pruning rules are used to delete candidates at level  $k - 1$  before generating candidates at level  $k$ . Pruning rule 1 is justified by Lemmas 2 and 3. This rule reduces the size of the search space by eliminating redundant candidates. Pruning rule 2 is justified by Theorem 9. This rule indicates that  $X \rightarrow Z$  can be inferred from  $X - Y^* \rightarrow Z$ . Pruning rule 3 is justified by Theorem 10. This rule indicates that for a candidate  $X$  and its superset  $S$ ,  $S \rightarrow X^*$  can be inferred given  $X^*$ . Thus, for candidate  $S$ , we need to check only the FDs  $S \rightarrow v_i, v_i \in U - S - X^*$ . Pruning rule 4 is justified by Lemma 4. This rule indicates that for any superset  $S$  of  $X$ ,  $S \rightarrow U - S$  can be inferred.

When no FDs are found in the relation, the checking required for the four pruning rules is an overhead that may increase the computation time. In Section 3.4, a series of experiments are reported that provide evidence that it is worthwhile to use these pruning rules for many datasets, since FDs are discovered that reduce the overall time cost.

### 3.3 The FD\_Mine Algorithm

In this section, the FD\_Mine algorithm for finding FDs in data is described. FD\_Mine combines a level-wise search, similar to that used in the Apriori algorithm [2], and the four pruning rules given in Section 3.2. In a level-wise search, results from level  $k$  are used to explore level  $k + 1$ . The FD\_Mine algorithm is shown in Figure 3.2. First, at level 1, the singleton candidates in  $U$  are stored in  $C_1$ . At level 2, each candidate  $v_i$  of  $C_1$  is used to generate all candidates of the form  $v_i v_j$ , where  $v_j \in C_1$

and  $v_i \neq v_j$ , which are stored in  $C_2$ . Then all FDs of the form  $v_i \rightarrow U - v_i$  are found and stored in  $F$ . At level 3, the candidate set  $C_2$  is used to generate the three attribute candidates, which are stored in  $C_3$ . All FDs of the form  $v_i v_j \rightarrow U - v_i v_j$  are checked and added to  $F$ . This procedure is repeated until  $C_k = \emptyset$ . In addition to storing a set of FDs in  $F$ , this algorithm also stores equivalent candidates in  $E$ .

In more detail, for each level, the `FD_Mine` algorithm works as follows. First, in line 6,  $k$  is incremented. All candidates at level  $k$  are generated in line 7, then the partitions of each candidate at level  $k$  are calculated in line 8 and their non trivial closures are initialized to the empty set in line 9. In line 10, for each  $X \in C_{k-1}$ , all FDs of the form  $X \rightarrow v_i$  are checked. The equivalent candidates are discovered in line 11. In line 12, candidates may be deleted from  $C_k$  by using the four pruning rules given in Section 3.2.

**FD\_Mine**( $r(U)$ )

Input: A relation  $r(U)$  over  $U = \{v_1, \dots, v_m\}$

Output: A set  $F$  of functional dependencies over  $r(U)$ .

1.  $F = \emptyset$ ;  $E = \emptyset$ ;  $C_1 = U$ ;  $k = 1$ ;
2.  $C_k = \text{CalculatePartition}(C_k, r(U))$ ;
3.  $C_k = \text{InitialClosure}(C_k)$ ;
4. **while** ( $|C_k| > 0$ )
5. {
6.      $k = k + 1$ ;
7.      $C_k = \text{Apriori-Gen}(C_{k-1})$ ;
8.      $C_k = \text{CalculatePartition}(C_k, r(U))$ ;
9.      $C_k = \text{InitialClosure}(C_k)$ ;
10.      $F = F \cup \text{ObtainFDs}(C_{k-1})$ ;
11.      $E = E \cup \text{ObtainEquivalences}(C_{k-1}, F)$ ;
12.      $C_k = \text{Prune}(C_{k-1}, C_k, E)$ ;
13. };
14. **return** ( $F$ );

Figure 3.2: The `FD_Mine` algorithm.

The algorithms called by the `FD_Mine` algorithm are *CalculatePartition*, *Ini-*

*InitializeClosure*, *Apriori-Gen*, *ObtainFDs*, *ObtainEquivalences*, and *Prune*. The *CalculatePartition* algorithm accesses the relation to calculate the actual partition of each candidate  $X$  in  $C_k$ . We refer the reader to [41] for a thorough discussion of this algorithm. The *Apriori-Gen* algorithm, which is shown in Figure 2.2, generates all possible candidates in  $C_k$  from the candidates in  $C_{k-1}$ . More details concerning the *Apriori-Gen* algorithm can be found in [2]. The remaining algorithms are given in Figures 3.3 to 3.6.

The *InitializeClosure* algorithm, which is shown in Figure 3.3, initializes the nontrivial closure of  $X$  to the empty set. It is assumed that each candidate  $X$  has a *nontrivial closure* field, denoted  $X^*$ , for storing the nontrivial closure of  $X$ .

**InitializeClosure( $C_k$ )**

1. **for** each candidate  $X$  in  $C_k$
2.      $X^* = \emptyset$ ;
3. **return** ( $C_k$ );

Figure 3.3: The *InitializeClosure* algorithm of FD\_Mine.

The *ObtainFDs* algorithm, which is shown in Figure 3.4, checks the FDs of each candidate  $X$  in  $C_k$ . More precisely, FDs of the form  $X \rightarrow v_i$ , where  $v_i \in U - X^+$ , are checked by comparing partitions  $\prod_X$  and  $\prod_{Xv_i}$ . Theorem 7 guarantees the correctness of this method. It is assumed that each candidate  $X$  has a *partition* field, denoted  $\prod_X$ , for storing the partition of  $X$ .

**ObtainFDs**( $C_{k-1}$ )

1.  $F = \emptyset$ ;
2. **for** each candidate  $X$  in  $C_{k-1}$
3.     **for** each  $v_i \in U - X^+$      // by Pruning rule 3
4.         **if** ( $|\prod X| == |\prod_{Xv_i}|$ ) **then**
5.         {
6.              $X^* = X^* \cup \{v_i\}$ ;
7.              $F = F \cup \{X \rightarrow v_i\}$ ;     // by Theorem 7
8.         }
9. **return** ( $F$ );

Figure 3.4: The *ObtainFDs* algorithm of FD\_Mine.

The *ObtainEquivalences* algorithm, which is shown in Figure 3.5, uses Theorem 8 to obtain equivalent candidates from the discovered FDs.

**ObtainEquivalences**( $C_{k-1}, F$ )

1.  $E = \emptyset$ ;
2. **for** each candidate  $X$  in  $C_{k-1}$
3.     **for** each  $Y \rightarrow v_i \in F$
4.         **if** ( $X \subseteq Y^+$  and  $Y \subseteq X^+$ ) **then**
5.              $E = E \cup \{X \leftrightarrow Y\}$ ;     // by Theorem 8
6. **return** ( $E$ );

Figure 3.5: The *ObtainEquivalences* algorithm of FD\_Mine.

The *Prune* algorithm, which is shown in Figure 3.6, exploits the four pruning rules given in Section 3.2 to reduce the size of search space.

```

Prune( $C_{k-1}, C_k, E$ )
1.   for each  $S \in C_k$ 
2.     for each  $X \in C_{k-1}$ 
3.       if ( $X \subset S$ ) then
4.         {
5.           if ( $X \in \{Z \mid Y \leftrightarrow Z \in E\}$ ) then
6.             { delete  $S$  from  $C_k$ ; // by Pruning rule 1
7.               break;
8.             }
9.           if ( $X^* \subset S$ ) then
10.            { delete  $S$  from  $C_k$ ; // by Pruning rule 2
11.              break;
12.            }
13.             $S^* = S \cup X^*$ ; // by Pruning rule 3
14.            if ( $U == S \cup S^*$ ) then
15.              { delete  $S$  from  $C_k$ ; // by Pruning rule 4
16.                break;
17.              }
18.          }
19.   return ( $C_k$ );

```

Figure 3.6: The *Prune* algorithm of FD\_Mine.

We use the relation in Table 3.2 to provide an example of how FD\_Mine works.

**Example 22** At level 1,  $C_1$  is set to  $C_1 = \{A, B, C, D, E\}$  and the partition of each candidate in  $C_1$  is computed by *CalculatePartition*. For example, the partition of  $A$  is  $\prod_A = \{\{t_1, t_2, t_3, t_4, t_7\}, \{t_5, t_6\}\}$ . At level 2,  $C_2 = \{AB, AC, AD, AE, BC, BD, BE, CD, CE, DE\}$  is generated by *Apriori-Gen* and the partition of each candidate in  $C_2$  is computed by *CalculatePartition*. For example, the partition of  $AB$  is  $\prod_{AB} = \{\{t_1, t_7\}, \{t_2\}, \{t_3\}, \{t_4\}, \{t_5\}, \{t_6\}\}$ . Since  $|\prod_A| = |\prod_D| = |\prod_{AD}| = 2$ ,  $A^* = D$  and  $D^* = A$  are obtained in line 6 of *ObtainFDs*, and FDs  $A \rightarrow D$  and  $D \rightarrow A$  are discovered in line 7 of *ObtainFDs*. Using *ObtainEquivalences*, the equivalence  $A \leftrightarrow D$  is deduced. In the *Prune* algorithm, since  $A \leftrightarrow D$  holds, then by pruning rule 1, all candidates that include  $D$ , namely,  $AD, BD, CD$ , and  $DE$ ,



are removed from  $C_2$ , i.e.,  $C_2 = \{AB, AC, AE, BC, BE, CE\}$ . At level 3,  $C_3 = \{ABC, ABE, ACE, BCE\}$  is generated by *Apriori-Gen* and the partition of each candidate in  $C_3$  is computed by *CalculatePartition*. Since  $|\prod_{AB}| = |\prod_{ABE}| = 6$ ,  $|\prod_{BE}| = |\prod_{ABE}| = 6$ , and  $|\prod_{CE}| = |\prod_{ACE}| = 6$ , *ObtainFDs* adds candidate  $E$  to  $(AB)^*$  and adds candidate  $A$  to  $(BE)^*$  and  $(CE)^*$ . *ObtainFDs* also discovers FDs  $AB \rightarrow E$ ,  $BE \rightarrow A$ , and  $CE \rightarrow A$ , and adds them to  $F$ . Using *ObtainEquivalences*, the equivalence  $AB \leftrightarrow BE$  is deduced from  $AB \rightarrow E$  and  $BE \rightarrow A$ . In the *Prune* algorithm, since  $AB \leftrightarrow BE$  is satisfied by  $r(U)$ , then by pruning rule 1,  $ABE$ , and  $BCE$  are removed from  $C_3$ , i.e.,  $C_3 = \{ABC, ACE\}$ . Since  $CE \rightarrow A$  is satisfied by  $r(U)$ ,  $(CE)^+ = ACE$ . By pruning rule 2,  $ACE$  is removed from  $C_3$ , i.e.,  $C_3 = \{ABC\}$ . Since  $(AB)^* = \{D, E\}$ ,  $(AE)^* = \{D\}$ , and  $(BE)^* = \emptyset$ , then, in line 15 of the *Prune* algorithm,  $(ABC)^* = \{D, E\}$ , which means  $(ABC)^+ = U$ . Thus, by pruning rule 4,  $ABC$  is removed from  $C_3$  in line 18 of the *Prune* algorithm, i.e.,  $C_3 = \emptyset$ . As no other candidates remain when the *Prune* algorithm finishes, *FD\_Mine* halts.

The portion of the semi-lattice illustrated in Figure 3.7 shows the search space for *FD\_Mine* when applied to the relation shown in Table 3.2. Each node represents a combination of candidates. If an edge is shown between nodes  $X$  and  $X \cup v_i$ , then FD  $X \rightarrow v_i$  needs to be checked. A candidate is shown in bold face if its partition needs to be computed by accessing the relation. Hence, the number of edges is the number of FDs to be checked, and the number of candidates in bold face is the number of partitions to be computed by accessing the relation. Figure 3.7 shows that *FD\_Mine* checks 32 FDs and computes 19 partitions of candidates, while the semi-lattice shown in Figure 3.1 corresponds to checking 75 FDs and computing 31

partitions.

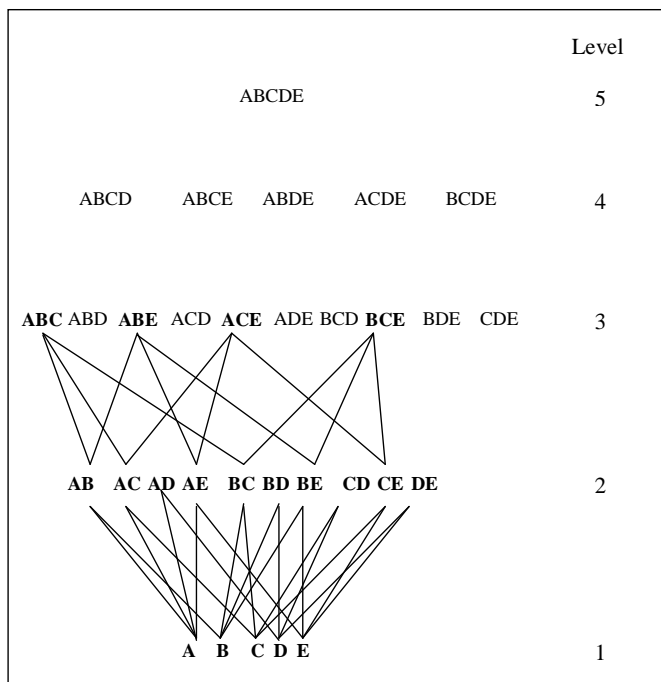


Figure 3.7: Portion of semi-lattice accessed by FD\_Mine.

The crucial point about Example 22 is that equivalences are discovered and used as the basis for pruning. First, the discovery at level 2 that FDs  $A \rightarrow D$  and  $D \rightarrow A$  are satisfied indicates that  $A \leftrightarrow D$  is also satisfied. Thus, all candidates involving  $D$  at levels 3 and higher do not need to be checked, as reflected in Figure 3.7. Secondly, the discovery at level 3 that FDs  $AB \rightarrow E$  and  $BE \rightarrow A$  are satisfied indicates that  $AB \leftrightarrow BE$  is also satisfied. Hence, all candidates that includes  $BE$  do not need to be checked at level 4. As shown in Figure 3.7, all FDs in the sample relation have already been discovered after checking level 3. In this case, FD\_Mine saves a considerable fraction of the computation compared to searching the semi-lattice shown in Figure 3.1.

**Theorem 11** *Let  $F$  be the set of FDs that are obtained by applying FD\_Mine to a relation  $r(U)$  and  $F^+$  be the closure of  $F$ . Then, an FD  $X \rightarrow v_i$  is satisfied by  $r(U)$  if and only if  $X \rightarrow v_i \in F^+$ .*

*Proof:* When FD\_Mine is applied to  $r(U)$ , the possible set of candidates includes all nonempty, proper subsets of  $U$ . For any candidate  $X$ , we first prove that if  $X \rightarrow v_i$  is satisfied by  $r(U)$  then  $X \rightarrow v_i \in F^+$ . Suppose  $X \rightarrow v_i$  is satisfied by  $r(U)$ . In this case,  $X \rightarrow v_i$  is either discovered by FD\_Mine or not. If  $X \rightarrow v_i$  is discovered by FD\_Mine, then  $X \rightarrow v_i \in F$  and it follows that  $X \rightarrow v_i \in F^+$ . If  $X \rightarrow v_i$  is not discovered by FD\_Mine, then according to lines 2 and 3 in *ObtainFDs*,  $X \notin C_k$  or  $v_i \notin U - X^+$ , which could only occur if candidate  $X$  is pruned by pruning rules 1, 2, or 4, or  $v_i$  is pruned by pruning rule 3. Since Theorems 8, 9, 10, and Lemma 4 guarantee that pruning rules 1, 2, 3, and 4, respectively, are correct, it follows that  $X \rightarrow v_i \in F^+$ . Now we prove that if  $X \rightarrow v_i \in F^+$  then  $X \rightarrow v_i$  is satisfied by  $r(U)$ . For any FD  $X \rightarrow v_i \in F^+$ , either  $X \rightarrow v_i \in F$  or  $X \rightarrow v_i \notin F$ . If  $X \rightarrow v_i \in F$ , then by Theorem 7, FD  $X \rightarrow v_i$  is satisfied by  $r(U)$ . Otherwise, if FD  $X \rightarrow v_i \notin F$ , then by Theorem 6,  $X \rightarrow v_i$  is satisfied by  $r(U)$ .  $\square$

The time complexity of the FD\_Mine algorithm depends on the number of attributes  $m$ , the number of tuples  $n$ , and the degree of correlation among the attributes. The time required varies for different relations, since the number and levels of the equivalent candidates and the FDs that are discovered vary for different relations. The worst case occurs when no FDs are found in the relation, and all combinations of the attributes are tested. The number of combinations of all attributes is  $C_1^m + C_2^m + \dots + C_m^m = 2^m - 1$ . Since the combination  $U = v_1 \dots v_m$  does not need to be tested, only  $2^m - 2$  candidates will be tested, but this small

constant is ignored in the complexity analysis. Thus, the worst case time complexity is  $O(n \cdot 2^m)$ , assuming that the partition of a candidate can be computed in time  $O(n)$ , as indicated in [41]. Suppose an equivalence  $X \leftrightarrow Y$  is discovered at level  $k$ ,  $k < m$ . Then all supersets of  $Y$  above level  $k$  will be eliminated by the pruning rules. Since the number of supersets of  $Y$  above level  $k$  is  $2^{m-k} - 1$ , then time complexity will be reduced to  $O(n \cdot (2^m - 2^{m-k}))$ . If more FDs or equivalent candidates are discovered in the relation, then more of the search space can be pruned by using the discovered information.

## 3.4 Experimental Results

In this section, experiments on fifteen UCI datasets [84] are summarized, and then a comparison between FD\_Mine and another algorithm called TANE [41] is presented.

### 3.4.1 Experimental Summary

FD\_Mine was applied to fifteen datasets obtained from the UCI Machine Learning Repository [84]. The results are summarized in Table 3.4, where column 4 represents the number of FDs that need to be checked on data, column 5 represents the number of discovered FDs, column 6 represents the number of discovered equivalences, and column 7 is the CPU time in seconds, measured on a SGI R12000 processor.

The timing results show that in practice the processing time is mainly determined by the number of attributes  $m$ , as expected from the  $O(n \cdot 2^m)$  theoretical complexity. For example, in the *Imports-85* dataset containing 26 attributes, the running time is long, even though it only has 205 tuples. The *Chess* dataset has nearly 30,000 tuples, but only 6 attributes, and its running time is much shorter.

Table 3.4: Experimental results for FD\_Mine on fifteen datasets.

Dataset Name	# of Attributes	# of Tuples	# of FD checked	# of FDs found	# of Equivalents	Time (secs)
Balance-scale	5	625	70	1	0	0
Iris	5	150	70	4	0	0
Chess	7	28,056	434	1	0	3
Abalone	8	4,177	594	60	8	1
Led	8	50	477	12	1	0
Nursery	9	12,960	2,286	1	0	16
Cancer-Wisconsin	10	699	4,562	19	0	1
Breast-cancer	10	191	5,095	3	0	0
Glass	10	142	405	11	1	0
Bridge	13	108	15,397	61	48	2
Echocardiogram	13	132	2,676	536	30	0
Crx	16	690	79,418	494	235	10
Pendigits	17	7,494	223,143	27,501	671	920
Hepatitis	20	155	1,161,108	7,381	4,862	1,327
Imports-85	26	205	2,996,737	3,971	19,888	8,322

### 3.4.2 Algorithm Comparison

A comparison between FD\_Mine, TANE [41], and an exhaustive search algorithm is given. The exhaustive search algorithm was obtained by using the FD\_Mine algorithm with the Prune function commented out. The search space for the exhaustive algorithm corresponds to the semi-lattice shown in Figure 3.1. The comparison with the exhaustive algorithm was used to confirm the correctness of the implementation and to place any efficiency gains in an overall context. The results indicate that FD\_Mine finds correct FDs, because every FD found by FD\_Mine was also found by the exhaustive search.

TANE was selected for comparison because work on TANE established the theoretical framework for the problem and TANE has been tested on an extensive set of UCI datasets [84]. For the relation given in Table 3.2, Figure 3.8 shows the portion of the semi-lattice accessed by TANE. The portions of semi-lattices shown in Figure 3.7 for FD\_Mine and Figure 3.8 for TANE both have fewer edges than the

semi-lattice shown in Figure 3.1. In addition, the portion of the semi-lattice accessed by FD\_Mine has fewer edges than that accessed by TANE. TANE checks 55 FDs and computes 30 partitions of candidates, which is more than the 32 FDs checked and 19 partitions computed by FD\_Mine.

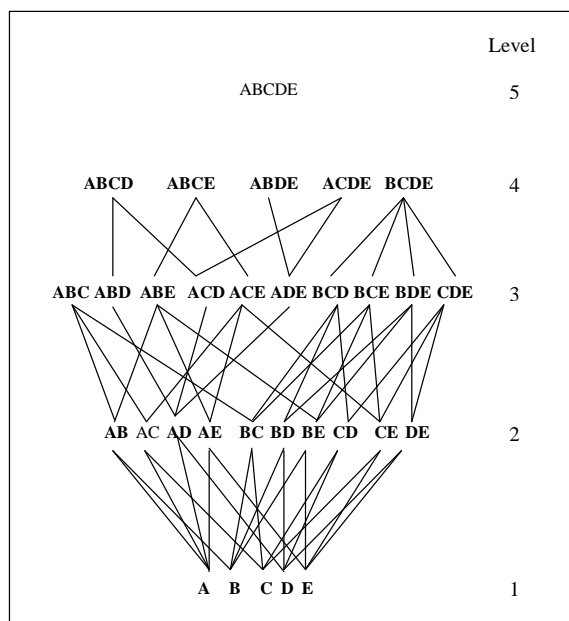


Figure 3.8: Portion of the semi-lattice accessed by TANE.

Table 3.5 further compares the number of FDs checked on this sample relation at each level. FD\_Mine checks fewer FDs at level 3, since candidates  $AD$ ,  $BD$ ,  $CD$ ,  $ED$ ,  $ABD$ ,  $ACD$ ,  $ADE$ ,  $BCD$ ,  $BDE$ , and  $CDE$  are removed after  $A \leftrightarrow D$  is found to hold. At level 4, FD\_Mine does not check any FDs, while TANE checks 11 FDs.

Table 3.6 compares the number of FDs that are checked by FD\_Mine, TANE, and an exhaustive search for the same fifteen UCI datasets used in Table 3.4. For example, in the *Crx* dataset, FD\_Mine checks 79,418 FDs, while TANE checks 130,605 FDs, and the exhaustive search checks 524,272 FDs. In every case, FD\_Mine checks at most as many FDs as TANE. The "\*" in the table indicates that the

Table 3.5: Number of FDs checked for the sample relation in Table 3.2.

	FD_Mine	TANE
Level 1	0	0
Level 2	20	20
Level 3	12	24
Level 4	0	11
Total	32	55

exhaustive search ran out of memory on *Import-85*. It used up all 12GB of available memory while generating candidates at level 11. Recall that there are  $n2^{n-1} - n$  edges in the exhaustive search space. The experimental results conform to this analysis, because in each case, the exhaustive search checks  $n2^{n-1} - n$  FDs. We can predict for the *Imports-85* dataset that the number of FDs to be checked by the exhaustive algorithm would be 872, 415, 206. It is worth mentioning that as the number of attributes increases, FD\_Mine is relatively more useful than the other methods. One reason for this is that more candidates can be pruned due to using discovered equivalences, as shown in Table 3.6.

Table 3.6: The number of FDs checked in the UCI datasets.

Dataset Name	# of Attributes	# of Tuples	# of FDs Checked		
			FD_MINE	TANE	Exhaustive
Abalone	8	4177	594	594	1,016
Balance-scale	5	625	70	70	75
Breast-cancer	10	191	5,095	5,095	5,110
Bridge	13	108	15,397	15,626	53,235
Cancer-Wisconsin	10	699	4,562	4,562	5,110
Chess	7	28,056	434	434	441
Crx	16	690	79,418	130,605	524,272
Echocardiogram	13	132	2,676	2,766	53,235
Glass	10	142	405	455	5,110
Hepatitis	20	155	1,161,108	1,272,789	10,485,740
Imports-85	26	205	2,996,737	3,564,176	*
Iris	5	150	70	70	75
Led	8	50	295	477	1,016
Nursery	9	12,960	2,286	2,286	2,295
Pendigits	17	7,494	223,143	227,714	1,114,095

Table 3.7 shows more detailed results for the *Imports-85* and *Hepatitis* datasets. At levels 1 through 3, both algorithms check approximately the same number of FDs, but at levels 4 through 14, FD\_Mine checks fewer FDs than TANE.

Table 3.7: Number of FDs checked on the *Imports-85* and *Hepatitis* Datasets.

LEVEL	FD_MINE	TANE
1	650	650
2	7,309	7,309
3	44,021	44,021
4	145,647	154,985
5	342,114	377,366
6	573,815	656,106
7	689,825	817,849
8	596,476	731,203
9	370,399	469,283
10	164,187	216,510
11	50,794	71,000
12	10,270	15,703
13	1,176	2,070
14	54	121
Total	2,996,737	3,564,176

(a) Imports-85 Dataset

LEVEL	FD_MINE	TANE
1	380	380
2	3,420	3,420
3	19,074	19,074
4	70,099	70,484
5	165,906	172,024
6	262,905	279,461
7	284,865	313,801
8	207,780	237,601
9	102,808	122,133
10	34,332	42,300
11	8,098	10,210
12	1,320	1,735
13	116	160
14	5	6
Total	1,161,108	1,272,789

(b) Hepatitis Dataset

FD\_Mine finds at most as many FDs as the other methods. In every case, any FD not found by FD\_Mine can be inferred from the set  $F$  of discovered FDs using Armstrong’s Axioms. Table 3.8 compares the number of FDs that are found by FD\_Mine, TANE, and the exhaustive search for the same fifteen UCI datasets as in Tables 3.4 and 3.6.

### 3.5 Constructing a Bayesian Network

In this section, we describe a practical application of discovering FDs from data, namely, constructing a Bayesian network. We begin by reviewing Bayesian networks in Section 3.5.1. In Section 3.5.2, we describe the FD2BN algorithm for constructing a sound Bayesian network from the functional dependencies obtained by FD\_Mine.



Table 3.8: The number of FDs found in the UCI datasets UCI Datasets.

Dataset	FD_Mine	TANE	Exhaustive
Abalone	60	60	270
Balance-scale	1	1	1
Breast-cancer	3	3	8
Bridge	61	62	28,164
Cancer-Wisconsin	19	19	139
Chess	1	1	1
Crx	494	1,099	214,556
Echocardiogram	536	583	41,771
Glass	27	27	1,533
Hepatitis	7,381	8,250	6,639,417
Imports-85	3,971	4,176	*
Iris	4	4	5
Led	12	12	480
Nursery	1	1	1
Pendigits	27,501	29,934	703,082

Finally, in Section 3.5.3, we establish the correctness of the FD2BN algorithm and analyze its complexity.

### 3.5.1 Bayesian Networks

A *directed graph* is a pair  $G = (U, E)$ , where  $U$  is a finite set of *vertices* (variables)  $\{v_1, v_2, \dots, v_m\}$  and  $E \subseteq U \times U$  is a set of directed *edges*. A directed graph  $G = (U, E)$  is *acyclic*, if the transitive closure of  $E$  is irreflexive [1]. We refer to a directed graph that is acyclic as a *directed acyclic graph (DAG)*  $G$  and denote it as  $D = (U, E)$ . For variables  $v_i, v_j \in U$ , if there is a directed edge  $(v_i, v_j)$ , then  $v_i$  is called a *parent* of  $v_j$  and  $v_j$  is called a *child* of  $v_i$ . We write  $P_i$  for the set of all parents of  $v_i$ .

**Definition 26** A *joint probability distribution (JPD)* on  $dom(U)$  is a function  $p$  on  $dom(U)$  such that  $p$  satisfies the following two constraint conditions: (i)  $0 \leq p(u) \leq 1$ , for each configuration  $u \in dom(U)$ , and (ii)  $\sum_{u \in dom(U)} p(u) = 1.0$ .

A joint probability distribution can be represented as a relation  $r$ . The relation  $r$  representing the JPD  $p(U)$  has attributes  $U \cup p(U)$ , where the column labelled by  $p(U)$  stores the probability value. For example, the relation  $r$  representing a JPD  $p(U)$  on the set of variables  $U = \{v_1, v_2\}$  is shown in Figure 3.9.

Table 3.9: A joint probability distribution  $p(v_1, v_2)$ .

$v_1$	$v_2$	$p(v_1, v_2)$
0	0	0.12
0	1	0.18
1	0	0.46
1	1	0.24

**Definition 27** The *marginal probability distribution* on a subset of random variables  $Y \subset X \subseteq U$  is a function such that  $p(Y) = \sum_{X-Y} p(X)$ .

**Definition 28** [77]. Let  $v$  be a variable and  $X$  be a finite, possibly empty set of variables. A *conditional probability table (CPT)*, also known as a *conditional probability distribution*, for  $v$  given  $X$ , is a distribution, denoted  $p(v|X)$ , satisfying the condition: for each configuration  $x \in \text{dom}(X)$ ,  $\sum_{c \in \text{dom}(v)} p(v = c|X = x) = 1.0$ .

**Example 23** Let  $U = \{a, b, c, \dots, k\}$  be a set of binary variables. Eleven CPTs  $p(a)$ ,  $p(b|a)$ ,  $p(c)$ ,  $p(d|c)$ ,  $p(e|c)$ ,  $p(f|d, e)$ ,  $p(g|b, f)$ ,  $p(h|c)$ ,  $p(i|h)$ ,  $p(j|g, h, i)$  and  $p(k|g)$  are depicted in Table 3.10. Note that the missing conditional probabilities can be obtained by Definition 28; for instance,  $p(a = 0) = 1 - p(a = 1) = 1 - 0.504 = 0.496$  and  $p(b = 0|a = 0) = 1 - p(b = 1|a = 0) = 1 - 0.052 = 0.948$ .

Table 3.10: Eleven conditional probability tables (CPTs).

$a$	$p(a)$	$c$	$p(c)$	$d$	$e$	$f$	$p(f d, e)$	$g$	$h$	$i$	$j$	$p(j g, h, i)$		
1	0.504	1	0.577	0	0	1	0.710	0	0	0	1	0.178		
				0	1	1	0.193	0	0	1	1	0.565		
$a$	$b$	$p(b a)$	$c$	$d$	$p(d c)$	1	0	1	0.485	0	1	0	1	0.446
0	1	0.052	0	1	0.714	1	1	1	0.602	0	1	1	1	0.729
1	1	0.358	1	1	0.627					1	0	0	1	0.931
										1	0	1	1	0.582
$c$	$e$	$p(e c)$	$c$	$h$	$p(h c)$	$b$	$f$	$g$	$p(g b, f)$	1	1	0	1	0.403
0	1	0.383	0	1	0.214	0	0	1	0.027	1	1	1	1	0.222
1	1	0.286	1	1	0.651	0	1	1	0.123					
						1	0	1	0.898					
$h$	$i$	$p(i h)$	$g$	$k$	$p(k g)$	1	1	1	0.405					
0	1	0.104	0	1	0.593									
1	1	0.369	1	1	0.416									

**Definition 29** [18]. Any JPD of a ordered set of variables  $U = \{v_1, v_2, \dots, v_m\}$  can be expressed as a product of  $m$  CPTs of the form

$$p(v_1, v_2, \dots, v_m) = \prod_{i=1}^m p(v_i|U_i), \quad (3.1)$$

where  $U_i = \{v_1, \dots, v_{i-1}\}$ . Equation(3.1) is called the *chain rule factorization* of  $p(U)$  w.r.t. the ordering  $\langle v_1, \dots, v_m \rangle$ .

For instance, let  $p(U)$  be a JPD on  $U = \{a, b, c, d, \dots, k\}$ . The chain rule factorization of  $p(U)$  w.r.t. the ordering  $\langle a, b, c, d, \dots, k \rangle$  is  $p(U) = p(a) \cdot p(b|a) \cdot p(c|a, b) \cdot p(d|a, b, c) \cdot \dots \cdot p(k|a, b, c, d, e, f, g, h, i, j)$ .

**Definition 30** A *Bayesian network (BN)* [68] on  $U$  is a pair  $(D, C)$ .  $D$  is a directed acyclic graph on  $U$ .  $C$  is a set of CPTs defined as: for each variable  $v_i \in D$ , there is a CPT  $p(v_i|P_i)$  for  $v_i$  given its parents  $P_i$ .

**Example 24** Let  $U = \{a, b, c, \dots, k\}$  be a set of binary variables. A real-world Bayesian network for *coronary heart disease* (CHD) [32] on  $U$  is the DAG in Figure 3.9 together with the CPTs  $p(a)$ ,  $p(b|a)$ ,  $p(c)$ ,  $p(d|c)$ ,  $p(e|c)$ ,  $p(f|d, e)$ ,  $p(g|b, f)$ ,  $p(h|c)$ ,  $p(i|h)$ ,  $p(j|g, h, i)$  and  $p(k|g)$  that are depicted in Table 3.10. This screenshot was obtained by applying the software for semantic modelling jointree propagation developed in Chapter 4 to the CHD DAG given in [32].

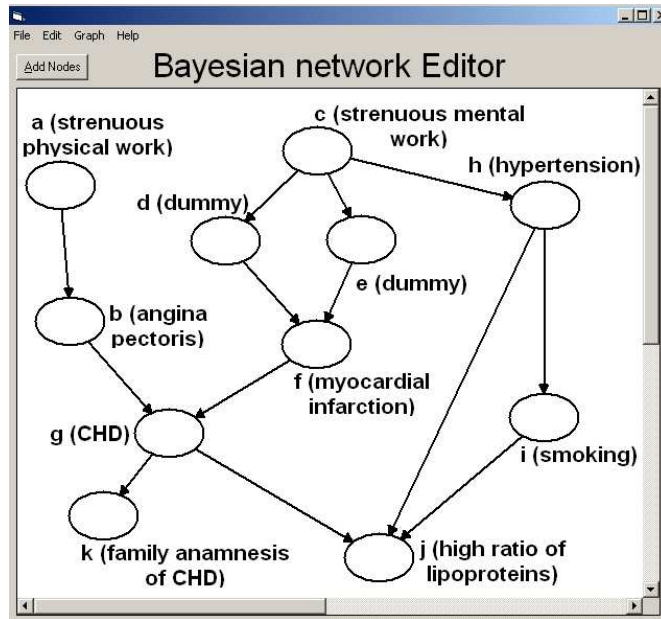


Figure 3.9: A directed acyclic graph (DAG) on  $U = \{a, b, c, d, e, f, g, h, i, j, k\}$  [32].

Given a Bayesian network, the product of the CPTs in  $C$  of the Bayesian network is a JPD. That is,

$$p(U) = \prod_{v_i \in U} p(v_i | P_i). \quad (3.2)$$

Equation (3.2) is called the *Bayesian factorization* of  $p(U)$  w.r.t. the DAG  $D$  [68].

For instance, the Bayesian factorization of  $p(U)$  w.r.t. the DAG in Figure 3.9 is a JPD on  $U = \{a, b, c, d, \dots, k\}$ , namely,  $p(U) = p(a) \cdot p(b|a) \cdot p(c) \cdot p(d|c) \cdot \dots \cdot p(k|g)$ .

**Definition 31** [85]. Let  $X$ ,  $Y$ , and  $Z$  be disjoint subsets of variables in  $U$ . We say  $X$  and  $Z$  are *conditionally independent* given  $Y$  under the joint probability distribution  $p$ , denoted  $I_p(X, Y, Z)$ , if  $p(X|Y, Z) = p(X|Y)$ , whenever the marginal probability distribution  $p(Y, Z) > 0$ . If  $Y = \emptyset$ , then we say  $X$  and  $Z$  are *unconditionally independent*, denoted  $I(X, \emptyset, Z)$ .

Henceforth, we may write  $I_p(X, Y, Z)$  as  $I(X, Y, Z)$  if the joint probability distribution  $p$  is understood.

The next theorem shows that functional dependency logically implies conditional independence.

**Theorem 12** [15]. Let  $r(U)$  be a relation over  $U$ . Let  $p(U)$  be a JPD over  $r(U)$ . Let  $X, Y \subseteq U$  and  $Z = U - XY$ . If  $r(U)$  satisfies the functional dependency  $X \rightarrow Y$ , then  $p(U)$  satisfies the conditional independence  $I(Y, X, Z)$ .

**Example 25** Given a JPD  $p(U)$  on  $U = \{a, b, c, d, \dots, k\}$ . If the FD  $ghi \rightarrow j$  is satisfied by  $r(U)$ , then  $I(j, ghi, abcdefk)$  is satisfied by  $p(U)$ .

### 3.5.2 The FD2BN Algorithm

We describe an algorithm, called FD2BN, to construct a Bayesian network from FDs obtained by FD\_Mine. A crucial part of FD2BN is the *ObtainOrdering* algorithm, which is shown in Figure 3.10. This algorithm finds an ordering  $\mathcal{O}$  of the variables of  $U$  such that given a set  $F$  of FDs, for any two different variables  $v_i, v_j \in U$ , if

$v_i$  is before  $v_j$  in  $\mathcal{O}$ , then the FD of the form of  $X \rightarrow v_i \in F$ , where  $v_j \in X$  and  $X \subseteq U$ , cannot be inferred from  $F$  using Armstrong's Axioms.

**ObtainOrdering**( $U, F$ ).

Input:  $U = \{v_1, \dots, v_m\}$ , and a  $F$  set of FDs.

Output: an ordered list  $\mathcal{O}$  of the variables of  $U$ .

```

1.   $\mathcal{O} = \langle \rangle$ ;
2.  while ( $F \neq \emptyset$ )
3.  {
4.    for each  $X \rightarrow v_i \in F$ 
5.      if (for all  $Y \rightarrow v_j \in F$  and  $v_i \neq v_j$ , such that  $v_i \notin Y$ ) then
6.        {
7.           $U = U - \{v_i\}$ ;
8.          prepend  $v_i$  to the head of  $\mathcal{O}$ ;
9.          remove any  $X \rightarrow v_i$  from  $F$ ;
10.       }
11.     if ( $F \neq \emptyset$ ) then
12.       {
13.         obtain a FD  $X \rightarrow v_i$  from  $F$ ;
14.         remove all  $Y \rightarrow v_j$  that satisfies  $v_i \in Y$  from  $F$ ;
15.       }
16.     }
17.   prepend  $U$  to the head of  $\mathcal{O}$ ;
18.   return( $\mathcal{O}$ )

```

Figure 3.10: The *ObtainOrdering* algorithm of FD2BN.

We use following example to illustrate how the *ObtainOrdering* algorithm works.

**Example 26** Suppose  $U = \{v_1, \dots, v_{13}\}$  and  $F = \{v_1v_5 \rightarrow v_3, v_1v_5 \rightarrow v_6, v_1v_5 \rightarrow v_{11}, v_1v_5 \rightarrow v_{13}, v_1v_8 \rightarrow v_7, v_4v_5v_9 \rightarrow v_2, v_1v_5v_{10} \rightarrow v_4, v_1v_2v_5 \rightarrow v_8, v_1v_5v_{10} \rightarrow v_9, v_1v_5v_{10} \rightarrow v_{12}\}$ . In line 4, the first FD,  $v_1v_5 \rightarrow v_3$ , is selected. In line 5,  $v_3$  is not in  $Y$  for any  $Y \rightarrow v_j \in F$ , where  $v_j \neq v_3$ . Therefore, in line 7, variable  $v_3$  is removed from  $U$ , in line 8,  $\mathcal{O} = \langle v_3 \rangle$  is obtained, and in line 9, FD  $v_1v_5 \rightarrow v_3$  is removed from  $F$ . As  $F$  is not empty, Algorithm *ObtainOrdering* repeats lines 4 to 10. The second FD,  $v_1v_5 \rightarrow v_6$ , is selected in line 4. Variable  $v_6$  is removed from  $U$  in line 7, and  $\mathcal{O}$

$= \langle v_6, v_3 \rangle$  is obtained in line 8. By repeatedly performing lines 4 to 10, an ordering  $\mathcal{O} = \langle v_9, v_4, v_2, v_{12}, v_8, v_7, v_{13}, v_{11}, v_6, v_3 \rangle$  is obtained and  $F$  becomes empty. Since  $F$  is empty, lines 11 to 15 are skipped, and the algorithm finishes the *while* loop. At the end of line 16,  $U = \{v_1, v_5, v_{10}\}$ . In line 17,  $U$  is prepended to the head of  $\mathcal{O}$  to give  $\mathcal{O} = \langle v_1, v_5, v_{10}, v_9, v_4, v_2, v_{12}, v_8, v_7, v_{13}, v_{11}, v_6, v_3 \rangle$ .

The FD2BN algorithm is shown in Figure 3.11. The algorithms called by the FD2BN algorithm are *FD\_Mine* and *ObtainOrdering*, which are presented in Figures 3.2 and 3.10, respectively.

**FD2BN**( $r(U)$ )

Input: A relation  $r(U)$  over variable set  $U$ .

Output: A DAG  $\mathcal{D}\langle U, E \rangle$  of a BN learned from  $r(U)$ .

1.  $F = \text{FD\_Mine}(r(U));$
2.  $\mathcal{O} = \text{ObtainOrdering}(U, F);$
3.  $U_i = \{\};$
4. **while**  $\mathcal{O}$  is not empty
5. {
6.  $v_i = \text{pophead}(\mathcal{O});$
7. **if**  $(\exists X \rightarrow v_i \in F \text{ and } X \in U_i)$  **then**
8.  $P_i = X;$
9. **else**
10.  $P_i = U_i;$
11.  $U_i = U_i \cup \{v_i\};$
12. }
13. Construct a DAG  $\mathcal{D}\langle U, E \rangle$  such that  $\{(b, v_i) \in E \mid b \in P_i, v_i \in U\};$
14. **return**( $\mathcal{D}\langle U, E \rangle$ )

Figure 3.11: The FD2BN algorithm.

We illustrate the FD2BN algorithm using the heart disease dataset obtained from the UCI Machine Learning Repository [84].

**Example 27** The UCI heart disease dataset contains thirteen attributes, i.e.,  $U = \{v_1, \dots, v_{13}\}$ , and 230 tuples. We apply FD2BN to the heart disease dataset as

follows. After applying FD\_Mine, the discovered set of FDs is  $F = \{v_1v_5 \rightarrow v_3, v_1v_5 \rightarrow v_6, v_1v_5 \rightarrow v_{11}, v_1v_5 \rightarrow v_{13}, v_1v_8 \rightarrow v_7, v_4v_5v_9 \rightarrow v_2, v_1v_5v_{10} \rightarrow v_4, v_1v_2v_5 \rightarrow v_8, v_1v_5v_{10} \rightarrow v_9, v_1v_5v_{10} \rightarrow v_{12}\}$ . The ordering  $\mathcal{O}$  is obtained in line 2, as shown in Example 26. Using ordering  $\mathcal{O}$ , we obtain  $P_1 = \{\}$ ,  $P_5 = \{v_1\}$ ,  $P_{10} = \{v_1, v_5\}$ ,  $P_9 = \{v_1, v_5, v_{10}\}$ ,  $P_4 = \{v_1, v_5, v_{10}\}$ ,  $P_2 = \{v_4, v_5, v_9\}$ ,  $P_{12} = \{v_1, v_5, v_{10}\}$ ,  $P_8 = \{v_1, v_5, v_2\}$ ,  $P_7 = \{v_1, v_8\}$ ,  $P_{13} = \{v_1, v_5\}$ ,  $P_{11} = \{v_1, v_5\}$ ,  $P_6 = \{v_1, v_5\}$ , and  $P_3 = \{v_1, v_5\}$ . By making each element of  $P_i$  a parent of  $v_i$ , the DAG depicted in Figure 3.12 can be constructed for the heart disease dataset. For example,  $P_2 = \{v_4, v_5, v_9\}$ , edges  $v_4 \rightarrow v_2$ ,  $v_5 \rightarrow v_2$ , and  $v_9 \rightarrow v_2$  are added to the DAG for variable  $v_2$  with parents  $v_4, v_5$ , and  $v_9$ .

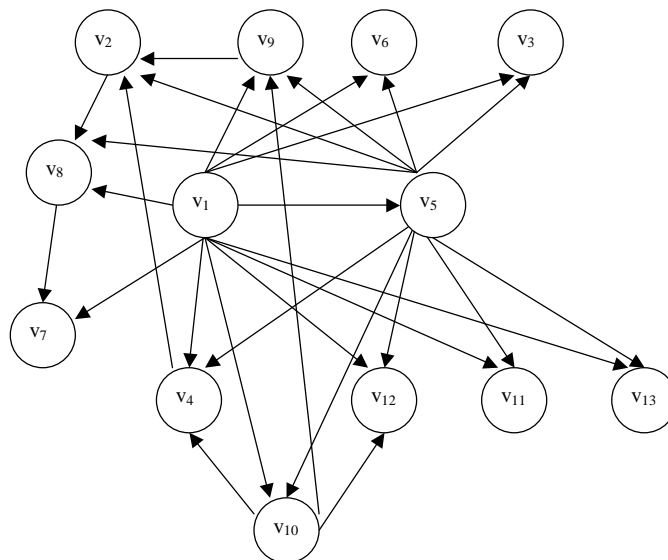


Figure 3.12: The DAG of a Bayesian network for the heart disease dataset.

### 3.5.3 Correctness and Complexity

In this subsection, we establish the correctness and complexity of FD2BN.



**Theorem 13** Given an input dataset  $r(U)$ , then the product of the CPTs of the constructed BN obtained using the FD2BN algorithm is  $p(U)$ , the joint distribution w.r.t.  $r(U)$ .

*Proof:* Suppose  $\mathcal{O} = \langle v_1, \dots, v_m \rangle$  and  $U_i = \{v_1, \dots, v_{i-1}\}$ . According to Definition 29, the JPD  $p(U)$  can be expressed as the following chain rule factorization w.r.t. the ordering  $\mathcal{O}$ .

$$\begin{aligned} p(U) &= p(v_1) \dots p(v_j | v_1, \dots, v_{j-1}) \dots p(v_m | v_1, \dots, v_{m-1}) \\ &= p(v_1) \dots p(v_j | U_j) \dots p(v_m | U_m) \end{aligned} \quad (3.3)$$

For any  $v_k \in P_i$ ,  $v_k$  is always before  $v_i$  in the ordering  $\mathcal{O}$  returned by *Obtain-Ordering*. It follows that the graph  $D$  obtained in line 13 is a DAG.

We now prove that this DAG is a DAG of a Bayesian network. Due to lines 7 to 10 of FD2BN, each  $P_i$  satisfies  $P_i = X$  or  $P_i = U_i$ . We first consider the case where  $P_i = X$ , which indicates that FD  $X \rightarrow v_i$  is satisfied by  $r(U)$  in line 7. By Theorem 12, the conditional independence  $I(v_i, X, U - X - v_i)$  is satisfied by  $p(U)$ . Since  $X \subseteq U_i \subseteq U$ ,  $I(v_i, X, U_i - X - v_i)$  holds by the decomposition axiom of conditional independencies [68]. This conditional independence indicates that the CPT  $p(v_i | v_1, \dots, v_{i-1})$  of  $v_i$  satisfies  $p(v_i | v_1, \dots, v_{i-1}) = p(v_i | X)$ . Thus, Equation (3.3) can be rewritten as:

$$p(U) = p(v_1) \dots p(v_j | U_j) p(v_{j+1} | P_{j+1}) \dots p(v_m | P_m). \quad (3.4)$$

Otherwise, consider the case where  $P_i = U_i$ . In this case, Equation (3.4) can be rewritten as:

$$p(U) = p(v_1|P_1) \dots p(v_j|P_j) \dots p(v_m|P_m) = \prod_{v_i \in U} p(v_i|P_i),$$

which is the Bayesian factorization of  $p(U)$  defined in Equation (3.2).  $\square$

Given a relation  $r(U)$  with  $n$  tuples and  $m$  attributes, the time complexity of the FD2BN algorithm depends mainly on the time complexity of FD\_Mine, which is  $O(n \cdot 2^m)$ . The time complexity of *ObtainOrdering* is  $O(p^3)$ , where  $p$  is the number of FDs in  $F$  returned by FD\_Mine. The loop from lines 4 to 12 of the FD2BN algorithm requires  $O(m \cdot p)$  time. As a result, the complexity of FD2BN is  $O(n \cdot 2^m + p^3 + m \cdot p)$ .

### 3.6 Summary

This chapter addressed the problem of discovering functional dependencies in a relation. We showed how to simplify the problem by finding equivalences among attributes and calculating the nontrivial closures of candidate sets of attributes, and then developed the FD\_Mine based on this approaches. We also described how to construct a sound Bayesian network from the FDs discovered by FD\_Mine. The experimental results on the UCI datasets show that the pruning rules used by FD\_Mine are valuable for the datasets examined, since they reduce the number of candidates and the overall number of FDs to be checked.

# Chapter 4

## A Jointree Probability

## Propagation Architecture for

## Semantic Modeling

Bayesian networks are well established as a model for representing and reasoning with uncertain information using probability. Although various methods exist for probabilistic inference directly in a Bayesian network [23, 48, 97], the task of probabilistic inference is usually carried out on a jointree constructed from a BN [42, 45, 54, 55, 78]. Shafer [76] explicitly stated that jointree propagation is central to the theory and practice of probabilistic expert systems. No study, however, has focused on identifying the exact information being passed between jointree nodes, or on the exact information remaining at each node after propagation finishes. In this chapter, we are interested in identifying the probability information that is passed during propagation and that which remains after propagation terminates. As a result, we propose a jointree probability propagation architecture for semantic

modelling, and demonstrate its usefulness with experimental results.

## 4.1 LAZY Propagation

In this section, we review LAZY propagation [54]. LAZY propagation is a significant advance in the study of jointree probability propagation. Based on the experimental results in [54], LAZY propagation is regarded as the most efficient probabilistic inference algorithm currently known.

The DAG of a Bayesian network (BN) is normally transformed through moralization and triangulation into a jointree on which the jointree propagation is applied. The task of constructing a jointree from the DAG of a BN has been extensively studied in probabilistic reasoning literature. We refer the reader to [66] for discussions on the detail of how to construct a jointree from a DAG.

**Definition 32** [68, 77] A *jointree* (*JT*) is a tree with sets of variables as nodes such that any variable in two nodes is also in any node on the path between the two. The *separator*  $S$  between any two neighbour nodes  $N_i$  and  $N_j$  in a jointree is  $S = N_i \cap N_j$ .

**Example 28** One possible jointree for the Bayesian network in Figure 3.9 is depicted in Figure 4.1 with circles for the nodes and rectangles for the separators. We label the nodes of this jointree as  $ab$ ,  $bfg$ ,  $cdefgh$ ,  $ghij$ , and  $gk$ . The separators are  $\{b\}$ ,  $\{f, g\}$ ,  $\{g, h\}$ , and  $\{g\}$ .

LAZY propagation maintains a structure of messages in the form of a multiplicative factorization of the potentials at each JT node and each JT separator when a node is ready to send its messages to a neighbour.

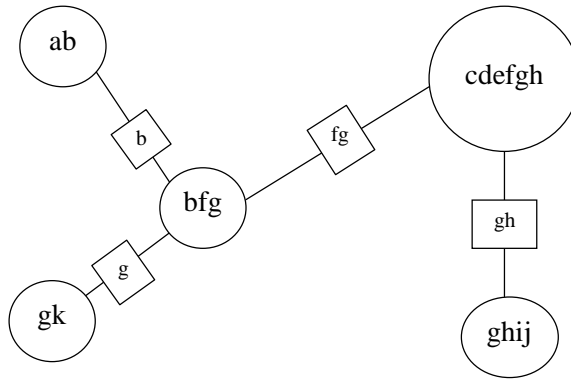


Figure 4.1: One possible jointree for the Bayesian network in Figure 3.9.

**Definition 33** [77] A *potential* on  $dom(X)$  is a function  $\phi$  on  $dom(X)$  such that  $\phi(x) \geq 0$ , for each  $x \in dom(X)$ , and at least one  $\phi(x)$  is positive.

For instance, five potentials  $\phi(b)$ ,  $\phi(f, g)$ ,  $\phi(g, h)$ ,  $\phi(f)$ , and  $\phi(g)$  are illustrated in Table 4.1.

Table 4.1: Five potentials  $\phi(b)$ ,  $\phi(f, g)$ ,  $\phi(g, h)$ ,  $\phi(f)$ , and  $\phi(g)$ .

b	$\phi(b)$	f	g	$\phi(f, g)$	g	h	$\phi(g, h)$	f	$\phi(f)$	g	$\phi(g)$
0	0.796	0	0	0.796	0	0	0.432	0	0.469	0	0.808
1	0.204	0	1	0.204	0	1	0.376	1	0.531	1	0.192
		1	0	0.819	1	0	0.103				
		1	1	0.181	1	1	0.089				

**Definition 34** Let  $\phi$  be a potential on  $U$  and  $X \subseteq U$ . Then the *marginal* of  $\phi$  onto  $X$ , denoted  $\phi(X)$ , is defined as: for each configuration  $x \in dom(X)$ ,

$$\phi(x) = \sum_{y \in dom(Y)} \phi(x, y),$$

where  $Y = U - X$ , and  $x, y$  is the configuration of  $U$  that we get by combining the configurations  $x$  of  $X$  and  $y$  of  $Y$ .

When a node is ready to send its messages to a neighbour, the structure of message is modeled to remove irrelevant potentials from the multiplicative factorization by exploiting barren variables [54] and independencies induced by evidence [54]. Given a query, a variable is a barren variable if it is neither an evidence nor a target variable and it only has barren descendants [54]. The independency relations induced by a set of evidence in a BN can be determined using the d-separation [54]. Next, physical computation is performed on the relevant potentials by marginalizing on relevant potentials. After LAZY propagation terminates, the potentials at each JT node  $N$  are a factorization of the marginal  $p(N)$  of  $p(U)$ . Example 29 illustrates the messages passed in LAZY and the probability information that remains after propagation.

**Example 29** Consider the *coronary heart disease* (CHD) [32] BN shown in Figure 3.9 and one possible JT with assigned CPTs shown in Figure 4.2. The distributions of the five potentials  $\phi(b)$ ,  $\phi(f, g)$ ,  $\phi(g, h)$ ,  $\phi(f)$ , and  $\phi(g)$  in Figure 4.2, passed by LAZY propagation, are illustrated in Table 4.1. After propagation, the marginal distribution at each node is:

$$\begin{aligned}
 p(a, b) &= p(a) \cdot p(b|a), \\
 p(b, f, g) &= p(g|b, f) \cdot \phi(b) \cdot \phi(f), \\
 p(c, d, e, f, g, h) &= p(c) \cdot p(d|c) \cdot p(e|c) \cdot p(f|d, e) \cdot p(h|c) \cdot \phi(f, g), \\
 p(g, h, i, j) &= p(i|h) \cdot p(j|g, h, i) \cdot \phi(g, h), \\
 p(g, k) &= p(k|g) \cdot \phi(g).
 \end{aligned}$$

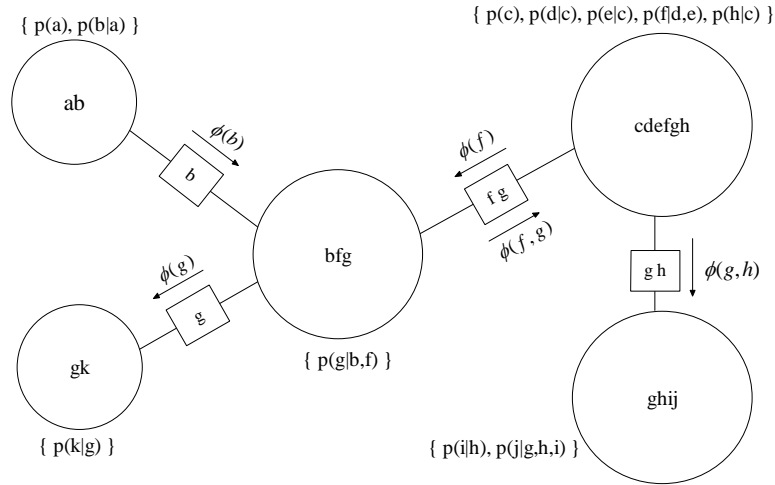


Figure 4.2: A jointree for the BN in Figure 3.9 with assigned CPTs.

We now make some remarks on LAZY propagation. From close inspection of the five potentials in in Table 4.1, it can be verified that the five potentials  $\phi(b)$ ,  $\phi(f, g)$ ,  $\phi(g, h)$ ,  $\phi(f)$ , and  $\phi(g)$  to be propagated by LAZY propagation are the respective CPTs  $p(b)$ ,  $p(g|f)$ ,  $p(g, h)$ ,  $p(f)$ , and  $p(g)$  of the joint distribution  $p(U)$ . Therefore, while LAZY propagation can efficiently compute the five distributions in Table 4.1, it does not clearly articulate the semantics of the messages being propagated between jointree nodes. Unlike a CPT, a potential [32] does not have clear physical meaning [18]. Consequently, by propagating messages in the form of potentials, the semantics of the probability tables being passed are lost. The reason that LAZY cannot articulate the semantics of the messages being propagated is that LAZY only uses the independencies that can identify irrelevant potentials, any remaining independencies in the relevant potentials are immaterial. By ignoring these independencies, LAZY propagation is not able to precisely articulate the probability

information being passed from one node to another. Passing potentials blurs the exact probability information remaining at each JT node when propagation finishes.

## 4.2 Modeling Inference Not Involving Evidence

To address the problems mentioned in Section 4.1, instead of developing yet another architecture for performing probabilistic inference, our architecture models probabilistic inference. We begin by introducing some relevant concepts.

Recall from Chapter 3, that given a DAG, if there is a directed edge  $(v_i, v_j)$ , then  $v_i$  is called a *parent* of  $v_j$  and  $v_j$  is called a *child* of  $v_i$  [18]. The set consisting of a variable and its parents is called the *family* of the variable [18]. For example, the family of variable  $f$  in the DAG of Figure 3.9 is  $\{f, d, e\}$ . In a DAG  $D = (U, E)$ , the *ancestral set*  $An(v_j)$  of a vertex  $v_j \in U$  is the set of all vertexes  $v_i$  such that  $(v_i, v_j)$  is a member of the transitive closure of  $E$  [1]. For instance, the ancestral set of variable  $f$  in the DAG of Figure 3.9 is  $\{c, d, e\}$ . The *ancestral set*  $An(X)$  for a set  $X \subseteq U$  of variables is defined as the union of the ancestral sets of the variables in  $X$ . For example, the ancestral set of variables  $\{c, e, f, h\}$  in the DAG of Figure 3.9 is  $An(cefh) = \{c, d, e\}$ . A numbering of the vertices in a DAG is called *ancestral* [18], if the number corresponding to any vertex is lower than the numbers corresponding to all of its children. For example, recall the DAG in Figure 3.9. One ancestral numbering of these vertexes is  $a = 1, b = 2, \dots, k = 11$ . For variable  $v_i$ , a *unity-potential*  $1(v_i)$  is a potential 1 assigning value 1.0 to each configuration of  $v_i$ . For a subset  $X \subseteq U$ , the unity-potential  $1(X)$  is defined as the product of the unity-potentials  $1(v_i)$ ,  $v_i \in X$ . Following the usage of [77], a set of rules for governing jointree propagation is called an architecture.



Our simple semantic architecture identifies the probability information being passed between JT nodes. It uses five rules, given below, for filling the storage registers in the JT separators with CPTs or unity-potential labels. The key to our architecture is the *IdentifyCPTMessages* (ICM) algorithm used in Rule 4.

**Rule 1.** For every variable in every separator, allocate two empty storage registers, one for a label in each direction.

**Rule 2.** Fix an ancestral numbering  $\prec$  of the variables in the BN.

**Rule 3.** Each node waits to identify its label(s) to a given neighbour until it has received labels from all its other neighbours.

**Rule 4.** When a node  $N_i$  is able to send its label(s) to a neighbour  $N_j$ , it calls the ICM algorithm, passing its assigned CPT labels and all CPT labels from its other neighbours, as well as the variables  $N_i - N_j$  to be eliminated.

**Rule 5.** For each CPT label  $p(v_k|P_k)$  returned by ICM, fill the storage register for variable  $v_k$  from  $N_i$  to  $N_j$  with label  $p(v_k|P_k)$ . For any variable  $v_l$  with a storage register from  $N_i$  to  $N_j$  still empty, fill the register with the unity-potential label  $1(v_l)$ .

We illustrate Rules 1-3 with the following example.

**Example 30** Consider the JT with assigned BN CPTs in Figure 4.3. By Rule 1, the separator  $gh$ , for instance, has four storage registers, which initially are empty.

For Rule 2, we fix the ancestral numbering as  $a \prec b \prec \dots \prec k$ . According to Rule 3, node  $bfgh$ , for example, can only send labels to node  $ab$  after it has received labels from both node  $cdefgh$  and node  $gk$ .

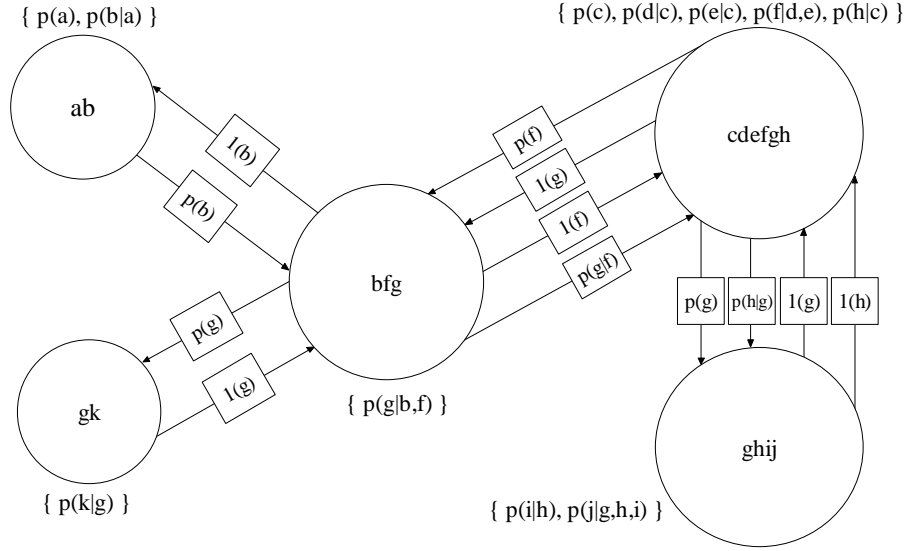


Figure 4.3: The probability information being passed between JT nodes.

The ICM algorithm is built upon the *FindRelevantCPTs* (FRC) and *MaintainCPTLabels* (MCL) algorithms.

**Definition 35** Given a set  $C$  of CPT labels and a variable  $v_i$  to be eliminated, the *FindRelevantCPTs* (FRC) algorithm returns the set  $C'$  of CPT labels in  $C$  involving  $v_i$ , where FRC first sorts the CPT labels in  $C'$  according to  $\prec$  in Rule 2, say  $C' = \{p(v_i|P_i), p(v_1|P_1), \dots, p(v_k|P_k)\}$ , where  $v_i \prec v_1 \prec \dots \prec v_k$ .

**Example 31** Suppose FRC is called with  $C = \{p(c), p(d|c), p(e|c), p(f|d,e), p(h|c)\}$  and variable  $d$  is to be eliminated. By  $\prec$  in Example 30, FRC returns  $C' = \{p(d|c), p(f|d,e)\}$  and not  $C' = \{p(f|d,e), p(d|c)\}$ .

**Definition 36** To eliminate  $v_i$ , suppose FRC returns  $\{p(v_i|P_i), p(v_1|P_1), \dots, p(v_k|P_k)\}$ . Consider a variable  $v_j \in v_i v_1 \cdots v_k$ . The *parent-set* of  $v_j$  is  $P_j$ .

**Example 32** To eliminate  $c$ , suppose FRC returns  $\{p(c), p(e|c), p(f|c, e), p(h|c)\}$ . The parent-sets of variables  $c, e, f$ , and  $h$  are  $\{\}, \{c\}, \{c, e\}$ , and  $\{c\}$ , respectively.

**Definition 37** Suppose FRC returns  $\{p(v_i|P_i), p(v_1|P_1), \dots, p(v_k|P_k)\}$  for eliminating variable  $v_i$ . We call  $C_i = \{v_1, \dots, v_k\}$  the *child-set* of  $v_i$ , where  $v_1 \prec \dots \prec v_k$  in Rule 2.

**Example 33** To eliminate  $c$ , suppose FRC returns  $\{p(c), p(d|c), p(e|c), p(h|c)\}$ . Besides  $p(c)$ , variable  $c$  appears in the CPT labels for  $\{h, d, e\}$ . By  $\prec$  in Example 30,  $e \prec h$ , while  $d \prec e$ . By definition, the child-set of  $v_i = c$  is  $C_i = \{v_1 = d, v_2 = e, v_3 = h\}$ .

To eliminate  $v_i$ , given that FRC returns  $\{p(v_i|P_i), p(v_1|P_1), \dots, p(v_k|P_k)\}$ , those CPT labels  $p(v_1|P_1), \dots, p(v_k|P_k)$  of the variables  $v_j \in C_i$  are modified. While variable  $v_i$  is deleted from  $P_j$ , only certain variables preceding  $v_j$  in  $\prec$  of Rule 2 may be added to  $P_j$ .

**Definition 38** To eliminate  $v_i$ , suppose FRC returns  $\{p(v_i|P_i), p(v_1|P_1), \dots, p(v_k|P_k)\}$ . Consider a variable  $v_j \in v_i v_1 \cdots v_k$ . The *family-set*, denoted  $F_j$ , of  $v_j$  is  $v_j P_j$ .

**Example 34** To eliminate  $c$ , suppose FRC returns  $\{p(c), p(e|c), p(f|c, e), p(h|c)\}$ . The family-sets of variables  $c, e, f$ , and  $h$  are  $\{c\}, \{c, e\}, \{c, e, f\}$ , and  $\{c, h\}$ , respectively.

**Definition 39** Given a set of CPT labels  $\{p(v_1|P_1), \dots, p(v_m|P_m)\}$ , the directed graph defined by these CPT labels, called the *CPT-graph*, has variables  $F_1 \dots F_m$ , and a directed edge from each variable in  $P_k$  to  $v_k$ ,  $k = 1, \dots, m$ .

**Example 35** Consider the set of CPT labels  $\{p(c), p(e|c), p(f|c, e), p(h|c)\}$  in Example 34. The CPT-graph is shown in Figure 4.4.

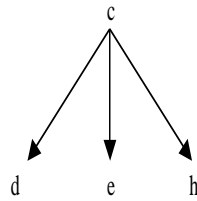


Figure 4.4: The CPT-graph for Example 35.

**Example 36** Consider another set of CPT labels  $\{p(a), p(c|a), p(d), p(e|b), p(g|d), p(h), p(j|c, d, e), p(k|c, d, e, h, j), p(l|c, d, e, h, i, j, k), p(m|j)\}$ . The CPT-graph is shown in Figure 4.5. Note that variables  $b$  and  $i$  do not have CPT labels in the given set.

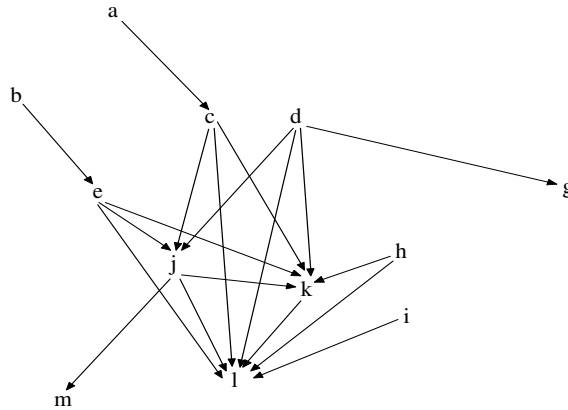


Figure 4.5: The CPT-graph defined by the CPT labels in Example 36.

**Definition 40** To eliminate  $v_i$ , suppose FRC returns  $\{p(v_i|P_i), p(v_1|P_1), \dots, p(v_k|P_k)\}$ . The child-set  $C_i = \{v_1, \dots, v_k\}$  is defined by  $\prec$  in Rule 2. For each variable  $v_j \in C_i$ , the *elder-set*  $E_j$  is defined as  $E_1 = F_i - v_i$ ,  $E_2 = (E_1 F_1) - v_i$ ,  $E_3 = (E_2 F_2) - v_i, \dots, E_k = (E_{k-1} F_{k-1}) - v_i$ .

For simplified notation, we will write  $(E_j F_j) - v_i$  as  $E_j F_j - v_i$ . Definition 40 says that the elder-set  $E_j$  of variable  $v_j$  in  $C_i$  is  $F_i$  together with the family-sets of the variables in  $C_i$  preceding  $v_j$ , with  $v_i$  subsequently removed.

**Example 37** Consider the set of CPT labels  $\{p(a), p(c|a), p(d), p(e|b), p(f|c, d), p(g|d), p(h), p(j|e, f), p(k|d, f, h), p(l|f, i), p(m|j)\}$ . The CPT-graph defined by these CPT labels is shown in Figure 4.6. Let us assume that  $\prec$  in Rule 2 orders this subset of variables in alphabetical order. Consider eliminating variable  $v_i = f$ . The call to FRC returns  $\{p(f|c, d), p(j|e, f), p(k|d, f, h), p(l|f, i)\}$ . With respect to  $\prec$  in Rule 2, the elder-set of the variables in the child-set  $C_i = \{v_1 = j, v_2 = k, v_3 = l\}$  are  $E_1 = \{c, d\}$ ,  $E_2 = \{c, d, e, j\}$ , and  $E_3 = \{c, d, e, h, j, k\}$ , respectively, as depicted in Figure 4.6.

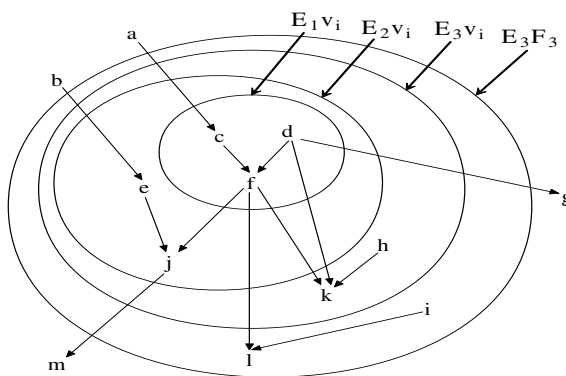


Figure 4.6: The elder-sets  $E_1, E_2$ , and  $E_3$  for the variables in the child-set  $C_i = \{v_1 = j, v_2 = k, v_3 = l\}$ .

The MCL algorithm can now be introduced.

**MCL**( $C'$ )

Input:  $C' = \{p(v_i|P_i), p(v_1|P_1), \dots, p(v_k|P_k)\}$  from FRC for eliminating  $v_i$

Output: the modified CPT labels for all  $k$  variables  $v_j$  in  $C_i = \{v_1, \dots, v_k\}$

//Determine the elder-set  $E_j$  and parent-set  $P_j$  for all  $k$  variables  $v_j \in C_i$

1. **for**  $j = k, \dots, 1$
2.      $P_j = (E_j P_j) - v_i$
3. **return**( $\{p(v_1|P_1), \dots, p(v_k|P_k)\}$ )

To simplify notation, we will write  $(E_j P_j) - v_i$  as  $E_j P_j - v_i$ .

**Example 38** To eliminate variable  $v_i = f$ , suppose the set of CPT labels obtained by FRC in Example 37 is passed to MCL. For  $\prec$  in Example 37, consider variable  $v_3 = l$ . By Definition 40,  $E_3 = \{c, d, e, h, j, k\}$  and  $P_3 = \{f, i\}$ . Then  $p(l|f, i)$  is adjusted to  $p(l|c, d, e, h, i, j, k)$ , as  $E_3 P_3 - v_i$  is  $\{c, d, e, h, i, j, k\}$ . Similarly,  $p(k|d, f, h)$  is changed to  $p(k|c, d, e, h, j)$ , and  $p(j|e, f)$  is modified to  $p(j|c, d, e)$ . By Definition 39, the CPT-graph defined by the CPT labels remaining after the elimination of variable  $f$  is shown in Figure 4.5.

**Lemma 5** *According to our architecture, let  $C'' = \{p(v_1|P_1), \dots, p(v_k|P_k)\}$  be the set of CPT labels returned by the MCL algorithm. Then the CPT-graph defined by  $C''$  is a DAG.*

*Proof:* For any BN CPT  $p(v_j|P_j)$ , by Rule 2,  $v \prec v_j$ , where  $v \in P_j$ . Moreover, by the definition of elder-set,  $v \prec v_j$ , where  $v \in E_j$ . Since the parent set  $P_j$  of  $v_j$  is modified as  $P_j E_j - v_i$ , for any CPT label  $p(v_j|P_j)$  returned by the MCL algorithm, it is still the case that  $v \prec v_j$ , where  $v \in P_j$ . Therefore, the CPT-graph defined by  $C''$  is acyclic, namely, it is a DAG.  $\square$

We now present the ICM algorithm.

**ICM**( $C, X$ )

Input: a set  $C$  of CPT labels,

the set  $X$  of variables to be eliminated from  $C$

Output: the set  $C$  of CPT labels sent from a node to a neighbour

1. **for** each variable  $v$  in  $X$
2. {
3.      $C' = FRC(C, v)$
4.      $C'' = MCL(C')$
5.      $C = (C - C') \cup C''$
6. }
7. **return**( $C$ )

We use two examples to illustrate the subtle points of our architecture.

**Example 39** Let us demonstrate how our architecture determines the CPT labels  $p(g)$  and  $p(h|g)$  sent from  $cdefgh$  to  $ghij$ , shown in Figure 4.3. By Rule 2, we use  $\prec$  in Example 30. By Rule 4,  $cdefgh$  has collected the CPT label  $p(g|f)$  from  $bfgh$ , but not  $1(f)$  as this is not a CPT label, and calls ICM with  $C = \{p(c), p(d|c), p(e|c), p(f|d, e), p(g|f), p(h|c)\}$  and  $X = \{c, d, e, f\}$ . For pedagogical purposes, let us eliminate the variables in the order  $d, c, e, f$ . ICM calls FRC, passing it  $C$  and variable  $d$ . FRC returns  $\{p(d|c), p(f|d, e)\}$ , which ICM initially assigns to  $C'$  and subsequently passes to MCL. Here  $v_i = d, C_i = \{v_1 = f\}, P_1 = \{d, e\}$ , and  $E_1 = \{c\}$ . As  $E_1 P_1 - v_i$  is  $\{c, e\}$ , MCL returns  $\{p(f|c, e)\}$ , which ICM assigns to  $C''$ . Next, the set  $C$  of CPT labels under consideration in ICM is adjusted to be  $C = \{p(c), p(e|c), p(f|c, e), p(g|f), p(h|c)\}$ . For variable  $c$ , the call to FRC results in  $C' = \{p(c), p(e|c), p(f|c, e), p(h|c)\}$ . To eliminate  $v_i = c$ , the elder-set of each variable in the child-set  $C_i = \{v_1 = e, v_2 = f, v_3 = h\}$  is  $E_1 = \{\}$ ,  $E_2 = \{e\}$ , and  $E_3 = \{e, f\}$ . Moreover, the parent-set of each variable in the child-set is  $P_1 = \{c\}$ ,  $P_2 = \{c, e\}$ , and  $P_3 = \{c\}$ . In MCL, then  $p(h|c)$  is adjusted to  $p(h|e, f)$ , as  $E_3 P_3 - v_i$  is  $\{e, f\}$ . Similarly,  $p(f|c, e)$  is changed to  $p(f|e)$  and  $p(e|c)$  is modified to  $p(e)$ . Therefore, MCL returns the set  $\{p(e), p(f|e), p(h|e, f)\}$  of CPT labels, which

ICM assigns to  $C''$ . Then  $C$  is updated to be  $C = \{p(e), p(f|e), p(g|f), p(h|e, f)\}$ . After eliminating variable  $e$ , the set of labels under consideration is modified to  $C = \{p(f), p(g|f), p(h|f)\}$ . Moreover, after considering the last variable  $f$ , the set of labels under consideration is  $C = \{p(g), p(h|g)\}$ . ICM returns  $C$  to  $cdefgh$ . By Rule 5,  $cdefgh$  places the CPT labels  $p(g)$  and  $p(h|g)$  in the storage registers of  $g$  and  $h$  from  $cdefgh$  to  $ghij$ , respectively.

**Example 40** Now let us show how our architecture determines the labels  $p(f)$  and  $1(g)$  sent from  $cdefgh$  to  $bfgh$ , shown in Figure 4.3. By Rule 2, we use  $\prec$  in Example 30. By Rule 4,  $cdefgh$  does not collect the unity-potential labels  $1(g)$  and  $1(h)$  from  $ghij$  and calls ICM with  $C = \{p(c), p(d|c), p(e|c), p(f|d, e), p(h|c)\}$ , and  $X = \{c, d, e, h\}$ . For simplicity, let us eliminate the variables in the order  $d, c, e, h$ . Similar to Example 39, the set of CPT labels under consideration, after the elimination of variables  $c, d, e$ , is  $C = \{p(f), p(h|f)\}$ . After eliminating the last variable  $h$ , the set of labels under consideration is  $C = \{p(f)\}$ . ICM returns  $C$  to  $cdefgh$ . By Rule 5,  $cdefgh$  places the CPT label  $p(f)$  in the storage register of  $f$  from  $cdefgh$  to  $bfgh$  and places the unity-potential label  $1(g)$  in the empty storage register for variable  $g$  from  $cdefgh$  to  $bfgh$ .

All of the identified CPT messages for Figure 4.3 are illustrated in Figure 4.7, which is a screen shot of our implemented system.



Sender	Receiver	CPT Message
ab	bfg	p(b)
bfg	cdefgh	p(g f)
bfg	gk	p(g)
cdefgh	bfg	p(f)
cdefgh	ghij	p(g)
cdefgh	ghij	p(h g)

Figure 4.7: Identification of the CPT messages in Figure 4.3.

**Example 41** Let us review Example 39 in terms of equations:

$$\sum_{c,d,e,f} p(c) \cdot p(d|c) \cdot p(e|c) \cdot p(f|d,e) \cdot p(g|f) \cdot p(h|c) \quad (4.1)$$

$$= \sum_f p(g|f) \cdot \sum_e \sum_c p(c) \cdot p(e|c) \cdot p(h|c) \cdot \sum_d p(d|c) \cdot p(f|d,e) \quad (4.2)$$

$$= \sum_f p(g|f) \cdot \sum_e \sum_c p(c) \cdot p(e|c) \cdot p(h|c) \cdot p(f|c,e) \quad (4.3)$$

$$= \sum_f p(g|f) \cdot \sum_e p(e) \cdot p(f|e) \cdot p(h|e,f)$$

$$= \sum_f p(g|f) \cdot p(f) \cdot p(h|f)$$

$$= p(g) \cdot p(h|g).$$

The derivation of  $p(g)$  and  $p(h|g)$  is not correct without independencies. For instance, the independencies  $I(d, c, e)$  and  $I(f, de, c)$  are necessary when moving from Equation (4.2) to Equation (4.3). In fact, without the unconditional independence  $I(b, \emptyset, f)$ , the message  $p(g|f)$ , from  $bfg$  to  $cdefgh$ , used in Equation (4.1), is not

correct either:

$$\sum_b p(b) \cdot p(g|b, f) = \sum_b p(b) \cdot \frac{p(b, f, g)}{p(b, f)} = \sum_b p(b) \cdot \frac{p(b, f, g)}{p(b) \cdot p(f)} = p(g|f).$$

Thus, the importance of showing that we can identify these independencies and utilize them, as shown above, is made obvious.

### 4.3 Complexity and Correctness

Here we establish the time complexity of the ICM algorithm and the correctness of our architecture for modeling inference not involving evidence.

**Lemma 6** *Let  $n$  be the number of CPT labels in  $C'$  given as input to the MCL algorithm for eliminating variable  $v_i$ . The time complexity of MCL is  $O(n)$ .*

*Proof:* Let  $\{p(v_i|P_i), p(v_1|P_1), \dots, p(v_{n-1}|P_{n-1})\}$  be the input set  $C'$  of CPT labels given to MCL. The elder-sets  $E_1, \dots, E_{n-1}$  and parent-sets  $P_1, \dots, P_{n-1}$  can be obtained in one pass over  $C'$ . Given that  $C'$  has  $n$  labels, the time complexity to determine the required parent-sets and elder-sets is  $O(n)$ . Similarly, for each label in  $\{p(v_1|P_1), \dots, p(v_{n-1}|P_{n-1})\}$ , the parent-set is modified exactly once. Hence, this *for-loop* executes  $n - 1$  times. Therefore, the time complexity of MCL is  $O(n)$ .  $\square$

Our main complexity result, given in Theorem 14, is that the CPT labels sent from a JT node can be identified in polynomial time.

**Theorem 14** *In the input to the ICM algorithm, let  $n$  be the number of CPT labels in  $C$ , and let  $X$  be the set of variables to be eliminated. The time complexity of ICM is  $O(n^2 \log n)$ .*

*Proof:* As the number of CPT labels in  $C$  is  $n$ , the maximum number of variables to be eliminated in  $X$  is  $n$ . The loop body is executed  $n$  times, once for each variable in  $X$ . Recall that the loop body calls the FRC and MCL algorithms. Clearly, FRC takes  $O(n)$  time to find those CPT labels involving  $v_i$ . According to  $\prec$  in Rule 2, these CPT labels can be sorted using the merge sort algorithm in  $O(n \log n)$  time [21]. Thus, FRC has time complexity  $O(n \log n)$ . By Lemma 6, MCL has time complexity  $O(n)$ . Thus, the loop body takes  $O(n \log n)$  time. Therefore, the time complexity of the ICM algorithm is  $O(n^2 \log n)$ .  $\square$

We now turn to the correctness of our architecture.

A graphical procedure, namely *d-separation* [68], has been developed to identify all the conditional independencies that are encoded in a DAG  $D$ . We refer the reader to [68] for a thorough discussion on the details of d-separation. While Pearl [68] uses the d-separation method for testing separation in DAGs, Lemma 7 shows how d-separation in DAGs can be more simply viewed as conventional separation in undirected graphs. Lemma 7 involves the notion of moralization. The *moralization* of a DAG  $D$  is the undirected graph defined by a three step process: (i) copy  $D$  as  $D'$ ; (ii) add an edge  $(v_i, v_j)$  to  $D'$ , if  $v_i$  and  $v_j$  have a common child in  $D'$ ; and (iii) drop the directionality of all edges in  $D'$ .

**Lemma 7** [46] *Let  $X, Y$  and  $Z$  be three disjoint subsets of vertexes in a DAG  $D$ . Let  $D'$  be the sub-DAG of  $D$  restricted to the vertexes in  $XYZ \cup An(XYZ)$ . Then  $Y$  d-separates  $X$  and  $Z$  in  $D$  if and only if  $Y$  separates  $X$  and  $Z$  in the moralization of  $D'$ .*

The next result shows that certain independencies involving elder-sets hold in any BN.

**Lemma 8** *Let  $(D, C)$  be a BN defining a JPD  $p(U)$ . Given the set  $C$  of CPT labels and any variable  $v_i \in D$ , the FRC algorithm returns  $\{p(v_i|P_i), p(v_1|P_1), \dots, p(v_k|P_k)\}$ , where the variables in the child-set  $C_i = \{v_1, \dots, v_k\}$  are written according to  $\prec$  in Rule 2. For each  $v_j \in C_i$  with elder-set  $E_j$ , the independencies  $I(v_i, E_j, P_j - v_i)$  and  $I(v_j, P_j, E_j)$  hold in the JPD  $p(U)$ .*

*Proof:* Observe that the parents, children, and family of any variable  $v$  in  $D$  are precisely the parent-set, child-set, and family-set of  $v$ , which are defined by  $C'$ , respectively. Let us first show that  $I(v_j, P_j, E_j)$  holds for  $j = 1, \dots, k$ . Pearl [68] has shown that  $I(v, P_v, N_v)$  holds for every variable  $v$  in a BN, where  $N_v$  denotes the set of all non-descendants of  $v$  in  $D$ . By definition, no variable in the elder-set  $E_j$  can be a descendant of  $v_j$  in  $D$ . Thereby,  $E_j \subseteq N_j$ . By the *decomposition axiom* of probabilistic independence [68, 85], the JPD  $p(U)$  satisfying  $I(v_j, P_j, N_j)$  logically implies that  $p(U)$  satisfies  $I(v_j, P_j, E_j)$ .

We now show that  $I(v_i, E_j, P_j - v_i)$  holds, for  $j = 1, \dots, k$ . We write  $I(v_i, E_j, P_j - v_i)$  as  $I(v_i, E_j, P_j - v_i - E_j)$ . By Lemma 7, the ancestral set of the variables in  $I(v_i, E_j, P_j - v_i - E_j)$  is  $An(v_i) \cup An(E_j) \cup An(P_j - v_i - E_j)$ , which is the same as  $An(E_j P_j)$ , since  $v_i \in P_j$ . Note that  $v_j$ , and any child of  $v_i$  in  $D$  succeeding  $v_j$  in  $\prec$  of Rule 2, are not members in the set  $An(E_j P_j)$ . Hence, in the constructed sub-DAG  $D'$  of DAG  $D$  onto  $An(E_j P_j)$ , the only directed edges involving  $v_i$  are those from each variable in  $P_i$  to  $v_i$  and from  $v_i$  to every child preceding  $v_j$ . By Corollary 6 in [68],  $E_j$  is a Markov blanket of  $v_i$  in  $D'$ . By definition,  $I(v_i, E_j, U - E_j - v_i)$  holds in  $p(U)$ . By the decomposition axiom,  $p(U)$  satisfies  $I(v_i, E_j, P_j - v_i)$ .  $\square$

Let us first focus on the elimination of just one variable. We can maintain a CPT factorization after  $v_i \in X$  is eliminated, provided that the corresponding elder-set

independencies are satisfied by the JPD  $p(U)$ . Note that, as done previously, we write  $(E_j P_j) - v_i$ ,  $(E_j F_k) - v_i$ , and  $(E_j F_j) - v_i v_j$  more simply as  $E_j P_j - v_i$ ,  $E_j F_k - v_i$ , and  $E_j F_j - v_i v_j$ , respectively.

**Lemma 9** *Given the output  $\{p(v_i|P_i), p(v_1|P_1), \dots, p(v_k|P_k)\}$  of FRC to eliminate variable  $v_i$  in our architecture. By Rule 2,  $v_1 \prec \dots \prec v_k$  in the child-set  $C_i = \{v_1, \dots, v_k\}$  of  $v_i$ . For each variable  $v_j \in C_i$  with elder-set  $E_j$ , if the independencies  $I(v_i, E_j, P_j - v_i)$  and  $I(v_j, P_j, E_j)$  hold in the JPD  $p(U)$ , then*

$$\sum_{v_i} p(v_i|P_i) \cdot p(v_1|P_1) \cdot \dots \cdot p(v_k|P_k) = \prod_{j=1}^k p(v_j|E_j P_j - v_i).$$

*Proof:* We first show, by mathematical induction on the number  $k$  of variables in  $C_i$ , that the product of these relevant CPTs can be rewritten as follows:

$$p(v_i|P_i) \cdot p(v_1|P_1) \cdot \dots \cdot p(v_k|P_k) = p(v_i|E_k F_k - v_i) \cdot \prod_{j=1}^k p(v_j|E_j P_j - v_i).$$

(Basic step:  $j = 1$ ). By definition,  $E_1 = P_i$ . Thus,  $p(v_i|P_i) \cdot p(v_1|P_1) = p(v_i|E_1) \cdot p(v_1|P_1)$ . By definition,

$$p(v_i|E_1) \cdot p(v_1|P_1) = \frac{p(v_i E_1)}{p(E_1)} \cdot \frac{p(v_1 P_1)}{p(P_1)}. \quad (4.4)$$

When  $j = 1$ , by assumption,  $I(v_i, E_1, P_1 - v_i)$  and  $I(v_1, P_1, E_1)$  hold. We can apply these independencies consecutively by multiplying the numerator and denominator

of the right side of Equation (4.4) by  $p(E_1P_1 - v_i)$ , namely,

$$\begin{aligned}
\frac{p(v_iE_1)}{p(E_1)} \cdot \frac{p(v_1P_1)}{p(P_1)} &= \frac{p(v_iE_1) \cdot p(E_1P_1 - v_i)}{p(E_1) \cdot p(E_1P_1 - v_i)} \cdot \frac{p(v_1P_1)}{p(P_1)} \\
&= \frac{p(E_1P_1)}{p(E_1P_1 - v_i)} \cdot \frac{p(v_1P_1)}{p(P_1)} \\
&= \frac{p(v_1P_1E_1)}{p(E_1P_1 - v_i)}. \tag{4.5}
\end{aligned}$$

By definition,

$$\begin{aligned}
\frac{p(v_1P_1E_1)}{p(E_1P_1 - v_i)} &= \frac{p(E_1F_1)}{p(E_1F_1 - v_iv_1)} \\
&= p(v_iv_1|E_1F_1 - v_iv_1). \tag{4.6}
\end{aligned}$$

By the *product rule* [87] of probability, which states that  $p(X, Y|Z) = p(X|Y, Z) \cdot p(Y|Z)$  for pairwise disjoint subsets  $X$ ,  $Y$ , and  $Z$ , Equation (4.6) is expressed as:

$$\begin{aligned}
p(v_iv_1|E_1F_1 - v_iv_1) &= p(v_i|E_1F_1 - v_i) \cdot p(v_1|E_1F_1 - v_iv_1) \\
&= p(v_i|E_1F_1 - v_i) \cdot p(v_1|E_1P_1 - v_i). \tag{4.7}
\end{aligned}$$

By Eqs. (4.4) - (4.7),

$$p(v_i|P_i) \cdot p(v_1|P_1) = p(v_i|E_1F_1 - v_i) \cdot p(v_1|E_1P_1 - v_i). \tag{4.8}$$

(Inductive hypothesis:  $j = k - 1$ ,  $k \geq 2$ ). Suppose

$$p(v_i|P_i)p(v_1|P_1) \cdots p(v_{k-1}|P_{k-1}) = p(v_i|E_{k-1}F_{k-1} - v_i) \prod_{j=1}^{k-1} p(v_j|E_jP_j - v_i).$$

(Inductive step:  $j = k$ ). Consider the following product

$$p(v_i|P_i)p(v_1|P_1) \dots p(v_k|P_k) = p(v_i|P_i)p(v_1|P_1) \dots p(v_{k-1}|P_{k-1})p(v_k|P_k). \quad (4.9)$$

By the inductive hypothesis,

$$\begin{aligned} & (p(v_i|P_i)p(v_1|P_1) \dots p(v_{k-1}|P_{k-1})) \cdot p(v_k|P_k) \\ = & (p(v_i|E_{k-1}F_{k-1} - v_i) \prod_{j=1}^{k-1} p(v_j|E_jP_j - v_i)) \cdot p(v_k|P_k) \\ = & p(v_i|E_{k-1}F_{k-1} - v_i) \cdot p(v_k|P_k) \cdot \prod_{j=1}^{k-1} p(v_j|E_jP_j - v_i). \end{aligned} \quad (4.10)$$

Since  $E_k = E_{k-1}F_{k-1} - v_i$ , Equation (4.10) can be rewritten as

$$p(v_i|P_i)p(v_1|P_1) \dots p(v_k|P_k) = p(v_i|E_k)p(v_k|P_k) \prod_{j=1}^{k-1} p(v_j|E_jP_j - v_i). \quad (4.11)$$

It can easily be shown that

$$p(v_i|E_k) \cdot p(v_k|P_k) = p(v_i|E_kF_k - v_i) \cdot p(v_k|E_kP_k - v_i), \quad (4.12)$$

by following Eqs. (4.4) - (4.7) replacing  $j = 1$  with  $j = k$ . Substituting Equation (4.12) into Equation (4.11), the desired result follows:

$$\begin{aligned} & p(v_i|P_i) \cdot p(v_1|P_1) \dots p(v_k|P_k) \\ = & p(v_i|E_kF_k - v_i) \cdot p(v_k|E_kP_k - v_i) \cdot \prod_{j=1}^{k-1} p(v_j|E_jP_j - v_i) \\ = & p(v_i|E_kF_k - v_i) \cdot \prod_{j=1}^k p(v_j|E_jP_j - v_i). \end{aligned} \quad (4.13)$$

Having rewritten the factorization of the relevant CPTs, we now consider the elimination of variable  $v_i$  as follows:

$$\begin{aligned}
& \sum_{v_i} p(v_i | E_k F_k - v_i) \cdot \prod_{j=1}^k p(v_j | E_j P_j - v_i) \\
= & \prod_{j=1}^k p(v_j | E_j P_j - v_i) \cdot \sum_{v_i} p(v_i | E_k F_k - v_i) \\
= & \prod_{j=1}^k p(v_j | E_j P_j - v_i) \cdot 1.0 \\
= & \prod_{j=1}^k p(v_j | E_j P_j - v_i). \tag{4.14}
\end{aligned}$$

By Eqs. (4.13) and (4.14), we obtain the desired result

$$\sum_{v_i} p(v_i | P_i) \cdot p(v_1 | P_1) \cdot \dots \cdot p(v_k | P_k) = \prod_{j=1}^k p(v_j | E_j P_j - v_i). \quad \square \tag{4.15}$$

Equation (4.15) explicitly demonstrates the form of the factorization in terms of CPTs after variable  $v_i$  is eliminated. Hence, the right-side of Equation (4.15) is used in the MCL algorithm to adjust the label of the CPT for each variable in  $v_i$ 's child-set. That is,  $\prod_{j=1}^k$  corresponds to the *for-loop* construct running from  $j = k, \dots, 1$ , while  $p(v_j | E_j P_j - v_i)$  corresponds to the statement  $P_j = E_j P_j - v_i$  for one iteration of the *for-loop*.

**Example 42** Recall eliminating variable  $d$  in Equation (4.2). By Example 39,  $v_i = d$ ,  $C_i = \{v_1 = f\}$ ,  $P_1 = \{d, e\}$ , and  $E_1 = \{c\}$ . By Lemma 8,  $I(v_i, E_j, P_j - v_i)$  and  $I(v_j, P_j, E_j)$  hold, namely,  $I(d, c, e)$  and  $I(f, de, c)$ . By Lemma 9,  $\sum_d p(d|c) \cdot p(f|d, e)$  is equal to  $p(f|c, e)$ , as shown in Equation (4.3).



We now establish the correctness of our architecture by showing that our messages are equivalent to those of the SS architecture [76, 77]. Since their architecture computes physical distributions, while our architecture determines labels, let us assume that the label in each storage register of our architecture is replaced with its corresponding physical probability distribution.

**Theorem 15** *Given a BN  $D$  and a JT for  $D$ . Apply our architecture and also the SS architecture. For any two neighbouring JT nodes  $N_i$  and  $N_j$ , the product of the distributions in the storage registers from  $N_i$  to  $N_j$  in our architecture is the message in the storage register from  $N_i$  to  $N_j$  in the SS architecture.*

*Proof:* When a JT node  $N_i$  receives a message from a neighbour  $N_j$ , it is also receiving, indirectly, information from the nodes on the other side of  $N_j$  [77]. Thus, without a loss of generality, let the JT for  $D$  consist of two nodes,  $N_1$  and  $N_2$ . Consider the message from  $N_2$  to  $N_1$ . Let  $Z$  be those variables  $v_m$  of  $N_2$  such that the BN CPT  $p(v_m|P_m)$  is assigned to  $N_2$ . The variables to be eliminated are  $X = N_2 - N_1$ . Let  $Y = N_2 - (XZ)$ . By SS architecture [77], the SS message  $\phi(N_2 \cap N_1)$  from  $N_2$  to  $N_1$  is:

$$\begin{aligned}
\phi(N_2 \cap N_1) &= \sum_X \phi(N_2) \\
&= \sum_X 1(N_2) \cdot \prod_{v_m \in Z} p(v_m|P_m) \\
&= \sum_X 1(Y) \cdot 1(XZ) \cdot \prod_{v_m \in Z} p(v_m|P_m) \\
&= 1(Y) \cdot \sum_X \prod_{v_m \in Z} p(v_m|P_m).
\end{aligned}$$

Consider the first variable  $v_i$  of  $X$  eliminated from  $\prod_{v_m \in Z} p(v_m | P_m)$ . By Lemma 8, for each  $v_j \in C_i$ , the independencies  $I(v_i, E_j, P_j - v_i)$  and  $I(v_j, P_j, E_j)$  hold in  $p(U)$ . By Lemma 9, the exact form of the CPTs is known after the elimination of  $v_i$ . By [77], the product of all remaining probability tables in the entire JT is the marginal distribution  $p(U - v_i)$  of the original joint distribution  $p(U)$ . Let us more carefully examine the remaining CPTs in the entire JT. First, each variable in  $U - v_i$  has exactly one CPT. It follows from Lemma 5 that the CPT-graph defined by all CPT labels remaining in the JT is a DAG. Therefore, the CPTs for the remaining variables  $U - v_i$  are a Bayesian network defining the marginal distribution  $p(U - v_i)$  of the original JPD  $p(U)$ . By the definition of probabilistic conditional independence, an independence holding in the marginal  $p(U - v_i)$  necessarily means that it holds in the joint distribution  $p(U)$ . Thereby, we can recursively apply Lemmas 5, 8, and 9 to eliminate the other variables in  $X$ . The CPTs remaining from the marginalization of  $X$  from  $\prod_{v_m \in Z} p(v_m | P_m)$  are exactly the distributions of the CPT labels output by the ICM algorithm when called by  $N_2$  to eliminate  $X$  from  $C = \{p(v_m | P_m) \mid v_m \in Z\}$ . In our architecture, the storage registers from  $N_2$  to  $N_1$  for those variables in  $Y$  are still empty. Filling these empty storage registers with unity-potentials  $1(v_l)$ ,  $v_l \in Y$ , follows directly from the definition of the unity-potential  $1(Y)$ . Therefore, the product of the distributions in the storage registers from  $N_2$  to  $N_1$  in our architecture is the message in the storage register from  $N_2$  to  $N_1$  in the SS architecture.  $\square$

**Example 43** It can be verified that the identified CPTs shown in Figure 4.3 are correct. In particular, the SS message  $\phi(f, g)$  from  $cdefgh$  to  $bfgh$  is  $p(f) \cdot 1(g)$ .

**Corollary 1** *The marginal distribution  $p(N)$  for any JT node  $N$  can be computed*

by collecting all CPTs sent to  $N$  by  $N$ 's neighbours and multiplying them with those CPTs assigned to  $N$ .

## 4.4 Modeling Inference Involving Evidence

Pearl [68] emphasizes the importance of structure by opening his chapter on Markov and Bayesian networks with the following quote:

Probability is not really about numbers; it is about the structure of reasoning. – G. Shafer

In this section, we extend our architecture to *model* the processing of evidence and show that it can still identify CPT messages. Modeling the processing of evidence is faster than the physical computation needed for evidence processing. By allowing our architecture to take full responsibility for modeling structure, we empirically demonstrate that LAZY can finish its work sooner.

It is important to realize that the processing of evidence  $E = e$  can be viewed as computing marginal distributions [74, 77, 90]. For each JT node  $N$ , compute the marginal  $p(NE)$ , from whence  $p(N - E, E = e)$  can be obtained. The desired distribution  $p(N - E|E = e)$  can then be determined via normalization.

Rules 6 and 7 extend our architecture to *model* the processing of evidence.

**Rule 6.** Given evidence  $E = e$ . For each node  $N$  in the JT, set  $N = N \cup E$ . On this augmented JT, apply Rules 1-5 of our architecture for modeling inference not involving evidence.

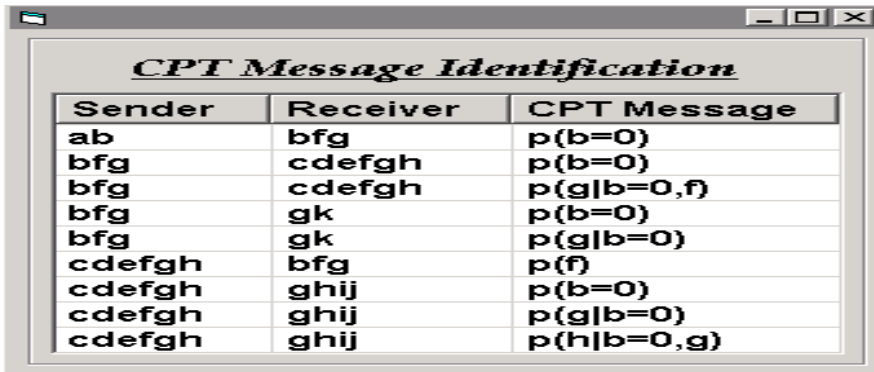
**Rule 7.** For each evidence variable  $v \in E$ , change each occurrence of  $v$  in an assigned or propagated CPT label from  $v$  to  $v = \varepsilon$ , where  $\varepsilon$  is the observed value of  $v$ .

We now show the correctness of our architecture for modeling inference involving evidence.

**Theorem 16** *Given a BN  $D$ , a JT for  $D$ , and observed evidence  $E = e$ . After applying our architecture, extended by Rules 6 and 7 for modeling inference involving evidence, the probability information at each JT node  $N$  defines  $p(N - E, E = e)$ .*

*Proof:* Apply Rule 6. By Corollary 1, the probability information at each node  $N$  is  $p(NE)$ . By selecting those configurations agreeing with  $E = e$  in Rule 7, the probability information at each node is  $p(N - E, E = e)$ .  $\square$

**Example 44** Consider evidence  $b = 0$  in the real-world BN for CHD in Figure 3.9. With respect to the CHD JT in Figure 4.3, the CPT messages to be propagated are depicted in Figure 4.8.

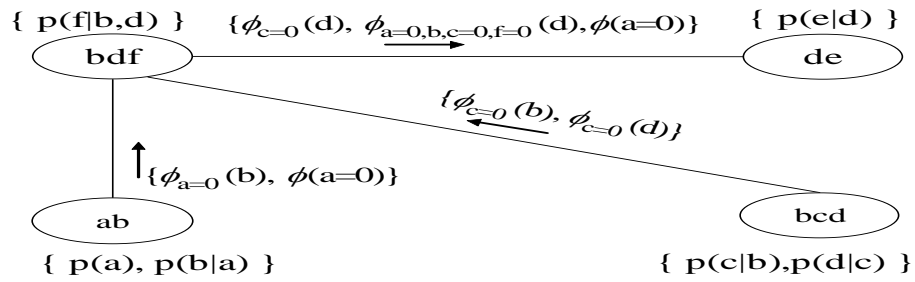


Sender	Receiver	CPT Message
ab	bfg	$p(b=0)$
bfg	cdefgh	$p(b=0)$
bfg	cdefgh	$p(g b=0, f)$
bfg	gk	$p(b=0)$
bfg	gk	$p(g b=0)$
cdefgh	bfg	$p(f)$
cdefgh	ghij	$p(b=0)$
cdefgh	ghij	$p(g b=0)$
cdefgh	ghij	$p(h b=0, g)$

Figure 4.8: The propagated CPT labels given evidence  $b = 0$ .

The next example involves three evidence variables.

**Example 45** Consider the JT with assigned BN CPTs in Figure 4.9 (top). Given evidence  $a = 0, c = 0, f = 0$ , the LAZY potentials propagated towards node  $de$  are shown [54]. In comparison, all CPT labels identified by our architecture are depicted in Figure 4.9 (bottom).



Sender	Receiver	CPT Message
ab	bdf	$p(a=0)$
ab	bdf	$p(b a=0)$
bcd	bdf	$p(c=0 b)$
bcd	bdf	$p(d c=0)$
bdf	ab	$p(c=0 b)$
bdf	ab	$p(f=0 b,c=0)$
bdf	bcd	$p(a=0)$
bdf	bcd	$p(b a=0)$
bdf	bcd	$p(f=0 b,d)$
bdf	de	$p(a=0)$
bdf	de	$p(c=0 a=0)$
bdf	de	$p(d c=0)$
bdf	de	$p(f=0 a=0,c=0,d)$

Figure 4.9: [54] A JT with assigned BN CPTs (top) and all identified CPT labels given evidence  $a = 0, c = 0, f = 0$  (bottom).

We can identify messages faster than they can be physically computed.

**Example 46** Given evidence  $b = 0$  in Example 44, physically constructing the distribution  $p(b = 0)$  for the message from node  $ab$  to  $bfg$  required 1.813 milliseconds. Identifying all CPT messages in Figure 4.8 required only 0.954 milliseconds.

Although semantic modeling can be done significantly faster than physical computation, the LAZY architecture applies these two tasks iteratively. The consequence, as the next two examples show, is that semantic modeling must wait while the physical computation catches-up.

**Example 47** [54] Consider the BN in Figure 4.10(a) and JT with assigned CPTs in Figure 4.10(b). Suppose evidence  $d = 0$  is collected. Before node  $bcdef$  can send its messages to node  $efg$ , it must first wait for the message  $\phi(b, c)$  to be physically constructed at node  $abc$  as:

$$\phi(b, c) = \sum_a p(a) \cdot p(b|a) \cdot p(c|a). \quad (4.16)$$

Upon receiving distribution  $\phi(b, c)$  at node  $bcdef$ , LAZY exploits the independence  $I(bc, d, ef)$  induced by the evidence  $d = 0$  to identify that  $\phi(b, c)$  is irrelevant to the computation for the messages to be sent from  $bcdef$  to  $efg$ .

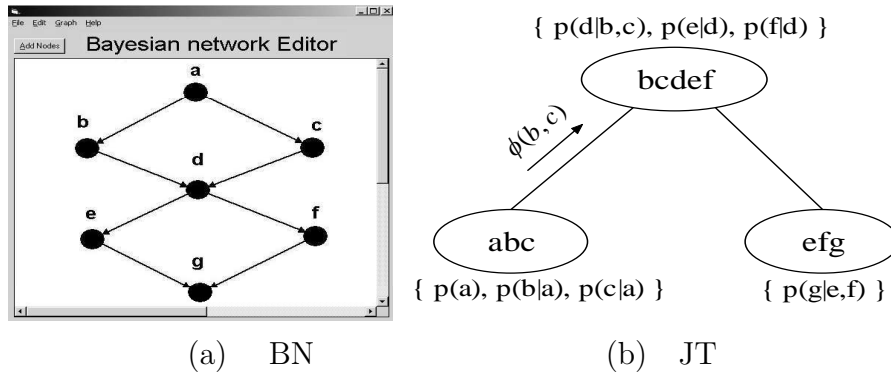


Figure 4.10: [54] A BN and a JT with assigned CPTs.

In the exploitation of independencies induced by evidence, Example 47 explicitly demonstrates that LAZY forces node  $bcdef$  to wait for the physical construction of

the irrelevant message  $\phi(b, c)$ .

**Example 48** [54] Consider the BN in Figure 4.11(a) and the JT with assigned CPTs in Figure 4.11(b). Before node  $acde$  can send its messages to node  $ef$ , it must first wait for the message  $\phi(c, d|a)$  to be physically constructed at node  $abcd$  as:

$$\phi(c, d|a) = \sum_b p(b|a) \cdot p(c|a, b) \cdot p(d|a, b). \quad (4.17)$$

Upon receiving distribution  $\phi(c, d|a)$  at  $acde$ , LAZY exploits barren variables  $c$  and  $d$  to identify that  $\phi(c, d|a)$  is irrelevant to the computation for the messages to be sent from  $acde$  to  $ef$ .

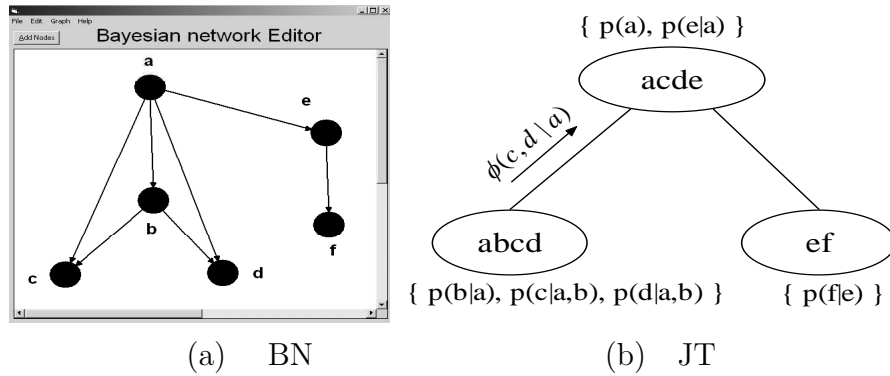


Figure 4.11: [54] A BN and a JT with assigned CPTs.

In the exploitation of barren variables, Example 48 explicitly demonstrates that LAZY forces node  $acde$  to wait for the physical construction of the irrelevant message  $\phi(c, d|a)$ . These unnecessary delays in Examples 47 and 48 are inherently built into the main philosophy of LAZY propagation [54]:

The bulk of LAZY propagation is to maintain a multiplicative [factorization of the CPTs] and to postpone combination of [CPTs]. This

gives opportunities for exploiting barren variables . . . during inference.  
. . . Thereby, when a message is to be computed only the required  
[CPTs] are combined.

The notion of a barren variable, however, is relative. For instance, in Example 48, variables  $c$  and  $d$  are not barren at the sending node  $abcd$ , but are barren at the receiving node  $acde$ . Thus, the interweaving of modeling structure and physical computation underlay these unnecessary delays.

We advocate the uncoupling of these two independent tasks. More specifically, we argue that modeling structure and physical computation should be performed separately. As our architecture can model structure faster than the physical computation can be performed, it can scout the structure in the JT and organize the collected information in three kinds of work schedules.

Our first work schedule pertains to non-empty messages that are irrelevant to subsequent message computation at the receiving node, as evident in Examples 47 and 48. More specifically, for three distinct nodes  $N_i$ ,  $N_j$  and  $N_k$  such that  $N_i$  and  $N_j$  are neighbours, as are  $N_j$  and  $N_k$ , our first work schedule indicates that the messages from  $N_i$  to  $N_j$  are irrelevant in the physical construction of the messages to be sent from  $N_j$  to  $N_k$ .

**Example 49** Given evidence  $d = 0$  in Example 47, the work schedule in Figure 4.12 indicates that the messages from node  $abc$  are irrelevant to node  $bcdef$  in the physical construction of the messages to be sent from  $bcdef$  to node  $efg$ . Now reconsider Example 48. The work schedule in Figure 4.13 indicates that the messages from node  $abcd$  are irrelevant to node  $acde$  in the physical construction of the messages to be sent from  $acde$  to node  $ef$ .



Node 1	Node 2	Node 3
abc	bcdef	efg

Figure 4.12: The irrelevant message from  $abc$  to  $bcdef$  given evidence  $d = 0$ .

Node 1	Node 2	Node 3
abcd	acde	ef

Figure 4.13: The irrelevant message from  $abcd$  to  $acde$ .

Our second work schedule indicates that an empty message will be propagated from a non-leaf node to a neighbour. (As LAZY could begin physical computation at the leaf JT nodes, we focus on the non-leaf JT nodes.) For instance, given evidence  $b = 0$  in the extended CHD BN and JT with assigned CPTs in Figure 4.14, the work schedule in Figure 4.15 indicates that  $bfg$  will send  $ab$  an empty message. Our second work schedule saves LAZY time.

**Example 50** Recall the extended CHD BN and JT in Figure 4.14. Given evidence  $b = 0$ , LAZY determines that  $bfg$  will send  $ab$  an empty message only after  $bfg$  receives the physical message  $\phi(f)$  from  $cdefgh$ . Physically constructing  $\phi(f)$  involves eliminating the four variables  $c, d, e, h$  from the five potentials  $p(c), p(d|c), p(e|c), p(f|d, e), p(h|c)$ . In less time than is required for this physical computation, our architecture has generated the work schedule in Figure 4.15. With-

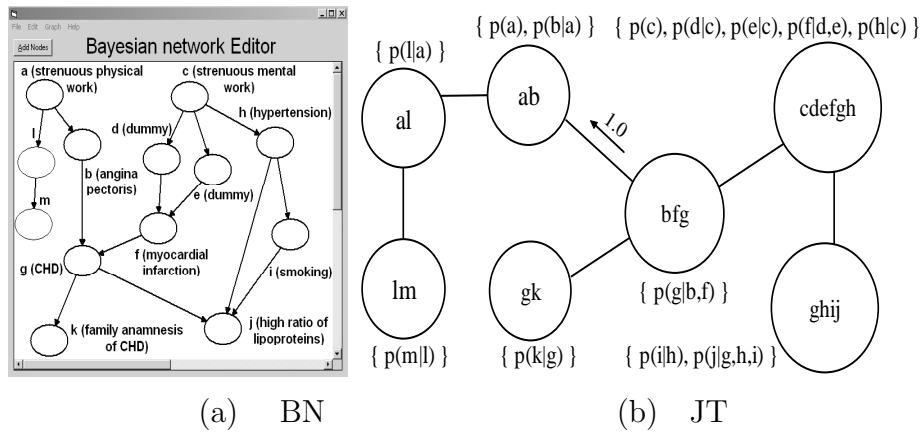


Figure 4.14: An extended CHD BN and a JT with assigned CPTs.

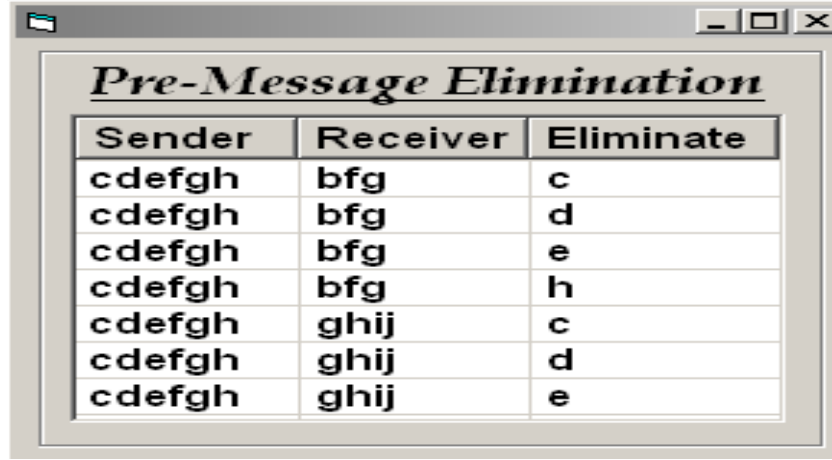
<i>Empty Message Identification</i>		
Sender	Receiver	Message
bfg	ab	1.0

Figure 4.15: All empty messages sent by non-leaf nodes.

out waiting for LAZY to eventually consider *bfg*, node *ab* can immediately send its messages  $\{p(a|b=0), p(b=0)\}$  to node *al*, which, in turn, can send its respective messages  $\{p(b=0), p(l|b=0)\}$  to node *lm*.

Last, but not least, our architecture can generate another type of beneficial work schedule. This third work schedule indicates the variables that can be eliminated at a non-leaf node with respect to the messages for a particular neighbour, before the sending node has received any messages. Given evidence  $b=0$  in the CHD JT of Figure 4.3, the work schedule in Figure 4.16 indicates that, for instance, non-leaf node *cdefgh* can immediately eliminate variables *c, d, e* in its construction of the message to *ghij*. Assisting LAZY to eliminate variables early saves time, as the

next example clearly demonstrates.



Sender	Receiver	Eliminate
cdefgh	bfg	c
cdefgh	bfg	d
cdefgh	bfg	e
cdefgh	bfg	h
cdefgh	ghij	c
cdefgh	ghij	d
cdefgh	ghij	e

Figure 4.16: The variables that can be eliminated at non-leaf nodes before any messages are received.

**Example 51** Given evidence  $b = 0$  in the CHD JT of Figure 4.3, consider the message from  $cdefgh$  to  $ghij$ . LAZY waits to eliminate variables  $c, d, e, f$  at  $cdefgh$  until  $cdefgh$  receives its messages  $\{\phi(b = 0), \phi_{b=0}(f, g)\}$  from  $bfg$ , which, in turn, has to wait for message  $\phi(b = 0)$  from  $ab$ . Thus, LAZY computes the message from  $cdefgh$  to  $ghij$  as:

$$\sum_{cdef} p(c) \cdot p(d|c) \cdot p(e|c) \cdot p(f|d, e) \cdot p(h|c) \cdot \phi(b = 0) \cdot \phi_{b=0}(f, g). \quad (4.18)$$

In contrast, the work schedule of Figure 4.16 allows LAZY to immediately eliminate variables  $c, d, e$  as:

$$\phi(f, h) = \sum_{c,d,e} p(c) \cdot p(d|c) \cdot p(e|c) \cdot p(f|d, e) \cdot p(h|c).$$

The benefit is that when the LAZY messages  $\{\phi(b = 0), \phi_{b=0}(f, g)\}$  from  $bf g$  are received, only variable  $f$  remains to be eliminated. That is to say, LAZY's Equation (4.18) is simplified to

$$\sum_f \phi(f, h) \cdot \phi(b = 0) \cdot \phi_{b=0}(f, g). \quad (4.19)$$

In our CHD example, LAZY will eliminate  $c, d, e$  twice at node  $cdefgh$ , once for each message to  $bf g$  and  $ghij$ . This duplication of effort is a well-known undesirable property in JT propagation [77]. Utilizing our third work schedule, such as in Figure 4.16, to remove this duplication remains for future work.

We conclude this section by providing some empirical results illustrating the usefulness of our jointree architecture. In short, we provide the time taken for LAZY propagation and the time saved by allowing our architecture to guide LAZY.

The source code for LAZY propagation was obtained from [26], as was the code for generating a jointree from a Bayesian network. Table 4.2 describes five real-world Bayesian networks and their corresponding jointree representations used in the experiments.

Table 4.2: Information about five Bayesian networks.

Bayesian Network	Number of Variables in the BN	Number of Jointree Nodes	Maximum Jointree Node Size
Alarm	37	85	5
CHD	11	24	4
Hailfinder	56	127	5
Insurance	27	64	9
Mildew	35	73	5

We first measure the time cost of performing propagation without evidence. Table 4.3 shows the time taken for : (i) LAZY propagation by itself, (ii) LAZY propagation guided by our architecture, (iii) time saved by using our approach, and (iv) time saved as a percentage. All propagation was timed in seconds using a SGI R12000 processor. Note that our architecture lowered the time taken for propagation in all five real-world Bayesian networks. The time percentage saved ranged from 11.76% to 69.26% with an average percentage of 37.02%.

Table 4.3: Experimental results on five Bayesian networks without evidence.

Bayesian Network	LAZY Propagation Time	LAZY Propagation Time Guided by our Architecture	Time Saved	Percentage of Time Saved
Alarm	0.758	0.233	0.525	69.26%
CHD	0.083	0.054	0.029	34.94%
Hailfinder	2.56	2.259	0.301	11.76%
Insurance	8.932	7.223	1.709	19.13%
Mildew	882.276	441.032	441.244	50.01%

Next, we measure the time cost of performing propagation involving evidence. In each BN, approximately twenty percent of the variables are randomly instantiated as evidence variables, such as was done for the largest BN used in the experimental results of [54]. Table 4.4 shows how time can be saved by allowing our architecture to guide LAZY in its propagation involving evidence. Note that once again our architecture lowered the time taken for propagation in all five real-world Bayesian networks. The time percentage saved ranged from 1.86% to 14.89% with an average percentage of 6.12%.

It is worth mentioning that our architecture is more useful with fewer evidence variables. One reason for this is that the cost of physical computation is lowered with

Table 4.4: Experimental results on five Bayesian networks with 18% of the variables randomly instantiated as evidence variables.

Bayesian Network	LAZY Propagation Time	LAZY Propagation Time Guided by our Architecture	Time Saved	Percentage of Time Saved
Alarm	0.579	0.545	0.034	5.87%
CHD	0.047	0.040	0.007	14.89%
Hailfinder	0.783	0.762	0.021	2.68%
Insurance	0.453	0.429	0.024	5.29%
Mildew	15.03	15.00	0.028	1.86%

collected evidence, since the probability tables to be propagated are much smaller than would be without evidence. For instance, LAZY takes over 882 seconds to perform propagation without evidence on the Mildew BN, but LAZY only takes about 15 seconds to perform propagation if 6 variables are instantiated as evidence variables. Therefore, LAZY needs the most help with little or no evidence and this is precisely when our architecture is most beneficial.

## 4.5 Local BNs and Practical Applications

After applying our architecture, as described in Section 4.2, to identify the probability information being passed in a JT, let us apply either method from [54, 97] to physically construct the identified CPT messages. That is, we assume that the label in each storage register of our architecture is replaced with its corresponding physical probability distribution. Unlike all previous JT architectures [42, 45, 54, 76], in our architecture, after propagation not involving evidence, each JT node  $N$  has a *sound, local* BN preserving all *conditional* independencies of the original BN involving variables in  $N$ . Practical applications of local BNs include an automated

modeling procedure for multiply sectioned BNs and a method for exploiting localized queries in direct computation techniques.

**Lemma 10** *Given a BN  $D$  and a JT for  $D$ , apply Rules 1-5 in our architecture. For any JT node  $N$ , the CPTs assigned to  $N$ , together with all CPTs sent to  $N$  from  $N$ 's neighbours, define a local BN  $D_N$ .*

*Proof:* As previously mentioned, when a JT node  $N_i$  receives a message from a neighbour  $N_j$ , it is also receiving, indirectly, information from the nodes on the other side of  $N_j$  [77]. Thus, without a loss of generality, let the JT for  $D$  consist of two nodes,  $N_1$  and  $N_2$ . We only need focus on variables in the separator  $N_1 \cap N_2$ . Consider a variable  $v_m$  in  $N_1 \cap N_2$  such that the BN CPT  $p(v_m|P_m)$  is assigned to  $N_1$ . Thus,  $v_m$  is without a CPT label with respect to  $N_2$ . In  $N_1$ 's call to ICM for its messages to  $N_2$ , variable  $v_m$  is not in  $X = N_1 - N_2$ , the set of variables to be eliminated, Thus, ICM will return a CPT label for  $v_m$  to  $N_1$ . By Rule 5,  $N_1$  will place this CPT label in the empty storage register for  $v_m$  from  $N_1$  to  $N_2$ . Therefore, after propagation, variable  $v_m$  has a CPT label at node  $N_2$ . Conversely, consider  $N_2$ 's call of ICM for its messages to  $N_1$ . Since the BN CPT  $p(v_m|P_m)$  is assigned to  $N_1$ , there is no CPT label for  $v_m$  passed to ICM. Thus, ICM does not return a CPT label for  $v_m$  to  $N_2$ . By Rule 5,  $N_2$  fills the empty storage register of  $v_m$  from  $N_2$  to  $N_1$  with the unity-potential label  $1(v_m)$ . Therefore, after propagation, both  $N_1$  and  $N_2$  have precisely one CPT label for  $v_m$ . Moreover, for each node, it follows from Lemma 5 that the CPT-graph defined by the assigned CPTs and the message CPTs is a DAG. By definition, each JT node  $N$  has a local BN  $D_N$ .  $\square$

**Example 52** Recall the JT with assigned CPTs in Figure 4.3. Each JT node has a local BN after propagation not involving evidence, as depicted by a screen shot of

our implemented system in Figure 4.17.

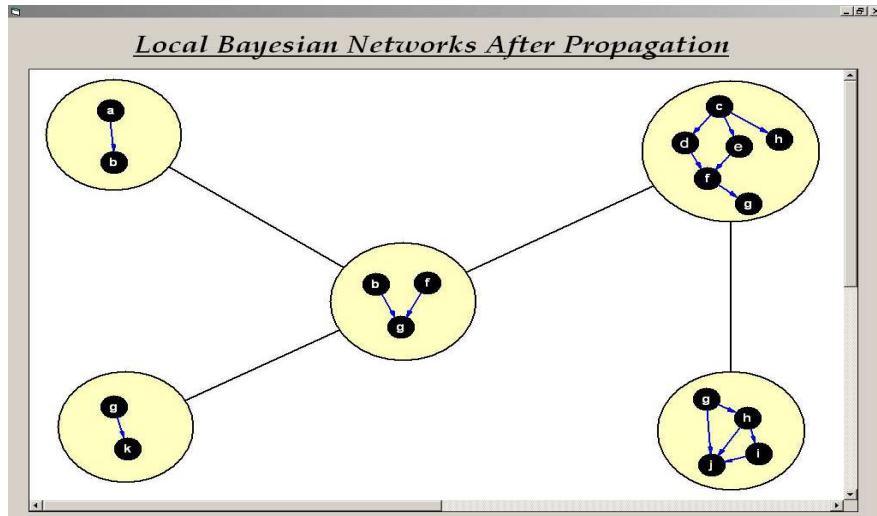


Figure 4.17: *Local* BNs after propagation not involving evidence for the CHD BN.

**Theorem 17** *Given a BN  $D$  and a JT for  $D$ , let  $D_N$  be the local BN for a JT node  $N$ . If an independence  $I(X, Y, Z)$  holds in a local BN  $D_N$  for a JT node  $N$ , then  $I(X, Y, Z)$  holds in the original BN  $D$ .*

*Proof:* We will show the claim by removing variables in the JT towards the node  $N$ . Let  $v_i$  be the first variable eliminated in the JT for  $D$ . After calling MCL, let  $D'$  be the CPT-graph defined by the CPTs remaining at this node, together with the CPTs assigned to all other nodes. It follows from Lemma 5 that  $D'$  is a DAG. We now establish that  $I(X, Y, Z)$  holding in  $D'$  implies that  $I(X, Y, Z)$  holds in  $D$ . Note that since  $v_i$  is eliminated,  $v_i \notin XYZ$ . By contraposition, suppose  $I(X, Y, Z)$  does not hold in  $D$ . By Lemma 7, let  $D_m$  be the moralization of the sub-DAG of  $D$  onto  $XYZ \cup An(XYZ)$ . There are two cases to consider. Suppose  $v_i \notin An(XYZ)$ . In this case,  $I(X, Y, Z)$  does not hold in  $D'$  as  $D_m$  is also the moralization of the



sub-DAG of  $D'$  onto  $XYZ \cup An(XYZ)$ . Now, suppose  $v_i \in An(XYZ)$ . Let  $D'_m$  be the moralization of the sub-DAG of  $D'$  onto  $XYZ \cup An(XYZ)$ . By assumption, there is an undirected path in  $D_m$  from  $X$  to  $Z$ , which does not involve  $Y$ . If this path does not involve  $v_i$ , this same path must exist in  $D'_m$  as the MCL algorithm only removes those edges involving  $v_i$ . Otherwise, as  $v_i \notin XYZ$ , this path must include edges  $(v, v_i)$  and  $(v_i, v')$ , where  $v, v' \in P_i C_i$ . Since the moralization process will add an undirected edge between every pair of variables in  $P_i$ , and since the MCL algorithm will add a directed edge from every variable in  $P_i$  to every variable in  $C_i$  and also from every variable  $v_j$  in  $C_i$  to every other variable  $v_k$  in  $C_i$  such that  $v_j \prec v_k$  in Rule 2, it is necessarily the case that  $(v, v')$  is an undirected edge in the moralization  $D'_m$  of  $D'$ . As there is an undirected path in  $D'_m$  from  $X$  to  $Z$  not involving  $Y$ , by definition,  $I(X, Y, Z)$  does not hold in  $D'$ . Thus, every independence  $I(X, Y, Z)$  encoded in  $D'$  is encoded in the original BN  $D$ . Therefore, by recursively eliminating all variables  $v \notin N$ , all conditional independencies encoded in  $D_N$  are also encoded in  $D$ .  $\square$

**Example 53** In Figure 4.17, it is particularly illuminating that our architecture correctly models  $I(g, c, h)$ , the conditional independence of  $g$  and  $h$  given  $c$ , in the local BN for  $cdefgh$ , yet at the same time correctly models  $I(g, h, i)$ , the conditional independence of  $g$  and  $i$  given  $h$ , in the local BN for  $ghij$ .

Although unconditional independencies of the original BN might not be saved in the local BNs, Theorem 18 shows that conditional independencies are.

**Theorem 18** *Given Lemma 10. If an independence  $I(X, Y, Z)$  holds in the original BN  $D$ , where  $Y \neq \emptyset$  and  $XYZ$  is a subset of a JT node  $N$ , then  $I(X, Y, Z)$  holds in the local BN  $D_N$ .*

*Proof:* Let  $v_i \in U$  be the first variable eliminated by our architecture. Suppose FRC returns the set of CPT labels  $C' = \{p(v_i|P_i), p(v_1|P_1), \dots, p(v_k|P_k)\}$ . By  $\prec$  in Rule 2, the child-set of  $v_i$  is  $C_i = \{v_1, \dots, v_k\}$ . The set of all variables appearing in any label of  $C'$  is  $E_k F_k$ . Clearly, the CPT-graph  $D'$  defined by  $C'$  is a DAG. By Corollary 6 in [68], variable  $v_i$  is independent of all other variables in  $U$  given  $E_k F_k - v_i$ . Thus, we only need to show that any conditional independence  $I(X, Y, Z)$  holding in  $D'$  with  $v_i \notin XYZ$  is preserved in  $D''$ , where  $D''$  is the CPT-graph defined by the set  $C''$  of CPT labels output by MCL given  $C'$  as input. By Lemma 5,  $D''$  is a DAG. By contraposition, suppose a conditional independence  $I(X, Y, Z)$  does not hold in  $D''$ . According to the method for testing independencies in Section 2, let  $D'_m$  be the moralization of the sub-DAG of  $D'$  onto  $XYZ \cup An(XYZ)$ . There are two cases to consider. Suppose  $v_i \notin An(XYZ)$ . In this case,  $I(X, Y, Z)$  does not hold in  $D'$  as  $D'_m$  is also the moralization of the sub-DAG of  $D''$  onto  $XYZ \cup An(XYZ)$ . Now, suppose  $v_i \in An(XYZ)$ . By definition, the moralization process makes families complete. Observe that the family of each variable  $v_m$  in  $D'$  is, by definition, the family-set of  $v_m$  in the CPT label  $p(v_m|P_m)$  of  $C'$ . For every variable  $v_m$  in the sub-DAG of  $D'$  onto  $XYZ \cup An(XYZ)$ ,  $v_i$  is a member of the family-set  $F_m$ . Therefore, there is an edge  $(v_i, v)$  in  $D'_m$  between  $v_i$  and every other variable  $v$  in  $D'_m$ . Then, in particular, there is a path  $(x, v_i), (v_i, z)$  from every variable  $x \in X$  to every variable  $z \in Z$ . Since  $v_i \notin Y$ , by definition,  $I(X, Y, Z)$  does not hold in  $D'$ . Hence, all conditional independencies on  $U - v_i$  are kept. Therefore, by recursively eliminating all variables  $v \notin N$ , all conditional independencies on  $N$  are kept. That is, by Lemma 10, all conditional independencies  $I(X, Y, Z)$  with  $XYZ \subseteq N$  are preserved in the local BN  $D_N$ .  $\square$

**Example 54** Recall the CPT-graphs in Figs. 4.6 and 4.5 defined by the CPT labels before and after the elimination of variable  $v_i = f$ , respectively, where  $C_i = \{v_1 = j, v_2 = k, v_3 = l\}$ . It may seem as though some conditional independencies are lost due to the additional directed edges like  $(c, l)$ ,  $(e, l)$ , and  $(j, l)$ , where  $c \in P_i$ ,  $e \in P_1$ , and  $j, l \in C_i$ . On the contrary, although  $I(c, ejkhi, l)$ ,  $I(e, bcdhjk, l)$ , and  $I(j, cde, l)$  do not hold in Figure 4.5, these independencies do not hold in Figure 4.6 either. Some unconditional independencies were lost, however, such as  $I(be, \emptyset, kl)$ .

Our first practical application of local BNs concerns *multiply sectioned BNs* (MSBNs) [86–89]. A MSBN is a representation of a BN as a collection of local BNs that are organized in a JT structure. Before any inference takes place, a given BN needs first be modeled as a MSBN. Several problems, however, with the manual construction of a MSBN from a BN have recently been acknowledged [88]. We resolve this modeling problem as follows.

Recently, Olesen and Madsen [66] gave a simple method for constructing a special jointree based on the maximal prime decomposition (MPD) of a Bayesian network. One desirable property of a MPD jointree is that the jointree nodes are unique for a given Bayesian network. We favour MPD JTs over conventional JTs, since they facilitate inference in the LAZY architecture while still only requiring polynomial time for construction [66]. For example, given the CHD BN in Figure 3.9, the JT shown in Figure 4.3 is the unique MPD JT.

We now propose the following automated procedure for constructing a MSBN from a given BN. First, construct a MPD JT for the given BN. Next, assign the BN CPTs as usual. Instead of applying the LAZY architecture on the MPD JT as Olesen and Madsen suggest, apply our architecture to label the messages to be

propagated during JT propagation. After the distributions of these identified CPTs have been physically computed using any of the available inference algorithms, such as [54,97], each JT node has a local BN. By definition, the representation is a MSBN.

**Example 55** We illustrate our automated MSBN modeling method using the real-world CHD BN in Figure 3.9. The MPD JT with assigned BN CPTs is shown in Figure 4.3. The CPTs identified by our architecture are listed in Figure 4.7. Apply any probabilistic inference algorithm to physically construct these CPTs. By definition, Figure 4.17 is a MSBN for the CHD BN.

The significance of our suggestion is not aimed at an improvement in MSBN computational efficiency. Instead, an automated procedure for the semantic modeling of MSBNs overcomes the recently acknowledged problems with manually constructing MSBNs [88].

Our second practical application of local BNs concerns *direct computation* (DC) [23, 48,97]. MSBNs are well established in the probabilistic reasoning community due, in large part, to the presence of *localized* queries [86,87]. That is, practical experience has previously demonstrated that queries tend to involve variables in close proximity within the BN [89]. We conclude our discussion by showing how our semantic architecture allows direct computation to exploit localized queries.

Direct computation is usually better than JT propagation, if one only is interested in updating a small set of non-evidence variables [54], where small is shown empirically to be twenty or fewer variables in [97]. However, direct computation processes every query using the original BN. Therefore, it is not exploiting localized queries.

Our semantic architecture models the original BN as a set of *local* BNs, once the physical distributions corresponding to the identified CPT labels have actually been constructed. Hence, direct computation techniques can process localized queries in local BNs. The following empirical evaluation is styled after the one reported by Schmidt and Shenoy [74].

**Example 56** We suggest that the CHD BN in Figure 3.9 be represented as the smaller local BNs in Figure 4.17, after the physical construction of CPTs  $p(b)$ ,  $p(f)$ ,  $p(g)$ ,  $p(g|f)$ , and  $p(h|g)$ . Table 4.5 shows the work needed by direct computation to answer five localized queries using the original CHD BN of Figure 3.9 in comparison to using the local BNs of Figure 4.17.

Table 4.5: The computation needed in DC to process five localized queries in the original CHD BN in Figure 3.9 versus the local BNs in Figure 4.17.

Localized query	Original BN			Local BNs		
	+	×	÷	+	×	÷
$p(a b = 0)$	1	2	2	1	2	2
$p(b f = 0, g = 0)$	3	8	2	1	4	2
$p(d h = 0)$	3	6	2	3	6	2
$p(g h = 0, i = 0, j = 0)$	19	42	2	1	6	2
$p(g k = 0)$	19	38	2	1	2	2

While it is acknowledged that our suggestion here is beneficial only for localized queries, practical experience with BNs, such as in neuromuscular diagnosis [89], has long established that localized queries occur in practice.

## 4.6 Summary

The LAZY architecture has demonstrated a remarkable improvement in efficiency over the traditional methods by actively exploiting independencies to remove irrelevant potentials before variable elimination. LAZY propagation, however, does not utilize the independencies holding in the relevant potentials. In our architecture, we introduce the concepts of parent-set and elder-set in order to take advantage of these valuable independencies. Based on this exploitation of independence information, we proposed an architecture for semantic modeling. We believe that the computationally efficient LAZY method, and the semantically rich architecture proposed here, serve as complementary examples of second-generation jointree probability propagation architectures.

# Chapter 5

## Conclusion

In this thesis, we have highlighted the use of semantics in knowledge discovery and uncertainty reasoning. In order to use more semantics in itemset mining, we proposed utility based itemset mining. Using equivalences among attributes, we showed that functional dependencies can be discovered efficiently. By considering the logical implication relationship between functional dependency and conditional independence, we showed that a Bayesian network can be constructed from the resulting functional dependencies. We established a jointree architecture for semantic modelling by applying conditional independencies that had remained unnoticed in previous architectures.

We summarize our research on utility based itemset mining in Section 5.1. We present the conclusions concerning learning functional dependencies and Bayesian networks in Section 5.2. We summarize the results of our study of a jointree propagation architecture for semantic modelling in Section 5.3. Future work is discussed in Section 5.4.

## 5.1 Utility based Itemset Mining

Our research on utility based itemset mining has made two contributions to the theories and techniques of KDD. First, our theoretical work provides a basis for the study of itemset mining from the view of utility. Liu et al. have already used our theoretical work as a basis for developing their own algorithm [50,51] and it has also been cited by Zhou and Wang [98]. Although previous research on itemset mining resulted in fast scalable algorithms for discovering frequent itemsets [4,33,67], it did not emphasize the quality of the resulting itemsets. By allowing users to quantify their preferences concerning the usefulness of itemsets as utility values, our approach is capable of expressing and dealing with more complex types of user preferences than previous research. Based on the utility value of itemsets, we introduced the utility constraint. As we showed in Chapter 2, utility constraints may not satisfy the Apriori property [2] or the convertible property [69] commonly used in itemset mining. Thus, we evaluated a novel mathematical property for this constraint, namely, the utility upper bound property.

Our second contribution is that we provided an efficient algorithm, called UMining, for finding all itemsets satisfying a utility constraint. Where a user's interests can be specified as utility constraints, this algorithm is effective. Our theoretical results for utility constraints were used to design an efficient pruning strategy. Using this pruning strategy, the UMining algorithm can efficiently find all high utility itemsets from a database, while methods for frequent itemset mining [2], convertible constraint based mining [17, 19, 49, 53, 70], and share based mining [7, 92] cannot. Experimental results showed that the proposed algorithm is effective on synthetic and real world datasets. We also developed a heuristic algorithm, called UMining\_H,



which typically finds most high utility itemsets based on a heuristic pruning strategy. Experimental results indicate that UMining\_H provides a reasonable balance between accuracy and efficiency for some applications.

## 5.2 Learning Functional Dependencies and Bayesian Networks

Our research on learning functional dependencies and Bayesian networks made two contributions to the techniques and practical applications of learning functional dependencies from data. First, we provided an efficient algorithm, called FD\_Mine, for finding functional dependencies from data. By exploiting Armstrong's Axioms for functional dependencies, we identified equivalences among attributes, which can be used to reduce both the size of the dataset and the number of functional dependencies to be checked. We summarized four effective pruning rules to reduce the number functional dependencies to be checked. Based on these pruning rules, the FD\_Mine algorithm was developed. Experimental results showed that FD\_Mine can prune more candidates than previous methods without eliminating any valid candidates.

Our second contribution is that we showed a practical application of learning functional dependencies from data, namely, constructing a sound Bayesian network. Using the implication relationship that functional dependency logically implies conditional independence, we developed another algorithm, called FD2BN, to construct a Bayesian network using the resulting functional dependencies, which are efficiently discovered from data by FD\_Mine. We proved the correctness of FD2BN. Since functional dependency is only a sufficient condition for conditional independence, we ac-

knowledge that our approach may not utilize all conditional independencies encoded in data. However, other polynomial learning algorithms also suffer this disadvantage, since Bouckaert [12] proved that discovering all conditional independencies in a probability distribution is a NP-hard problem.

### 5.3 Jointree Probability Propagation for Semantic Modelling

Our research on jointree probability propagation for semantic modelling made three contributions to the theories and techniques of inference in probabilistic expert system. We proposed the first architecture to precisely model jointree propagation in terms of conditional probability tables. Our architecture can identify independencies that are not utilized in previous architectures. The identified independencies are very useful, since they allow us to maintain a conditional probability table factorization after a variable is eliminated. We showed that after our semantic modelling, each jointree node has a sound, local Bayesian network preserving all conditional independencies of the original Bayesian network involving variables in this node.

Our second contribution is showing that the proposed architecture allows us to assist LAZY propagation [54], the state-of-the-art architecture, by producing three work schedules, when modeling inference involving evidence. Our first work schedule identified all irrelevant non-empty messages, as depicted in Figures 4.12 and 4.13. This information is useful, since LAZY's lack of semantic information could force a receiving node to wait for the physical construction of a message that is irrelevant to its subsequent message computation, as explicitly demonstrated in Examples 47

and 48. Our second work schedule identified all empty messages propagated from non-leaf jointree nodes, as shown in Figure 4.15. Our third work schedule identified variables that can be eliminated at a non-leaf node before the node has received any messages, as the screen shot in Figure 4.16 indicates. Our experimental results, given in Tables 4.3 and 4.4, showed that the three work schedules reduced the time required for LAZY propagation for all five real-world Bayesian networks evaluated.

Our third contribution is showing that our architecture is useful to multiply sectioned Bayesian network techniques and direct computation techniques, even when modeling inference not involving evidence. Using our architecture, we developed an automated procedure for constructing a multiply sectioned Bayesian network from a given Bayesian network. This procedure may be of wide interest, since several problems with the manual construction of a multiply sectioned Bayesian network from a Bayesian network have recently been acknowledged [88]. We also have suggested a method for exploiting localized queries in direct computation techniques. Practical experience, such as that gained from neuromuscular diagnosis [89], demonstrated that queries tend to involve variables in close proximity within a Bayesian network. Our approach allows direct computation to process localized queries in local Bayesian networks. The experimental results in Table 4.5 showed promise.

## 5.4 Future Research

In this section, some possible directions for future research in itemset mining, learning functional dependencies, and jointree probability propagation are given.

Several interesting problems related to utility based itemset mining remain as possibilities for future research. First, the UMining algorithm is based on a level

wise approach. Since depth first search approaches, such as MAFIA [14] and FP-growth [33], have several advantages over level wise approaches, future research could consider whether the pruning strategies described in Section 2.3 could be incorporated in these approaches. As well, following the suggestion of Kleinberg et al. [44] that a discovered pattern is interesting only to the extent that it can be used in the decision making process of the user to increase utility, the ability to suggest a highly profitable action for the user based on discovered high utility itemsets could be investigated. Finally, although the focus of utility based mining has been on itemset mining, the results of the present study may be useful in other related problem domains, such as information retrieval [73] and web mining [20]. For example, Table 2.1 can be regarded as representing a document set used in information retrieval, where each column represents a term, and each row represents a document, and the value in each cell indicates the frequency of a term appearing in a document. Table 2.1 can also be regarded as representing web pages used in web mining, where each column represents a keyword, and each row represents a web page, and the value in each cell indicates the frequency of a keyword appearing in a web page. In each case, the documents matching a user's interest could be retrieved using the UMining algorithm.

We see three possibilities worth investigating in the study of learning functional dependencies and Bayesian networks from data based on the results obtained in this thesis. First, it may be possible to use attributes identified as being equivalent to perform data cleaning by detecting and removing errors and inconsistencies from data in order to improve its quality. Secondly, our approach might be helpful for other heuristic search algorithms [28, 37] used for learning a Bayesian network from data. These algorithms use greedy search techniques to pick a Bayesian network.

However, it is usually difficult for a person to determine a proper initial Bayesian network [37]. Instead, the Bayesian network constructed in our approach could be used as the initial Bayesian network in these other approaches. Finally, rules that are more general than functional dependencies, such as association rules, may possibly be organized into a Bayesian network to support reasoning.

There is still more work to be done in jointree propagation. A graph might be used to represent the relationship among conditional probability tables found by the *IdentifyCPTMessages* algorithm. Increasing the parallelism in jointree propagation is another research direction. Also, the potential relationship between jointree propagation and expectation propagation, as proposed by Geng and Hamilton [29] for mining interesting summaries in a generalization space, needs to be investigated.

# References

- [1] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, Boston, Massachusetts, 1995.
- [2] R. Agrawal, T. Imielinski, and A.N. Swami, Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Washington, D.C., 1993, 207–216.
- [3] R. Agrawal and R. Srikant, Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases*, Santiago, Chile, 1994, 487–499.
- [4] R. Agarwal, C.C. Aggarwal, and V.V.V. Prasad, A tree projection algorithm for generation of frequent item sets. *Journal of Parallel and Distributed Computing*, 61(3), 2001, 350-371.
- [5] K. Ali, S. Manganaris, and R. Srikant, Partial classification using association rules. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, Newport Beach, California, 1997, 115-118.
- [6] F. Bacchus and A. Grove, Graphical models for preference and utility. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, Montreal, 1995, 3-10.
- [7] B. Barber and H.J. Hamilton, Extracting share frequent itemsets with infrequent subsets. *Data Mining and Knowledge Discovery*, 7(2), 2003, 153–185.

- [8] R.J. Bayardo, Efficiently mining long patterns from databases. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, Seattle, Washington, 1998, 85–93.
- [9] R. J. Bayardo, R. Agrawal, and D. Gunopulos, Constraint based rule mining in large dense databases. In *Proceedings of the 15th International Conference on Data Engineering*, Sydney, Australia, 1999, 188-197.
- [10] I. Beinlich, G. Suermondt, R. Chavez, and G. Cooper, The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks, In *Proceedings of the 2nd European Conference on Artificial Intelligence in Medicine*, London, UK, 1989, 247–256.
- [11] J. Binder, D. Koller, S. Russell, and K. Kanazawa, Adaptive probabilistic networks with hidden variables, *Machine Learning*, 29(2-3), 1997, 213–244.
- [12] R. Bouckaert, Properties of learning algorithms for Bayesian belief networks. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, Seattle, Washington, 1994, 102–109.
- [13] C. Boutilier, R. Patrascu, P. Poupart, and D. Schuurmans, Constraint-based optimization and utility elicitation using the minimax decision criterion, *Artificial Intelligence*, 170(8), 2006, 686-713.
- [14] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. Yiu, MAFIA: A maximal frequent itemset algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 17(11), 2005, 1490-1504.

- [15] C.J. Butz, S.K.M. Wong, and Y.Y. Yao, On data and probabilistic dependencies. In *Proceeding of the IEEE Canadian Conference on Electrical and Computer Engineering*, Edmonton, Alberta, Canada, 1999, 1692-1697.
- [16] C.J. Butz, H. Yao, and H. Hamilton, Towards jointree propagation with conditional probability distributions. In *Proceeding of the 4th International Conference on Rough Sets and Current Trends in Computing*, Uppsala, Sweden, 2004, 368-377.
- [17] C.H. Cai, A.W.C. Fu, C.H. Cheng, and W.W. Kwong, Mining association rules with weighted items. In *Proceedings of the IEEE International Database Engineering and Applications Symposium*, Cardiff, UK, 1998, 68-77.
- [18] E. Castillo, J. Gutiérrez, and A. Hadi, *Expert Systems and Probabilistic Network Models*. Springer, New York, 1997.
- [19] R. Chan, Q. Yang, and Y.D. Shen, Mining high utility itemsets. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, Melbourne, Florida, 2003, 19-26.
- [20] R. Cooley, B. Mobasher, and J. Srivastava. Web mining: Information and pattern discovery on the world wide web. In *Proceedings of the International Conference on Tools with Artificial Intelligence*, Newport Beach, California, 1997, 558-567.
- [21] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 2001.



- [22] R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter, *Probabilistic Networks and Expert Systems*. Springer, New York, 1999.
- [23] R. Dechter, Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence*, Portland, Oregon, 1996, 211-219.
- [24] G. Dong and J. Li, Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining*, San Diego, California, 1999, 43–52.
- [25] J. Doyle, T. Dean, et al. Strategic directions in Artificial Intelligence. *ACM Computing Surveys*, 28(4), 1996, 653-670.
- [26] Elvira Consortium, Elvira: An environment for probabilistic graphical models. In *Proceedings of the 1st European Workshop on Probabilistic Graphical Models*, Cuenca, Espana, 2002, 222-230.
- [27] U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT, Menlo Park, California, 1996, 1-34.
- [28] N. Friedman and M. Goldszmidt, Learning Bayesian networks with local structure. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence*. San Francisco, California, 1996, 252-262.
- [29] L. Geng and H.J. Hamilton, Finding Interesting Summaries in GenSpace Graphs Efficiently. In *Proceedings of the 17th Canadian Artificial Intelligence Conference*, London, Ontario, Canada, 2004, 89-104.

- [30] L. Geng and H.J. Hamilton, Interestingness measures for data mining: A survey. *ACM Computing Surveys*. To appear.
- [31] E. G. Goodaire and M.M. Parmenter, *Discrete Mathematics with Graph Theory*. Prentice Hall, New Jersey, 1992.
- [32] P. Hájek, T. Havránek, and R. Jiroušek, *Uncertain Information Processing in Expert Systems*. CRC Press, Ann Arbor, Michigan, 1992.
- [33] J. Han, J. Pei, Y. Yin, and R. Mao, Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1), 2004, 53-87.
- [34] J. Han and M. Kamber, *Data mining: Concepts and Techniques*. Morgan-Kaufman, New York, 2001.
- [35] D. Heckerman, E. Horvitz, and B. Nathwani, Towards normative expert systems: Part I the Pathfinder project. *Methods of Information in Medicine*, 31(2), 1992, 90-105.
- [36] D.E. Heckerman, J. Breese, and K. Rommelse, Decision-theoretic troubleshooting. *Communications of the ACM*, 38(3), 1995, 49-57.
- [37] D. Heckerman, A tutorial on learning with Bayesian network. *Microsoft Research Technical Report, MSR-TR-95-06*, 1995.
- [38] R.J. Hilderman and H.J. Hamilton, Measuring the interestingness of discovered knowledge: A Principled Approach. *Intelligent Data Analysis*, 7(4), 2003, 347-382.

- [39] E. Horvitz, S. Srinivas, C. Rouokangas, and M. Barry, A decision-theoretic approach to the display of information for time-critical decisions: The Vista project. In *Proceedings of the 6th Annual Workshop on Space Operations Applications and Research*, Houston, Texas, 1992.
- [40] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The Lumire project: Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the 14th Conference on Uncertainty in AI*, Madison, Wisconsin, 1998, 256-265.
- [41] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen, TANE: An efficient algorithm for discovering functional and approximate dependencies. *The Computing Journal*, 42(2), 1999, 100-111.
- [42] F.V. Jensen, S.L. Lauritzen, and K.G. Olesen, Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4, 1990, 269–282.
- [43] F.V. Jensen, *An Introduction to Bayesian Networks*. UCL Press, London, UK, 1996.
- [44] J.M. Kleinberg, C.H. Papadimitriou, and P. Raghavan, A Microeconomic view of data mining. *Data Mining and Knowledge Discovery*, 2(4), 1998, 311-324.
- [45] S.L. Lauritzen and D.J. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems. *Royal Statistical Society, Series B*, 50(2), 1988, 157-244.

- [46] S.L. Lauritzen, A.P. Dawid, B.N. Larsen, and H.G. Leimer, Independence properties of directed Markov fields. *Networks*, 20(5), 1990, 491–505.
- [47] IBM Synthetic data.  
<http://www.almaden.ibm.com/software/quest/Resources/index.shtml>, 2005.
- [48] Z. Li and B. D’Ambrosio, Efficient inference in Bayes networks as a combinatorial optimization problem. *International Journal of Approximate Reasoning*, 11(1), 1994, 55-81.
- [49] T.Y. Lin, Y.Y. Yao, and E. Louie, Value added association rules. In *Proceedings of the 6th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Taipei, Taiwan, 2002, 328-333.
- [50] Y. Liu, W.K. Liao, and A. Choudhary, A two-phase algorithm for fast discovery of high utility itemsets. In *Proceedings of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Hanoi, Vietnam, 2005, 689-695.
- [51] Y. Liu, W.K. Liao, and A. Choudhary, A Fast High Utility Itemsets Mining Algorithm. In *Proceedings of the Workshop on Utility-Based Data Mining in conjunction with the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2005)*, Chicago, Illinois, 2005, 90-99.
- [52] W. Long, Medical diagnosis using a probabilistic causal network. *Applied Artificial Intelligence*, 3, 1989, 367-383.
- [53] S. Lu, H. Hu, and F. Li, Mining weighted association rules. *Intelligent Data Analysis*, 5(3), 2001, 211–225.

- [54] A.L. Madsen and F.V. Jensen, LAZY propagation: A junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*, 113(1-2), 1999, 203-245.
- [55] A.L. Madsen, An empirical evaluation of possible variations of lazy propagation. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, Banff, Alberta, Canada, 2004, 366-373.
- [56] D. Maier, *The Theory of Relational Databases*. Computer Science Press, Rockville, Maryland, 1983.
- [57] H. Mannila, H. Toivonen, and A.I. Verkamo, Efficient algorithms for discovering association rules. In *Proceeding of the AAAI Workshop on Knowledge Discovery in Databases*, Seattle, Washington, 1994, 181-192.
- [58] H. Mannila and K.J. Raiha, Algorithms for inferring functional dependencies from relations. *Data and Knowledge Engineering*, 12(1), 1994, 83-99.
- [59] H. Mannila and H. Toivonen, Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3), 1997, 241-258.
- [60] K. Murphy, Y. Weiss, and M. Jordan, Loopy belief propagation for approximate inference: An empirical study, In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, San Francisco, California, 1999, 467-475.
- [61] R.E. Neapolitan, *Probabilistic Reasoning in Expert Systems*. John Wiley & Sons, Toronto, Canada, 1990.

- [62] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang, Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, Seattle, Washington, 1998, 13-24.
- [63] <http://www.nokia.com/nokia/0,,53720,00.html>
- [64] N. Novelli and R. Cicchetti, FUN: An efficient algorithm for mining functional and embedded dependencies. In *Proceeding of the International Conference on Database Theory*, London, UK, 2001, 189-203.
- [65] N. Novelli and R. Cicchetti, Functional and embedded dependency inference: A data mining point of view. *Information Systems*, 26(7), 2001, 477-506.
- [66] K.G. Olesen and A.L. Madsen, Maximal prime subgraph decomposition of Bayesian networks. *IEEE Transactions on Systems, Man, and Cybernetics, B*, 32(1), 2002, 21-31.
- [67] J.S. Park, M. Chen, and S.Y. Philip, An effective hash-based algorithm for mining association rules. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, San Jose, California, 1995, 175-186.
- [68] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, California, 1988.
- [69] J. Pei and J. Han, Can we push more constraints into frequent pattern mining? In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining*, Boston, Massachusetts, 2000, 350–354.

- [70] J. Pei, J. Han, and L.V.S. Lakshmanan, Pushing convertible constraints in frequent itemset mining. *Data Mining and Knowledge Discovery*, 8(3), 2004, 227-252.
- [71] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, California, 1993.
- [72] R. Ramakrishnan and J. Gehrke. *Database Management Systems*, McGraw-Hill, New York, 1998.
- [73] G. Salton, *Automatic Text Processing*. Addison-Wesley, Boston, Massachusetts, 1989.
- [74] T. Schmidt and P.P. Shenoy, Some improvements to the Shenoy-Shafer and Hugin architectures for computing marginals, *Artificial Intelligence*, 102, 1998, 323-333.
- [75] R. Shachter, Evaluating influence diagrams. *Operation Research*, 34 (6), 1986, 871–882.
- [76] G. Shafer and P.P. Shenoy, Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2, 1990, 327–352.
- [77] G. Shafer, *Probabilistic Expert Systems*, SIAM, Philadelphia, Pennsylvania, 1996.
- [78] P.P. Shenoy, Binary join trees for computing marginals in the Shenoy-Shafer architecture. *International Journal of Approximate Reasoning*, 17(2-3), 1997, 239–263.

- [79] M. Singh and M. Valtorta, Construction of Bayesian network structures from data: A survey and an efficient algorithm. *International Journal of Approximate Reasoning*, 12, 1995, 111–131.
- [80] A. Silberschatz and A. Tuzhilin, On subjective measures of interestingness in knowledge discovery. In *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining*, Montreal, Quebec, Canada, 1996, 275–281.
- [81] C. Silverstein, S. Brin, and R. Motwani, Beyond market baskets: generalizing association rules to dependence rules. *Data Mining and Knowledge Discovery*, 2(1), 1998, 39-68.
- [82] P.N. Tan, V. Kumar, and J. Srivastava, Selecting the right objective measure for association analysis. *Information Systems*, 29(4), 2004, 293-313.
- [83] P.N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison-Wesley, Boston, Massachusetts, 2005.
- [84] UCI Repository of machine learning databases.  
<http://www.ics.uci.edu/~mllearn/MLRepository.html>, 2005.
- [85] S.K.M. Wong, C.J. Butz and D. Wu, On the implication problem for probabilistic conditional independency. *IEEE Transactions on Systems, Man, and Cybernetics, A*, 30(6), 2000, 785-805.
- [86] Y. Xiang, A probabilistic framework for cooperative multi-agent distributed interpretation and optimization of communication, *Artificial Intelligence*, 87(1-2), 1996, 295-342.



- [87] Y. Xiang, *Probabilistic Reasoning in Multiagent Systems: A Graphical Models Approach*, Cambridge University Press, New York, 2002.
- [88] Y. Xiang, K.G. Olesen, and F.V. Jensen, Practical issues in modeling large diagnostic systems with multiply sectioned Bayesian networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 14(1), 2000, 59-71.
- [89] Y. Xiang, B. Pant, A. Eisen, M.P. Beddoes, and D. Poole, Multiply sectioned Bayesian networks for neuromuscular diagnosis. *Artificial Intelligence in Medicine*, 5, 1993, 293-314.
- [90] H. Xu, Computing marginals for arbitrary subsets from marginal representation in Markov trees, *Artificial Intelligence*, 74(1), 1995, 177-189.
- [91] H. Yao, H.J. Hamilton, and C.J. Butz, FD\_Mine: discovering functional dependencies in a database using equivalences. In *Proceedings of the 2nd IEEE International Conference on Data Mining*, Maebashi City, Japan, 2002, 729-732.
- [92] H. Yao, H.J. Hamilton, and C.J. Butz, A foundational approach to mining itemset utilities from databases. In *Proceedings of the 3rd SIAM International Conference on Data Mining*. Orlando, Florida, 2004, 482-486.
- [93] H. Yao, Decision mining with user preferences, In *Proceedings of the 17th Canadian Artificial Intelligence Conference*. London, Ontario, Canada, 2004, 576-577.
- [94] H. Yao and H.J. Hamilton, Mining itemset utilities from transaction databases, *Data & Knowledge Engineering*. To appear.

- [95] H. Yao, C.J. Butz, and H.J. Hamilton, Causal discovery, In *The Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach (Eds.), Springer, 2005, 945-955.
- [96] Y.Y. Yao and N. Zhong, An analysis of quantitative measures associated with rules, Methodologies for Knowledge Discovery and Data Mining. In *Proceedings of the 3rd Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Beijing, China, 1999, 26-28.
- [97] N.L. Zhang, Computational properties of two exact algorithms for Bayesian networks. *Applied Intelligence*, 9(2), 1998, 173-184.
- [98] S. Zhou and K. Wang, Profit mining, *the Encyclopedia of Data Warehousing and Mining*, Idea Group Reference, 2004