Waterford Institute of Technology

Department of Computing, Maths and Physics

Master Thesis

# Distributed Algorithms for Computing Closed Itemsets Based on an Iterative MapReduce Framework

**Author:**
Biao Xu

**Supervisor:**
Dr. Mícheál Ó Foghlú

Submitted to Waterford Institute of Technology, Dec. 2011

# Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Master of Science is entirely my own work and has not been taken from the work of others save to the extent that such work has been cited and acknowledged within the text of my work.

Signed: ......................ID: 20038100
Date: Oct. 2012

# Acknowledgments

I have been a master student in the Department of Computing, Mathematics & Physics at the Waterford Institute of Technology for 3 years. Most of the time I study and work in the TSSG which I found is a wonderful place for research and to work. It is my pleasure to engage in dynamic, state of the art research projects in TSSG which motivated my research and inspired this thesis. I would like to give my great thanks and best wishes to WIT, TSSG, and their people who supported me so much.

Mícheál Ó Foghlú, my supervisor, gave me great guidance and tireless support on my research. He always can help me out of the dilemmas and suggest better ways to step forward. Dr. Ruairí de Fréin in TSSG is so good at writing and research. He gave me great help on paper and thesis writing. Appreciate your great effort, Mícheál and Ruairí.

I would like to express my gratitude to Eric Robson in DM&SC group who helped me to balance the project and research work. He helped with technical issues and paper writing and he was a great help to me.

This work would have not been possible without the unending love, understanding and support of my parents, although we are actually thousands of miles away and separated by vast oceans. My sincere thanks to you, dear father and mother!

# Contents

# List of Figures

# List of Tables

# List of Acronyms

**DM** Data Mining

**ARM** Association Rules Mining

**FI** Frequent Itemset

**CI** Closed Itemset

**FCI** Frequent Closed Itemset

**IFCI** Infrequent Closed Itemset

**FC** Formal Concepts

**FCA** Formal Concept Analysis

**MRF** MapReduce Framework

**MRGanter** a distributed version of NextClosure algorithm based on the MapReduce framework

**MRGanter+** an enhanced version of MRGanter

**MRCbo** a distributed version of CloseByOne based on the MapReduce framework

**MRGeneral** a general merging model for frequent closed itemset mining algorithms based on the MapReduce framework

**HIL** Horizontal Item-List

**VTL** Vertical TID-List

**HIV** Horizontal Item-Vector

| | |
|---|---|
| **VTV** | Vertical TID-Vector |
| **FIST** | Frequent Item Sets Tree |

# Nomenclature

| | |
|---|---|
| $\mathcal{D}$ | the exemplar dataset used in this thesis |
| $\mathcal{D}_i$ | a dataset partition, $i = 1, \ldots, n$ |
| $n$ | the number of data partitions |
| $T$ | the set of objects (transactions) |
| $P$ | the set of attributes (items) |
| $t \in T$ | $t$ is an element in $T$, i.e., an object or transaction |
| $p \in P$ | $p$ is an element in $P$, i.e., an attribute or item |
| $m$ | the number of attributes in $P$, i.e., the cardinality of $P$ |
| $N$ | the number of objects in $T$ |
| $A \Longrightarrow B$ | an association rule with regard to an itemset $X \subseteq P$, where $A \cup B = X$ and $A \cap B = \emptyset$ |
| $N(X)$ | the number of objects which contain $X$ in $T$ |
| $N(A, B)$ | the number of objects which contain both $A$ and $B$ in $T$ |
| $S$ | the support count (absolute support) of an itemset $X$ |
| $\sup(X)$ | the (relative) support of itemset $X$ |
| $\text{conf}(A \Longrightarrow B)$ | the confidence of the association rule $A \Longrightarrow B$ |
| $\sigma$ | the minimal support specified by the user, for example, 30% |
| $\varsigma$ | the minimal confidence specified by the user, for example, 90% |

| | |
|---|---|
| $(T, P, R)$ | the formal context, where $R$ is the binary relation between $T$ and $P$ |
| $(\{a, b, c, \cdots\} : S)$ | denotes an itemset, $\{a, b, c, \cdots\}$, consisting of elements $a, b, c, \cdots$ with its support count |
| $F_i = \langle Y, X\rangle$ | a formal concept of $(T, P, R)$, where $Y \subseteq T$, $X \subseteq P$ |
| $F^X_{\mathcal{D}_i}$ | a formal concept based on itemset $X$ in the database partition $\mathcal{D}_i$ |
| $\mathcal{F}_{\mathcal{D}_i}$ | a set of formal concepts derived from $\mathcal{D}_i$ |
| $\mathcal{F}$ | the set of all formal concepts for all data partitions |
| $\mathcal{F}_{\mathcal{D}_i \cap \mathcal{D}_j}$ | the set of intersections between the sets of formal concepts $\mathcal{F}_{\mathcal{D}_i}, \ldots, \mathcal{F}_{\mathcal{D}_j}$, for the corresponding partitions $\mathcal{D}_i, \cdots, \mathcal{D}_j$ |
| $\Psi(\mathcal{F}_{\mathcal{D}_i})$ | denotes a set of intersections between $\mathcal{F}_{\mathcal{D}_i}$. See Theorem 3 for more details |
| $\leq_i$ | the lectic order adopted by Ganter's algorithm for comparing itemsets, see Equation 2.5 |
| $\mathcal{B}(T, P, R)$ | the set of all formal concepts which forms a complete lattice together with $\leq_i$ |
| $X'$ | the derivation operator for mapping between a set of objects and a set of attributes in Formal Concept Analysis |
| $X''$ | the closure generation operator in Formal Concept Analysis, for example, $X'' = Y$, where $X \subseteq P$ and $Y \subseteq T$ |
| $X \oplus p$ | the intent generation operator in Ganter's algorithm, see Section 2.2.3 |
| $I_i$ | the i-th frequent closed itemset |
| $\mathcal{C}_i$ | a set of frequent closed itemsets produced by data partition $\mathcal{D}_i$ |
| $\mathcal{C}_{ij}$ | the union of $\mathcal{C}_i$ and $\mathcal{C}_j$. See Example 7 |
| $\widetilde{\mathcal{C}}_i$ | a set of infrequent closed itemsets produced by data partition $\mathcal{D}_i$ |
| $\widetilde{\mathcal{C}}_{i\cup j}$ | an union of $\widetilde{\mathcal{C}}_i, \ldots, \widetilde{\mathcal{C}}_j$ |
| $\mathcal{C}_\alpha$ | the set of frequent closed itemsets which always appear as local FCIs on the data partitions. $\alpha$ refers to this scenario. $I_\alpha$ is a FCI of this type |

| | |
|---|---|
| $\mathcal{C}_\beta$ | the set of frequent closed itemsets which are infrequent on some of the data partitions. $\beta$ refers to this scenario. $I_\beta$ is a FCI of this type |
| $\mathcal{C}$ | the set of all frequent closed itemsets existing in every $\mathcal{D}_i$ |
| $\mathcal{C}_1 \boxplus \mathcal{C}_2$ | global frequent closed itemsets generation operator proposed in [1] |
| $\overline{\mathcal{C}}$ | the set of global FCIs produced by $\mathcal{C}_1 \boxplus \mathcal{C}_2 \boxplus \cdots \boxplus \mathcal{C}_i$. See Theorem 4 |
| $\mathcal{C}_i \otimes \widetilde{\mathcal{C}_j}$ | the operation for extracting frequent closed itemsets which are locally infrequent in some of data partitions. Note that $i \neq j$ and see Definition 5 for detail |
| $\{\mathcal{C}_i, \mathcal{C}_j, \mathcal{C}_k\} \biguplus \{\widetilde{\mathcal{C}_i}, \widetilde{\mathcal{C}_j}, \widetilde{\mathcal{C}_k}\}$ | the operator introduced to calculate global frequent closed itemsets which are existing on some of data partitions. See Definition 5 and Example 7 for more detail |
| $\mathcal{S}_i = \Gamma(\mathcal{S}_{i-1}, \mathcal{S}_1)$ | the set of frequent closed itemsets which are only frequent on $i$ out of the $n$ data partitions |
| $\overline{\mathcal{S}_i} = \Upsilon(\mathcal{S}_i, \mathcal{S}_n, \overline{\mathcal{S}_{(i+1)\to n}}, \widetilde{\mathcal{C}}_{1\to n})$ | the set of frequent closed itemsets which are existing or included in $i$ out of $n$ data partitions, where $\overline{\mathcal{S}_{(i+1)\to n}}$ is the union of $\overline{\mathcal{S}_{(i+1)}}$,..., $\overline{\mathcal{S}_n}$. See Proposition 2 for more details |

## Abstract

Closed Itemset mining is a major task both in Data Mining and Formal Concept Analysis. It is an efficient way to determine patterns which are hidden in raw data. These patterns may be expressed as association rules which capture relations between variables in databases. In the case of Formal Concept Analysis, Closed Itemsets form the basis of formal concepts. A concept consists of an extent and intent. It turns out that the intent is exactly a closed itemset whose support is the cardinality of extent. With the increasing application of Data Mining and Formal Concept Analysis in knowledge discovery, Closed Itemset mining is fast becoming an attractive research area.

While many prominent algorithms have been developed to mine closed itemsets in both Formal Concept Analysis and Frequent Closed Itemset mining area, they are typically unsuitable for distributed implementation. The root cause is that the theoretical foundation of closed itemset mining is based on a centralized data representation. By examining the features of closed itemset, we propose distributed closed itemset computing theories for Formal Concept Analysis and Frequent Closed Itemset mining in a distributed environment. Additionally, taking the MapReduce framework as our inspiration we propose a general distributed approach for performing closed itemset mining. We then provide representative exemplars of how the classic centralized algorithms in Formal Concept Analysis and Frequent Closed Itemset mining can be implemented in a distributed fashion using our methodology and present a family of MR$^*$ algorithms, where the abbreviation stands for MapReduce and * stands for the underlying algorithm that has been adapted. To analyse the factors which impact a distributed algorithm's performance, we compare our MR$^*$ algorithms with the state-of-the-art. Experiments conducted on real datasets demonstrate that the MR$^*$ algorithms for Formal Concept Analysis are efficient and scalable. The MR$^*$ algorithm and theory for Frequent Closed Itemset mining of this work set the scene for future developments. We analyze our experimental results and discuss the bottlenecks which we will focus on in future work.

Chapter 1

# Introduction

This master thesis addresses the data mining area known as closed itemset mining. The work programme involved implementing a number of well-known algorithms from the literature, and then modifying these algorithms in order to optimize their performance. The modifications were primarily directed towards allowing the resulting algorithms to run in parallel, as a distributed processing task.

This chapter starts by describing the current open problems in closed itemset mining for distributed data. A crucial problem is the lack of distributed computing theories for Closed Itemsets (CIs). A second problem is how to extend distributed principles to existing algorithms to obtain distributed algorithms. These problems are becoming more prominent due to the following reasons. Modern datasets are rapidly increasing in size and are typically located on groups of servers. Many existing algorithms may not be suited to these scenarios. The aim of this thesis is to exploit the power of closed itemset mining techniques in scenarios where the algorithms deal with data which is distributed across multiple locations. Further objectives include developing formal computing theories and efficient distributed algorithms. Our research is focused on mining formal concepts and frequent closed itemset respectively.

## 1.1 Research Motivation

In recent years, there has been great increase in the requirement for data analysis [2], knowledge discovery [3] [4] and information retrieval [5]. This is because mobile operators, service providers etc. realize that their products can be potentially improved by learning and exploiting information from customer datasets [3], for example, through personalization and prediction services. Among numerous techniques, Association Rules Mining (ARM) [6, 7], which defined by Agrawal et al., is usually used to search groups between variables, attributes. Association algorithms are used for recommendation engine that originally suggests products to customers based on what they bought earlier. To construct graph between people, events and places, people normally adopt Formal Concept Analysis (FCA) developed by Wille [8, 9] which refers to both an unsupervised machine learning technique and, more broadly, a method of data analysis. In FCA, a concept refers to a pair containing a closed attributes set and a closed objects set. Closed itemset [10] mining is a crucial technique for generating association rules and Formal Concepts (FC) and can be found both in Data Mining (DM) and FCA. A Closed itemset (closure) is a minimal present of a set of items. We use the fraction of objects (*support*) which support the given closed itemset X to measure how often X is applicable to a given dataset (See Section 2.3.1 for more details). Naturally, a closed itemset is frequent when its support meets a setting value. We present both ARM and FCA in this thesis because they have the same mathematical foundation, i.e., closure mining. Note that in ARM, we are interested in Frequent Closed Itemset (FCI), but pay attention to all CIs in FCA. An important reason for this is that ARM aims to form associated rules between attributes, while FCA places extra emphasis on constructing the concept lattice by arranging the concepts by order. In other words, ARM relies on support knowledge but FCA does not.

In the past 20 years, researchers have been focusing on distributed computing [11] and have proposed many efficient FCA algorithms [12] and FCI mining algorithms [13]. Much work has been done to develop novel distributed algorithms that suit very specific distributed settings (e.g. a cluster with many nodes that do not need frequent communication [1]), while in comparison very little effort has been expended on the basic theory for merging frequent closed itemsets. Most existing closed itemset mining algorithms are designed for centralized datasets and their performance degrades for large datasets, that is, datasets with a large number of records [14]. Moreover, social network research is becoming increasingly prominent. Many researchers have studied social networks in order to find potential patterns and rules [15, 16, 17]. Both FCA and ARM have been used to find interesting patterns. However, the data source of social network normally comes from distributed servers. This scenario makes a case from a computing perspective for distributed closed itemsets mining algorithms some of which we have to propose in this thesis. In short, the recent explosion in datasets size and the distributed nature of the system that have been deployed to analyze them, due to privacy protection concerns and increasing scale, suggests that efficient distributed algorithms are required.

## 1.2 Problem Statement

FCA is a discipline that studies the hierarchical structures induced by the binary relation between a pair of sets [8, 9]. The structure is made up of closed itemsets ordered by set-theoretical inclusion and the core of FCA lies in the calculation for the closed itemsets. An efficient method to form association rules is to use frequent closed itemsets, and this is also the key process of ARM. From this point of view, ARM has the same sub-task as FCA, i.e., closed itemset mining. In fact, some FCA-based methods could be applied on ARM and a lot of work has been done in this area [18, 19].

However, even to this day, it is still a challenge to compute closed itemsets in a distributed environment. Due to increasing data size, existing FCA and FCI mining algorithms are no longer able to process the huge datasets [12, 20, 21, 22, 23]. We list the following open problems:

- Most closed itemset mining algorithms are designed to operate on a central database and normally they maintain the dataset in the memory of a single machine. The increasing dataset size makes mining of closed itemsets using one algorithm on a single machine a bottle neck.
- Distributed theories for computing (frequent) closed itemsets are not well developed. There is little work on distributed FCA, whereas distributed FCI mining only has some elementary results.
- Traditional distributed architectures, such as client-server, peer-to-peer, have to either access slave nodes or access each other to obtain necessary information. This feature severely restricts algorithm performance.
- The iterative feature of FCA and some DM algorithms make it is even harder to execute them in a distributed way, because decomposing an iterative task is relatively difficult.

This thesis aims to address these problems. Our objectives are listed below.

## 1.3 Thesis Objectives and Scope

Given the aforementioned description of current problems in distributed closed itemset mining, the scope and objectives of this thesis can now be defined.

As an essential step in both FCA and ARM namely closed itemset mining is discussed from both FCA and ARM perspectives in this thesis. The aim of closed itemset mining is to obtain closely related items. One problem in distributed closed itemset mining stems from the lack of distributed computing theories for closed itemset. Thus, the primary aims of this thesis are:

**(1)** To provide theories supporting distributed closed itemset computing. In light of the lack of distributed theories for Formal Concept and Frequent Closed Itemset mining, this thesis develops distributed theories for them separately.

**(2)** To utilize the MapReduce Framework (MRF) to construct distributed CI mining algorithms and to introduce a distributed architecture.

**(3)** To process large datasets in a distributed manner. Large data is normally hard to deal with because it is too large to fit the single machine. The methods proposed in this thesis divide the data into small partitions and process the individual partition of data on different machines, so that both distributed data and large datasets can be handled.

In this thesis, we emphasize the scalability of the MRF for redesigning existing FCA and FCI mining algorithms and deploy new algorithms. Specifically, we adopt the iterative MapReduce implementation, Twister, as an infrastructure to underpin our approach. Although there are many other MapReduce frameworks, such as Apache Hadoop MapReduce, we use Twister because Twister is suitable for iterative algorithms [24]. Twister takes care of the inter-process communications and the management of the set of processes in distributed settings efficiently. The main advantage of Twister is that Twister supports iterative MapReduce computations which makes Twister particularly suited to the work in this thesis.

## 1.4 Thesis Structure

This thesis provides a basic overview of existing distributed frequent closed itemset mining algorithms. It also provides a new solution based on the MapReduce framework, which is an efficient programming tool to deal with large and distributed datasets [20, 25]. The solution proposed in this thesis modifies existing frequent closed itemset mining algorithms using the MapReduce framework. This thesis is organized as follows.

Chapter 2 presents the background knowledge and the state-of-the-art of closed itemset mining. By introducing the MapReduce framework and an iterative implementation: Twister, we show that Twister is a efficient way to solve the problems highlighted in Chapter 1.

Chapter 3 presents the drawback of existing distributed closed mining solutions. The methodology that this work follows is presented subsequently. Distributed formal concept mining and distributed frequent closed itemset mining theories are proposed respectively in Section 3.3.

Chapter 4 begins by describing the design of MRGanter. Then MRGanter+, and MRCbo are introduced. A general merging model for distributed frequent closed itemset mining, MRGeneral, is presented subsequently. These algorithms are based on existing algorithms, thus detailed descriptions are presented which explain how the existing approaches are improved. Results from testing and analysis of these algorithms are then shown. This chapter concludes with a brief discussion on the experimental results.

Chapter 5 describes some ancillary points of interest which arise during the course of this research work, and explores some work which we intend to undertake to improve the performance of these algorithms.

Chapter 6 summarizes the main points of this thesis and the rationale for having performed this research. The contributions of this thesis are listed here for the reader's convenience.

### Contributions

1. Proposed distributed Formal Concept and Frequent Closed Itemset mining theories.

2. Developed two distributed versions for NextClosure and distributed CloseByOne based on the iterative MapReduce framework Twister, they are MRGanter, MRGanter+ and MRCbo.

3. Designed a general model MRGeneral for computing frequent closed itemsets in distributed manner, and presented the application of Closet+ algorithm on this model.

Chapter 2

# Background and Literature Review

## 2.1 Introduction

This chapter describes the background to techniques related to the approach proposed in this thesis. We start by introducing the concepts behind Formal Concept Analysis (FCA) and Association Rules Mining (ARM). ARM originated from market basket analysis problem [26]. For example, a stake owner might be interested in knowing if a customer who purchased bread possibly chooses milk as well. Its theoretical foundation is based on FCA [7]. Some FCA techniques can be utilized in ARM [27].

In ARM, rules can be produced by finding Frequent Itemsets (FIs) and Frequent Closed Itemsets (FCIs). A set of attributes is called a frequent itemset if its occurrence is greater than the threshold which the user specifies for that dataset. A frequent closed itemset is a condensed set of some frequent itemsets [26]. In recent years, finding closed itemsets has become a popular research direction because FCI is more compact than FI and typically searching for the FCIs takes less time [28]. For very large datasets, smaller CIs can be more computationally efficient [28, 29].

Two types of algorithms will be explored in this chapter so that we can show how to extend the operation of closed itemset techniques to a distributed environment in the following chapters. The first type of algorithm discussed are the formal concept mining algorithms. They will be discussed in Section 2.2.3. The other type we called frequent closed itemset mining algorithms, which have strong connection to FCA, and they will be discussed in Section 2.3.2.

We adopt the MapReduce programming model when we modify common

Table 2.1: Selected attributes from the Eircom dataset.

| Movie | Music | Technology | News | Lifestyle | FunGame | Sport |
|---|---|---|---|---|---|---|

closure mining algorithms. Hadoop MapReduce[1] and Twister MapReduce runtime[2] will be introduced and the difference between them will also be discussed. We conclude that Twister is most suited to the iterative algorithms, this is supported by the literation [24].

In this thesis, some examples are taken from the Software as a Service Implementation of Predictive Analytics (SaaSiPA) system, developed by the Telecommunications Software and Systems Group (TSSG) located in Waterford Institute of Technology (WIT).

## 2.2 Formal Concept Analysis

### 2.2.1 Definitions

In this chapter, we explain many concepts using a real example provided by Eircom net[3]. Eircom net, a subsidiary of Eircom, provides Ireland's largest telecom's website portal. Through their on-line portal, Eircom offer a range of services and applications to their subscribers and regular internet users. These services and applications range from a news and weather service, to an email facility, crosswords, jokes and horoscopes. The users can discover, personalize and use the best of Eircom's content and services. They can also take these services and share them across the Web, the PC desktop and social networking sites such as Facebook. In the meantime, Eircom keeps logs of user profiles, their activities and typical interactions, such as application sharing, user application intersection and application views. This type of data contains the users' favorites and ARM is used to determine user interests. Note that, ethical rules and privacy preservation are beyond the scope of this thesis.

This data was formatted in the SaaSiPA system and after that only part of the attributes were selected as showed in Table 2.1.

Formal Concept Analysis refers to an unsupervised machine learning method. It was invented in the early 80s by Rudolf Wille [8] and takes

[1]Official website: http://hadoop.apache.org/mapreduce/
[2]Official website: http://www.iterativemapreduce.org/
[3]See the official website: http://widgets.eircom.net/

Table 2.2: Real example from the Eircom dataset. This table indicates the user preferences.

| User | Atrributes |
|---|---|
| 1 | Movie,Music,News,FunGame |
| 2 | Movie,Technology,Lifestyle,Sport |
| 3 | Music,Technology,News,FunGame,Sport |
| 4 | Music,News,Lifestyle |
| 5 | Movie,Music,News,Lifestyle,FunGame |
| 6 | Music,Technology,FunGame,Sport |

as input a matrix specifying a set of objects and the attributes, and finds all the clusters of attributes and objects in the input data. FCA is based on computing closure, ie., mining closed itemset. For this reason we start by introducing it as a method for finding closed itemset in this thesis.

The input of FCA is a binary data table describing the relationship between a set of objects and attributes. We obtain this kind of data table by converting Table 2.2 to a Horizontal Item-Vector (HIV) layout, i.e., using '1' and '0' to denote the occurrence and absence of an attribute. More detail is discussed in Section 2.3.3. Note that, if an object has an attribute, we mark the corresponding entry using an $\times$ symbol, otherwise leave it blank. The data shown in Table 2.3 is called a formal context[4] and is typically presented in the form of $(T, P, R)$, where $T$ is the set of all objects and $P$ is the set of all attributes, and $R$ is the binary relation between $T$ and $P$. In Table 2.3, $T = \{1, 2, 3, 4, 5, 6\}$, $P = \{a, b, c, d, e, f, g\}$, and the object $\{2\}$ has attributes $\{a, c, e, g\}$.

Let $t \in T$ and $p \in P$, for a formal context $R \subseteq T \times P$. To introduce the notion of a cluster in the formal context, we define a derivation operator. For any $Y \subseteq T$ and $X \subseteq P$ we have:

$$Y' = \{p \in P \mid \forall t \in Y : (t, p) \in R\} \tag{2.1}$$

$$X' = \{t \in T \mid \forall p \in X : (t, p) \in R\} \tag{2.2}$$

The operation $Y'$ generates the set of attributes which are shared by all objects in $Y$. Similarly, $X'$ generates the set of all objects which are common to all attributes in $X$. A pair $\langle Y, X \rangle$ is called a formal concept of $(T, P, R)$ if

[4]We use round brackets for arbitrary attribute-object pairs and the formal context, and use angular brackets for formal concepts which are introduced below. Note that, the key/value pair discussed in Section 2.4.2 is also in the form of round brackets.

Table 2.3: An example of a Formal Context. The × symbol in each row indicates that an object has the corresponding attribute.

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| 1 | × | × | | × | | × | |
| 2 | × | | × | | × | | × |
| 3 | | × | × | × | | × | × |
| 4 | | × | | × | × | | |
| 5 | × | | | × | × | × | |
| 6 | | × | × | | | × | × |

and only if $Y \subseteq T$, $X \subseteq P$, $Y' = X$, and $X' = Y$, where $Y$ and $X$ are called the extent and intent. The crucial property of a formal concept is that the mappings $Y \mapsto Y''$ and $X \mapsto X''$ are closure operators. The itemset, $Y''(X'')$, is the closure of $Y(X)$. The closure operator can be used to calculate the extent and intent that form a formal concept.

We define an ordering of all concepts. Given $T_i$, $T_j \subseteq T$ and $P_i$, $P_j \subseteq P$ the concepts of a context are ordered as follows:

$$(T_i, P_i) \leqslant (T_j, P_j) :\Longleftrightarrow T_i \subseteq T_j \Longleftrightarrow P_j \subseteq P_i. \tag{2.3}$$

The set of all formal concepts is denoted by $\mathcal{B}(T, P, R)$. The set $\mathcal{B}(T, P, R)$ and $\leq$ (see section 2.2.3) form a complete lattice [9]. In the following sections we introduce the traditional algorithms that are used to compute formal concepts.

### 2.2.2 Related Work

Some of the best known algorithms for performing FCA include NextClosure [30], Lindig's algorithm [31] and CloseByOne [32, 33] and their variants [34, 35]. Ganter introduces *lectic* ordering so that all possible attribute subsets do not have to be scanned when performing FCA. Ganter's algorithm computes concepts iteratively based on the previous concept without incurring exponential memory requirements. In contrast, CloseByOne produces many concepts in each iteration. Bordat's algorithm which is described in [36] runs in almost the same amount of time as Ganter's algorithm, however, it takes a local concept generation approach. Bordat's algorithm introduces a data structure to store previously found concepts, which results in considerable time savings. Berry proposed an efficient algorithm based on Bordat's approach which did not require a data structure of exponential size, in [37].

A detailed comparison between these algorithms and other algorithms can be found in [12].

The main dis-advantage of the batch algorithms discussed above is that they typically have to construct the lattice from scratch if the database changes. Incremental algorithms address this problem by updating the lattice structure when a new object is added to database. Incremental approaches have been made popular by Norris in [38], Dowling in [39], Godin et al., in [40], Capineto and Romano in[41], Valtchev et al., in [42] and Yu et al., in [43]. In recent years some parallel and distributed algorithms have been proposed. Petr Krajca proposed a parallel version based on CloseByOne in [35]. The first distributed algorithm [20] was developed by Krajca in 2009 using the MapReduce framework [44]. To date, the theory underlying distributed FCA has not been developed. In this thesis, we address this short-coming and develop efficient distributed FCA algorithms.

### 2.2.3 Iterative Closure Mining Algorithm: Ganter's Algorithm

Ganter's algorithm, namely NextClosure, describes a method for generating new closures which guarantees every closure is only generated once. Closures are generated according to a pre-defined order. We continue by describing this ordering, namely lectic ordering, in order to lay the foundations for NextClosure. Let us arrange the elements of $P$ in an arbitrary linear order $P = \{p_1 < p_2 < \ldots < p_i < \ldots < p_m\}$, where $m$ is the cardinality of the attribute set $P$. Subsets of $P$ are ordered linearly according to *lectic* ordering, denoted by $\leq$. For example, given two subsets $P_1$, $P_2 \subseteq P$, $P_1$ is lectically smaller than $P_2$ if the smallest element in which $P_1$ and $P_2$ differ from each other belongs to $P_2$. Formally,

$$P_1 \leq P_2 :\Longleftrightarrow \exists_{p_i}(p_i \in P_2, p_i \notin P_1, \forall_{j<i}(p_j \in P_1 \Longleftrightarrow p_j \in P_2)). \qquad (2.4)$$

We can write this relationship in terms of the smallest element $p_i$ in which $P_1$ and $P_2$ differ.

$$P_1 \leq_{p_i} P_2 :\Longleftrightarrow p_i \in P_2, p_i \notin P_1, \forall_{j<i}(p_j \in P_1 \Longleftrightarrow p_j \in P_2). \qquad (2.5)$$

For example, if the order is defined as $P = \{a < b < c < d < e < f < g\}$, $P_1 = \{a, c, e, g\}$ and $P_2 = \{a, b, e, g\}$ then $P_1 < P_2$ because the smallest element in which the two sets differ is $b$ and this element belongs to $P_2$. Each subset $X \subseteq P$ gives a closure $X'' \subseteq P$. NextClosure attempts to find the list of all closures: this is achieved according to lectic ordering. The basic operation in NextClosure is the $\oplus$ construct which generates a new intent by applying $\oplus$ on an existing intent and an attribute. Note that (Eqn 2.5) is used to check if a new formal concept has been found. We compare the candidate formal concept with the previous. If the condition in (Eqn 2.5) is satisfied, we keep the candidate. The operator $\oplus$ consists of intersection, union and closure operations. Let the ordering be $P = \{p_1 < p_2 < \ldots < p_i < \ldots < p_n\}$, and consider the subset $X \subseteq P$ then the $\oplus$ construct is defined as:

$$X \oplus p_i := ((X \cap \{p_1, \ldots, p_{i-1}\}) \cup \{p_i\})'', \quad \text{where } X \subseteq P \text{ and } p_i \in P. \tag{2.6}$$

(Eqn. 2.6) consists of two operations. First, compute $X \cap \{p_1, \ldots, p_{i-1}\}) \cup \{p_i\}$ which entails removing all elements that are greater or equal to $p_i$ and then adding the element $p_i$ to the set. Second, form the closure. Given an initial closure $X$, let us go through how NextClosure produces new concepts by Example 1.

We are going to calculate concepts for the formal context in Table 2.3. Assume we have a concept $\langle\{1,5\},\{a,d,f\}\rangle$ and $X = \{a,d,f\}$. Now let us calculate $X \oplus e$. First $\{a,d,f\} \cap \{a,b,c,d\} = \{a,d\}$, then by appending $e$ we have $\{a,d\} \cup \{e\} = \{a,d,e\}$. Performing $\{a,d,f\} \oplus e = \{a,d,e\}'' = \{5\}' = \{a,d,e,f\}$. Similarly, we have that $X \oplus c = \{a,c,e\}$. According to (Eqn. 2.5), $\{a,d,e,f\} \leq_c \{a,c,e\}$. $\{a,c,e\}$ is bigger and should be kept. By repeating this process, NextClosure can determine 21 formal concepts totally in this example as shown in Table 2.4.

**Example 1:** Calculate all concepts for the formal context in Table 2.3

**Algorithm 1** First_Closure

```
Input: Closure operator $''$;
Output: $X$.
1: $X \leftarrow \emptyset''$;
2: return $X$
```

Algorithms 1-3 are NextClosure's pseudo code. Algorithm 1 applies the closure operator on an empty attribute set and outputs the first intent $X$, which is the base to later concepts.

Algorithm 2 accepts $P$ and subset $X$, as inputs. Note that, the base set $P$ is sorted in descending order. Line 3 in Algorithm 2 applies (Eqn. 2.6) and produces all the concepts one by one and incurs most of computation cost. Line 4 utilizes (Eqn. 2.5) to verify whether a new candidate should be kept. An candidate satisfied the condition in (Eqn. 2.5) is eligible and is used to replace $X$ in the next iteration.

Table 2.4: Formal Concepts mined from Table 2.3. Note that the empty concepts are included for completeness.

| | | | |
|---|---|---|---|
| $F_1$: | $\langle\{1,2,3,4,5,6\},\{\}\rangle$ | $F_{12}$: | $\langle\{4\},\{b,d,e\}\rangle$ |
| $F_2$: | $\langle\{1,3,5,6\},\{f\}\rangle$ | $F_{13}$: | $\langle\{3\ 6\},\{b,c,f,g\}\rangle$ |
| $F_3$: | $\langle\{2,4,5\},\{e\}\rangle$ | $F_{14}$: | $\langle\{3\},\{b,c,d,f,g\}\rangle$ |
| $F_4$: | $\langle\{1,3,4,5\},\{d\}\rangle$ | $F_{15}$: | $\langle\{1,2,5\},\{a\}\rangle$ |
| $F_5$: | $\langle\{1,3,5\},\{d,f\}\rangle$ | $F_{16}$: | $\langle\{2,5\},\{a,e\}\rangle$ |
| $F_6$: | $\langle\{4,5\},\{d,e\}\rangle$ | $F_{17}$: | $\langle\{1,5\},\{a,d,f\}\rangle$ |
| $F_7$: | $\langle\{2,3,6\},\{c,g\}\rangle$ | $F_{18}$: | $\langle\{5\},\{a,d,e,f\}\rangle$ |
| $F_8$: | $\langle\{1,3,4,6\},\{b\}\rangle$ | $F_{19}$: | $\langle\{2\},\{a,c,e,g\}\rangle$ |
| $F_9$: | $\langle\{1,3,6\},\{b,f\}\rangle$ | $F_{20}$: | $\langle\{1\},\{a,b,d,f\}\rangle$ |
| $F_{10}$: | $\langle\{1,3,4\},\{b,d\}\rangle$ | $F_{21}$: | $\langle\{\},\{a,b,c,d,e,f,g\}\rangle$ |
| $F_{11}$: | $\langle\{1,3\},\{b,d,f\}\rangle$ | | |

**Algorithm 2** Next_Closure

```
Input: $P$, $X$;
Output: $X$.
1: for $p_i$ from $p_n$ down to $p_1$ do
2:   if $p_i \notin X$ then
3:     candidate $\leftarrow X \oplus p_i$;
4:     if candidate $\leq_{p_i} X$ then
5:       $X \leftarrow$ candidate;
6:       break;
7:     end if
8:   end if
9: end for
10: return $X$
```

**Algorithm 3** All_Closure

```
Input: $\emptyset$: null attributes set;
Output: $\mathcal{F}$.
1: First_Closure;
2: while $X$ is not the last Closure do
3:   $X \leftarrow$ Next_Closure;
4:   $\mathcal{F} \leftarrow \mathcal{F} \cup X$;
5: end while
6: return $\mathcal{F}$
```

Algorithm 3 takes charge of the calls for Algorithm 1 and 2. The set $\mathcal{F}$ is the set of all formal concepts. The null set is passed to Next_Closure as an initial formal concept.The while loop from line 2 to line 5 generates new concepts using the previous one until all concepts are found. As the intent is extended with new attributes, the last one should include all attributes (line 2), this feature can be used to measure if the loop comes to the end.

CloseByOne works in a similar fashion to NextClosure, i.e., generating new formal concept based on previous one(s) and verifying using $\leq_i$ operator. The difference is that CloseByOne generates many concepts in each iteration until there are no more concepts satisfying (Eqn. 2.5). While NextClosure only finds the first qualified one. Thus CloseByOne requires much less iterations.

## 2.3 Association Rule Mining Techniques

The origins of ARM can be traced back to market basket analysis, where shop owners wanted to analyse the purchasing patterns of their customers [26]. The resulting analysis illustrated interesting patterns which were used to advise shop owners to engage in certain product placement strategies. Furthermore, it has been applied to many other fields such as healthcare [45, 46, 47] and social network analysis [48, 49] to determine potential rules. The typical dataset for ARM consists of an ensemble of transactions. Rules are derived between the sets of attributes shared by the transactions.

Algorithms for generating rules from transactional data might go as follows: (1) mine frequent itemsets; (2) generate strong association rules from frequent itemsets [50]. Most of the computation involved is due to the first step. The performance of the first step determines the overall performance of association rule mining. A Frequent Itemset is the set of items (or attributes) whose occurrence is greater than a user specified threshold. The threshold determines interestingness for rules. In other words, the higher value causes the fewer rules which have higher possibility. Since its introduction in 1993 by Rakesh Agrawal in the paper [6], FI mining has been an active research topic. To calculate complete frequent itemsets, the first algorithm: Apriori [50] was proposed by Rakesh Agrawal in 1994. Many variants based on Apriori have been developed. FI mining is quite straightforward in that it simply lists all the frequent itemsets and forms rules using their subsets. However, a large amount of evidence indicates that mining frequent pattern by finding all frequent itemsets is not efficient when the dataset to be processed is large and dense [28, 29]. The two drawbacks with this approach are summarized as follows. First, the number of frequent itemsets mined from the transactional data can be large in practice [28]. Second, there is redundancy in the rules particularly when the data is dense [51, 52, 53, 14]. These factors limit the efficiency and effectiveness when generating rules from large datasets using FIs.

Several useful condensed representations of frequent itemsets have been designed over last 10 years, including maximal frequent itemset (MFI) [54], CI [55, 56, 10], $\delta$-Free itemset [57, 58], Disjunction-Free itemset [59, 60], Generalized Disjunction-Free itemset [61], Non-Derivable itemset [62, 63] and the unified framework presented in [64]. All of these alternatives reduce the size of result set because they remove the redundant frequent itemsets.

Among the itemsets, some may have supersets which have the same supports with them. Normally, the itemsets like this can be included by their supersets. Unlike itemset, a closed itemset does not have superset with the same support. A closed itemset approach provides a minimal representation of itemsets without any loss of the support information. Moreover, with closed itemsets, one can directly generate a reduced set of association rules without having to determine all frequent itemsets. Thus, the problem of mining association rules is reduced to determining frequent closed itemsets.

In recent years, maximal frequent itemsets and frequent closed itemsets mining algorithms have been improved in terms of performance and efficiency. For instance, MAFIA in [65] proposed by Doug Burdick, CHARM in [66] by Mohammed J. Zaki, CLOSET+ in [14] by Jianyong Wang, DCI_ CLOSED in [13] by Salvatore Orlando. These algorithms are good for dealing with the small size datasets. However, modern datasets are increasingly distributed and large. Even for some centralized datasets, these algorithms struggle because the datasets are increasingly too large to fit the memory. Hence, it is necessary to develop efficient distributed algorithms to process datasets come from different locations, so that the data does not have to be merged in order to preserve privacy and security.

Mining closed itemsets in distributed environment is still an undeveloped field. Claudio Lucchese gave some preliminary research results in [1] and [21] to combine FCIs from different locations. However, this approach can only be applied with some limitations, for instance, the global FCIs must be local frequent as well in every data partition. We will discuss this by Example 3. Chunhua Ju proposed an algorithm to merge separate local FCIs under different distributed settings in [67]. The drawback of Chunhua's approach lies in scanning datasets many times to find the global frequent closed itemsets [67]. Chun Liu also proposed a novel algorithm to obtain global FCIs with exact support counts (as you will see in the next subsection) in [68] although the evaluation of efficiency was not presented. In the sequel, we are going to introduce FI and FCI mining.

### 2.3.1 Frequent Itemset Mining

Recall Table 2.2 which consists of user identification (ID) and the 7 names of the attributes. Each row records all the attributes that have been accessed by the corresponding user. The attributes (items) form a set $P$. An itemset with $m$ items is called m-itemset. Each row in Table 2.2 is called an object and normally we use the user ID to refer the object. We denote $T$ as the set of objects (we use the same symbols to be consistent with FCA). In this example, P={Movie, Music, Technology, News, Lifestyle, FunGame, Sport} is the set of all items, and subset {Music, News, Sport} is a 3-itemset. There are 6 objects in Table 2.2.

Given a dataset with $N$ objects, the *support*[5] for an itemset $X$, denoted as sup(X), refers to the fraction of objects which contain $X$, i.e., sup(X)= $\frac{N(X)}{N}$, where $N(X)$, is the number of objects which contain $X$. An *association rule* has the form: $A \Rightarrow B$, and denotes the relationship between two itemsets $A$ and $B$ where $A, B \subset P$, $A \cup B = X$ and $A \cap B = \emptyset$. Take the 3-itemset {Music, News, FunGame} as an example, $A$={Music, News}, $B$={FunGame}, $A \cup B = X$, and $A \cap B = \emptyset$, then a possible association rule is {Music, News} $\Rightarrow$ {FunGame}. The support of a rule is defined as the percentage of the objects that contain both $A$ and $B$, denoted as below:

$$\sup(A \Rightarrow B) = P(A \cup B) = \frac{N(A,B)}{N}, \tag{2.7}$$

where $N(A,B)$ is the number of object which contain both $A$ and $B$ at the same time, and $N$ is the number of objects. The confidence of the rule is taken to be the conditional probability and denoted by

$$conf(A \Rightarrow B) = P(B|A) = \frac{N(A,B)}{N(A)}, \tag{2.8}$$

measures the percentage of objects containing $A$ that also contain $B$. The $N(A)$ is the count number of the objects which contains $A$. For the association rule above, the set $X$ = {Music, News, FunGame} occurs in 3 objects and 4 objects contain $A$ = {Music, News}. This means that $N(A,B) = 3$ and $N(A) = 4$, in addition $N = 6$, as indicated by Table 2.2. Hence the support and confidence of this rule are 50% and 75% respectively.

An itemset is frequent if and only if its support is more than or equal to a user-specified minimal support value, $\sigma$, i.e., sup(X)$\geqslant \sigma$. A user prefers setting a lower $\sigma$ to obtain many more rules, and vice verse. In this case, association rule mining can be viewed as a two-step process:

[5]The support is defined here as relative occurrence frequency to total number of objects. Note that it is defined in some of literature as the absolute one, i.e., the support count.

Table 2.5: Frequent itemsets and association rules mined from the data in Table 2.2 when $\sigma = 50\%$ and $\varsigma = 75\%$. The support and confidence for each association rule is listed in the two rightmost columns.

| Frequent Itemsets | Association Rules | sup() | conf() |
|---|---|---|---|
| ({Movie}:3) | | | |
| ({Music}:4) | | | |
| ({Technology}:3) | {Music}⇒{FunGame} | 50% | 75% |
| ({News}:4) | {FunGame}⇒{Music} | 50% | 75% |
| ({Lifestyle}:3) | {Technology}⇒{Sport} | 50% | 100% |
| ({FunGame}:4) | {Sport}⇒{Technology} | 50% | 100% |
| ({Sport}:3) | {News}⇒{FunGame} | 50% | 75% |
| ({Music,Lifestyle}:3) | {FunGame}⇒{News} | 50% | 75% |
| ({Music,FunGame}:3) | | | |
| ({Technology,Sport}:3) | | | |
| ({News,FunGame}:3) | | | |

1. Frequent Itemsets Search: do exhaustive search all frequent itemsets. An itemset is called a frequent itemset if its support is greater than or equal to $\sigma$.

2. Rule Generation: for each frequent itemset $X$ found, generate all association rules $A \Rightarrow B$, if its confidence is greater than or equal to the confidence threshold $\varsigma$. Like $\sigma$, the $\varsigma$ is also specified by users, and it determines the confidence level of rules. Therefore $\varsigma$ is critical for generating strong association rules which more probably occur in the corresponding dataset.

Let us examine the example in Table 2.2. Assume $\sigma = 50\%$ and $\varsigma = 75\%$, then we can obtain 11 frequent itemsets and 8 association rules as shown in Table 2.5.

Note that there exists some redundancy in the frequent itemsets in Table 2.5, for example, $(\{\text{Technology}\} : 3)$ and $(\{\text{Sport}\} : 3)$, which are subsets of $(\{\text{Technology}, \text{Sport}\} : 3)$. In fact, frequent itemset mining often generates a huge number of itemsets satisfying the minimum support threshold $\sigma$, especially when $\sigma$ is set low. This is because the subsets of a frequent itemset is frequent as well. For a n-itemset, the number of its subsets up to

$$\binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{n} = 2^n - 1.$$

Hence, the computational requirements for frequent itemset generation are generally more expensive than those of rule generation [6, 69]. The overall performance of mining association rules is determined by the first step. So, we mainly discuss the algorithm for mining frequent itemsets. However, as the reader will see soon, finding frequent closed itemset is more efficient for rules generation than the traditional frequent itemset mining, we start introducing it in the following section.

### 2.3.2 Frequent Closed Itemset Mining

Recall the closure operator introduced in Section 2.2.1, which provides a method to calculate closure. Normally, the result of closure operation is known as closure in FCA, while we call it closed itemset in ARM. The closure operation in FCA is the theoretical foundation of frequent closed itemset, however, FCA does not consider support information of the closed itemsets which is needed by FCI. In the following, we restate the definition of FCI, denoted by $I$, and introduce some of its properties.

**Definition 1** *An itemset $X$ is closed if none of its immediate supersets has exactly the same support count as $X$.*

In this definition, an immediate superset refer to the most closed superset when there are many supersets. Assume $X_1$ and $X_2$ are subset and immediate superset of $X$ respectively. According to the support definition, the support of an itemset is never greater than the support of its subsets. So we have $\sup(X_1) \geqslant \sup(X) \geqslant \sup(X_2)$. When the support of $X$ is greater than the support of all $X$'s immediate supersets, i.e., $\sup(X) > \sup(X_2)$, we call $X$ closed itemset. Naturally, $X$ is not closed if at least one of its immediate supersets has the same support as $X$.

**Definition 2** *An itemset is a Frequent Closed Itemset if it is closed and its support is greater than or equal to the minimal support $\sigma$.*

Formally we have that:

**Theorem 1** *An itemset $I$ is said to be closed if and only if*

$$I'' = (I')' = I$$

*where $I''$ is the Galois operator or closure operator.*

Theorem 1 shows that, if an itemset $I$ goes through the closure operator and returns the same itemset, then $I$ is closed.

### 2.3.3 General Process for Mining FCI

Our approach for mining frequent closed itemset in a distributed way is on the basis of centralized FCI mining algorithm. The performance and structure of adopted centralized FCI mining algorithms effect the distributed algorithm partially. Naturally, it is necessary to know general process for mining FCIs in centralized dataset. We divide the process of mining frequent pattern into 5 sub problems and describe them below. They are:

1. Database Representation and Compressed Structure;
2. Searching Strategy;
3. Projecting Method;
4. Checking and Pruning Techniques;
5. Results Storing.

#### Database Representation and Compressed Structure

The first problem we have to face when mining frequent patterns from a given database is how to represent them. The representation format of a dataset influences the performance of the mining algorithm. In practical applications, objects can be presented in Horizontal Item-List (HIL), Vertical TID-List (VTL), HIV and Vertical TID-Vector (VTV) form. The HIL layout stores one object in each row. It is an intuitive method of bookkeeping for the objects. Each object is recorded as a list of items, as shown in Table 2.2. HIL is adopted by most association rule mining algorithms, such as Apriori, CLOSET, and CLOSET+. The VTL layout stores the list identifiers associated with each item, instead of recording the transactions explicitly [70], as shown in Table 2.6. VTL were- proposed by Holsheimer and Savasere on 1995 and it is adopted in CHARM and Eclat [71] algorithms. HIV is similar to HIL, except that a bit-vector is used to represent the presence or absence of each item in each row respectively, as shown Table 2.7. In Table 2.8, the VTV layout is similar to VTL except that a bit-vector is used to represent the presence or absence of each item in terms of objects. The difference between HIV and VTV is that the later organizes the data as a set of columns.

When the dataset is huge, it should be compressed and only the information related to the mining is kept. The compressed structure can be array-based [72, 73], tree-based [14, 28, 72, 74], which is the key factor in the efficiency of the projection operation and the counting operation. Among of

Table 2.6: The data in Table 2.2 arranged by vertical TID-list layout.

| Movie | Music | Technology | News | Lifestyle | FunGame | Sport |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 1 | 2 | 1 | 2 |
| 2 | 3 | 3 | 3 | 4 | 3 | 3 |
| 5 | 4 | 6 | 4 | 5 | 5 | 6 |
|  | 6 |  | 5 |  | 6 |  |

Table 2.7: The data in Table 2.2 arranged in horizontal item-vector layout.

| User | Movie | Music | Technology | News | Lifestyle | FunGame | Sport |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 2 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 3 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 5 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

Table 2.8: The data in Table 2.2 arranged in vertical TID-vector layout.

| User | Movie | Music | Technology | News | Lifestyle | FunGame | Sport |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 2 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 3 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 5 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

the data representations, the tree-based structure is often used and is efficient for projection. We will discuss FP-tree in detail in Section 3.1.

### Searching Strategy

Mining frequent closed itemsets is actually a process of searching for FCIs from a lattice structure. The search strategy employed by an algorithm dictates how the lattice is traversed during the frequent itemset generation process. Three types of strategies often used are:

- **General-to-specific and Specific-to-general**: In the general-to-specific (GTS) approach, candidate closed itemsets are obtained by merging pairs of frequent (k-1)-itemsets, and then checking candidate k-itemsets to see if they are frequent. The first algorithm which adopted this kind of traversal was Apriori.

- **Equivalent Classes**: This approach partitions the lattice into disjoint group nodes first and then processes each part in turn. As an example, the level-wise strategy used in Apriori partitions the lattice on the basis of itemsets size, i.e., the algorithm discovers all frequent 1-itemset first before proceeding to larger-sized itemsets. See Figure 2.1, the prefix-tree and suffix-tree belong to this type.

- **Breadth-first and Depth-first**: The Apriori algorithm also can be implemented in a breadth-first manner traversal. It first discovers all frequent 1-itemsets, followed by the frequent 2-itemsets, and so on, until no new frequent itemsets are generated.

### Projecting Method

The projecting operation is a process of filtering and reorganizing datasets, which could be stored in database or compressed structure, into different smaller sub sets. When the original database is large, it maybe unrealistic to read the dataset into memory or even construct a memory-based storing structure, such as a tree and an array. Moreover, when we adopt pattern-growth and depth-first approach to mining frequent patterns, it is also needed to project the dataset recursively to find out local frequent itemsets. Hence, we need to first partition the database into a set of projected databases.

The existing projecting methods could be classified into three categories according to the way the datasets are stored: database projection [28, 75], tree projection [14, 73], and array projection [72, 76].

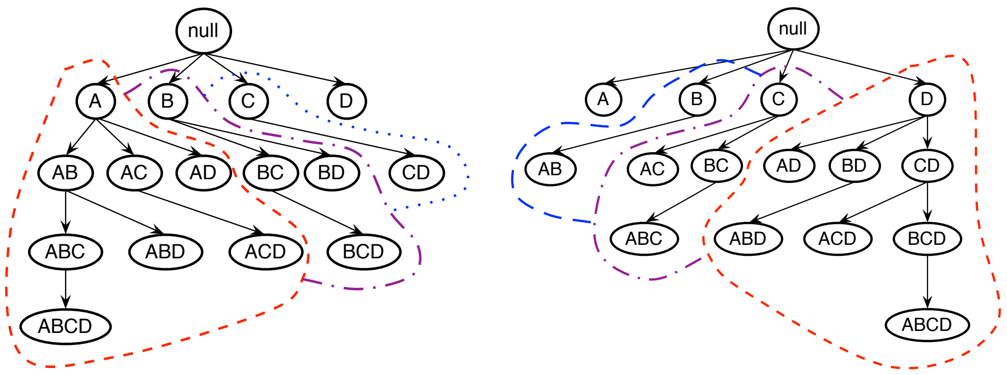

Figure 2.1: Equivalent classes based on prefix label (left) and suffix label (right) of itemsets. The itemsets having the same prefix label and suffix label (circled by dashed lines) are explored first.

### Checking and Pruning Techniques

We may need to check the candidate itemset in order to recognize one FCI quickly. Although we can achieve this by directly counting its support, sometimes it is not necessary because there are some basic properties that can be used. For instance, pruning techniques can be used to eliminate the itemset which is never used to generate FI (MFI, or FCI). This is useful for further pruning the search space and speeding up mining.

### Results Storing

Most existing algorithms store the frequent (closed) itemsets in memory or disk immediately without the need for special compressed structures, because they do not use the results. Apriori, FP-growth, A-close, CLOSET, etc. are particularly good examples of this type of algorithm.

There also exist some algorithms which mine frequent pattern by constructing a Frequent Item Sets Tree (FIST) [73]. Once constructed, all frequent itemsets can be obtained by reading the path of the FIST.

CHARM [66] improves mining efficiency by exploring an item based tree structure and only the frequent closed itemsets can be kept in the tree. To some degree, we can view this approach as a tree based results storing approach. CLOSET+ adopts two techniques to store the results and assist subset-checking: Two-level hash-indexed result tree and Pseudo-projection based upward checking. Two-level hash-indexed result tree, which is a special hybrid structure, can provide the ability to store results and verify the closure of frequent itemsets when they are imported. In contrast, pseudo-projection based upward checking removes the requirement of maintaining the set of closed itemsets in memory for subset-checking. It uses only the global FP-tree to check if a newly found frequent itemset is closed. Hence, the results, FCIs, can be directly stored in an output file.

This thesis uses CLOSET+ as a representation for recursive FCI mining algorithm because it can choose different mining strategies to handle sparse and dense data. The detail will be discussed in the following subsection.

### 2.3.4 Recursive Closure Mining Algorithm: CLOSET+

CLOSET+ follows the popular divide-and-conquer paradigm and the depth-first search strategy. It uses FP-tree as the compression technique.

Figure 2.2 shows a flowchart of CLOSET+. In the tree building process, we compute the average count of an FP-tree's nodes. After the tree has been built, we judge whether the dataset is dense or sparse according to the average count of an FP-tree node. At this moment, we use a hybrid tree-projection method to construct its projected databases. For dense datasets, the algorithm chooses a bottom-up physical tree-projection method; whereas for sparse datasets, the algorithm uses top-down pseudo tree-projection method. Closet+ deals with FP-tree in a top-down manner for sparse datasets, and in a bottom-up manner for dense datasets. During the mining process, use the item merging, item skipping, and sub-itemset pruning methods to prune search space. For each candidate frequent closed itemset, use the two-level hash indexed result tree method for dense datasets or pseudo-projection based upward checking method for sparse datasets to do closure checking. When all the items in the global header table have been mined the algorithm stops.

CLOSET+ works well on single machine to process data which fit the memory. However, CLOSET+ and other similar algorithms could not handle the data distributed on different places because all information it needs, such as the list of frequent itemsets, are relative to the whole dataset. We figure this issue out in section 3.3.2.

## 2.4 Distributed Closed Itemset Mining Algorithm

### 2.4.1 Related Work

The current work on distributed frequent closed itemset mining is still at an early stage in its development [21]. The traditional distributed data mining

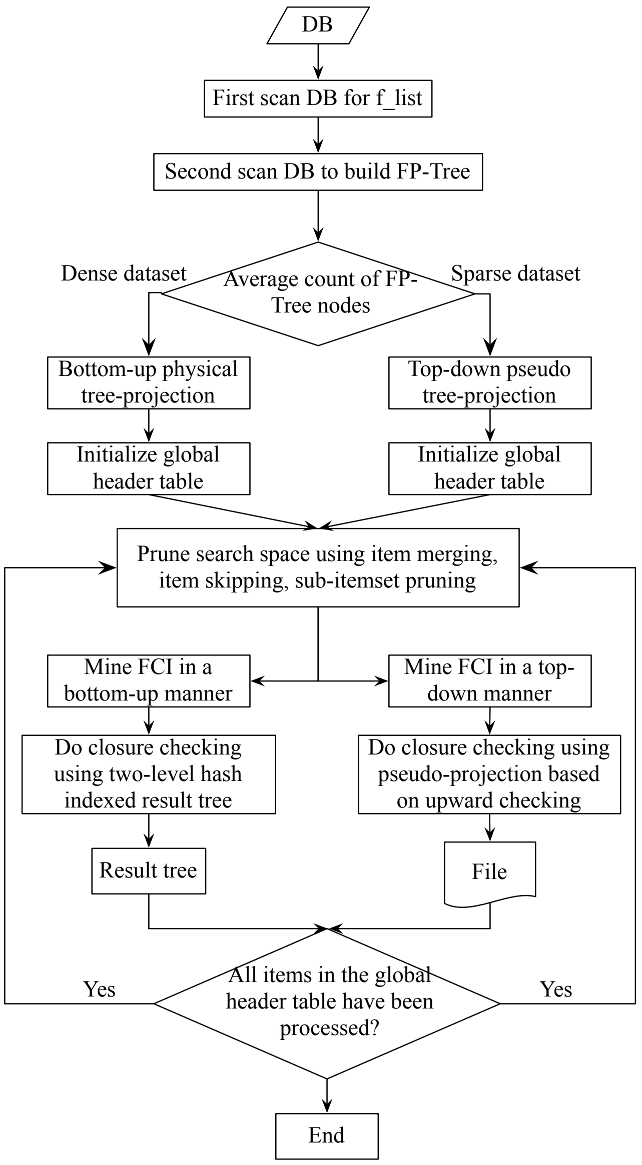
DB
First scan DB for f_list
Second scan DB to build FP-Tree
Dense dataset
Average count of FP-Tree nodes
Sparse dataset
Bottom-up physical tree-projection
Top-down pseudo tree-projection
Initialize global header table
Initialize global header table
Prune search space using item merging, item skipping, sub-itemset pruning
Mine FCI in a bottom-up manner
Mine FCI in a top-down manner
Do closure checking using two-level hash indexed result tree
Do closure checking using pseudo-projection based on upward checking
Result tree
File
Yes
All items in the global header table have been processed?
Yes
End

Figure 2.2: Flowchart of CLOSET+. CLOSET+ handles dense data and sparse data in a bottom-up physical tree-projection and a top-down pseudo tree-projection respectively, and adopts different closure checking strategies. For dense data, the mined FCIs are stored in a result tree. They are stored in a file in the case of sparse data.

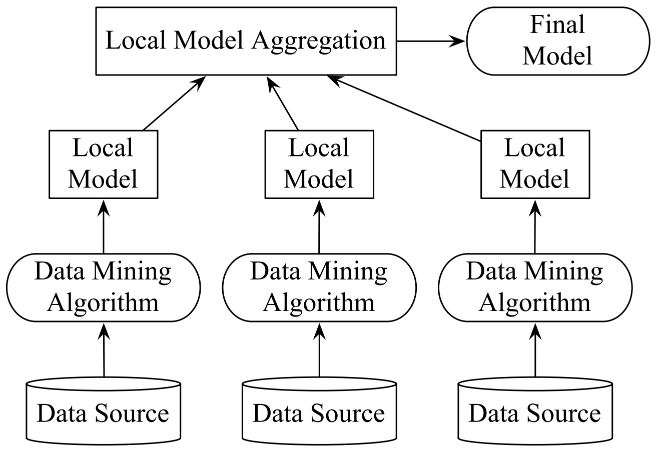

Figure 2.3: Traditional distributed data mining framework: the same algorithm works on each data site and the outputs are aggregated in a central site.

framework shown in Figure 2.3 was applied in an early distributed ARM algorithm in [77]. In this framework, the same algorithm operates on each distributed data site concurrently, producing one local model per site. Subsequently all local models are aggregated to produce the final model. Each local model represents the local itemsets but lacks details that may be required to induce globally meaningful knowledge. Moreover, the traffic from a Local Model to the Final Model may be heavy. Therefore, data transfer and synchronization are the most important considerations for this kind of approach. Two existing distributed FCI mining algorithms based on this framework are given in [67, 68]. In order to focus on algorithm design, we introduce the MapReduce framework to deal with distributed communication and management. The details of MapReduce are discussed in Section 2.4.2.

The other crucial problem that must be addressed when designing distributed FCI mining algorithms is the process of merging the local FCIs. Local FCIs from every part of distributed data are supersets of the global FCIs. Some important information such as support, which is required by global FCIs, hide in the local Infrequent Closed Itemset (IFCI)s and is prone to be lost when the merging procedure is performed. To explain this by Example 2, we divide the data in Table 2.3 into 2 parts $\mathcal{D}_1$ and $\mathcal{D}_2$ , as shown in Table 2.9.

Consider a distributed database $\mathcal{D}$ consisting of $n$ partitions $\mathcal{D}_i$, where $i = 1, \ldots, n$.

**Property 1** *If $I$ is a globally frequent closed itemset, then $I$ must be locally*

Table 2.9: Distributed data partitions $\mathcal{D}_1$ and $\mathcal{D}_2$ derived from Table 2.3.

$\mathcal{D}_1$

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| 1 | × | × | | × | | × | |
| 2 | × | | × | | × | | × |
| 3 | | × | × | × | | × | × |

$\mathcal{D}_2$

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| 4 | | × | | × | × | | |
| 5 | × | | | × | × | × | |
| 6 | | × | × | | | × | × |

Here are two closed itemsets $I_1 = (\{b, d, f\} : 2)$ and $I_2 = (\{b, c, f, g\} : 1)$ determined from $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively in Table 2.9. Given a minimal support $\sigma = 50\%$, then $(\{b, c, f, g\} : 1)$ is infrequent and should be discarded. However, $(\{b, c, f, g\} : 1)$ can contribute support information to itemsets $\{b\}$ and $\{f\}$, and result in a global FCI $I_3 = (\{b, f\} : 3)$. $I_3$ can be found in $F_9$ in Table 2.4.

**Example 2:** Support information loss of IFCIs when merging local FCIs

*frequent in at least one partition $\mathcal{D}_i$.*

**Proof.** This property can be proved by contradiction. Suppose a globally frequent closed itemset $I$ is infrequent in every partition. That means $I$'s support count is lower than the minimal support count in every partition. Then the total support count of $I$ must be smaller than $\sigma$, so $I$ is infrequent with respect to the whole data. The hypothesis does not hold.

There are some preliminary methods for merging FCIs. The authors of [1, 21] presented an operation denoted by the symbol, $\boxplus$, to compute global FCIs[6] in a distributed manner as shown in the following definitions.

**Definition 3** *Given the two sets of frequent closed itemsets $\mathcal{C}_1$ and $\mathcal{C}_2$, mined respectively from the two partitions $\mathcal{D}_1$ and $\mathcal{D}_2$, we have that:*

$$\overline{\mathcal{C}} = \mathcal{C}_1 \boxplus \mathcal{C}_2 =$$

$$(\mathcal{C}_1 \cup \mathcal{C}_2) \cup \{I_1 \cap I_2 \mid (I_1, I_2) \in (\mathcal{C}_1 \times \mathcal{C}_2)\} \equiv \mathcal{C}.$$

[6]The authors of [1, 21] adopted $\oplus$ instead of $\boxplus$. We use $\boxplus$ to distinguish from $\oplus$-structure in Section 2.2.3 in order to avoid notational confusion

For example in Table 2.9, we set the support $\sigma$ to 33% so that $\mathcal{C}_1 = \{(\{b,d,f\} : 2), (\{a\} : 2)\}$ and $\mathcal{C}_2 = \{(\{a,d,e,f\} : 1)\}$, then the global solution will be $\overline{\mathcal{C}} = \{(\{b,d,f\} : 2), (\{d,f\} : 3), (\{a\} : 3), (\{a,d,e,f\} : 1)\}$.

Definition 3 is trivially extended to $n$ partitions.

**Definition 4** *Given the sets of closed itemsets $\mathcal{C}_1$,$\cdots$,$\mathcal{C}_n$, mined respectively from $n$ disjoint horizontal partitions $\mathcal{D}_1$,$\cdots$,$\mathcal{D}_n$, we have that:*

$$\overline{\mathcal{C}} = (\cdots((\mathcal{C}_1 \boxplus \mathcal{C}_2) \boxplus \cdots \boxplus \mathcal{C}_n)\cdots).$$

Definitions 3 and 4 suggest that we may have to perform many intersection and union operations between the local closed itemsets. However, the $\boxplus$ operation only works when all local FCIs are also globally frequent. The Example 3 shows an exception:

Consider the data in Table 2.9, and let $\sigma = 66\%$. We can find that $(\{c,g\} : 2)$ is a local frequent closed itemset in $\mathcal{D}_1$. According to Definition 3, $(\{c,g\} : 2)$ is included in global FCIs. However, $(\{c,g\} : 2)$ is infrequent in terms of the whole data.

**Example 3:** An exception for Definition 3

The author of [68] develops another method which fully utilizes the infrequent closed itemsets in each data source. As a supporting lemma, Lemma 7 in [68] considers the relation between global FCIs and local IFCIs. It has only been proved for a special case, i.e. when distributed data consists of two partitions. We propose a novel method which utilizes infrequent itemsets to achieve the same aim. The details are discussed in Section 3.3.2.

The existing distributed work on FCA is mentioned in Section 2.2.2. We continue by introducing the MapReduce framework in the following sections.

### 2.4.2 MapReduce Framework

MapReduce adopts a divide-conquer strategy to deal with huge datasets on various kinds of problems which can be tackled in a distributed manner by a large number of computers, collectively referred to as a cluster. The benefit of using MapReduce lies in running an algorithm in its ability to tackle the problem in a distributed way.

MapReduce is inspired by the map and reduce functions commonly used in functional programming, such as Lisp. It was introduced by Google [44] and then implemented by Google, Yahoo! and Twister, and organizations such as Apache. These implementations provide automatic parallelization and distribution, fault-tolerance, I/O scheduling, status and monitoring. The

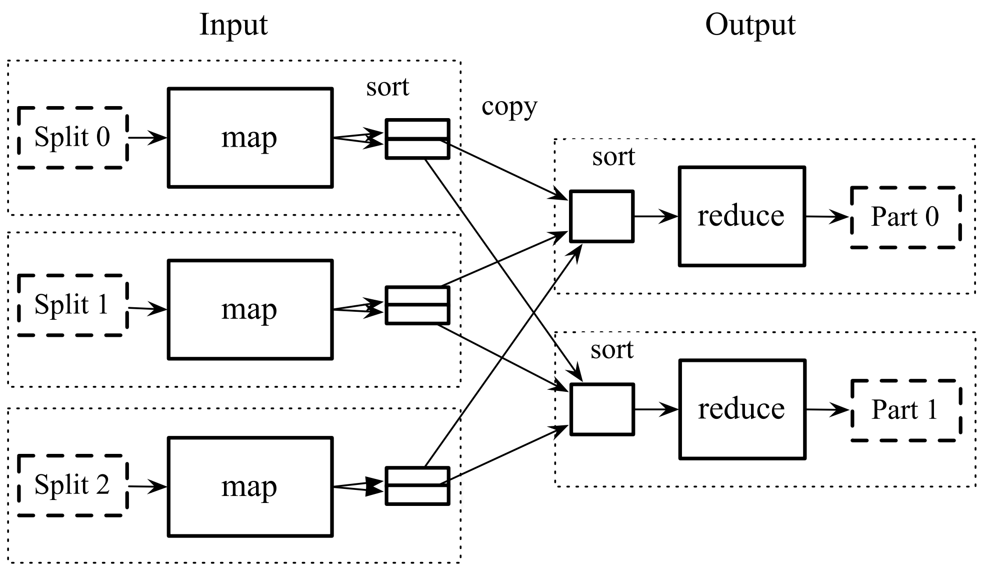

Figure 2.4: MapReduce data flow: the blocks refer to nodes and the arrows indicate the directions the data flows in. The map and reduce functions normally lie on the same node and deal with the data partition in that node. The reduce function combines the outputs of the map phases.

only thing that needs to be done by the user is to design the map and reduce functions for the specific problem.

The map function[7], or map(), takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library provides the ability to acquire an input pair from files or databases which are stored in a distributed way. Additionally, the map function can group all intermediate values associated with the same intermediate key K and pass them to the same reducer.

The reduce function, or reduce(), accepts an intermediate key K and a set of values associated with K. It merges these values to form a possibly smaller set of values.

Figure 2.4 depicts how MapReduce works. The flow of control of MapReduce is described as follows.

1. The MapReduce library splits the input files into $M = 3$ partitions in Figure 2.4. It then runs many copies of the program on a cluster of machines (nodes). One partition may be allocated to many nodes.

[7]Note that, we often use mapper and reducer to stand for map function and reduce function respectively.

2. One of the nodes is assigned to be the master, and the rest of the nodes are common workers nodes or slaves. In this example, node 0 is assigned to act as the master. The master assigns the map and reduce tasks to idle slaves and it is also responsible for managing the slaves and all communication. All the slave nodes must response to the master once they have finished or returned a faulted so that the master is able to take further actions.
3. A mapper parses the (key,value) pairs produced from the input data and then processes them by the user-defined map function. New intermediate (key,value) pairs are produced and buffered in memory.
4. Periodically, the buffered pairs are written to local disk. The locations of these buffered pairs are passed back to master, who is responsible for forwarding this information to each of the reducers.
5. Once the mappers are finished, the master sends the mappers' locations to the reducers and uses remote procedure calls to read the data buffered by the mappers. Normally, the intermediate keys are sorted in order to group the same keys.
6. A reducer then iterates over the sorted intermediate data and handles the key and the corresponding values using the user's reduce function. The output is stored in a local file.
7. The master wakes up the user main program once all map tasks and reduce tasks have been completed.

When the MapReduce framework was designed by Google, it followed the principles above. However, MapReduce implementations have been modified to solve particular problems. Note that, Twister is particularly suitable for iterative tasks.

### 2.4.3 Twister MapReduce

Twister [24] was designed in order to make MapReduce suitable for iterative algorithms. Normally, there are two types of data in iterative algorithms, static and dynamic data. The static data is the distributed data in a local machine, while dynamic data is produced by the last iteration. For some data mining tasks, static data is needed during each iteration in conjunction with dynamic data to generate fresh dynamic data. The requirement for both static and dynamic data is the crucial difference between Twister and other MRF implementations. The static data can be set up in the "configure"

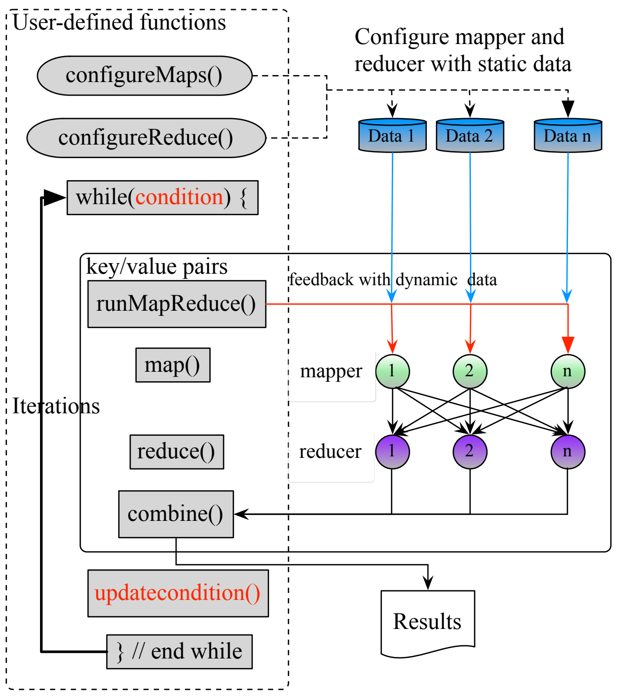

Figure 2.5: Iterative MapReduce programming model supported by Twister. This model introduces static and dynamic data and iteratively executes the tasks.

phase for the map and the reduce task. The "configure" phase allows the user to specify the source for static data and how to send the dynamic data back to mappers at the same time, as shown in Figure 2.5.

In order to avoid reading static data in each execution of MapReduce, Twister designs the MapReduce as long running tasks which last for the life of the whole computation. All of the communication between the nodes and between the mappers and the reducers is handled by a broker network. Twister supports NaradaBrokering which is described in [78] and ActiveMQ[8]. Figure 2.6 shows Twister architecture.

Apart from the iterative feature, Twister provides a combining phase (combine()) which can further merge the outputs from reducers, and gives user the ability to customize the broker networks in order to manage communication as need.

[8]More details can be found at http://activemq.apache.org/

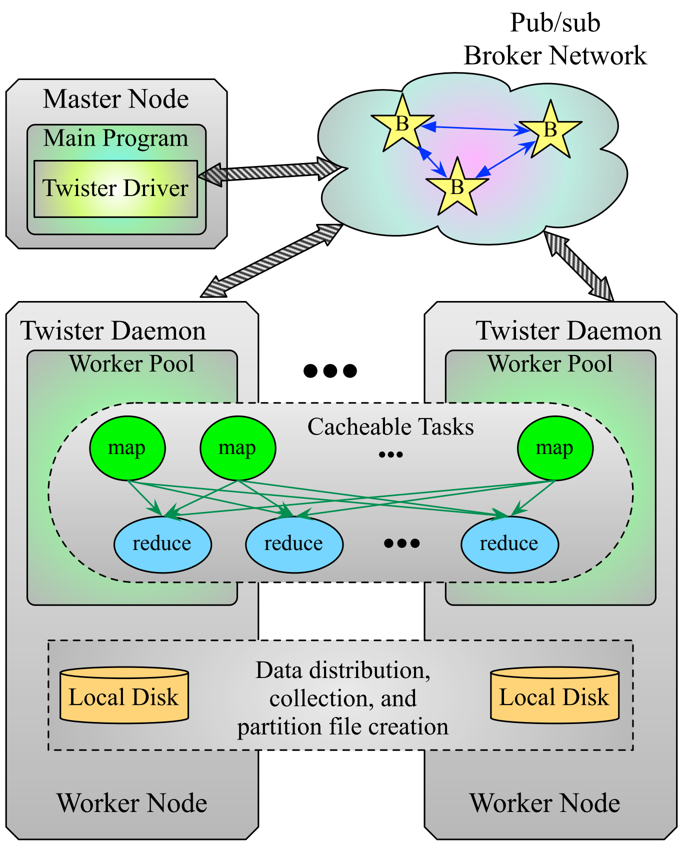

Figure 2.6: Architecture of Twister MapReduce. Twister maintains pools to cache mappers and reducers, and to read the input from the local disk of each node. The third party broker network is used to manage the communication between the worker nodes.

## 2.5 Summary

Closed itemset mining is an important concept in both ARM and FCA, and a novel MapReduce framework can be applied on it. Section 2.3 defined key words and terminologies, such as frequent itemset, frequent closed itemset. The background knowledge, related work and classic frequent closed itemset mining algorithms were also presented. Section 2.2 first defined formal concept analysis and its related properties, and then discussed recent related work including centralized and distributed algorithms. The NextClosure and CloseByOne algorithms were introduced at the end of this chapter. Centralized algorithms were discussed in detail in order to set the sense for the following chapters. in following sections. Section 2.4 covered concepts related to distributed implementations, such as the existing distributed frequent closed itemset mining methods, MapReduce framework and Twister MapReduce runtime. In the next chapter we will discuss some techniques for distributed implementation of these algorithms.

Chapter 3

# Theoretical Basis

In Chapter 2 we described the origin and current status of distributed closed itemset mining. We now list some open problems to clarify the situation and introduce potential solutions.

## 3.1 Domain Related Issues

To achieve distributed computation of Frequent Closed Itemsets (FCIs), a first approach might be to develop or modify existing algorithms, and then to deploy them over different network architectures. This approach is impractical because:

1. Most frequent closed itemset mining algorithms are not suitable for execution in a distributed environment. This is because FCI mining requires support and occurrence information for the whole dataset for some itemsets as the procedure progresses. That is to say, the algorithms need to access and search the dataset several times. It is inefficient to do so in distributed settings.

2. The existing FCI mining algorithms are good at processing datasets with certain characteristics: some are good for dense data, and some are good for sparse data. It is unfeasible to modify them to suit different structures and implementations as discussed in Section 2.3.3. Moreover, they differ from each other in the way the problem is modelled and solved, consequently it may reduce their performance if they are made distributed without due consideration of the characteristics of the approach.

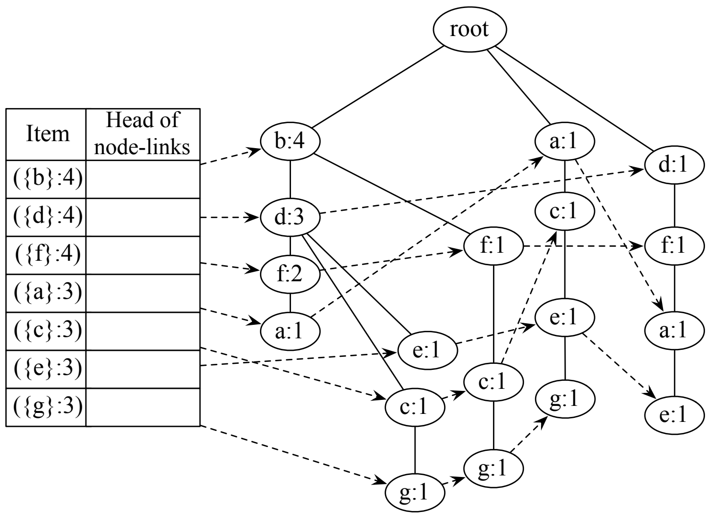

Figure 3.1: FP-tree structure for data in Table 2.3, which consists of diverse itemsets and is linked with an ordered head table which is located on the left. The same item from different itemsets forms an item node which is labelled by the item and its support count in the path. All nodes are linked ordered by descending frequency as indicated in the head table.

To understand the problems above, we analyze some prominent FCI mining algorithms.

Case 1: FP-growth, OP, CLOSET and CLOSET+ are FP-tree (FP refers to Frequent Pattern) based algorithms [74, 79], which project the whole dataset into a tree structure and then recursively mine the FCIs in the tree. The FP-tree contains all information of the original dataset and is stored in memory, as shown in Figure 3.1. In distributed cases, we may have to acquire knowledges from each data partition in order to construct the FP-tree which could be very large. It is a challenge to store it in (distributed) memory because of the size of the dataset.

Thinking in another way, we can obtain global FCIs by combining the local FCIs. Recall Definitions 3 and 4, we must ensure that all the local FCIs are also globally frequent. In fact, it is hard to do so. A globally frequent closed itemset $I$ might be infrequent on a given partition $\mathcal{D}_i$, so $I$ will not be returned as frequent closed itemset by the $i^{th}$ node. As consequence of this, the master node can not count the support of $I$ because of the missing knowledge on $\mathcal{D}_i$. Till now, we have to re-access all datasets to achieve the goal above.

Case 2: In practice, in the distributed settings some choices must be made with regard to trading off the number of nodes, the bandwidth between the nodes, given the characteristics of the datasets. To obtain good performance, we need to look for a workaround. In addition, as discussed in [28, 74], FP-growth is suitable for sparse datasets, while CLOSET is good for dense datasets. In other words, even using the same algorithm, differences in dataset characteristics will result in the variation of the communication flow. The main bottleneck, in practice, is that large amounts of data have to be transmitted among different work nodes because items have to be shared.

## 3.2 Methodology

We adopt the following methodology to address these issues:

1. Assumptions

    - We use the Twister distributed infrastructure in this thesis. We assume the bandwidth in the network is large enough.
    - We assume that the datasets used for the experiments are even distributed across the nodes, i.e., the closed itemsets are well-distributed. We divide the data randomly and the data partitions have the same size and similar density.
    - There are many techniques, such as closure pruning, to optimize the closure computation for itemsets. We do not focus on these techniques in order to make our algorithms more scalable and applicable to a wide variety of scenarios.
    - All algorithms in this thesis execute closure mining rather than rules generation or concept lattice construction. This is because rules generation and concept lattice construction consumes extra time.

2. Analysis Plan

    - We have surveyed the status of frequent closed itemset mining and implemented some prominent algorithms. By analysing them, we identified some similarities and differences between them, and grouped them into two categories.
    - We consider distributed FCI mining and distributed formal concept mining separately because FCI mining relies on support but formal concept mining does not.

- We use the CPU time of our algorithms to measure their performance. Different types of datasets are tested over various numbers of machines to determine the algorithms' overall performance and scalability. For example, the types of different datasets are characterized by their density, the number of attributes (dimension) and the number records (data size). The number of nodes was varied from 1 to 10. The comparisons between distributed algorithms, and distributed versions and the corresponding centralized versions are given as a result.
- All algorithms are implemented in the Java programming language and executed as Java applications in the Twister runtime, in the same hardware environment to ensure that all experiments are comparable.

## 3.3 Distributed Closed Itemset Mining Algorithms

In light of the issues mentioned in Section 3.1, we adopt the MapReduce framework to support both distributed formal concept mining and FCI mining. Note that, we do not suggest that all the problems can be solved by simply implementing map and reduce functions. On the contrary, we classify the centralized algorithms using two categories, so that they can be treated by different approaches. Basically, the prerequisite for applying MapReduce is that the task must be amenable to decomposition in to a map function and a reduce function. Thus, according to this criterion, we classify the common closure mining algorithm into two categories.

- Category 1: algorithms which work in a non-recursive manner, and can be decomposed into two or more tasks. For this type of algorithm, we need to adapt them to the MapReduce framework by modifying the inner functions.
- Category 2: recusive algorithms.

The FCA algorithms in this thesis belong to Category 1, while most of FCI mining algorithms belong to Category 2. To develop distributed algorithms, we are going to introduce some theories to support this process for the two categories respectively.

### 3.3.1 Distributed Formal Concept Mining

In Section 2.2.3 we said that the computing resource consumed by the $\oplus$ construct will increase significantly as the dataset size grows. A potential solution to this problem is to deploy this task on different computing units. The approach taken by MapReduce. Naturally, the question is how to decompose NextClosure, so that each component can be executed in parallel. This section talks about this in detail.

Given a dataset $\mathcal{D}$, we partition its objects into $n$ subsets and distribute the subsets over $n$ different nodes. Without loss of generality, it is convenient to limit $n = 2$ here. We denote the partitions by $\mathcal{D}_1$ and $\mathcal{D}_2$. Alternatively we can think in terms of formal contexts and write the formal context, $(T, P, R)$, in terms of the partitioned formal contexts $(T_{\mathcal{D}_1}, P, R_{\mathcal{D}_1})$ and $(T_{\mathcal{D}_2}, P, R_{\mathcal{D}_2})$. To fix ideas, we use the dataset in Table 2.9 as an exemplar. The partitions are non-overlapping: the intersection of the partitions is the null set, $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$ and their union gives the full dataset $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2$. It follows that the partitions, $\mathcal{D}_1$, $\mathcal{D}_2$, have the same attributes sets, $P$, as the entire dataset $S$, however, the set of objects is different in each partition, e.g. $T_{\mathcal{D}_1}$ and $T_{\mathcal{D}_2}$.

Let $X_{\mathcal{D}}$, $X_{\mathcal{D}_1}$ and $X_{\mathcal{D}_2}$ denote an arbitrary attribute set $X$ with respect to the entire dataset $S$, and each of its partitions $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively. By construction they are equivalent: $X_{\mathcal{D}} \equiv X_{\mathcal{D}_1} \equiv X_{\mathcal{D}_2}$. Similarly, $X'_{\mathcal{D}}$, $X'_{\mathcal{D}_1}$ and $X'_{\mathcal{D}_2}$ are the sets of objects derived by the derivation operation in each of the partitions $\mathcal{D}_1$, $\mathcal{D}_2$ and the entire dataset $\mathcal{D}$ respectively.

In this subsection, we use $F^X_{\mathcal{D}_i}$ to denote the formal concept derived from itemset $X$ in terms of $\mathcal{D}_i$, and use $\mathcal{F}_{\mathcal{D}_i}$ to denote the set of formal concepts mined from $\mathcal{D}_i$. We now propose some basic properties for distributed FCA in the following content.

**Property 2** *Given the formal context, $(T, P, R)$, the two partitions $(T_{\mathcal{D}_1}, P, R_{\mathcal{D}_1})$ and $(T_{\mathcal{D}_2}, P, R_{\mathcal{D}_2})$ and an arbitrary itemset, $X \subseteq P$, the property $X'_{\mathcal{D}} = X'_{\mathcal{D}_1} \cup X'_{\mathcal{D}_2}$ holds: the union of the sets of objects generated by the derivation of the attribute set X in each of the partitions is equivalent to the set of objects generated by the derivation of the attribute set over the entire dataset, $\mathcal{D}$.*

**Proof.** Appealing to the definition of the derivation operator proposed by Wille in [8], the set, $X'_S$, is a subset of $T$, $X'_{\mathcal{D}} \subseteq T$. Moreover, $X'_{\mathcal{D}_1} \subseteq T_{\mathcal{D}_1}$ and $X'_{\mathcal{D}_2} \subseteq T_{S_2}$. Given $\mathcal{D}_1 \cup \mathcal{D}_2 = \mathcal{D}$ and $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$, it follows that, $T_{\mathcal{D}_1} \cup T_{\mathcal{D}_2} = T$ and $T_{\mathcal{D}_1} \cap T_{\mathcal{D}_2} = \emptyset$; Therefore, $X'_{\mathcal{D}_1} \subseteq X'_{\mathcal{D}}$ and $X'_{\mathcal{D}_2} \subseteq X'_{\mathcal{D}}$. Finally, $X'_{\mathcal{D}_1} \cup X'_{\mathcal{D}_2} \equiv X'_{\mathcal{D}}$. As a counterexample, an object $t$ that exists in $X'_{\mathcal{D}}$, but not in $X'_{\mathcal{D}_1}$ or $X'_{\mathcal{D}_2}$, cannot exist because $T_{\mathcal{D}_1} \cup T_{\mathcal{D}_2} = T$ and $T_{\mathcal{D}_1} \cap T_{\mathcal{D}_2} = \emptyset$ and $X_{\mathcal{D}} = X_{\mathcal{D}_1} = X_{\mathcal{D}_2}$. If $t$ is in $X'_{\mathcal{D}}$ it must appear in $X'_{\mathcal{D}_1}$ or $X'_{\mathcal{D}_2}$.

In short, Property 2 allows us to process all objects independently: the objects can be distributed and processed in an arbitrary order and this will

For an itemset X={b,d} in Table 2.9, we have that $X'_{\mathcal{D}_1} = \{1, 3\}$ and $X'_{\mathcal{D}_2} = \{4\}$. We consider $X'$ in terms of the data in Table 2.3 $X'_{\mathcal{D}} = \{1, 3, 4\}$. Thus $X'_{\mathcal{D}} = X'_{\mathcal{D}_1} \cup X'_{\mathcal{D}_2}$.

**Example 4:** An example for Property 2

not affect the result of $X'$. Property 2 is trivially extended to the case of $n$ partitions. Now we describe how formal concepts can be combined from different partitions.

**Property 3** *Given the formal context, $(T, P, R)$, the two partitions $(T_{\mathcal{D}_1}, P, R_{\mathcal{D}_1})$ and $(T_{\mathcal{D}_2}, P, R_{\mathcal{D}_2})$ and an arbitrary itemset, $X \subseteq P$, the property $X''_{\mathcal{D}} = X''_{\mathcal{D}_1} \cap X''_{\mathcal{D}_2}$ holds: The intersection of the closures of the attribute set, $X$, with respect to each of the partitions $S_1$ and $S_2$ is equivalent to the closure of the attribute set, $X$, with respect to the entire dataset $\mathcal{D}$.*

**Proof.** By the definition of the partition construction method above, $\mathcal{D}_1 \cup \mathcal{D}_2 = \mathcal{D}$, which implies that, $\mathcal{D}_1 \subset \mathcal{D}$ and $\mathcal{D}_2 \subset \mathcal{D}$. Recall that, $X'_{\mathcal{D}_1} \subset X'_{\mathcal{D}}$ and $X'_{\mathcal{D}_2} \subset X'_{\mathcal{D}}$, and from Property 2 we have that $X'_S = X'_{\mathcal{D}_1} \cup X'_{\mathcal{D}_2}$. Appealing to the properties of the derivation operators, in [8], we have, $X''_{\mathcal{D}_1} \supseteq X''_{\mathcal{D}}$ and $X''_{\mathcal{D}_2} \supseteq X''_{\mathcal{D}}$. It is clear that $X''_{\mathcal{D}_1}$ and $X''_{\mathcal{D}_2}$ need not equal $X''_{\mathcal{D}}$, but by the definition of a closure $(X'_{\mathcal{D}_1} \cup X'_{\mathcal{D}_2})' = (X'_{\mathcal{D}})' = X_{\mathcal{D}}$, thus, $(X'_{\mathcal{D}_1} \cup X'_{\mathcal{D}_2})' = X''_{\mathcal{D}_1} \cap Y''_{\mathcal{D}_2}$ follows trivially from the definition of the derivations operators.

Taking itemset $X$ ={b,d} as an example. We have $X''_{\mathcal{D}_1}$ ={b,d,f} in terms of $\mathcal{D}_1$, $X''_{\mathcal{D}_2}$ ={b,d,e} in terms of $\mathcal{D}_2$, and $X''_{\mathcal{D}}$ ={b,d} in terms of $\mathcal{D}$. Therefore $X''_{\mathcal{D}} = X''_{\mathcal{D}_1} \cap X''_{\mathcal{D}_2}$.

**Example 5:** An example for Property 3

For a distributed dataset having two partitions, we propose Theorem 2 and give proof following it.

**Theorem 2** *Given a set of attributes $X$, $X \subset P$. Let $\mathcal{F}^X_{\mathcal{D}_1}$ and $\mathcal{F}^X_{\mathcal{D}_2}$ be the sets of closure based on $X$ in relation to $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively. Then the closure of $X$ in relation to $\mathcal{D}$ can be calculated from: $\mathcal{F}^X_{\mathcal{D}} = \mathcal{F}^X_{\mathcal{D}_1} \cap \mathcal{F}^X_{\mathcal{D}_2}$.*

**Proof.** This is simply a consequence of Property 3, $\mathcal{F}^X_{\mathcal{D}} = X''_{\mathcal{D}} = X''_{\mathcal{D}_1} \cap X''_{\mathcal{D}_2} = \mathcal{F}^X_{\mathcal{D}_1} \cap \mathcal{F}^X_{\mathcal{D}_2}$ and $X_{\mathcal{D}} \equiv X_{\mathcal{D}_1} \equiv X_{\mathcal{D}_2}$ by definition of the partition.

We are going to extend Theorem 2.

For an itemset X={b,d} from Table 2.9, we calculate its closure in terms of $\mathcal{D}_1$ so that $\mathcal{F}^X_{\mathcal{D}_1}$ = {b,d}$''_{\mathcal{D}_1}$ = {b,d,f}, and then its closure in terms of $\mathcal{D}_2$ is $\mathcal{F}^X_{\mathcal{D}_2}$ ={b,d}$''_{\mathcal{D}_2}$ = {b,d,e}. According to Theorem 2 we have $\mathcal{F}^X_{\mathcal{D}} = \mathcal{F}^X_{\mathcal{D}_1} \cap \mathcal{F}^X_{\mathcal{D}_2}$ = {b,d,f} ∩ {b,d,e}={b,d}.

**Example 6:** An example for Theorem 2

**Theorem 3** *Given the closures $\mathcal{F}^X_{\mathcal{D}_1}, \cdots, \mathcal{F}^X_{\mathcal{D}_n}$ from n disjoint data partitions, we have that: $\mathcal{F}^X_{\mathcal{D}} = \mathcal{F}^X_{\mathcal{D}_1} \cap \ldots \cap \mathcal{F}^X_{\mathcal{D}_n}$.*

**Proof.** A trivial inductive argument establishes that Theorem 3 is true. Theorem 2 provides the $n = 2$ case. A sketch of the proof of Theorem 3 follows by recognizing that the dataset $\mathcal{D}$ at the $(i-1)$-th step of the proof can be though as of consisting of two partitions only, the partition Supposing that Theorem 3 holds for the case of $(n-1)$, we would like to show that it holds for the case of $n$ as well. By hypothesis we know that the closures in the first $(n-1)$ partitions $\mathcal{D}_1 \cup \cdots \cup \mathcal{D}_{n-1}$ and the second partition $\mathcal{D}_n$. Recall, Theorem 2 proves the $n = 2$ case.

Calling on nothing more complex than the properties of the derivation operators, and by constructing the partitions in a non-overlapping manner we can now leverage Theorem 3 in order to apply the MapReduce framework, specifically the Twister variant, to calculate closures from arbitrary number of distributed nodes sure knowledge that the thoroughness of NextClosure is preserved.

### 3.3.2 Merging Model for Distributed Frequent Closed Itemset Mining

The difference between FCA and FCI is that FCI mining algorithms, such as CLOSET+, are harder to decompose into map and reduce functions, because they run recursively. An alternative approach is to treat the whole algorithm as a map. Each mapper deals with a partition of the dataset and produces local FCIs as intermediate results. The reduce function contains the merging function which is able to process local FCIs and to produce global ones by merging them together, as depicted in Figure 3.2, Which is a merging model for distributed FCI mining we propose in this work.

As introduced in Definition 3, the ⊞ operation applied on two local frequent closed itemsets results in information loss for some frequent closed itemsets. That is to say, a global FCI, $I$, will not appear in some partitions $\mathcal{D}_i$ if it is infrequent locally. As a consequence, the frequent closed itemsets cannot be mined completely. Researchers have proposed two solutions to address this issue. The first solution is to re-access the data partitions to

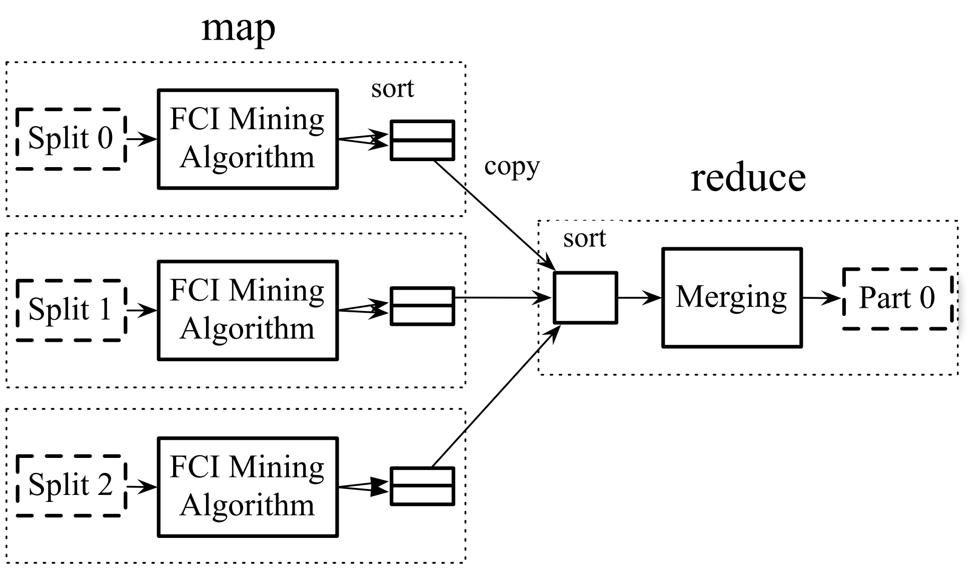

Figure 3.2: Frequent Closed Itemsets merging model based on the MapReduce framework. The map functions process each partition with the same FCI mining algorithm. The merging function combines the outputs of all sites.

acquire the information which is lost. The amount of lost information depends on the data distribution. The more information that is lost, the more computations needed to find the information. This is a potential waste of time and resources. The second solution is to retrieve the lost information for IFCIs [68]. This avoids re-accessing data partitions as long as both the FCIs and IFCIs are mined at a previous step. This thesis adopts the second approach to prevent information loss in distributed environments. The following theorems and properties support this from a mathematical point of view.

Suppose there are $n$ data partitions. For the purpose of this example, we take a dataset and form $n = 3$ partitions, $\mathcal{D}_i$ where $i = 1, 2, 3$, as shown in Table 3.1 The minimal support is defined before head as $\sigma = 40\%$. The minimal support is used to make a distinction between FCIs and IFCIs. In other words, all closed itemsets which have supports below $\sigma$ are IFCIs.

In the following discussion, we denote $\mathcal{C}_i$, $\widetilde{\mathcal{C}_i}$ as the sets of frequent closed itemsets and the set of IFCIs in terms of $\mathcal{D}_i$, and $\mathcal{C}$ as the set of global frequent closed itemsets. The set of local FCIs and the set of local IFCIs corresponding to each data partition, and the set of global FCIs are listed in Table 3.2. The number to the right of ":" is the support count of the current

Table 3.1: Distributed datasets for Frequent Closed Itemset Mining. Each dataset has 3 objects which have several attributes (the numbers).

| $\mathcal{D}_1$ | $\mathcal{D}_2$ | $\mathcal{D}_3$ |
|---|---|---|
| 2,3,5,6,7 | 1,3,4,6 | 2,5,6,7 |
| 1,3,4,7 | 1,3,5,7 | 2,3,5 |
| 1,2 | 3,4,7 | 1,2,3,5,6 |

Table 3.2: Local FCIs and IFCIs mined from datasets in Table 3.1. The entries in the rightmost column are the corresponding global FCIs.

| $\mathcal{C}_1$ | $\mathcal{C}_2$ | $\mathcal{C}_3$ | $\mathcal{C}$ |
|---|---|---|---|
| ({3,7}:2) | ({3,7}:2) | ({2,5,6}:2) | ({6}:4) |
| ({2}:2) | ({3,4}:2) | ({2,3,5}:2) | ({7}:5) |
| ({1}:2) | ({1,3}:2) | ({2,5}:3) | ({3,7}:4) |
| | ({3}:3) | | ({5}:5) |
| $\widetilde{\mathcal{C}}_1$ | $\widetilde{\mathcal{C}}_2$ | $\widetilde{\mathcal{C}}_3$ | ({3,5}:4) |
| ({2,3,5,6,7}:1) | ({1,3,4,6}:1) | ({2,5,6,7}:1) | ({2,5}:4) |
| ({1,3,4,7}:1) | ({1,3,5,7}:1) | ({1,2,3,5,6}:1) | ({2}:5) |
| ({1,2}:1) | ({3,4,7}:1) | | ({1}:5) |
| | | | ({1,3}:4) |
| | | | ({3}:7) |

itemset.

According to Property 1, the following Property 4 can be reached. Many researchers have discussed Property 4, such as [68].

**Property 4** *A global frequent closed itemset must be a local frequent closed itemset or included by the itemsets in* $\mathcal{C}_i, i = \{1, \ldots, n\}$.

**Proof.** Recall Property 1, a global frequent closed itemset $I$ is frequent in one partition at least. Therefore $I$ must be in one of $\mathcal{C}_i, i = \{1, \ldots, n\}$. In the rest of partitions, $I$ either appears as infrequent or never appear in order to meet the global support count. In other words, $I$ appears in the form of a subset of other locally frequent closed itemset.

Property 4 indicates that there are two cases for the existence of a global FCI. We define them as following:

- The first case is that we shall denote a global FCI by $I_\alpha$, which occurs exactly in some sets of local FCIs, $\mathcal{C}_i$. For example, the global FCI ({3}:7) in Table 3.2 is located in 3 of the sets of local FCIs, i.e. $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$. The local FCIs, ({3,7}:2), ({3}:3) and ({2,5,3}:2) could be intersected to form ({3}:7). For this case, we can obtain the global FCI by intersecting all the related $\mathcal{C}_i$ directly. We denote the set of $I_\alpha$-like FCIs by $\mathcal{C}_\alpha$.
- The other case is that a global FCI, which we denote by $I_\beta$, both occurs in some sets of local FCIs and also in some sets of infrequent closed itemsets $\widetilde{\mathcal{C}_i}$. For instance, ({5}:5) is subsumed in ({2,5}:3) in $\mathcal{C}_3$, but appears as infrequent itemsets, ({2,3,5,6,7}:1) and ({1,3,5,7}:1), in $\widetilde{\mathcal{C}_1}$ and $\widetilde{\mathcal{C}_2}$ respectively. To acquire the global FCI in this case, one needs to compute the related $\mathcal{C}_i$, in the meantime, looking for the missing support of $I_\beta$ in $\widetilde{\mathcal{C}_i}$. We denote the set of $I_\beta$-like FCIs by $\mathcal{C}_\beta$.

In the first case, all the global FCIs can be tracked from local frequent closed itemsets in two steps. First, we need to find the targeted sets of local frequent closed itemsets. And then we apply the $\boxplus$ operator on them. The basic operation of $\boxplus$ is composed of the intersection on two itemsets. The operation $\boxplus$ returns the common itemset with the sum of supports. Formally, combining Definition 4 and Property 4 we propose the following theorem:

**Theorem 4** *If the FCIs in a set of global frequent closed itemsets $\mathcal{C}$ exist only in $\mathcal{C}_i, i = 1, \ldots, n$, then $\mathcal{C} \subseteq \overline{\mathcal{C}} = (\cdots((\mathcal{C}_1 \boxplus \mathcal{C}_2) \boxplus \cdots \boxplus \mathcal{C}_n)\cdots)$.*

**Proof.** Continue the discussion in Definition 4, we see that $\overline{\mathcal{C}}$ is a superset of all globally frequent closed itemsets which are also locally frequent. For the case 1, we have $\mathcal{C} = \mathcal{C}_\alpha \subseteq \overline{\mathcal{C}}$, so then $\mathcal{C} \subseteq \overline{\mathcal{C}}$.

Theorem 4 can be used to find all the global FCIs which are only in $\mathcal{C}_\alpha$. However, there is still one remaining problem, that is, the result from Theorem 4 contains also some FCIs which belong to Case 2. This is a consequence of the fact that we are not able to exactly identify the $\mathcal{C}_i$ which contain(s) the corresponding global FCI. For example, when applying Theorem 4 on Table 3.2 we have that $\overline{\mathcal{C}}$ ={({3}:7), ({3,7}:4), ({1}:4), ({2}:5)}. You can see that ({1}:4) is not a global FCI due to the missing support information. This means that the itemset, {1}, appears in a set of IFCIs, $\widetilde{\mathcal{C}_i}$, but not in the corresponding $\mathcal{C}_i$. The other important reason is that $I_\alpha$ and $I_\beta$ appear in common $\mathcal{C}_i$ often so that they are mixed together. Our solution is to classify the global FCIs according to their distributions and then to utilize an elimination procedure. For the example in Table 3.2, the itemset ({7}:5) exists in the sets of FCIs, $\mathcal{C}_1$ and $\mathcal{C}_2$, while ({1,3}:4) occurs only in $\mathcal{C}_2$. One can distinguish between them by the ID of the partitions in which they are involved. To consider all the possibilities, it is needed to compute the combination of arbitrary number of the set of FCIs in terms of the data partition $i$, $\mathcal{C}_i$, out of $n$. Thus, there are $C_n^2 + C_n^3 + \ldots + C_n^n = 2^n - n - 1$ possibilities.

From another perspective, for a distributed dataset with $n$ partitions, there exists $1 + 2 + \ldots + n = n(n+1)/2$ possibilities for the distribution of a global FCI, $I$. $I$ could lie in $1, 2, \ldots, n-1$ or $n$ of the partitions. According to this property, we can classify all global FCIs into $n$ groups and give them names such as $\overline{\mathcal{S}_i}$, $i = 1, \ldots, n$. which refers to the set of FCIs which are only frequent on $i$ out of $n$ data partitions. Specifically, $I$ could appear in either $\mathcal{C}_\alpha$ or $\mathcal{C}_\beta$. In the case of $I \in \mathcal{C}_\alpha$, we can utilize Theorem 4 to find out a superset of $\mathcal{C}_\alpha$.

We propose the following proposition and define some notations for it: $\mathcal{S}_1$ refers to an union of the sets of FCIs, $\mathcal{C}_i$; $\mathcal{S}_i$ denotes the collection of the intersections between elements from $\mathcal{S}_{i-1}$ and from $\mathcal{S}_1$; Function $\Gamma(\mathcal{S}_{i-1}, \mathcal{S}_1)$ applies the $\boxplus$ operation on arbitrary two different FCIs from $\mathcal{S}_{i-1}$ and $\mathcal{S}_1$ separately and combines their result FCI into a set.

**Proposition 1** *Given a distributed dataset which consists of n partitions where the minimal support count is σ, the global frequent closed itemsets $\mathcal{C}_\alpha$, which appears only in $\mathcal{C}_i, i = 1, \ldots, n$, must satisfy the following equations:*

$$\mathcal{S}_1 = \{\mathcal{C}_1, \ldots, \mathcal{C}_n\} \tag{3.1}$$

$$\mathcal{S}_i = \Gamma(\mathcal{S}_{i-1}, \mathcal{S}_1) = \bigcup_{X \in \mathcal{S}_{i-1}, Y \in \mathcal{S}_1}^{X \neq Y} (X \boxplus Y)_\sigma, \textit{where } 1 < i \leqslant n \tag{3.2}$$

$$\mathcal{C}_\alpha \subseteq \bigcup_{i=1}^{n} \mathcal{S}_i \tag{3.3}$$

**Proof.** According to the discussion on $\mathcal{C}_\alpha$ in conjunction with Theorem 4, we know that $\mathcal{C}_\alpha \subseteq \overline{\mathcal{C}}$. As the definition of $\mathcal{S}_i$ indicates, it is clear that $\overline{\mathcal{C}} \subseteq \bigcup_{i=1}^{n} \mathcal{S}_i$. Naturally, (Eqn. 3.3) holds.

In Proposition 1, $\mathcal{S}_1$ is initialized as a collection of $\mathcal{C}_i$, $i = 1, \cdots, n$, as shown in (Eqn. 3.1). All the global FCIs, which are included by one of the $\mathcal{C}_i$, $i = 1, \cdots, n$, will be grouped to form a set $\mathcal{S}_n$. (Eqn. 3.2) can produce

$\mathcal{S}_i$ by intersecting $\mathcal{S}_1$ with $\mathcal{S}_{i-1}$ which is produced in last iteration, and the results are filtered with minimal support threshold. Note that, the itemsets from $\mathcal{S}_i$ only intersect with unequal itemsets in different data partition could not with itemsets in its own partition. Finally, (Eqn. 3.3) forms a union of the sets $\mathcal{S}_1$ through to $\mathcal{S}_n$.

In fact, Proposition 1 traverses all of the global FCIs with a forced traverse, and $\mathcal{S}_i$ contains some FCIs which belong to $\mathcal{C}_\beta$. So it is necessary to detach, $\mathcal{C}_\alpha$, from the union of $\bigcup_{i=1}^{n} \mathcal{S}_i$ in order to find out the FCIs which belong to $\mathcal{C}_\beta$. Consider $\mathcal{S}_{n-1}$ which comes from $(n-1)$ of all $\mathcal{C}_i$, $i = 1, \cdots, n$. In other words, there is only one $\mathcal{C}_i$ (suppose $i = x$) does not contribute to $\mathcal{S}_{n-1}$. To extract the FCIs that might be partly in $\widetilde{\mathcal{C}}_x$, we need to remove $\mathcal{S}_n$ from $\mathcal{S}_{n-1}$ because $\mathcal{S}_n$ is a subset of $\mathcal{S}_{n-1}$, and then intersect $\mathcal{S}_{n-1} - \mathcal{S}_n$ with $\widetilde{\mathcal{C}}_x$. Before going into deep, we define two operators $\biguplus$ and $\otimes$ in Definition 5.

**Definition 5** *Let us define a set of FCIs* $\{\mathcal{C}_i, \mathcal{C}_j, \mathcal{C}_k\}$ *and a set of IFCIs* $\{\widetilde{\mathcal{C}}_i, \widetilde{\mathcal{C}}_j, \widetilde{\mathcal{C}}_k\}$*, whose elements are from the data partitions* $i, j, k$ *respectively, then:*

$$\mathcal{C}_i \otimes \widetilde{\mathcal{C}}_j = \mathcal{C}_i \cup \{X \cap Y | (X, Y) \in (\mathcal{C}_i \times \widetilde{\mathcal{C}}_j)\}.$$

$$\{\mathcal{C}_i, \mathcal{C}_j, \mathcal{C}_k\} \biguplus \{\widetilde{\mathcal{C}}_i, \widetilde{\mathcal{C}}_j, \widetilde{\mathcal{C}}_k\} = (\mathcal{C}_i \otimes \widetilde{\mathcal{C}}_j \otimes \widetilde{\mathcal{C}}_k) \cup (\mathcal{C}_j \otimes \widetilde{\mathcal{C}}_i \otimes \widetilde{\mathcal{C}}_k) \cup (\mathcal{C}_k \otimes \widetilde{\mathcal{C}}_i \otimes \widetilde{\mathcal{C}}_j)$$

The symbol $\otimes$ first makes an intersection between the set of FCIs and the set of IFCIs, after that the results are united with the first set and finally returned as output[1].

The operator $\biguplus$ is introduced to compute the global FCIs by intersecting the set of FCIs in the $i$-th partition, $\mathcal{C}_i$, with the set of IFCIs in the $j$-th partition. The lost support information of FCIs in $\mathcal{C}_i$ can be re-acquired from related $\widetilde{\mathcal{C}}_i$. Formally, we define $\overline{\mathcal{S}_i}$, a set of frequent closed itemsets which appear in $i$ out of $n$ partitions and have the following principle. Note that, $\overline{\mathcal{S}_i}$ is a super set of $\mathcal{S}_i$.

**Proposition 2** *Given a distributed dataset which consists of* $n$ *partitions. With the minimal support count* $\sigma$*, the global frequent closed itemsets* $\mathcal{C}$ *can be generated by the following steps:*

[1]Note that, the $\oplus$ symbol which was used in NextClosure algorithm is different from the operator $\otimes$ here.

$$\mathcal{S}_n = \mathcal{C}_1 \cap \ldots \cap \mathcal{C}_n \tag{3.4}$$

$$\begin{aligned} \overline{\mathcal{S}_i} &= \Upsilon(\mathcal{S}_i, \mathcal{S}_n, \overline{\mathcal{S}_{(i+1)\cap n}}, \widetilde{\mathcal{C}}_{1\cup n}) \\ &= \{\mathcal{S}_i - (\mathcal{S}_n + \overline{\mathcal{S}_{i+1}} + \ldots + \overline{\mathcal{S}_{n-1}})\} \biguplus \{\widetilde{\mathcal{C}}_1, \ldots, \widetilde{\mathcal{C}}_n\}_\sigma \\ & \textit{where } 1 \leqslant i < n \end{aligned} \tag{3.5}$$

$$\mathcal{C} = \mathcal{S}_n + \bigcup_{i=1}^{n-1} \overline{\mathcal{S}_i} \tag{3.6}$$

**Proof.** Proposition 2 can be easily deduced from Proposition 1 which was discussed above. $\mathcal{S}_n$ is a special set among $\mathcal{S}_i, i = 1, \ldots, n$, its essence is the intersection of all $\mathcal{C}_i$ but does not contain any FCI belonging to $\mathcal{C}_\beta$. Hence, (Eqn. 3.4) has the same sense with (Eqn. 3.2) when $i = n$. In (Eqn. 3.5), the FCIs found in $\mathcal{S}_{i+1}, \ldots, \mathcal{S}_n$ need to be pruned away from $\mathcal{S}_i$. Subsequently, the reduced $\mathcal{S}_i$ will intersect with some $\widetilde{\mathcal{C}_i}$ to obtain $\overline{\mathcal{S}_i}$. Note that, the computation of $\overline{\mathcal{S}_i}$ begins from the $i = n - 1$ and ends at $i = 2$. The aggregation of $\overline{\mathcal{S}_i}$ and $\mathcal{S}_n$ is exactly complete set of global FCIs because $\overline{\mathcal{S}_i}$ and $\mathcal{S}_n$ cover all scenarios where frequent closed itemsets could be.

The contribution made here is that Proposition 2 does something different from the theorem by Chun Liu in [68]. For example, Proposition 2 works for general case, while the Liu's approach turns out to be valid for a special case, i.e., the distributed dataset has few partitions. A real example is shown in Example 7.

In the remaining work, we apply these properties, theories, and propositions to common FCI mining algorithms and design an algorithms based on the MapReduce framework. To mine FCIs in parallel, the dataset is divided into many disjoint parts, and each part is processed by a frequent closed itemsets mining algorithm. Actually, it is possible that different data parts can be processed with different FCIs mining algorithms. For simplicity, this thesis is going to adopt the same FCI mining algorithm (CLOSET+) for each data part.

Notice that at this stage, all the closed itemsets produced are local frequent only because they are from a part of the whole dataset. Based on Proposition 1 and 2 a merging function can be used to obtain global frequent ones. There are several steps need to do in the merging function. They will work in sequence to get global FCIs from local ones:

1 generate $\mathcal{S}_i$ using $\mathcal{C}_i$ produced by each data partition according to Proposition 1.

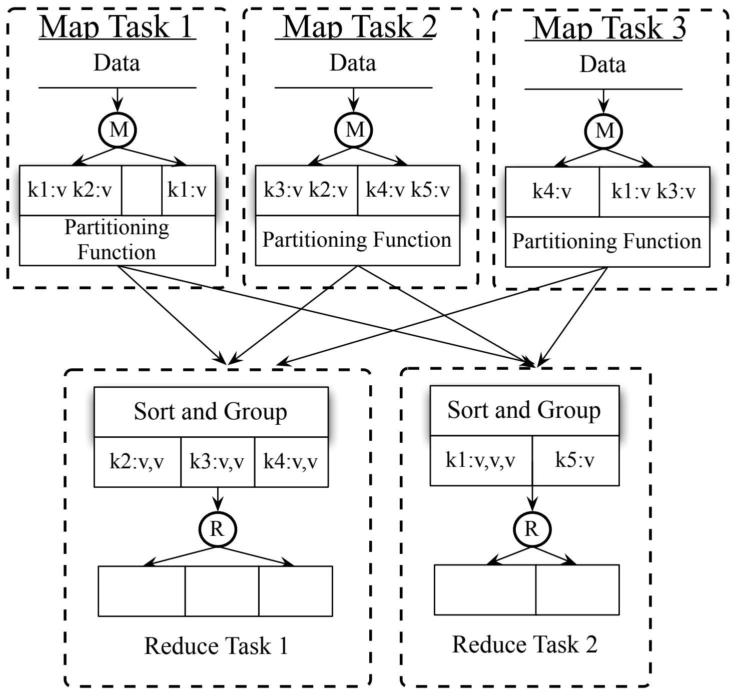

Figure 3.3: Parallel execution of MapReduce: the map function packages the outputs in the form of (key,value) pairs. The (key,value) pairs are sorted and grouped by the key in the reduce phase and are then processed by the reduce functions.

2 filter $\mathcal{S}_i$ with global FCIs which have already been found and combine ($\biguplus$) with $\mathcal{C}_i$ in order to generate $\overline{\mathcal{S}_i}$, according to (Eqn. 3.5) in Proposition 2.

3 combine the set of $\overline{\mathcal{S}_i}$ with $\mathcal{S}_n$ to form the final result, $\mathcal{C}$.

## 3.4 Algorithm Adaptation

Irrespective of whether or not distributed FCA or distributed frequent closed itemset mining is considered, they need to be adapted to fit the MapReduce framework. As introduced in Section 2.4.2, MapReduce generally consists of two functions, map() and reduce(). The first important thing we need to ensure is that the algorithm we are going to use is not a recursive one. The benefit of MapReduce model lies in executing algorithms in parallel, and it cannot call itself, as depicted in Figure 3.3.

To begin we set local computation to be the map phase. For example, for

FCA, computing closure(s) for each data partition is the local computation of map phase, whereas for distributed frequent closed itemset mining, the map phase should consist of the common FCIs mining algorithm, for instance Closet+. MapReduce requires that the inputs and the intermediate data produced by map() should be in the form of (key, value). In practice, the output of the map phase is set as $(keyElement, newLocalValue)$, where

- *newLocalValue* is a new FCI produced by map();
- *keyElement* is the corresponding attribute which contributes to the generation of *newLocalValue*.

In this example, the key, *keyElement*, will be required in the reduce phase. In general, however, the use of the key is optional and determined by the application's demand. The set is to constant map and reduce functions with subroutines which are based on the original centralized algorithm that is being adapted.

Now, we can summarize the general procedure for filling algorithms to the MapReduce framework.

1. Ensure that the target algorithm is non-recursive and can be decomposed. Note that in some special cases, even recursive algorithms can be converted into non-recursive one [20], such as the CloseByOne algorithm. Otherwise, we have to consider using the general model as discussed in Section 3.3.2.
2. Decompose the algorithm into two functions to satisfy the requirements of the MapReduce framework;
3. Design the (key,value) pairs for the map and reduce functions;
4. Implement the map and reduce functions with functions derived from the original algorithm.

In this thesis, we are going to implement the distributed versions of NextClosure, CloseByOne, and a merging model for distributed FCI mining by following this procedure. The concrete adaptation steps are shown in Section 4.

## 3.5 Summary

This chapter started by listing the domain related issues. It then proposed a methodology for this research. In Section 3.3, the common closed itemset mining algorithms were grouped into two categories, and, for each case, the corresponding distributed theories were developed. By analysing the MapReduce framework, general adaptation rules were proposed in Section 3.4.

Let us mine the data in Table 3.1 according to Proposition 1 and 2. First apply Proposition 1 on $\mathcal{C}_i$ in Table 3.2, we have:

$$\begin{aligned}\mathcal{S}_2 =&\{\mathcal{C}_{12} : \{\{3,7\} : 4, \{3\} : 5, \{1\} : 4\}, \mathcal{C}_{13} : \{\{2\} : 5\}, \mathcal{C}_{23} : \{\{3\} : 5\}\}\\ \mathcal{S}_3 =&\{\mathcal{C}_{123} : \{\{3\} : 7\}\}\end{aligned}$$

Then with the support of Proposition 2 plus $\widetilde{\mathcal{C}_i}$ in Table 3.2, we calculate $\overline{\mathcal{S}_2}$ and $\overline{\mathcal{S}_1}$ in turn. Note that, the operator $\otimes$ is always applied on $\mathcal{C}_i$ and $\widetilde{\mathcal{C}_j}$ which come from different and non-overlap partitions. $\overline{\mathcal{S}_2}$ contains only FCIs appeared on 2 data partitions.

$$\begin{aligned}\overline{\mathcal{S}_2} =&(\mathcal{S}_2 - \mathcal{S}_3) \biguplus (\widetilde{\mathcal{C}_1}, \widetilde{\mathcal{C}_2}, \widetilde{\mathcal{C}_3})\\ =&\{\mathcal{C}_{12} : \{\{3,7\} : 4, \{1\} : 4\}, \mathcal{C}_{13} : \{\{2\} : 5\}\} \biguplus (\widetilde{\mathcal{C}_2}, \widetilde{\mathcal{C}_3})\\ =&(\mathcal{C}_{12} \otimes \widetilde{\mathcal{C}_3}) \bigcup (\mathcal{C}_{13} \otimes \widetilde{\mathcal{C}_2})\\ =&\{\{\{3,7\} : 4, \{1\} : 4\} \otimes \{\{2,5,6,7\} : 1, \{1,2,3,5,6\} : 1\}\} \bigcup\\ &\{\{\{2\} : 5\} \otimes \{\{1,3,4,6\} : 1, \{1,3,5,7\} : 1, \{3,4,7\} : 1\}\}\\ =&\{\{3,7\} : 4, \{2\} : 5, \{7\} : 5, \{3\} : 5, \{1\} : 5\}\end{aligned}$$

$\overline{\mathcal{S}_1}$ includes only FCIs appeared on 1 data partition.

$$\begin{aligned}\overline{\mathcal{S}_1} =&(\mathcal{S}_1 - \mathcal{S}_3 - \overline{\mathcal{S}_2}) \biguplus (\widetilde{\mathcal{C}_1}, \widetilde{\mathcal{C}_2}, \widetilde{\mathcal{C}_3})\\ =&\{\mathcal{C}_1 : \{\{3,7\} : 2, \{2\} : 2, \{1\} : 2\}, \mathcal{C}_2 : \{\{3,7\} : 2, \{3,4\} : 2,\\ &\{1,3\} : 2, \{3\} : 3\}, \mathcal{C}_3 : \{\{2,5,6\} : 2, \{2,3,5\} : 2, \{2,5\} : 3\} - \{\{3\} : 7\} -\\ &\{\{7\} : 5, \{3\} : 5, \{1\} : 5\}\} \biguplus (\widetilde{\mathcal{C}_1}, \widetilde{\mathcal{C}_2}, \widetilde{\mathcal{C}_3})\\ =&\{\mathcal{C}_1 : \{\{3,7\} : 2, \{2\} : 2\}, \mathcal{C}_2 : \{\{3,7\} : 2, \{3,4\} : 2, \{1,3\} : 2\},\\ &\mathcal{C}_3 : \{\{2,5,6\} : 2, \{2,3,5\} : 2, \{2,5\} : 3\}\} \biguplus (\widetilde{\mathcal{C}_1}, \widetilde{\mathcal{C}_2}, \widetilde{\mathcal{C}_3})\\ =&(\{\{3,7\} : 2, \{2\} : 2\} \otimes \widetilde{\mathcal{C}_2} \otimes \widetilde{\mathcal{C}_3}) \bigcup (\{\{3,7\} : 2, \{3,4\} : 2, \{1,3\} : 2\}\\ &\otimes \widetilde{\mathcal{C}_1} \otimes \widetilde{\mathcal{C}_3}) \bigcup (\{\{2,5,6\} : 2, \{2,3,5\} : 2, \{2,5\} : 3\} \otimes \widetilde{\mathcal{C}_1} \otimes \widetilde{\mathcal{C}_2})\\ =&\{\{3\} : 4, \{5\} : 5, \{6\} : 4, \{1,3\} : 4, \{2,5\} : 4\}\end{aligned}$$

Finally, all scenarios should be covered to collect all FCIs.

$$\begin{aligned}\mathcal{C} =&\mathcal{S}_3 + \bigcup_{i=1}^{2} \overline{\mathcal{S}_i}\\ =&\{\{3\} : 7, \{1\} : 5, \{2\} : 5, \{5\} : 5, \{7\} : 5, \{6\} : 4, \{1,3\} : 4,\\ &\{2,5\} : 4, \{3,7\} : 4\}\end{aligned}$$

**Example 7:** Calculating global FCIs with Proposition 1 and 2 support.

Chapter 4

# Implementation and Validation

In Chapter 3, we discussed that candidate algorithms need to be decomposed into map and reduce functions in order to run them using the MapReduce framework. In this chapter, we give examples of classic closed itemset mining algorithms and show how to make them distributed. The centralized algorithms in this thesis are renamed using a prefix “MR”, specifically, we rename the algorithms and call them, MRGanter, MRGanter+, MRCbo and a general merging model for frequent closed itemset mining algorithms based on the MapReduce framework (MRGeneral).

## 4.1 Distributed Formal Concept Mining Algorithms

### 4.1.1 MRGanter

Reviewing Algorithm 2, it is clear that there are two stages involved in: 1) computing new candidate closure, and, 2)making a judgment on whether to keep it or not. Computing new closure corresponds to the map stage, and making a judgment on whether or not to keep the closure corresponds to the reduce phase.

Given that distributed datasets need to be computed independently, each mapper produces local closures because its input is only a part of the entire dataset. Accordingly, a merging operation is required to combine $n$ local closures computed by the mappers. Since the complete set of closures is produced in the collaboration by all of the mappers, the merging operation needs to be deployed on every reducer. The reducers are also responsible for making a judgment on whether or not a closure should be kept. Figure 4.1 describes the data flow for generating a new closure.

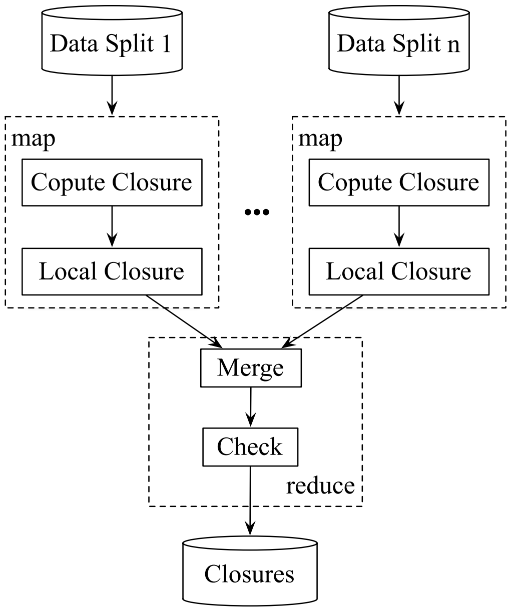

Figure 4.1: MRGanter iteration: MRGanter computes local FCIs in the map phase, generates global FCIs in the reduce phase and filters the global FCIs using the check function.

For the purpose of discussion, in the following distributed FCA algorithms, we only calculate the intent of a formal concept because the extent can be calculated using intent. Some variables and constants for these algorithms are shown in Table 4.1

The main operation in the merging function is the intersection operator, ∩, which is applied on the set of local closures L_i generated at each node. Algorithm 4 gives the pseudo code for the merging function based on Theorem 3. The merging function is deployed on the reduce phase and only processes the local closures derived from the same attribute (line 1).

**Algorithm 4** The merging Function for Distributed FCA Algorithms

```
Input: p_i, L_i, f.
Output: f.
1: l_i ← the local closure in L_i in terms of p_i
2: f ← $\Psi(l\_i, f)$;
3: return f
```

The MapTask described in Algorithm 5 produces all local closures. The output consists of the previous intent d and a set of local intents L_i. Note

Table 4.1: Table of variables and constants for distributed FCA algorithms.

| Variables/Constants | stands for |
|---|---|
| P | the set of attributes in a given dataset |
| p_i | an attribute in P, where $i = 1, \cdots, m$ |
| L_i | the set of local intents in data partition $i$ |
| l_i | an intent in L_i which is derived from p_i |
| d | the intent produced in previous iteration |
| f | the newly generated intent |
| G | an container in which storing the newly generated intents |
| partitionFile | a file contains full path for locating the datasets |
| numMap | the number of mappers |
| numReduce | the number of reducers |
| R | an container where storing all found concepts |
| toMap | an container for holding the local concepts and the attributes used to form the concepts |

**Algorithm 5** MapTask in MRGanter

**Input:** d.

**Output:** (d, L_i).

```
1: for p_i from p_m down to p_1 do
2:    if p_i is not in d then
3:       l_i $\leftarrow$ d $\oplus$ p_i;
4:       associate l_i with p_i;
5:       L_i $\leftarrow$ L_i $\cup$ l_i;
6:    end if
7:    return (d, L_i);
8: end for
```

that, in order to be used in the merging function the attribute which was used to form local closures should be recorded and passed, as Line 4 does. All pairs which have the same key, d, will be sent to the same reducer. After this, all the local intents are used to form global intents and then filtered by the closure validation condition, as shown by Line 7 in Algorithm 6.

**Algorithm 6** ReduceTask in MRGanter

```
Input: (d,L_i).
Output: f.
 1: for p_i in P do
 2:     f ← initialize new intent;
 3:     for $i$ from 1 up to $m$ do
 4:         f ← merging(p_i, L_i, f);
 5:     end for
 6:     set ← $\{0, \text{p\_1}, \cdots, \text{p\_i}\}$;
 7:     if f $\cap$ set is equal to d $\cap$ set then
 8:         break;
 9:     else
10:         continue;
11:     end if
12: end for
13: return f
```

The Algorithm 6 accepts (d,L_i) from the i-th *mapper*s (see Section 2.4.3), where $i = 1, \cdots, n$. Recall that, for a reducer, only the pairs who have the same key, d, are accepted. The line 4 generates an candidate closure f. After that, line 7 verifies the candidate. Finally, the successful candidate will be outputted as global closure f.

Figure 4.2 shows how MRGanter works iteratively. In Figure 4.2, the lines marked with “S” import static data from each partition, while the lines marked with “D” configure each map with the previous closure. The newly generated closure is tested whether it is the last one which has the most attributes. If it is not, the closure will be kept and MRGanter continues. Otherwise while loop comes to an end and the algorithm returns all the closures.

### 4.1.2 MRGanter+

In order to reduce redundancy, NextClosure calculates closures in lectic ordering to ensure every concept appears only once. This approach allows a single concept to be tested with the closure validation condition during each

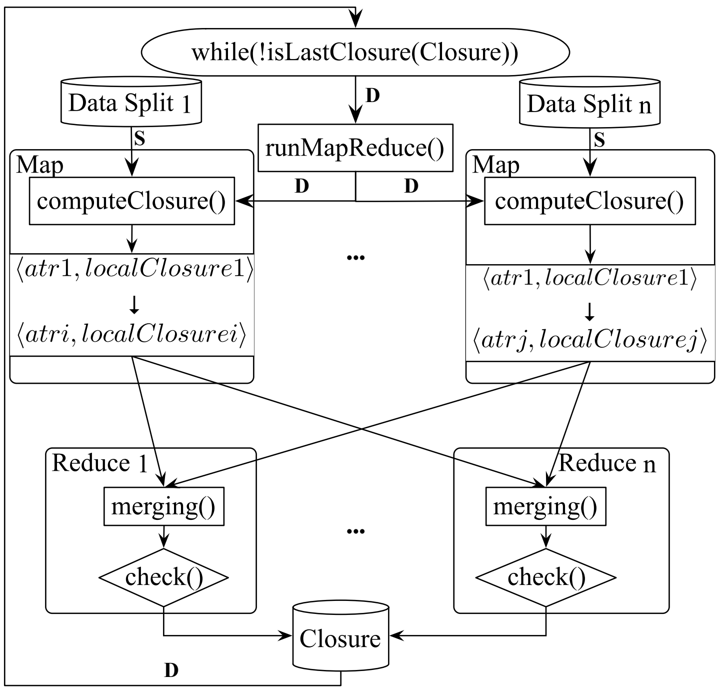

Figure 4.2: MRGanter work flow: static data is loaded at the start of the procedure (labeled by S) and the dynamic data (FCIs produce during each iteration) is passed and used in the next iteration (labeled by D).

iteration. This is efficient when the algorithm runs on a single machine. For multi-machine computation, the extra computation and redundancy resulting from keeping only one concept after each iteration across many machines. Hence, we modify NextClosure to reduce the number of iterations and name the corresponding distributed version, MRGanter+.

Rather than using redundancy checking, we keep as many closures as possible in each iteration. In other words, all closures appearing for the first time are maintained and used to generate the next batch of new closures. To do so, we just need to change strategy used in Algorithm 6. The MapTask remains the same as in Algorithm 5. Algorithm 7 indicates how the ReduceTask in MRGanter+ works.

**Algorithm 7** ReduceTask in MRGanter+

```
Input: (d, L_i).
Output: G.
 1: H ← initialize a two-level hash table;
 2: for $p_i$ in $P$ do
 3:     f ← initialize new intent;
 4:     for $i$ from 1 up to $m$ do
 5:         f ← merging(p_i, L_i, f);
 6:     end for
 7:     if f is not in H then
 8:         add f into H;
 9:         add f into G;
10:     end if
11: end for
12: return G
```

The ReduceTask in MRGanter+ merges local closures first in Line 5, and then recursively examines if they already exist in the set of global formal concepts, H (Line 7). The set, H, is used to index and search for a specified closure. It is designed as a two-level hash table to reduce searching costs. The first level is indexed by the head attribute of the closure, while the second level is indexed by the length of the closure. The new closures are stored in the container G.

Algorithm 8 is the core of the main class, that manages the calls of the MapTask and the ReduceTask. In Algorithm 8, inputs consist of partitionFile, numMap and numReduce. Using partitionFile, mappers are able to locate the corresponding datasets in the local nodes and load them for future use. It is possible to assign the number of mappers and reducers using the variables, numMap and numReduce. Line 1 initializes the first concept with

**Algorithm 8** MRGanter+

```
Input: partitionFile, numMap, numReduce.
Output: R.
 1: toMap ← initialize with ∅;
 2: toMap ← set current attribute;
 3: condition ← true;
 4: while (condition is true) do
 5:     monitor ← driver calls runMapReduceBCast(toMap);
 6:     monitor calls monitorTillCompletion();
 7:     combiner ← driver calls getCurrentCombiner();
 8:     concepts ← combiner calls getConcepts();
 9:     if concepts is not empty then
10:         add concepts to R;
11:         toMap ← concepts;
12:     else
13:         condition ← false;
14:         return R;
15:     end if
16: end while
```

an empty attribute set. Lines 5, 6 and 7 call the methods provided by the Twister runtime. The driver needs to be configured before use. The runMapReduceBCast is responsible for broadcasting concepts found in the last iteration to all mappers. The combiner can be used to further process the outputs from all reducers and pass the results back to main class. Once no new concept can be found (Line 9), the condition will be set as *false* (Line 13) and algorithm exits.

Comparing to the distributed version of CloseByOne proposed by Krajca Petr in [20], MRGanter+ generates more concepts in each iteration so that it needs fewer iterations. A more detailing analysis can be found in Section 5. We implement CloseByOne based on the MapReduce framework and give a name, MRCbo. The performance comparison between MRGanter+ and MRCbo are made in Section 4.3.2.

## 4.2 Distributed Frequent Closed Itemset Mining Algorithm: MRGeneral

With the support of Theorem 1, 2 and Twister MapReduce, we can now design a general model to mine FCIs from multiple data sources. Once we

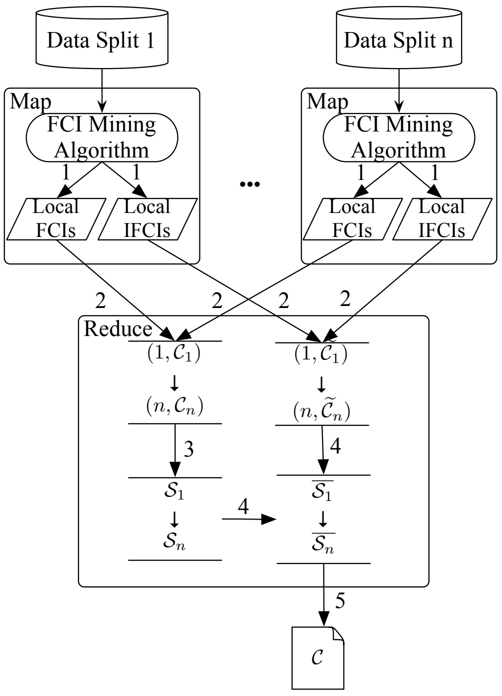

Figure 4.3: Frequent Closed Itemsets merging model, MRGeneral, which mines local FCIs ($\mathcal{C}_i$) and IFCIs ($\widetilde{\mathcal{C}_i}$) using the same FCI mining algorithm. In the reduce phase, the local FCIs, $\mathcal{C}_i$, generate a super-set of global FCIs, $\mathcal{S}_i$, (labeled by 3). The IFCIs $\widetilde{\mathcal{C}_i}$ are used to extract global FCIs $\overline{\mathcal{S}_i}$ from $\mathcal{S}_i$.

get the local FCIs using a traditional algorithm, the remaining work lies in the merging phase. Figure 4.3 shows how MRGeneral works.

In Figure 4.3, the dataset which is waiting to be processed is partitioned into $n$ small parts which are distributed among of the different nodes. The FCI mining algorithm in each node is the core component of the mappers and handles the corresponding data partition in each local disk. In this experiment, we use Closet+. In the map phase (see the lines marked by 1), Closet+ processes local data partitions and outputs local FCIs and IFCIs (lines labeled by 2). Each mapper outputs local FCIs and IFCIs with the corresponding number of partitions to the reducer. This is required by Twister model, and these numbers are needed in the reduce phase. By receiving local FCIs from every mapper, a reducer is able to combine them together and to produce an incomplete set of frequent closed itemsets, as indicated by the lines labelled by 3. In order to determine other hidden FCIs and remove the false ones, one has to further compute $\overline{\mathcal{S}_i}$ in the reducer. The set of IFCIs, $\widetilde{\mathcal{C}_i}$, is needed also to achieve this purpose. This procedure is indicated by

Table 4.2: Table of variables for distributed FCI mining algorithms.

| Variables/Constants | stands for |
|---|---|
| count | a counter for the number of data partitions, and, count $< n$ |
| S_current | a set of FCIs which are only frequent on count-1 data partitions |
| set | one set of $\mathcal{C}_i$ in S_current |
| S_tmp | storing $\mathcal{S}_{j-1}$ temporarily |

lines 4. In this procedure, every infrequent closed itemset in each node can contribute lost itemsets and support information. In last step we take an union of the sets of $\overline{\mathcal{S}_i}$ and obtain a set of all FCIs, $\mathcal{C}$, as shown by Line 5. In fact, there is a combiner in Twister, and the combiner can be used to further merge the outputs from the reducers, but this is not necessary in this example because we already obtain the global frequent closed itemsets in reduce phase. The combiner only takes charge of passing the final result to the main program.

As did for MRGanter and MRGanter+, we give pseudo for MRGeneral algorithm. See Table 4.2 for the variables.

In this example, there is only one iteration. That is to say, Twister is used like Hadoop MapReduce. We chose Twister because it is more light-weight. Note that, only one reducer is used in this work.

## 4.3 Evaluation

We provide evidence of the effectiveness and scalability of our algorithms in this section. Section 4.3.1 describes the experimental environment and the dataset characteristics for the datasets used to validate performance in this work. In Section 4.3.2, we present the experimental results and give analysis.

### 4.3.1 Test Environment and Datasets

In this work, all of the simulation work is done in Java. For the FCA algorithms, we simulate NextClosure and its distributed versions MRGanter and MRGanter+, and, CloseByOne and the corresponding distributed al-

**Algorithm 9** MRGeneral: Merging Model for Distributed FCI Mining Algorithms

```
Input:
    $\mathcal{C}_i$: the sets of local FCIs from each node, where $i = 1, \ldots, n$;
    $\widetilde{\mathcal{C}_i}$: the sets of local IFCIs from each node, where $i = 1, \ldots, n$;
Output:
    $\mathcal{C}$: A set of all global FCIs.
 1: Create $\mathcal{S}_1 \ldots \mathcal{S}_n$;
 2: Initialize $\mathcal{S}_1$ with $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$;
 3: counter $\leftarrow$ 1;
 4: while count $< n$ do
 5:     S_current $\leftarrow \mathcal{S}_{count-1}$;
 6:     for set in S_current do
 7:         initialize indexes with the identifiers of $C_i$;
 8:         for $i$ from 1 upto $n$ do
 9:             if indexes do not contain $i$ then
10:                 newSet $\leftarrow \Gamma(\text{set}, \mathcal{C}_i)$;
11:                 $\mathcal{S}_{count} \leftarrow \mathcal{S}_{count} \cup$ newSet;
12:             end if
13:         end for
14:         return S_count;
15:     end for
16: end while
17: found $\leftarrow \mathcal{S}_n$;
18: for $j$ from $n$ down to 1 do
19:     S_tmp $\leftarrow \mathcal{S}_{j-1}$;
20:     $\overline{\mathcal{S}_{j-1}} \leftarrow \Upsilon(\text{S\_tmp}, \text{found}, \mathcal{C}_{1 \cup n})$;
21:     found $\leftarrow$ found $\cup \overline{\mathcal{S}_{j-1}}$;
22:     $\mathcal{C} \leftarrow \mathcal{C} \cup \overline{\mathcal{S}_{j-1}}$;
23:     return $\mathcal{C}$;
24: end for
```

Table 4.3: UCI dataset characteristics: numbers of objects, attributes, and density.

| Dataset | mushroom | anon-web | census-income |
|---|---|---|---|
| objects | 8124 | 32711 | 103950 |
| attributes | 125 | 294 | 133 |
| density | 17.36% | 1.03% | 6.7% |

gorithm MRCbo. For the FCI mining algorithms, we simulate the classic Closet+ algorithm, and its distributed implementation in conjunction with MRGeneral. All distributed algorithms above are based on the Twister runtime. To show the performance improvement, we provide a fair comparison between the different distributed algorithms and compare them with their centralized versions.

The experiment was run on the Amazon EC2 cloud computing platform. We used High-CPU Medium Instances which have 1.7 GB of memory, 5 EC2 Compute Units (2 virtual cores with 2.5 EC2 Compute Units each), 350 GB of local instance storage, and 32-bit platforms. We selected 3 datasets from the UCI KDD machine learning repository[1], mushroom, anon-web[2], and census-income in our simulations. For the use in later experiment, some preprocesses are needed. We counted all the items that appeared in the dataset to be used as attributes and then give a '0' or '1' for the absence or occurrence of individual attribute in every record. The datasets produced consist of 8124, 32711, 103950 records and 125, 294, 133 attributes respectively. The percentage of 1s is used to measure the dataset density. See row 4 in Table 4.3 for more detail. We only test MRGeneral and Closet+ with mushroom at 2 different support. The CPU time is used as the metric for comparing the performance of each of the algorithms. For distributed FCA algorithms, we keep the CPU time from the algorithms initialization to all global intents are found, and timing MrGeneral from CLOSET+ execution on every machine to all global FCIs are generated. The number of iterations (IR) also impacts the algorithms performance significantly and we counted them in Table 4.6.

[1]See more detail at http://archive.ics.uci.edu/ml/index.html

[2]See http://archive.ics.uci.edu/ml/machine-learning-databases/anonymous/ for the original data.

Table 4.4: Execution time (in seconds) of the centralized FCA algorithms on the three datasets.

| Dataset | mushroom | anon-web | census-income |
|---|---|---|---|
| concepts | 219010 | 129009 | 96531 |
| NextClosure | 618 | 14671 | 18230 |
| CloseByOne | 2543 | 656 | 7465 |

### 4.3.2 Results and Analysis

#### MRGanter, MRGanter+ and MRCbo

In Table 4.4 and 4.5, we present the tested results for centralized algorithms, NextClosure and CloseByOne, and, distributed algorithms, MRGanter, MRCbo and MRGanter+ respectively. As a quick overview, it is clear that MRGanter+ has the best overall performance for the mushroom, anon-web and census-income datasets when 9 nodes and 11 nodes are used respectively. This is indicated by bold in Table 4.5. In comparison with NextClosure, MRGanter+ saves 68%, 96.6% and 98% in time when processing mushroom, anon-web and census-income dataset respectively. For census-income, MRGanter+ has the best performance. MRGanter+ runs 102 times faster than MRGanter and 1.4 times faster than MRCbo. MRCbo runs much faster than CloseByOne when 11 nodes are used. It presents a 90.5% saving in time when dealing with the mushroom dataset compared with CloseByOne, but there is not much of difference when the anon-web dataset is processed. MRGanter takes the longest time to calculate the formal concepts for both the mushroom and anon-web datasets. It is much slower than even the centralized version, NextClosure. The census-income dataset is an exception because MRGanter saves up to half the time with 11 nodes. Among the MR$^*$ algorithms and centralized algorithms, MRGanter+ achieved the best performance.

Taking scalability into account, we tested MR$^*$ algorithms on a range of nodes and plotted curves for each of them to demonstrate the ability of the algorithms to decrease computation time by utilizing more computers. These results are presented in Figure 4.4, 4.5 and 4.6 for each dataset.

In Figure 4.4, MRCbo is slower than MRGanter+ although the curve decreases faster than MRGanter+ when we increase the number of nodes. The execution time of MRGanter+ is fast even on a single node and keeps decreasing up to the maximum number of nodes, 11. The performance of MRGanter

Table 4.5: Execution time (in seconds) of MRGanter, MRCbo and MRGanter+ on the three datasets. These algorithms are tested on various numbers of nodes respectively.

| Dataset | | mushroom | anon-web | census-income |
|---|---|---|---|---|
| concepts | | 219010 | 129009 | 96531 |
| MRGanter | 1 Node | 22222 | 85642 | 34980 |
| | 2 Nodes | 20781 | 38188 | 21666 |
| | 3 Nodes | 20693 | **20110** | 15691 |
| | 5 Nodes | **20269** | 21937 | 10612 |
| | 7 Nodes | 21315 | 25831 | 9891 |
| | 9 Nodes | 20913 | 30029 | 9976 |
| | 11 Nodes | 24530 | 34242 | **9654** |
| MRCbo | 1 Node | 10576 | 4889 | 30193 |
| | 2 Nodes | 3481 | 1659 | 20944 |
| | 3 Nodes | 1588 | 1074 | 9384 |
| | 5 Nodes | 715 | 794 | 3437 |
| | 7 Nodes | 427 | 725 | 1795 |
| | 9 Nodes | 303 | 697 | 1121 |
| | 11 Nodes | **241** | **693** | **803** |
| MRGanter+ | 1 Node | 662 | 16603 | 16467 |
| | 2 Nodes | 399 | 3442 | 7687 |
| | 3 Nodes | 294 | 1092 | 6155 |
| | 5 Nodes | 227 | 626 | 909 |
| | 7 Nodes | 212 | 545 | 529 |
| | **9 Nodes** | **198** | **496** | 446 |
| | 11 Nodes | 202 | 559 | **358** |

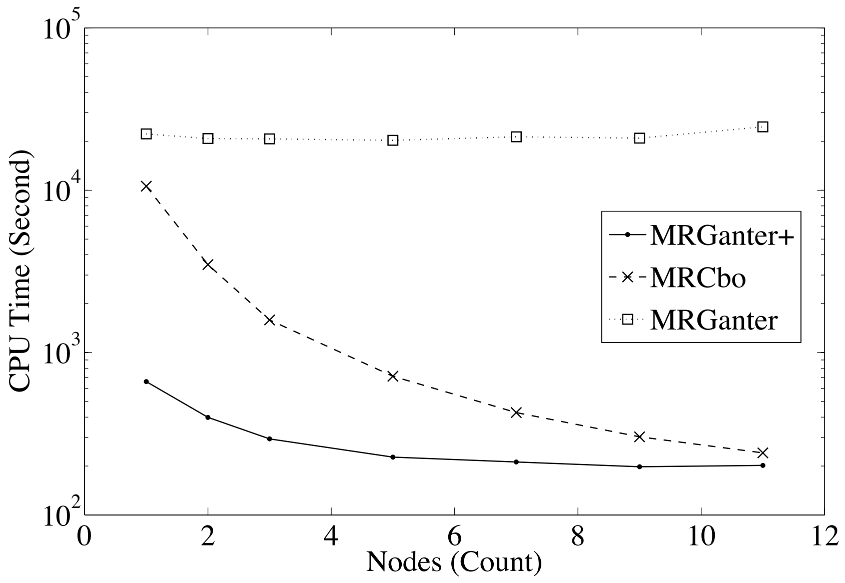

Figure 4.4: Mushroom dataset: comparison of MRGanter+, MRCbo and MRGanter. MRGanter+ outperforms MRCbo and MRGabter when dense data is processed.

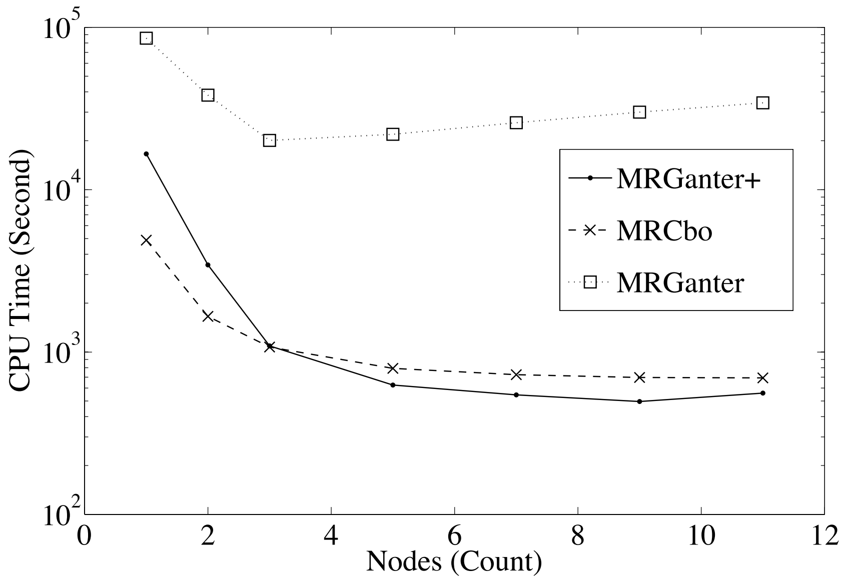

Figure 4.5: Anon-web dataset: comparison of MRGanter+, MRCbo and MRGanter. MRGanter+ is faster when more than 3 nodes are used.

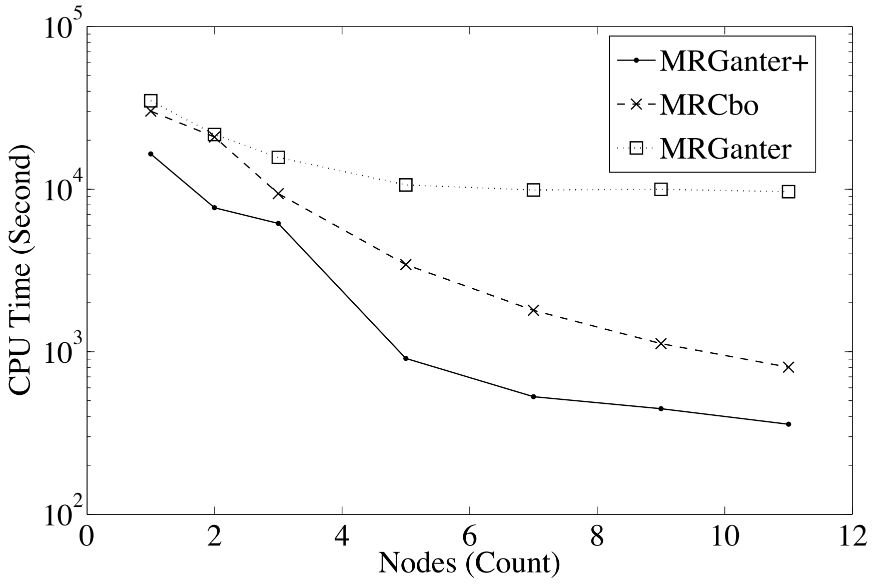

Figure 4.6: Census-income dataset: comparison of MRGanter+, MRCbo and MRGanter. MRGanter+ is fastest when a large dataset is processed.

is not beneficially affected by increasing the number of nodes. When using 11 nodes, the execution time of MRGanter even increases again. One explanation for this is the overhead incurred by distributing the computation, for example network communication overhead. This is markedly different from MRGanter+, because MRGanter+ produces substantially more intermediate data than MRGanter and MRCbo. Moreover, there is additional computation involved in the distributed algorithms in comparison with the centralized versions of these algorithms. Consider, for instance, the extra operation needed by the merging operation. The best number of nodes, in terms of performance speed, depends on the density characteristics of the dataset.

Figure 4.5 demonstrates that MRGanter+ outperforms MRGanter for the anon-web dataset. One reason for this performance improvement is that both algorithms produce different numbers of concepts during each iteration. Table 4.6 indicates that MRGanter+ requires 12, 11 and 9 iterations fore each of the datasets, whereas MRGanter requires 219010, 129009 and 96531 iterations to obtain all concepts. These additional iterations incur higher network communication costs. Figure 4.6 demonstrates that this is also the case for the census-income dataset. In addition, the curves in Figure 4.6 are steeper than the curves in Figure 4.4 and 4.5. These figures give evidence that the performance of the MR* algorithms is related to size and density of

Table 4.6: Number of iterations required for each of the three datasets.

| Dataset | mushroom | anon-web | census-income |
|---:|---:|---:|---:|
| concepts | 219010 | 129009 | 96531 |
| NextClosure | 219010 | 129009 | 96531 |
| CloseByOne | 14 | 11 | 11 |
| MRGanter | 219010 | 129009 | 96531 |
| MRCbo | 14 | 11 | 11 |
| MRGanter+ | 12 | 11 | 9 |

the data. Based on these results we posit that MR$^*$ algorithms scale well for large and sparse datasets. This evidence suggests that MR$^*$ algorithms may be a viable candidate tool for handling large datasets, particularly when it is impractical to use a traditional centralized technique.

Classical formal concept computing methods usually act on, and have local access to the entire database. Network communication is the primary concern when developing distributed FCA approaches: Frequent requests to remote databases incur significant time and resource costs. Performance improvements of the algorithms proposed in this paper may potentially arise from preprocessing the dataset so that the dataset is partitioned in a more efficient manner. One direction for improving these algorithms lies in making the partitions more even, in terms of density, so that the complexity is distributed more equably. In future work we intend to explore the effect of data distribution between cluster nodes in more detail. We propose to extend this empirical study in a companion paper which examines algorithm performance on larger dataset sizes. We will also study the affects the data distribution has on the optimal number of nodes. In addition, we intend to extend these methods so that they reduce the size of intermediate data produced in each iteration. We posit that further improvement of the methods proposed here could motivate a more widespread adoption of FCA using the MapReduce framework.

### MRGeneral

Table 4.7 shows the execution time of Closet+ on the mushroom dataset for different levels of support (50%, 20%). Table 4.8 and 4.9 list the results for

Table 4.7: Execution time (in seconds) of Closet+ for the mushroom data for a range of supports. The third row indicates the number of produced FCIs for the corresponding support.

| support | 50% | 20% | 5% | 0.5% | 0.1% |
|---|---|---|---|---|---|
| Time | 3.067 | 4.593 | 12.419 | 103.536 | 319.530 |
| FCIs | 45 | 1197 | 12843 | 76198 | 147905 |

Table 4.8: Execution time (in seconds) of MRGeneral for the mushroom data when the support is 50%, when different numbers of nodes is used. Forty five frequent closed itemsets are mined for each case.

| Number of nodes | Time |
|---|---|
| 2 | 339 |
| 3 | 249 |
| 5 | 474 |
| 7 | 991 |
| 9 | 1589 |

MRGeneral for 50% and 20% support respectively.

In Table 4.7-4.9, we conclude that MRGeneral takes a much longer time than Closet+. MRGeneral is 81 times slower than Closet+ even for the most favourable set of results. In Table 4.8, the execution time has a large range. Figure 4.7 exposes that MRGeneral consumes much more execution time once more than 3 nodes are used. In other words, using more nodes does not necessarily guarantee better performance. For example, running on only 3 nodes, MRGeneral is quite fast. Generally speaking, when the support is low and more nodes are added, MRGeneral consumes more processing time to process. In fact, it gets stuck on very low support such as 0.1%.

In addition, we recorded the number of local FCIs and local IFCIs produced in every local data partition. This helped to analyze the factors that impacted the performance of MRGeneral. Let us observe the relationship between the local FCIs and IFCIs. Before going further, we define an average ratio $r = L_1/L_2$, where $L_1$ is the sum of local IFCIs and $L_2$ is the sum

Table 4.9: Execution time (in seconds) of MRGeneral for the mushroom data when the support is 20%, when different numbers of nodes is used. One thousand one hundred and ninety seven frequent closed itemsets are mined for each case.

| Number of nodes | Time |
|---|---|
| 2 | 16117 |
| 3 | 14881 |
| 5 | 26576 |
| 7 | 40029 |
| 9 | 52836 |

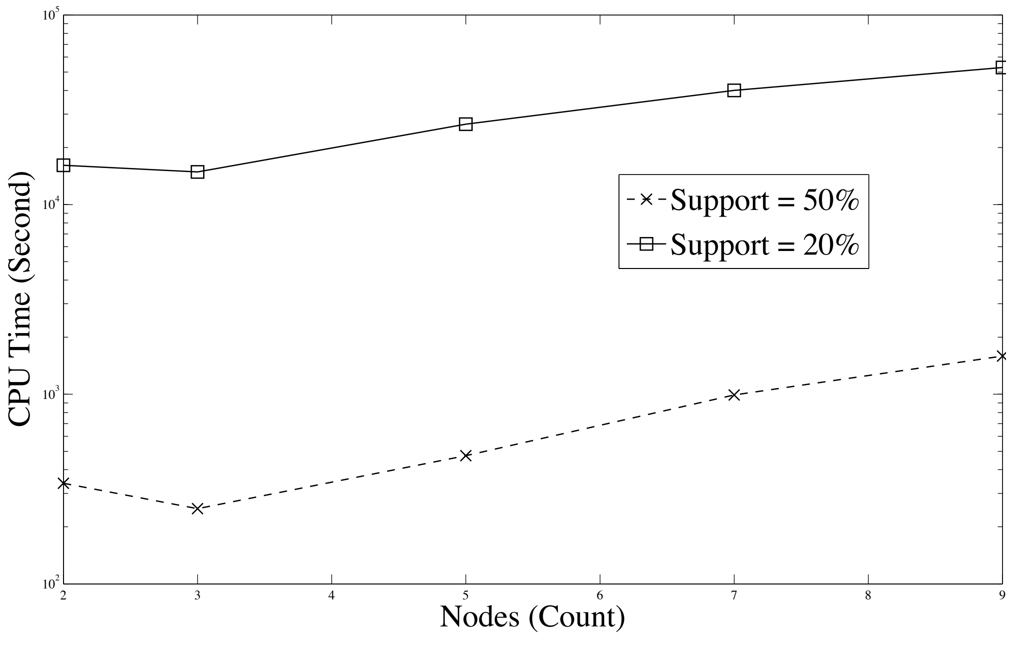

Figure 4.7: Scalability of MRGeneral versus different supports.

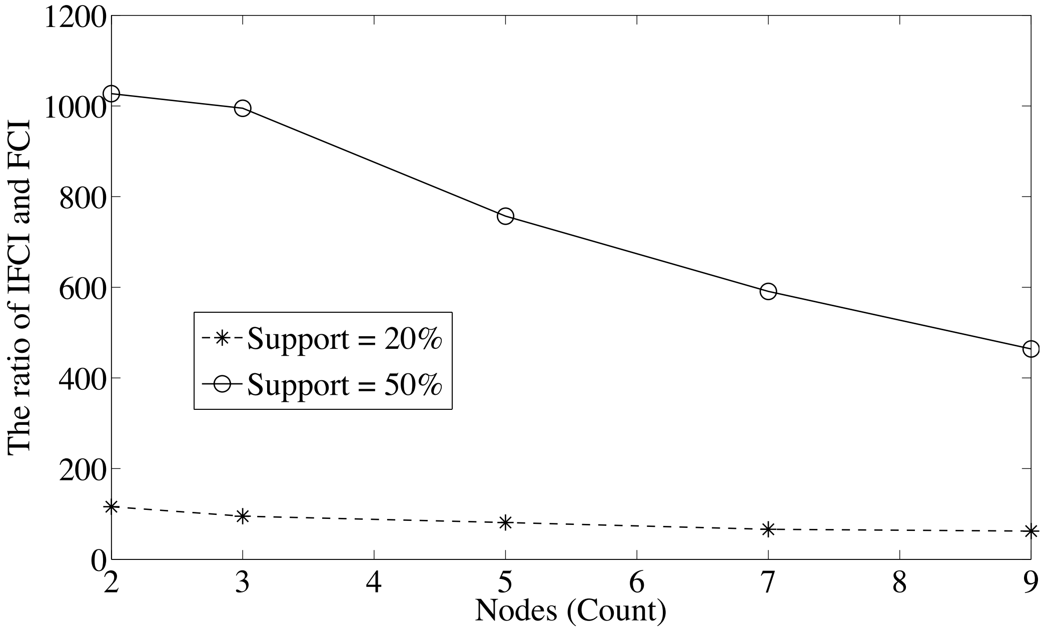

Figure 4.8: The ratios between the number of IFCIs and the number of FCIs.

of local FCIs respectively. The results are presented in Figure 4.8.

Figure 4.8 indicates that the radio decreases when more nodes are used, which is what we expect to happen. The quantity of local IFCIs, $L_1$, is normally much greater than the quantity of local FCIs, $L_2$. When a dataset is divided into more parts, more local IFCIs and local FCIs will be produced. This is understandable seeing as the IFCIs and FCIs are dispersed over more nodes. Although the number of both the local IFCIs and the local FCIs increase, the number of the local FCIs increases more significantly. This results in that the difference between them gets smaller thereby *ratio* decreases. Subsequently, more computation and longer time are required to deal with extra local closed itemsets. On the other hand, mining with low support will produce much more local FCIs but relative fewer local IFCIs, hence the ratio, $r$, is smaller in the case of high support. In addition, the ratio changes slower when more nodes are used as shown in Figure 4.8. This characteristic is reflected well in Figure 4.7 as well: the curve denoting low support approximately a constant. MRGeneral behaves better for low support. The important point to note is that MRGeneral has the ability to process distributed dataset although a long time are required currently.

In a word, MRGeneral performs correctly with distributed datasets, although it is not as scalable as MRGanter+. This is understandable because

MRGeneral has too much extra operations which are executed after the centralized FCI mining algorithm. Thus MRGeneral is promising for low support mining which is a difficult problem particularly for large distributed data, although it is not as fast as centralized algorithms when processing normal-size data. This is a useful property to have given the explosion in modern dataset size.

## 4.4 Summary

This chapter described the process of implementing distributed FCA and FCI algorithms All of the proposed distributed algorithms and their centralized versions were tested on several datasets with various number of computing nodes. We analyzed the execution time and the scalability of proposed algorithms, and a comparison was made between the proposed distributed algorithms and an exiting distributed algorithm which was proposed by [20] in the literature, and the original centralized versions of the algorithms. From the experimental results, MRGanter+ shows the impressive performance and scalability for the datasets in different size and density.

Chapter 5

# Discussion

By observing the nature of global closed itemsets and exploiting the relation between global and local closed itemsets, this work developed distributed algorithms to mine formal concepts and frequent closed itemsets. We redesigned some of the well known FCA algorithms in the literature and designed a general model for mining FCIs based on the MRF. Experimental results demonstrated that MRGanter+ was efficient, scalable and outperformed other distributed FCA algorithms. The merging model for distributed FCI mining proposed in this thesis is time-consuming, but we identified ways to improve it. This improvement constitutes one possible avenue for future work. Chapter 4 analysed the drawbacks in our approach and implementation, and indicated the steps for future improvement.

Theorem 3, distributed formal concept mining theory, is quite straight forward. However, MRGanter is much slower than NextClosure and in particular MRCbo and MRGanter+. In Section 4.3.2, we attributed this processing slowness to the fact that MRGanter only produces a single concept during each iteration. The comparison between MRGanter and MRCbo(MRGanter+) in Figure 5.1 demonstrated this.

Considering the three distributed FCA algorithms, MRGanter, MRCbo and MRGanter+, we demonstrated that MRGanter+ produces the most concepts in each iteration so that it needs the least number of iterations. However, for the same reason, MRGanter+ consumes the most communication resource because many concepts need to be held and passed in the distributed memory. Communication overhead presents the main bottleneck for MRGanter+ for large datasets.

Unlike Theorem 3, Proposition 1 and 2 have high complexity, they contain many intersection and union operations. Some operations are redundant as they duplicate effort. Proposition 1 and 2 could be better optimized, however,

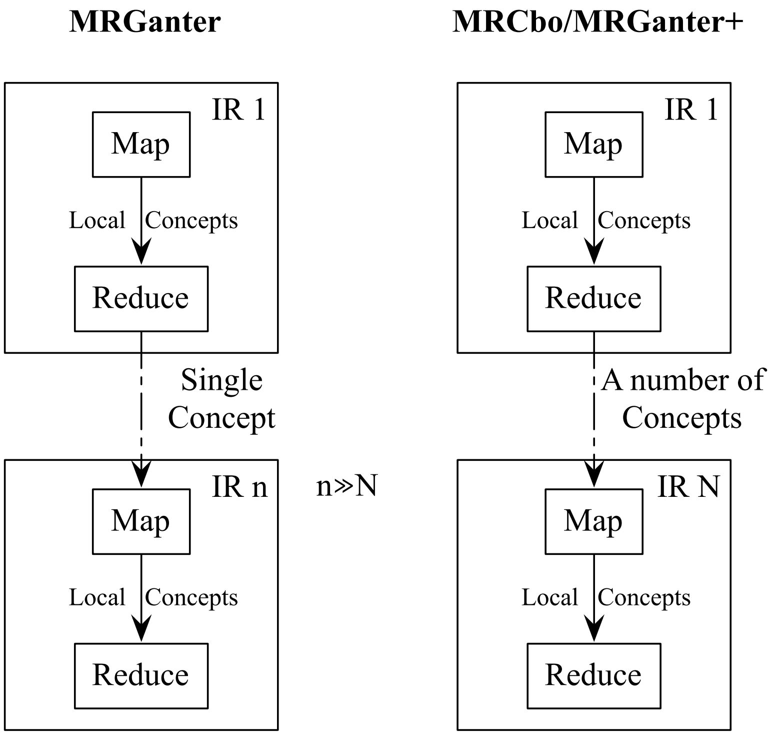

Figure 5.1: Comparison between the inner structure of MRGanter and MR-Cbo. MRGanter produces a single concept during each iteration, whereas MRCbo produces many concepts.

improvement may cause increased cost in terms of communication overhead. A further improvement in the implementations of these algorithms could include a refinement of methods for storing and accessing intermediate data more rapidly. All of these factors explain the latency of MRGeneral. We will address these issues and improve MRGeneral's performance in future work. For example, the merging model used in MRGeneral will be optimized to reduce unnecessary computation such as the duplication of intersections between local itemsets from different nodes.

We consider now the method used for preparing the data in this thesis. The datasets used in this work were randomly divided into blocks of equal number of records. In the real world, the distribution of data is not always uniform. The datasets could be processed faster if the inequality of the dataset was decreased, however, we did not consider the pre-processing of the partitions in these experiments, but rather focused on algorithm development. We will consider procedures for optimizing the pre-processing of the datasets in future work. However, it may often be the case, in practice, that due to the constraints of the application that re-organizing the data is not feasible and that the algorithms will have run on the data in its original partition.

Our future research directions are described as follows.

- We will consider optimizing the approach to reduce the creation of unnecessary local FCIs and IFCIs;
- We will enhance Twister by adding the ability to store intermediate data on a local disk rather than in distributed memory. This will help to avoid communicating all intermediate data via a broker network and reduce communication costs and time;
- We will consider the problem of eliminating inequality in the distribution of the dataset by partitioning the dataset evenly via preprocessing step that balances the computation load over every node.
- We will test the distributed algorithms over a broaden range of datasets and further extend the validation of these techniques.
- We will carry out more experiments for our algorithms on more computation nodes and higher specification hardware.

Chapter 6

# Conclusion

This thesis focuses on two types of closure mining in a distributed manner, Formal Conceptss (FCs) mining and Frequent Closed Itemset (FCI) mining. We explore the state of the art Formal Concept Analysis (FCA) and Association Rules Mining (ARM) and analyze the pros and cons of existing distributed solutions in FCA and FCI mining. We propose distributed FC and FCI mining algorithms based on the MRF. The contributions of this thesis include:

- Distributed Formal Concept and Frequent Closed Itemset mining theories.
- Application of an iterative MRF, Twister, on distributed FC and FCI mining.
- Two distributed Formal Concept mining algorithms, MRGanter and MRGanter+, which are based on a classic FCA algorithm, NextClosure in conjunction with Twister. Another distributed FCA algorithm, MRCbo which is based on centralized algorithm CloseByOne, is presented using Twister framework.
- A general model for computing Frequent Closed Itemsets in a distributed manner based on Twister framework.

The experimentation conducted on real datasets verified that: 1)MRGanter+ is more efficient and scalable than MRGanter, MRCbo, NextClosure and CloseByOne; 2) for MR$^\star$ algorithms, using more nodes does not guarantee better performance. There is an optimal number of nodes that gives the best performance; 3) when the support is low and mode nodes are added, MRGeneral consumes more processing time; 4) MRGeneral is promising for low support mining which is difficult problem particularly for large distributed data. We believe the contribution made in this thesis will be of benefit to the data mining practitioner, and will facilitate an increase in the usage of Formal Concept Analysis and Frequent Closed Itemset mining methods in distributed settings.

# Bibliography

[1] Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Claudio Silvestri. Mining Frequent Closed Itemsets from Highly Distributed Repositories. In *Proc. of the 1st CoreGRID Workshop on Knowledge and Data Management in Grids in conjunction with PPAM2005*, 2005.

[2] Michael Milton. Head first data analysis: A learner's guide to big numbers, statistics, and good decisions. 1, 2009.

[3] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, pages 37–54, 1996.

[4] Katharina Morik. Applications of knowledge discovery. In *Proceedings of the 18th international conference on Innovations in Applied Artificial Intelligence*, IEA/AIE'2005, pages 1–5, London, UK, 2005. Springer-Verlag.

[5] Amit Singhal. Modern information retrieval: A brief overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(4)::35–43, 2001.

[6] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, New York, NY, USA, 1993. ACM.

[7] Mohammed Javeed Zaki and Mitsunori Ogihara. Theoretical foundations of association rules. In *DMKD'98 workshop on research issues in Data Mining and Knowledge Discovery*, pages 1–8. ACM Press, June 1998.

[8] Rudolf Wille. Restructuring lattice theory: an approach based on hierarchies of concepts. In Ivan Rival, editor, *Ordered sets*, pages 445–470. Reidel, 1982.

[9] Bernhard Ganter and Rudolph Wille. *Formal concept analysis: Mathematical foundations.* Springer, Berlin-Heidelberg, 1999.

[10] Nicolas Pasquier, Yves, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24:25–46, 1999.

[11] Hillol Kargupta and Philip Chan, editors. *Advances in Distributed and Parallel Knowledge Discovery.* AAAI/MIT Press, 2000.

[12] Sergei O. Kuznetsov and Sergei A. Obiedkov. Comparing performance of algorithms for generating concept lattices. *Journal of Experimental and Theoretical Artificial Intelligence*, 14:189–216, 2002.

[13] Claudio Lucchese and Salvatore Orlando. Dci_closed: A fast and memory efficient algorithm to mine frequent closed itemsets. In *2nd IEEE ICDM Workshop on Frequent Itemset Mining Implementations 2004*, Brighton, UK, 2004.

[14] Jianyong Wang, Jiawei Han, and Jian Pei. Closet+: searching for the best strategies for mining frequent closed itemsets. In *KDD'03*, pages 236–245, 2003.

[15] Christoph Schmitz, Andreas Hotho, Robert Jäschke, and Gerd Stumme. Mining association rules in folksonomies. In *Data Science and Classification. Proceedings of the 10th IFCS Conf.*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 261–270, Heidelberg, July 2006. Springer.

[16] I-Hsien Ting. Web mining techniques for on-line social networks analysis. *2008 International Conference on Service Systems and Service Management*, (700):1–5, 2008.

[17] Amin Milani Fard and Martin Ester. Collaborative mining in multiple social networks data for criminal group discovery. In *CSE (4)'09*, pages 582–587, 2009.

[18] Petko Valtchev, Rokia Missaoui, and Robert Godin. Formal concept analysis for knowledge discovery and data mining: The new challenges. In Peter W. Eklund, editor, *ICFCA*, volume 2961 of *Lecture Notes in Computer Science*, pages 352–371. Springer, 2004.

[19] Lotfi Lakhal and Gerd Stumme. Efficient mining of association rules based on formal concept analysis. In *Formal Concept Analysis'05*, pages 180–195, 2005.

[20] Petr Krajca and Vilem Vychodil. Distributed algorithm for computing formal concepts using map-reduce framework. In *IDA '09: Proceedings of the 8th International Symposium on Intelligent Data Analysis*, pages 333–344, Berlin, Heidelberg, 2009. Springer-Verlag.

[21] Raffaele Perego Claudio Lucchese, Salvatore Orlando. Distributed mining of frequent closed itemsets: Some preliminary results. In *In Proceedings of HPDM'05- 8th SIAM Workshop on High Performance and Distributed Mining*, Newport Beach, California, USA, April 23 2005.

[22] Qiang Yang and Xindong Wu. 10 Challenging Problems in Data Mining Research. *International Journal of Information Technology & Decision Making*, 5(4):597–604, 2006.

[23] Sadok Ben Yahia, Tarek Hamrouni, and E. Mephu Nguifo. Frequent closed itemset based algorithms: a thorough structural and analytical survey. *SIGKDD Explor. Newsl.*, 8:93–104, June 2006.

[24] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In Salim Hariri and Kate Keahey, editors, *HPDC*, pages 810–818. ACM, 2010.

[25] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[26] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[27] Lotfi Lakhal and Gerd Stumme. Efficient mining of association rules based on formal concept analysis. In *Formal Concept Analysis'05*, pages 180–195, 2005.

[28] Jian Pei, Jiawei Han, and Runying Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery'00*, pages 21–30, 2000.

[29] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Mining frequent closed itemsets out-of-core. In *SDM*, 2006.

[30] B. Ganter. Two basic algorithms in concept analysis. *Formal Concept Analysis*, pages 312–340, 1984.

[31] C. Lindig. Fast concept analysis. *Working with Conceptual Structures-Contributions to ICCS*, 2000:235–248, 2000.

[32] Sergei O. Kuznetsov. A Fast Algorithm for Computing All Intersections of Objects in a Finite Semi-Lattice. *Automatic Documentation and Mathematical Linguistics*, 27(5):11–21, 1993.

[33] S Andrews. In-Close, a Fast Algorithm for Computing Formal Concepts. In *The Seventeenth International Conference on Conceptual Structures*, 2009.

[34] Vilem Vychodil. A new algorithm for computing formal concepts. *Cybernetics and Systems*, pages 15–21, 2008.

[35] Petr Krajca, Jan Outrata, and Vilem Vychodil. Parallel Recursive Algorithm for FCA. In *CLA 2008*, volume 433, pages 71–82. CLA2008, 2008.

[36] Jean-Paul Bordat. Calcul pratique du treillis de Galois d'une correspondance. *Mathématiques et Sciences Humaines*, 96:31–47, 1986.

[37] Anne Berry, Jean-Paul Bordat, and Alain Sigayret. A local approach to concept generation. *Annals of Mathematics and Artificial Intelligence*, 49(1):117–136, 2006.

[38] E. M. Norris. An algorithm for computing the maximal rectangles of a binary relation. *ACM*, 21:356–266, 1974.

[39] Cornelia E. Dowling. On the irredundant generation of knowledge spaces. *J. Math. Psychol.*, 37:49–62, March 1993.

[40] Robert Godin, Rokia Missaoui, and Hassan Alaoui. Incremental concept formation algorithms based on galois (concept) lattices. *Computational Intelligence*, pages 246–267, 1995.

[41] Claudio Carpineto and Giovanni Romano. A lattice conceptual clustering system and its application to browsing retrieval. *Machine Learning*, pages 95–122, 1996.

[42] Petko Valtchev, Rokia Missaoui, and Pierre Lebrun. A partition-based approach towards constructing galois (concept) lattices. *Discrete Mathematics*, pages 801–829, 2002.

[43] Yuan Yu, Xu Qian, Feng Zhong, and Xiao rui Li. An improved incremental algorithm for constructing concept lattices. *Software Engineering, World Congress on*, 4:401–405, 2009.

[44] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI*, page 13, 2004.

[45] E. Ramaraj and N. Venkatesan. Positive and negative association rule analysis in health care database. *IJCSNS International Journal of Computer Science and Network Security*, 8:325–330, 2008.

[46] S. Stilou, P. D. Bamidis, N. Maglaveras, and C. Pappas. Mining association rules from clinical databases: An intelligent diagnostic process in healthcare. 2001.

[47] Stefano Concaro, Lucia Sacchi, Carlo Cerra, Pietro Fratino, and Riccardo Bellazzi. Mining healthcare data with temporal association rules: Improvements and assessment for a practical use. In *Proceedings of the 12th Conference on Artificial Intelligence in Medicine: Artificial Intelligence in Medicine*, AIME '09, pages 16–25, Berlin, Heidelberg, 2009. Springer-Verlag.

[48] Walter Christian Kammergruber, Maximilian Viermetz, Karsten Ehms, and Manfred Langen. Using association rules for discovering tag bundles in social tagging data. In *Computer Information Systems and Industrial Management Applications (CISIM), 2010 International Conference on*, pages 414 –419, oct. 2010.

[49] Waleed Aljandal, Vikas Bahirwani, Doina Caragea, and William Hsu. Ontology-aware classification and association rule mining for interest and link prediction in social networks. In *Proceedings of the AAAI 2009 Spring Symposium on Social Semantic Web: Where Web 2.0 meets Web 3.0*, 2009.

[50] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *VLDB'94*, pages 487–499, 1994.

[51] Yves Bastide, Nicolas Pasquier, Rafik Taouil, Gerd Stumme, and Lotfi Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. In *Computational Logic'00*, pages 972–986, 2000.

[52] Mohammed Javeed Zaki. Generating non-redundant association rules. In *KDD'00*, pages 34–43, 2000.

[53] Mafruz Zaman Ashrafi, David Taniar, and Kate A. Smith. Redundant association rules reduction techniques. In *Australian Conference on Artificial Intelligence'05*, pages 254–263, 2005.

[54] Roberto J. Bayardo Jr. Efficiently mining long patterns from databases. In *SIGMOD Conference'98*, pages 85–93, 1998.

[55] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT'99*, pages 398–416, 1999.

[56] Mohammed Javeed Zaki and Mitsunori Ogihara. Theoretical foundations of association rules. In *In 3rd ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1998.

[57] Jean-Francois Boulicaut, Artur Bykowski, and Christophe Rigotti. Approximation of frequency queris by means of free-sets. In *PKDD '00: Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 75–85, London, UK, 2000. Springer-Verlag.

[58] Artur Bykowski. Free-sets: a condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery*, 7:2003, 2003.

[59] Artur Bykowski and Christophe Rigotti. A condensed representation to find frequent patterns. In *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 267–273, New York, NY, USA, 2001. ACM.

[60] Artur Bykowski and Christophe Rigotti. Dbc: a condensed representation of frequent patterns for efficient mining. *Inf. Syst.*, pages 949–977, 2003.

[61] Marzena Kryszkiewicz. Concise representation of frequent patterns based on disjunction-free generators. In *ICDM'01*, pages 305–312, 2001.

[62] Toon Calders and Bart Goethals. Mining all non-derivable frequent itemsets. In *PKDD'02*, pages 74–85, 2002.

[63] Toon Calders and Bart Goethals. Depth-first non-derivable itemset mining. In *In Proc. SIAM Int. Conf. on Data Mining SDM'05*, 2005.

[64] Toon Calders and Bart Goethals. Minimal k-free representations of frequent sets. In *PKDD'03*, pages 71–82, 2003.

[65] Douglas Burdick, Manuel Calimlim, and Johannes Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *ICDE'01*, pages 443–452, 2001.

[66] Mohammed Javeed Zaki and Ching-Jiu Hsiao. Charm: An efficient algorithm for closed itemset mining. In *SDM'02*, pages 457–473, 2002.

[67] Chunhua Ju and Dongjun Ni. Mining frequent closed itemsets from distributed dataset. In *ISCID '08: Proceedings of the 2008 International Symposium on Computational Intelligence and Design*, pages 37–41, Washington, DC, USA, 2008. IEEE Computer Society.

[68] Chun Liu, Zheng Zheng, Kai-Yuan Cai, and Shichao Zhang. Distributed frequent closed itemsets mining. *Signal-Image Technologies and Internet-Based System, International IEEE Conference on*, 0:43–50, 2007.

[69] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining.* Addison-Wesley, 2005.

[70] Ashok Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. In *VLDB'95*, pages 432–444, 1995.

[71] Mohammed Javeed Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12:372–390, 2000.

[72] Jian Pei, Jiawei Han, Hongjun Lu, Shojiro Nishio, Shiwei Tang, and Dongqing Yang. H-mine: Hyper-structure mining of frequent patterns in large databases. In *ICDM'01*, pages 441–448, 2001.

[73] Junqiang Liu, Yunhe Pan, Ke Wang, and Jiawei Han. Mining frequent item sets by opportunistic projection. In *In Proc. 2002 Int. Conf. on Knowledge Discovery in Databases (KDD'02*, pages 229–238. ACM Press, 2002.

[74] Jiawei Han and Jian Pei. Mining frequent patterns by pattern-growth: methodology and implications. *SIGKDD Explor. Newsl.*, 2(2):14–20, 2000.

[75] Jian Pei. *Pattern-growth methods for frequent pattern mining.* PhD thesis, Burnaby, BC, Canada, Canada, 2002. AAINQ81682.

[76] Hai-Tao He, Hai-Yan Cao, Rui-Xia Yao, Jia-Dong Ren, and Chang-Zhen Hu. Mining frequent itemsets based on projection array. In *Machine Learning and Cybernetics (ICMLC), 2010 International Conference on*, volume 1, pages 454 –459, july 2010.

[77] Byung-Hoon Park and Hillol Kargupta. Distributed data mining: Algorithms, systems, and applications. pages 341–358, 2002.

[78] Shrideep Pallickara and Geoffrey Fox. Naradabrokering: A distributed middleware framework and architecture for enabling durable peer-to-peer grids. In Markus Endler and Douglas C. Schmidt, editors, *Middleware*, volume 2672 of *Lecture Notes in Computer Science*, pages 41–61. Springer, 2003.

[79] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In Weidong Chen, Jeffrey Naughton, and Philip A. Bernstein, editors, *2000 ACM SIGMOD Intl. Conference on Management of Data*, pages 1–12. ACM Press, 2000.