MINING MULTIDIMENSIONAL DISTINCT PATTERNS

by

Thusjanthan Kubendranathan

B.Sc. (Hons.), University of Toronto, 2005

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Fall 2010

APPROVAL

Name: Thusjanthan Kubendranathan

Degree: Master of Science

Title of Thesis: Mining Multidimensional Distinct Patterns

Examining Committee: Dr. Diana Cukierman
Chair

Dr. Jian Pei
Associate Professor, Computing Science
Simon Fraser University
Senior Supervisor

Dr. Oliver Schulte
Associate Professor, Computing Science
Simon Fraser University
Supervisor

Dr. Wo-Shun Luk
Professor, Computing Science
Simon Fraser University
Examiner

Date Approved: November 15, 2010

Declaration of
Partial Copyright Licence

Abstract

How do we find the dominant groups of customers in age, sex and location that were responsible for at least 85% of the sales of iPad, Macbook and iPhone? To answer such types of questions we introduce a novel data mining task – mining multidimensional distinct patterns (DPs). Given a multidimensional data set where each tuple carries some attribute values and a transaction, multidimensional DPs are itemsets whose absolute support ratio in a group-by on the attributes against the rest of the data set passes a given threshold. A baseline algorithm uses BUC as our cubing algorithm, and passes two distinct sets of transactions associated to the tuples of the cell to a pattern mining algorithm called DPMiner. The use of several effective pruning techniques eliminates redundant processing of DPMiner and reduces the runtime. The empirical study between the baseline and advanced algorithm demonstrates that the advanced algorithm is significantly faster.

Keywords: distinct patterns, data mining, OLAP, data cube, DPMiner, BUC

To my loving family for all their support.

"If I have seen a little further it is by standing on the shoulders of Giants."

— Isaac Newton, 1676

Acknowledgments

It is difficult to find the words to express my gratitude to my senior supervisor Dr. Jian Pei. The quote by Isaac Newton which states "If I have seen a little further it is by standing on the shoulders of Giants." really does tell the story. I would like to thank Dr. Pei for showing me the sight of research, the guidance to put them to words and the voice to articulate the research. It would be impossible for me to have made this journey of enlightenment if it was not for the encouragement, the patience and suggestions provided by Dr. Pei.

I would like to thank my supervisor Dr. Oliver Schulte for the encouragement and insights on the thesis. Even though with his hectic research schedule in Germany, he found the time to address my questions and concerns. I would also like to thank Dr. Wo-Shun Luk, the examiner, for taking the time to review my work and to Dr. Diana Cukierman for acting as the Chair to my thesis defense and organizing the process.

I would like to thank my wife, Savitha. As I spent many tireless and sleepless nights working away on my thesis, she was there with encouragement and support. She gave me the strength and confidence to complete my research.

I would like to thank Greg Baker, Zhi Min Tang, Sharmila Sivasankaran, Annan Harley, Premini Manivannan, Thivya Sornalingam, Scott Hadfield, and Sajeev Vijayakumar for taking their precious time to review my thesis.

Last, but definitely not least, I would like to thank my parents, my sister, my brothers, Val Galat, Computing Science staff and my friends for all their support and encouragement throughout the years.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent times, data in the world has grown immensely in both size and complexity. Through the use of tools such as Bayes Theorem, decision trees and Support Vector Machines, data mining attempts to uncover hidden patterns in data; with the ultimate goal being to use those patterns to inform ones approach to problem solving in a particular domain. On the business front, data mining has become a crucial tool as a means to uncover hidden trends and patterns of sales to gain an informational advantage over the competition. Moreover, in recent decades, it has even been used in the biomedical field, marketing, fraud detection, natural languages and the list goes on. Thus, as the data size and complexity increases, researchers constantly try to find new and innovative ways of mining useful information.

Therefore we introduce a new kind of pattern mining called distinct patterns (DPs). A dataset is a set of tuples of items (e.g. items we buy from a store, symptoms from diseases), the support of an itemset X in a dataset is the total number of occurrences of X in that dataset. A frequent itemset is an itemset such that it must have a support that is greater than or equal to a given threshold. Thus, given two datasets of items D_1 and D_2, a distinct pattern is a frequent itemset X such that the absolute support ratio of X from D_2 to D_1 is greater than a given threshold.

1.1 Motivation

In this section we will motivate the distinct pattern problem. In Subsection 1.1.1, we will apply distinct patterns to potential real world scenarios. In Subsection 1.1.2, we will further

extend our examples in Subsection 1.1.1 to demonstrate the application of multidimensional distinct patterns . Finally in Subsection 1.1.3, we employ the use of data cubes to mine multidimensional distinct patterns.

1.1.1 Mining Distinct Patterns

This subsection answers the following two questions: (1) Why do we need to mine and use distinct patterns? (2) How can we use distinct patterns in the real world? We answer these questions by introducing real life scenarios.

Example 1.1.1. The following is a business model example. Suppose we own a chain of electronic stores across Canada and have the transactions of all the sales categorized by city. Given a percentage threshold, using distinct patterns we can find out all the different combinations of items that were bought in the different cities provided the sales of those items occur at least the percentage threshold amount relative to all transactions of those particular items. For example, given percentage threshold as 75%, if we find that the items iPad, Bose Companion 5 speaker and Canon MX 320 printer were bought at least 75% of the time in Toronto compared to all the sales of the items iPad, Bose Companion 5 speaker and Canon MX 320 printer, then we can either advertise those items more in Toronto or advertise more in the other cites where those items were not purchased as much. Another business strategy using the same sceario would be if a person from Toronto buys an iPad, we could provide suggestions of other items such as the Bose Companion 5 speaker and Canon MX 320 printer that were also bought together with iPad. The chances of that particular customer buying the other items suggested is increased as we have determined that 75% of items iPad, Bose Companion 5 speaker and Canon MX 320 printer are bought together in Toronto.

Example 1.1.2. The following is a biomedical field example. Suppose we have the symptoms of everyone across the world categorized by disease. Given a percentage threshold, using distinct patterns we can find out all the different combinations of the symptoms and the associated disease provided the symptoms occurs at least the percentage threshold amount relative to all of those particular symptoms. For example, given percentage threshold as 99.9% (albeit unlikely), if we find that the symptoms headache, vomiting, confusion, fear of bright lights and seizures occur 99.9% of the time when a person has meningitis.

Then when a doctor sees a patient that exhibits those symptoms, he or she can then have the patient tested for meningitis.

1.1.2 Mining Multidimensional Distinct Patterns

In this subsection, we will use the real life examples from Subsection 1.1.1 to illustrate multidimensional distinct patterns.

Example 1.1.3. Referring back to Example 1.1.1, we have the sales categorized by city. What if we further categorized the sales by a few more dimensions such as age group and sex? Given a multidimensional database, we can further categorize sales to pin point the target groups of the sales of different combinations of items. That is, we can find that iPad, Bose Companion 5 speaker and Canon MX 320 printer were bought in Toronto 75% of the time but also that they were bought 75% of the time by males that were 15-20 years old. Or we find out that 80% of the items bluetooth headphones, and printers were bought by females that were 30-40 years old. Using such information we can better advertise to a smaller set of target groups thus decreasing the cost of advertising and at the same time directing it at our target audience.

Example 1.1.4. Referring back to Example 1.1.2, we have the symptoms of people categorized by diseases. What if we further categorized by adding age group and location? Using this multidimensional database, we can now not only find the disease associated to a particular set of symptoms but we can also find the age group and location. For example, given percentage threshold as 99.9%, again although unlikely, if we find that the symptoms headache, vomiting, confusion, fear of bright lights and seizures occur 99.9% of the time when a person has meningitis that is 20-30 years old from Africa. Then when a doctor sees a patient from Africa who is 20-30 years old that exhibits those symptoms, he or she can then have the patient tested for meningitis. In addition, if we also add date as another dimension where it represents the date the person was diagnosed with that particular disease; we can use this with varying percentage thresholds to see if the disease is being migrated to another region. That is, if we know the symptoms with the particular disease was mostly diagnosed in Africa in 2007 but in 2008 it was mostly diagnosed in Canada. Then we can take measures to investigate why the disease is migrating to Canada.

Age Group	*Sex*	*Job*	*Location*	*Transaction*
15-20	M	Cashier	Toronto	$\{iPad, Canon, Bluetooth, Batteries\}$
15-20	M	Cashier	Toronto	$\{Canon, Bluetooth, Batteries\}$
15-20	M	Cashier	Vancouver	$\{iPad, Bluetooth, Batteries\}$
15-20	M	Tutor	Vancouver	$\{Canon, Bluetooth, Batteries\}$
15-20	M	Tutor	Toronto	$\{Canon, Batteries, TV\}$
20-25	F	Teacher	Toronto	$\{iPad, Batteries, TV\}$
20-25	F	Tutor	Toronto	$\{Canon, Bluetooth, Batteries, TV\}$
20-25	F	Teacher	Toronto	$\{iPad, Batteries, TV\}$
30-35	F	Tutor	Vancouver	$\{iPad, Bluetooth, TV\}$
30-35	F	Cashier	Toronto	$\{iPad, Bluetooth, Batteries\}$
30-35	F	Tutor	Toronto	$\{Bluetooth, Batteries, TV\}$
30-35	F	Teacher	Toronto	$\{iPad, Bluetooth, Batteries, TV\}$

Table 1.1: MDB on sales data

1.1.3 Motivation for Utilizing a Data Cube

Table 1.1 is an example of a multidimensional database. It represents the transactions and the attributes of the customer for each transaction. Using data cube [10], if we want to find all the transactions from Toronto, it can be done by matching the query {Age Group = *, Sex = *, Job = *, Location = Toronto} where the *'s represent any values from that column. If we want all the transactions that were bought by females, it can be done by matching the query {Age Group = *, Sex = F, Job = *, Location = *}. Using on-line analytical processing (OLAP) we can make use of the *roll-up* and *drill-down* operations where we set more dimensions to *'s or less dimensions to *'s respectively to find the transactions associated to the matching query. In the event that we have a multi-level hierarchy, we collapse the concept hierarchies to form extra dimensions in the multidimensional database. For example, suppose we have a dimension location which has a concept hierarchy of country, province and city. We would collapse the concept hierarchy and expand the multidimensional database where we now have the extra three dimensions, country, province and city. Thus, it is logically sound to apply the data cube model to be able to uncover distinct patterns from different matching queries. The cube based structure also allows OLAP query answering techniques to be applied such as point queries (seeking a cell, sub-cube queries (seeking an entire group-by), and top-k queries (seeking k most relevant cells).

1.2 Challenges and Contributions

Mining distinct patterns on two datasets can be easily done by passing the datasets to any frequent pattern mining algorithm, retrieving the supports of the itemsets and determining if the absolute support ratio is above a given threshold. However, we present a problem where we would like to mine distinct patterns from a multidimensional database. A multidimensional database is where each dimension can be such attributes as age, sex, location, time, symptom and etc and the transactions associated to each tuple in the multidimensional database are items from that particular domain such as transactions of purchases of store items, symptoms of disease and etc. Table 1.1 shows an example of such a multidimensional database of purchases from an electronic store where each tuple shows the age, sex, job and location of the person that made each of the associated transactions.

Given a threshold, we wish to find all possible different combinations of dimensions such that the absolute support ratios of the frequent itemsets are greater than or equal to the threshold. In which case, those frequent itemsets would be distinct patterns for those particular sets of dimensions. The Example 1.2.1 using Table 1.1 illustrates the idea and demonstrates why the complexity of the baseline approach to the problem is exponential.

Example 1.2.1. A local electronics merchant has been advertising a set of products in the places he believes his target group to be frequent. However, he is puzzled the sales do not reflect on the amount of advertising, and so he has doubts about his strategy. He conducts a survey with all his customers. The results of the survey are shown in Table 1.1. The merchant wishes to find the itemsets that are most bought together and the groups associated to them. So using the results from the table he continuously groups the table into two distinct sets, one that matches a criteria on the attributes such as age 15-20 and male and the other on the rest that are not 15-20 and not male. From this table of 4 dimensions, there are $2^4 = 16$ possible ways of grouping: [(age), (age,sex), (age,sex,job), (age,sex,job,location), (age,sex), (age,job), (age,job,location), (age,location), (sex), (sex,job), (sex,job,location), (sex,location), (job), (job,location), (location), (entire database)]. However, this is not the total number of times he needs to run the pattern miner on the two distinct datasets. In each of those possible ways of grouping, he would need to group by on each of the particular attributes. For example when grouping by on age, he would need to group by on (15-20, 20-25, 30-35) and when doing (age,sex) he would need to group by on (15-20,M), (15-20,F), (20-25,M), (20-25,F), (30-35,M) and (30-35,F) and etc. He soon realizes that doing this

experiment manually could be quite time consuming and frustrating.

If we were to conduct the experiment from Example 1.2.1 on Table 1.1 using a threshold of 75%, we would find such distinct patterns as follows: itemset {Canon, Bluetooth} are bought together atleast 75% of the time by males that are 15-20, itemset {iPad, Batteries, TV} are bought together by females from Toronto, itemset {iPad, Bluetooth, Batteries} were bought together by people who work as cashiers and so forth. We can then use this information to advertise to the appropriate target group.

Another business model example is where a grocery store owner is concerned that a lot of his goods have gone bad before they are bought by customer. For this case, we can have a multidimensional database where dimensions are month, days of the week (e.g. Saturday, Sunday, Monday) and hour and the transactions associated to those purchases. From this we can determine the distinct patterns where for a given threshold like 70%, all the itemsets of goods that are bought and the month, day of the week, and the hour they are bought. Suppose for example customers bought for 70% of the time milk, bread, lettuce on Sunday's from 6pm - 8pm. He can then make sure that he orders those items so that they are freshly available for Sunday afternoons.

1.2.1 Our Contributions

In this thesis, we present a new type of pattern mining called distinct patterns. To tackle the distinct pattern mining problem, we propose a novel data model by integrating an efficient frequent pattern miner called DPMiner [15] with the traditional data cube [10]. Conceptually, the data cube is an extended database with aggregates on multiple levels and multiple dimensions [14]. It generalizes the group-by operator by precomputing and storing group-bys with regard to all possible combinations of dimensions. We use Bottom Up Cubing (BUC) as our cubing algorithm because this type of a cubing strategy allows pruning of descendant cells based on ancestor cells. The baseline approach would call DPMiner 2^n times the size of each group-by where n is the number of dimensions. As can be seen from the experiments, this exponential baseline algorithm is not feasible as the number of tuples and dimensions grow.

We propose several interesting pruning techniques to the baseline algorithm. We report an extensive empirical study using synthetic datasets to verify our advanced algorithm is much faster than the baseline, and in some cases even several orders of magnitude faster.

1.3 Organization of the Thesis

The rest of the thesis is organized as follows. In Chapter 2, we formulate the problem of distinct pattern mining using data cubes. In Chapter 3, we introduce the baseline algorithm and provide psuedocode to the baseline algorithm. In Chapter 4, we provide several effective pruning techniques to the baseline algorithm. In Chapter 5, we provide an extensive empirical study comparing the baseline algorithm to the advanced algorithm. Finally, in Chapter 6 we conclude the thesis.

Chapter 2

Problem Formulation

2.1 Multidimensional Databases

Consider an $(n+1)$-dimensional table defined as $MDB = (D_1, D_2, ..., D_n, \mathcal{T})$, where D_1, D_2,...,D_n are the dimensions and $\mathcal{T}$ is the field of transactions. A transaction is defined as $T \subseteq \mathcal{I}$ where $\mathcal{I}$ is the set of alphabets and $\mathcal{T} = 2^{\mathcal{I}}$. Consider the following example of a survey of customer data on the items that were purchased from a particular store.

Age Group	*Sex*	*Job*	*Location*	*Transaction*
15-20	M	Cashier	Toronto	$\{t_1, t_2, t_3, t_4\}$
15-20	M	Cashier	Toronto	$\{t_2, t_3, t_4\}$
15-20	M	Cashier	Vancouver	$\{t_1, t_3, t_4\}$
15-20	M	Tutor	Vancouver	$\{t_2, t_3, t_4\}$
15-20	M	Tutor	Toronto	$\{t_2, t_4, t_5\}$
20-25	F	Teacher	Toronto	$\{t_1, t_4, t_5\}$
20-25	F	Tutor	Toronto	$\{t_2, t_3, t_4, t_5\}$
20-25	F	Teacher	Toronto	$\{t_1, t_4, t_5\}$
30-35	F	Tutor	Vancouver	$\{t_1, t_3, t_5\}$
30-35	F	Cashier	Toronto	$\{t_1, t_3, t_4\}$
30-35	F	Tutor	Toronto	$\{t_3, t_4, t_5\}$
30-35	F	Teacher	Toronto	$\{t_1, t_3, t_4, t_5\}$

Table 2.1: MDB on sales data

For simplicity, we will use Table 2.2 as our running example, where each attribute value is represented by a simple symbol.

We define the following notations: For any tuple t in MDB, $t.D_i$ is the value of dimension

Row ID	*A*	*S*	*J*	*L*	*Transaction*
r_1	a_1	s_1	j_1	l_1	$\{t_1, t_2, t_3, t_4\}$
r_2	a_1	s_1	j_1	l_1	$\{t_2, t_3, t_4\}$
r_3	a_1	s_1	j_1	l_2	$\{t_1, t_3, t_4\}$
r_4	a_1	s_1	j_2	l_2	$\{t_2, t_3, t_4\}$
r_5	a_1	s_1	j_2	l_1	$\{t_2, t_4, t_5\}$
r_6	a_2	s_2	j_3	l_1	$\{t_1, t_4, t_5\}$
r_7	a_2	s_2	j_2	l_1	$\{t_2, t_3, t_4, t_5\}$
r_8	a_2	s_2	j_3	l_1	$\{t_1, t_4, t_5\}$
r_9	a_3	s_2	j_2	l_2	$\{t_1, t_3, t_5\}$
r_{10}	a_3	s_2	j_1	l_1	$\{t_1, t_3, t_4\}$
r_{11}	a_3	s_2	j_2	l_1	$\{t_3, t_4, t_5\}$
r_{12}	a_3	s_2	j_3	l_1	$\{t_1, t_3, t_4, t_5\}$

Table 2.2: Simplified MDB on the survey of sales

i where $i \leq n$, and $t.T$ is the transactions for tuple t.

2.2 Aggregate Cells

The concept of data cubes was introduced by Gray *et al.* [10] as a way to answer On-line Analytical Processing (OLAP) queries on a multidimensional database. An **aggregate cell** (or a **cell** for short) in a multidimensional database is defined as follows:

Definition 2.2.1. (Aggregate Cell). Given $x_1, x_2, ..., x_n$, where $x_i \in D_i \cup \{*\}$ and $*$ is a meta symbol meaning that attribute is generalized, and t be a tuple in MDB, cell $c = (x_1, x_2, ..., x_n)$ is defined as: $cell\_tuples = \{t | t.D_i = x_i, x_i \neq' *'\}$, and the complement of c is $\overline{c} = \{t | t \notin c\}$

An aggregate cell is a **group-by** in a relational database.

A **data cube** is the set of all possible aggregate cells on a given relational table. These aggregate cells can be organized into cuboids. A **cuboid** is a set of aggregate cells generalizing the same dimensions. We use the notations A, AB, ABC and so on to refer to cuboids (A,*,*,*), (A,B,*,*), (A,B,C,*), respectively. Data cubing is the process of constructing a data cube based on a multidimensional table which is referred to as the **base**

table. In data cubing, an aggregated measure is computed for each aggregate cell. Commonly used measures include *count, sum, min, max* represented in cell notation as cell $c = (x_1, x_2, ..., x_n) : measure$. There exists an ancestor-descendant relationship between cells. For two cells $c_1 = (x_1, x_2, ..., x_n)$ and $c_2 = (y_1, y_2, ..., y_n)$, if $c_1 \neq c_2$ and either $x_i = y_i$ or $x_i =' *'$, for each $1 \leq i \leq n$ then, c_1 is an ancestor of c_2 and c_2 is a descendant of c_1 denoted by $c_1 \succ c_2$. Particularly, c_2 is a child of c_1 and c_1 is a parent of c_2 if for exactly one dimension $x_i =' *'$ and $y_i \neq' *'$.

Example 2.2.1. Using Table 2.2, consider the cells $a = \{a_1, *, *, *\}$, $b = \{a_1, *, j_1, *\}$, $c = \{a_1, *, j_1, l_1\}$ and $d = \{a_1, s_1, j_1, l_1\}$. Then we have the relations $a \succ d$, c, and b; $b \succ d$ and c; $c \succ d$. Further c is a parent of d and d is a child of c; b is a parent of c and c is a child of b; a is a parent of b and b is a child of a;

Figure 2.1 represents a lattice of cuboids. A data cube is represented as a lattice of cuboids where each cuboid represents a different group-by. Notice however that this data cube is bottom up where the apex cell is at the bottom and the base cell is at the top. The reason for which we will explain later in this chapter.

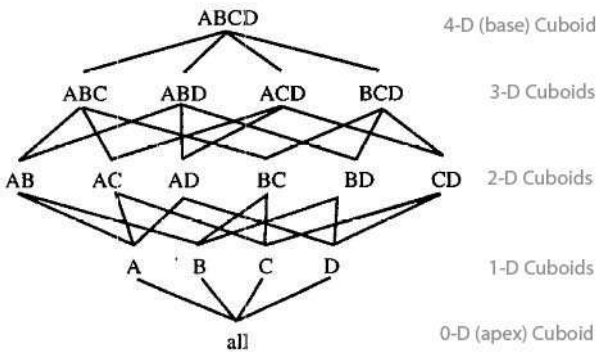

Figure 2.1: Lattice of Cuboids [4].

The base cuboid is the aggregate cells that do *not* generalize any dimensions. The **apex** is the most generalized cuboid, commonly represented as ALL. It contains one value, the aggregated measure M for all tuples stored in the base cuboid. A cell from the base cuboid is known as the base cell. Given an n-dimensional data cube. Let $c = (x_1, x_2, ..., x_n)$ represent a particular cell in the data cube. We say a cell $c = (x_1, x_2, ..., x_n)$ is an m-dimensional cell

if for $m \leq n$, m values of $x_1, x_2, ..., x_n$ are not '*'. If $m = n$, c represents a cell from the base cuboid.

Example 2.2.2. Using Table 2.2, Let us consider the cuboid (A,*,*,*). The cuboid (A,*,*,*) contains the cells: (a_1,*,*,*), (a_2,*,*,*), (a_3,*,*,*). The cells (a_1,*,*,*) and (*,*,s_1,*) are examples of 1-D cells, (*,s_1,j_1,*), (*,*,j_2,l_2) are 2-D cells, (a_2,*,j_3,l_1) is a 3-D cell and (a_3,s_2,j_3,l_1) is a 4-D cell. Here the 1-D, 2-D, and 3-D cells are aggregate cells whereas the 4-D cells are base cells.

The aggregate measure for each group-by is defined as follows:

Definition 2.2.2. (Aggregate Function). Given an MDB and a cell c, our aggregate function $F(c)$ is defined as:

$$F(c) = \langle PC, NC \rangle ,$$

where $PC = \{t.T \mid t \in c \}$ and $NC = \{t.T \mid t \in \overline{c} \}$. PC is the positive set of transactions and NC is the negative set of transactions.

We use the following notations to represent the measures of a cell: Given a cell $c = (a_1, s_1, *, *) :< PC, NC >$. $c.PC$ represents the PC set from the aggregate function $F(c)$ and $c.NC$ represents the NC set from the aggregate function $F(c)$

Example 2.2.3. Using Table 2.2. Suppose cell $c = (a_1, *, *, *)$. The measures of cell c are defined by the aggregate function $F(c)$ such that $c.PC = \{t.T | c.D_1 = a_1, t \in c\}$ and $c.NC = \{t.T | c.D_1 \neq a_1, t \in c\}$. Thus the set of transactions for $c.PC$ is: { {t_1,t_2,t_3,t_4}, {t_2,t_3,t_4}, {t_1,t_3,t_4}, {t_2,t_3,t_4}, {t_2,t_4,t_5} } and $c.NC$ is: { {t_1,t_4,t_5}, {t_2,t_3,t_4,t_5}, {t_1,t_4,t_5}, {t_1,t_3,t_5}, {t_1,t_3,t_4}, {t_3,t_4,t_5}, {t_1,t_3,t_4,t_5} } . Thus cell c is defined as c = (a_1,*,*,*): ⟨ { {t_1,t_2,t_3,t_4}, {t_2,t_3,t_4}, {t_1,t_3,t_4}, {t_2,t_3,t_4}, {t_2,t_4,t_5} }, { {t_1,t_4,t_5}, {t_2,t_3,t_4,t_5}, {t_1,t_4,t_5}, {t_1,t_3,t_5}, {t_1,t_3,t_4}, {t_3,t_4,t_5}, {t_1,t_3,t_4,t_5} } ⟩)

We now have the following property.

Property 2.2.1. Given a cell, $c.NC = \overline{c}.PC$, and $c.PC = \overline{c}.NC$

2.3 Distinct Patterns

Given the set of transactions T and an itemset X, the absolute support of X is defined as:

$$Sup_T(X) = |\{t | t \in T, X \subseteq t\}|$$

Definition 2.3.1. (Distinctiveness Ratio) Given an itemset $X \subseteq \mathcal{I}$, and set of transactions D_1 and D_2 where $D_1, D_2 \subseteq \mathcal{T}$. The Distinctiveness Ratio (DR) of X from D_2 to D_1 defined as:

$$DR(X) = \begin{cases} 0 & \text{if } Sup_{D_1}(X),\ Sup_{D_2}(X) = 0 \\ \infty & \text{if } Sup_{D_1}(X) = 0 \text{ and } Sup_{D_2}(X) \neq 0 \\ \frac{Sup_{D_2}(X)}{Sup_{D_1}(X)} & \text{Otherwise} \end{cases}$$

For a predefined *distinctiveness threshold* $\rho > 0$, a **Distinct Pattern** is defined as $DR(X) \geq \rho$.

The concept of Emerging Patterns (EP) was defined by Dong *et al.* [6]. An Emerging Pattern is defined as the itemsets whose relative support increases significantly from one background dataset D_1 to a target dataset D_2 [6]. The growth rate $GR(X)$ defined by Dong *et al.* is the rate of increase of itemset X from D_1 to D_2. The difference between $GR(X)$ and $DR(X)$ is that growth rate uses relative support defined as $Sup_{D_i}(X) = |t|t \in D_i.T|/|D_i|$ where as $DR(X)$ uses absolute support.

The difference between Distinct Pattern and Emerging Pattern is that Distinct Pattern uses the absolute support ratio of itemset X from D_2 to D_1 where as Emerging Pattern uses the increase in growth rate of itemset X from D_1 to D_2. Thus we cannot use the Emerging Patterns to compute Distinct Pattern.

Example 2.3.1. (Itemset X an EP but not a DP)
Using Table 2.2. Consider the following example where we show that an itemset X can be an EP but not be a DP in the same datasets with the same threshold. Consider the cell $c = (a_2, *, *, *) :< PC, NC >$ where $c.PC$ = { {t_1,t_4,t_5}, {t_2,t_3,t_4,t_5}, {t_1,t_4,t_5} } and $c.NC$ = { {t_1,t_2,t_3,t_4}, {t_2,t_3,t_4}, {t_1,t_3,t_4}, {t_2,t_3,t_4}, {t_2,t_4,t_5}, {t_1,t_3,t_5}, {t_1,t_3,t_4}, {t_3,t_4,t_5}, {t_1,t_3,t_4,t_5} }. Let $\rho = 3$ for both EP and DP. Suppose we have itemset $X = \{t_1, t_4, t_5\}$. We are testing to see if DR(X) ≥ 3, that is if X occurs 75% or more in PC or GR(X) > 3 that is X is an EP from NC to PC with a growth rate of 3. We will discuss later in this chapter how to compute threshold from percentage.
Distinct Pattern:

$$Sup_{D_{PC}}(X) = 2,\ Sup_{D_{NC}}(X) = 1$$
$$DR(X) = \tfrac{2}{1} = 2 < \rho$$

Emerging Pattern:

$$Sup_{D_{PC}}(X) = \frac{2}{3},\ Sup_{D_{NC}}(X) = \frac{1}{9}$$
$$GR(X) = \frac{2}{3} / \frac{1}{9} = 6 > \rho$$

As we can see, X is an EP from $\overline{c}$ to c with $GR(X) = 6 > 3$. However, the same itemset X is not a distinct pattern from $\overline{c}$ to c with a $DR(X) = 2 < 3$.

Example 2.3.2. (Itemset X a DP but not an EP)
Using Table 2.2. Consider opposite of the previous example where we now show that an itemset X can be a DP but not an EP. Consider the cell $c = (*, *, *, l_1) :< PC, NC >$ where $c.PC$ = { {t_1,t_2,t_3,t_4}, {t_2,t_3,t_4}, {t_2,t_4,t_5}, {t_1,t_4,t_5}, {t_2,t_3,t_4,t_5}, {t_1,t_4,t_5}, {t_1,t_3,t_4}, {t_3,t_4,t_5}, {t_1,t_3,t_4,t_5} } and $c.NC$ = { {t_1,t_3,t_4}, {t_2,t_3,t_4}, {t_1,t_3,t_5} }. Let $\rho = 3$ for both EP and DP. Suppose we have itemset $X = \{t_3, t_4\}$. We are testing to see if the DR(X) ≥ 3 or GR(X) > 3.
Distinct Pattern:

$$Sup_{D_{PC}}(X) = 6,\ Sup_{D_{NC}}(X) = 2$$
$$DR(X) = \frac{6}{2} = 3$$

Emerging Pattern:

$$Sup_{D_{PC}}(X) = \frac{6}{9},\ Sup_{D_{NC}}(X) = \frac{2}{3}$$
$$GR(X) = \frac{6}{9} / \frac{2}{3} = 1 < \rho$$

As we can see, 75% of the occurrences of itemset X in MDB occurs in cell c and so X is a distinct pattern of c. However, $GR(X) = 1 < \rho$ and so X is not an EP from $\overline{c}$ to c.

From the previous two examples, we can see that an emerging pattern cannot be used to find a distinct pattern and conversely a distinct pattern cannot be used to find an emerging pattern.

2.4 The Concentrate Cell Problem

A cell is a minimal cell of itemset X if X is not a distinct pattern in any descendant of the cell. We state this formally in the following theorem:

Theorem 1. *(Minimal Cell)*
Given an itemset X and a cell c where $Sup_c(X) = \rho$ and $Sup_{\overline{c}}(X) = 0$. For any c' such that $c \succ c'$ and any $X' \supseteq X$, if $Sup_{\overline{c'}}(X') > 0$ then X' is not a distinct pattern in c' and its descendants.

Proof.
Let $TotalSupport(X) = Sup_c(X) + Sup_{\overline{c}}(X)$ and $TotalSupport(X') = Sup_{c'}(X') + Sup_{\overline{c'}}(X')$. If X' is a distinct pattern in c' and its descendants then the following must hold:

$$TotalSupport(X) = TotalSupport(X') \tag{2.1}$$

Assume for contradiction that X' is a distinct pattern of c'. Thus TotalSupport(X) = TotalSupport(X'). However we know that $Sup_{\overline{c'}}(X') > 0$. In order for Equation 2.1 to hold, $Sup_{c'}(X') < \rho$. However, for X to be a distinct pattern in cell c, $Sup_{c'}(X') > \rho$. Thus we have a contradiction where we need to have $Sup_{c'}(X') \geq \rho$ in order for X to be a DP but we know that $Sup_{c'}(X') < \rho$ since $Sup_{\overline{c'}}(X') > 0$. Therefore, Equation 2.1 cannot hold and thus TotalSupport(X) $\neq$ TotalSupport(X'). Contradiction. Hence X' is not a distinct pattern of c' if $Sup_c(X) = \rho$, $Sup_{\overline{c}}(X) = 0$ and $Sup_{\overline{c'}}(X') > 0$. □

Based on Theorem 1, we formally define a minimal cell as follows:

Definition 2.4.1. (Minimal Cell). Given an itemset X and a cell c, c is called a minimal cell of X if and only if: (1) X is a DP in c, and (2) X is not a DP in any descendant of c.

The following example illustrates the definition of a minimal cell.

Example 2.4.1. Using Table 2.2. Consider the cells $c_1 = (a_1, s_1, *, *) :< PC, NC >$ and $c_2 = (a_1, s_1, *, l_1) :< PC, NC >$ where $c_1 \succ c_2$. Let itemset $X = \{t_2, t_3, t_4\}$. Given $\rho = 3$, we can compute the DR(X) from c_1 to $\overline{c_1}$ and DR(X) from c_2 to $\overline{c_2}$ as follows:
Distinctiveness ratio from c_1 to $\overline{c_1}$:

$$Sup_{D_{c_1.PC}}(X) = 3,\ Sup_{D_{c_1.NC}}(X) = 1$$
$$DR(X) = \tfrac{3}{1} = 3$$

Thus X is a DP from c_1 to $\overline{c_1}$.
Distinctiveness ratio from c_2 to $\overline{c_2}$:

$$Sup_{D_{c_2.PC}}(X) = 2,\ Sup_{D_{c_2.NC}}(X) = 2$$
$$DR(X) = \tfrac{2}{2} = 1$$

Thus X is not a DP from c_2 to $\overline{c_2}$.
Since the descendant of c_1 does not satisfy DR(X) condition, c_1 is considered the minimal cell of itemset X where $DR(X) \geq \rho$.

Definition 2.4.2. (Concentrate Ratio). For any itemset X and cell c. The *Concentrate Ratio* of X in c is defined as:

$$CR(X) = \frac{Sup_c(X)}{Sup_{\overline{c}}(X) + Sup_c(X)} \tag{2.2}$$

Given a user specified **concentrate threshold** α, c is a **Concentrate Cell** of X if $CR(X) \geq \alpha$

We now transform the concentrate cell mining problem into the distinct patterns problem.

Theorem 2. *(Concentrate Cell-Distinct Pattern). For any itemset X and cell c. C is a concentrate cell for X with respect to concentrate threshold α if and only if X is a distinct pattern in cell c with respect to distinctiveness threshold ρ where:*

$$\rho = \frac{\alpha}{1-\alpha} \tag{2.3}$$

Proof. (1). Assume c is a concentrate cell of X. That means:

$$\frac{Sup_c(X)}{Sup_{\overline{c}}(X) + Sup_c(X)} \geq \alpha$$

Then, the distinctiveness ratio (DR) of X in c is:

$$\begin{aligned}
\frac{Sup_c(X)}{Sup_{\overline{c}}(X)} &\geq \frac{\alpha}{1-\alpha} \\
Sup_c(X) - \alpha Sup_c(X) &\geq \alpha Sup_{\overline{c}}(X) \\
Sup_c(X) &\geq \alpha Sup_{\overline{c}}(X) + \alpha Sup_c(X) \\
Sup_c(X) &\geq \alpha(Sup_{\overline{c}}(X) + Sup_c(X)) \\
\frac{Sup_c(X)}{Sup_{\overline{c}}(X) + Sup_c(X)} &\geq \alpha
\end{aligned}$$

Thus, if c is a concentrate cell of X with respect to α, then X is a distinct pattern of c where $DR(X) \geq \rho$, where ρ defined by equation 2.3.

(2). Assume X is a distinct pattern of cell c. That means:

$$\frac{Sup_c(X)}{Sup_{\overline{c}}(X)} \geq \rho$$

Then, the Concentrate Ratio(CR) of X in cell c is:

$$\frac{Sup_c(X)}{Sup_{\overline{c}}(X) + Sup_c(X)} \geq \frac{\rho}{1+\rho},$$

Where α was derived by rearranging equation 2.3 as follows:

$$\begin{aligned} \rho &= \frac{\alpha}{1-\alpha} \\ \rho - \rho\alpha &= \alpha \\ \rho &= \alpha + \rho\alpha \\ \rho &= \alpha(1+\rho) \\ \alpha &= \frac{\rho}{1+\rho} \end{aligned}$$

Thus we have:

$$\begin{aligned} \frac{Sup_c(X)}{Sup_{\overline{c}}(X) + Sup_c(X)} &\geq \frac{\rho}{1+\rho}, \\ Sup_c(X) - \rho Sup_c(X) &\geq \rho Sup_c(X) - \rho Sup_{\overline{c}}(X) \\ \frac{Sup_c(X)}{Sup_{\overline{c}}(X)} &\geq \rho \end{aligned}$$

Thus, if X is a distinct pattern of c with respect to ρ, then c is a concentrate cell of X where $CR(X) \geq \alpha$. □

Using Theorem 2, an end user can ask such queries as: What are the concentrate cells where 75% of itemset X belong to those cells? The following example shows how we can answer such a query.

Example 2.4.2. Given the query with a concentrate threshold of 75%, we would compute ρ as follows:

$$\begin{aligned} \alpha &= 75/100 = .75 \\ \rho &= \frac{\alpha}{1-\alpha} = \frac{.75}{.25} = 3 \end{aligned}$$

Thus for itemset X and cell c, if $DR(X) \geq 3$ then 75% of itemset X from MDB occurs in cell c. In addition if $DR(X)$ of descendants of c is $< \rho$, then c is the minimal cell for itemset X.

2.5 Cubing Method

The time and space cost of cubing grows exponentially with respect to dimensionality. Given a table with n dimensions, up to 2^n group-bys are needed. Thus, scalability must be kept in mind when choosing the appropriate cubing algorithm.

There are three types of approaches to cubing in terms of the order the cells are materialized: top-down, bottom-up and hybrid. The top-down approaches such as the Multiway Array Aggregation construct the cube from the base cells towards the apex. The bottom-up approaches such as the BUC compute the cells from the apex towards the base. There are other methods that combine both the top-down and the bottom-up approach to construct the data cube such as the Star-Cubing.

We have used the Bottom Up Cubing (BUC) algorithm. BUC traverses each cuboid from the apex towards the base. We will describe BUC in detail in the next chapter. BUC was chosen because it allows pruning during the construction of the cubes using the Apriori property. The Apriori property states that all nonempty subsets of a frequent itemset must also be frequent [2]. The difference between BUC and Apriori is that the latter uses breath-first search to generate its candidates whereas BUC uses depth-first search. For example, Apriori would first compute all 1-D aggregate cells in one pass and prune based on those cells. Where as BUC would compute cuboids A, AB, ABC and ABCD where it would prune AB based on A and ABC based on AB and so on. The problem with using Apriori is the candidate set often cannot fit in the memory as little pruning is expected during the first few passes of the input [4]. In either case, the cells that do not pass a certain threshold are pruned out. In our case we prune out cell c where $|c.PC| <$ minimum support threshold. The partially materialized cubes are referred to as iceberg cubes. Because tuples are shared while traversing the tree, the aggregate function $F(c)$ holds the anti-monotonic property; that is, if the aggregate value on a cell does not satisfy the iceberg condition, then all of the cells descending from this cell cannot satisfy the iceberg condition either. Refer to example 2.5.1. These cells and all of their descendant cells can be pruned out because descendant cells, by definition are more specialized than their parent, as they are partitioned based on more dimensions values than their parents. Thus if the aggregate value of a parent cell fails the iceberg condition, then all its children must also fail the condition. We can formally state this in a property as follows:

Property 2.5.1. Let cell X be a child of cell Y. Let $P = X.PC$ and $Q = Y.PC$. The

following property holds:

$$P \subseteq Q$$

Example 2.5.1. Using Table 2.2. Consider the following two cells
$Y = (a_1,*,*,*)$: $\langle$ { $\{t_1,t_2,t_3,t_4\}$, $\{t_2,t_3,t_4\}$, $\{t_1,t_3,t_4\}$, $\{t_2,t_3,t_4\}$, $\{t_2,t_4,t_5\}$ }, { $\{t_1,t_4,t_5\}$, $\{t_2,t_3,t_4,t_5\}$, $\{t_1,t_4,t_5\}$, $\{t_1,t_3,t_5\}$, $\{t_1,t_3,t_4\}$, $\{t_3,t_4,t_5\}$, $\{t_1,t_3,t_4,t_5\}$ } $\rangle$)
$X = (a_1,*,j_1,*)$: $\langle$ { $\{t_1,t_2,t_3,t_4\}$, $\{t_2,t_3,t_4\}$, $\{t_1,t_3,t_4\}$ }, $\{t_2,t_3,t_4\}$, $\{t_2,t_4,t_5\}$, { $\{t_1,t_4,t_5\}$, $\{t_2,t_3,t_4,t_5\}$, $\{t_1,t_4,t_5\}$, $\{t_1,t_3,t_5\}$, $\{t_1,t_3,t_4\}$, $\{t_3,t_4,t_5\}$, $\{t_1,t_3,t_4,t_5\}$ } $\rangle$). X is a child of Y. The set of transactions $Y.PC$ is $Q =$ { $\{t_1,t_2,t_3,t_4\}$, $\{t_2,t_3,t_4\}$, $\{t_1,t_3,t_4\}$, $\{t_2,t_3,t_4\}$, $\{t_2,t_4,t_5\}$ } and $X.PC$ is $P =$ { $\{t_1,t_2,t_3,t_4\}$, $\{t_2,t_3,t_4\}$, $\{t_1,t_3,t_4\}$ }. Thus, $P \subseteq Q$. Suppose minimum support for the iceberg condition was set to 5, the parent Y was pruned out as $|P| < 5$ and so based on anti-monotonicity the child cell will also be pruned out as $|Q| < 5$.

Chapter 3

Related Work and Baseline Algorithm

In this chapter we review the related work and the baseline algorithm[20]. Particularly, in Section 3.1 we will discuss the BUC algorithm. Section 3.2 introduces Frequent Pattern Mining. In Section 3.3 we discuss the DPMiner algorithm. Section 3.4 shows the implementation [20] with some slight modifications. Finally, Section 3.5 shows how we can use DPMiner to extract distinct patterns.

3.1 Bottom Up Cubing

Bottom up cubing (BUC) constructs the data cube from the apex cuboid to the base cuboid. Figure 3.1 shows the BUC process tree. The numbers on the vertices indicate the order in which BUC materializes the aggregate cells.

In this section we will demonstrate the BUC algorithm by example and in Section 3.4 we will explain the BUC-DPMiner integrated algorithm. Let us see a simple example on a table with dimensions A, B, C and D. Suppose dimension A contains four distinct values, a_1, a_2, a_3, a_4; dimension B contains four distinct values, b_1, b_2, b_3, b_4; dimension C contains two distinct values, c_1, c_2, and dimension D contains two distinct values d_1, d_2. We must group-by based on every possible combination of dimensions. For simplicity we will use count as the aggregate function and set 1 as the minimum support. That is, if the cell count of the group-by is less than 1, then that group-by and any descendant group-by can

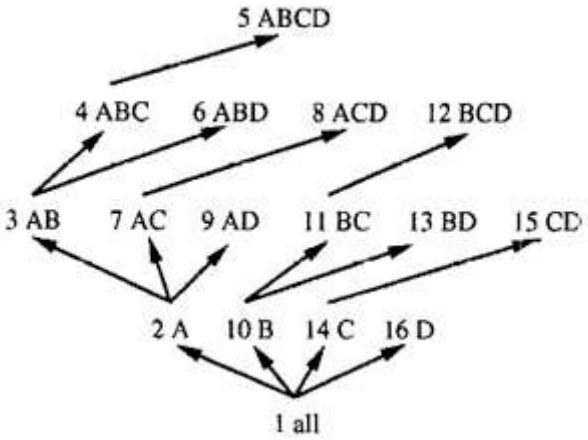

Figure 3.1: BUC Process Tree [4].

be pruned out based on the anti-monotonicity property. Figure 3.2 illustrates the group-bys during the first four calls to BUC.

As can be seen from the BUC process tree in Figure 3.1. BUC first materializes the most generalized group-by. Next, we group-by on dimension A producing the four cells $(a_1, *, *, *)$,$(a_2, *, *, *)$,$(a_3, *, *, *)$, and $(a_4, *, *, *)$. If cell $(a_1, *, *, *)$ passes the minimum threshold, we recurse on the cell (a_1,*,*,*) and group-by on dimension B of the tuples of the cell $(a_1, *, *, *)$. If count of $(a_1, b_1, *, *)$ satisfies the iceberg condition then output the aggregated measure to the cell $(a_1, b_1, *, *)$ and recurse on $(a_1, b_1, *, *)$ and group-by on dimension C of the tuples of the cell $(a_1, b_1, *, *)$. If the cell count of $(a_1, b_1, c_1, *)$ does not satisfy the minimum threshold, then all descendant group-bys are pruned out. That is the cells (a_1, b_1, c_1, d_1) and (a_1, b_1, c_1, d_2) are not materialized. BUC then backtracks to the $(a_1, b_1, *, *)$ group-by and recurses on $(a_1, b_1, c_2, *)$, and so on. BUC saves a great deal of processing time by using the iceberg condition to prune out cells that do not meet the minimum support.

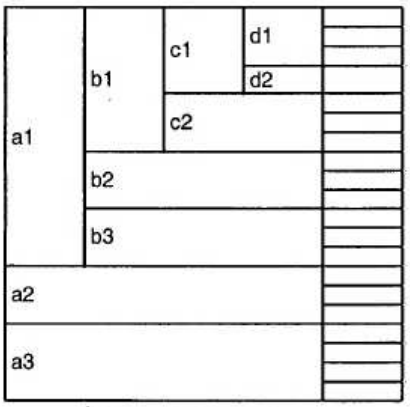

Figure 3.2: BUC Partitioning [4].

3.2 Frequent Pattern Mining

Frequent patterns are patterns (sets of items, sequence, etc.) that occur frequently in a database [1]. The supports of the frequent patterns must be greater than a predefined support threshold.

A naïve approach to finding frequent patterns would be to test the supports of $2^n - 1$ possible itemsets where n is the number of items in the dataset against a large database. For example, a transaction with 20 items needs to test the support of $2^{20} - 1 = 1,048,575$ itemsets. Thus reducing the number of itemsets that are needed to be checked and checking the supports of only a selected itemsets efficiently would optimize the algorithm greatly. Frequent pattern mining has been studied extensively. Apriori [2] and FP-Growth [12] are two notable algorithms for frequent pattern mining.

Apriori[2] uses breath-first search to traverse the lattice of itemsets as shown in Figure 3.3. Each vertex represents an item in the dataset. Frequent subsets are extended one item at a time, a process known as *candidate generation*. The support of each individual itemset is calculated separately. Any itemset that does not meet the minimum support threshold is pruned out. This is based on the property that any superset of an infrequent itemset must also be infrequent [2]. Some of the major costs of the Apriori algorithm are: huge number of candidate generation, multiple scans of the database, tedious support counting of candidates.

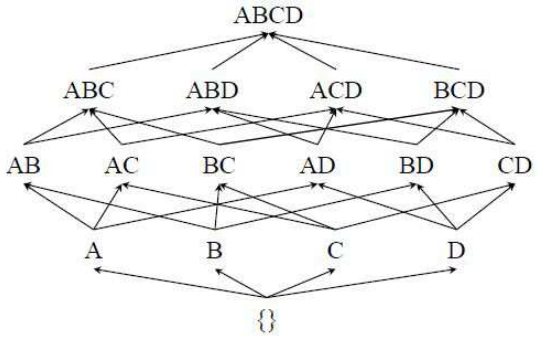

Figure 3.3: Lattice of Itemsets [21].

FP-Growth, on the other hand, is a depth-first search algorithm that addresses the limitations of Apriori. The first scan of the database finds all of the frequent items, ranks them in frequency descending order, and puts them into a header table. It then compresses the database into a prefix tree called FP-tree. For example, given a transaction database in Table 3.1 [23], and a minimum support of 3, FP-Growth would build the FP-tree shown in Figure 3.4.

TID	Items	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, a, c, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

Table 3.1: An sample transaction database [23].

The first scan of Table 3.1 derives the list of frequent items, $\langle(f : 4), (c : 4), (a : 3), (b : 3), (m : 3), (p : 3)\rangle$, where the number after":" indicates the support. As can be seen, any item with a support less than 3 is excluded from this list. Second, the FP-tree is constructed by scanning the transaction database again. The root of the tree is labeled with "*null*". Scanning the first transaction creates the first branch of the tree: $\langle(f : 1), (c : 1), (a : 1), (b : 1), (m : 1), (p : 1)\rangle$. When scanning the second transaction, the ordered frequent item list $\langle f, c, a, b, m\rangle$ shares the common prefix $\langle f, c, a\rangle$ with the existing $\langle f, c, a, m, p\rangle$. Thus increment the count of each of the shared items, create a new node

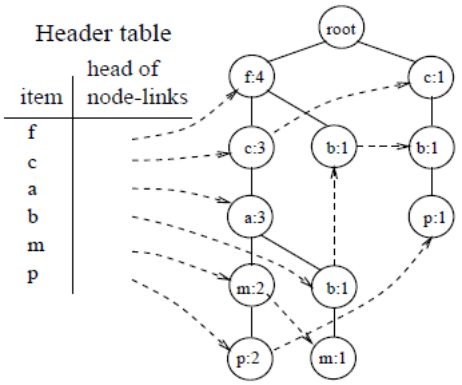

Figure 3.4: A example of FP-tree based on Table 3.1 [23].

for the remaining items and link each item back to the previous item in the list. This process is repeated for each transaction in the database. The benefits of FP-tree are as follows: long patterns in any transaction remain unbroken; complete information preserved for frequent pattern mining without having to scan the database anymore, all infrequent items are removed; the more frequent an item is, the more likely to be shared, and is never larger than the original database, not counting the node-links and the count fields. The FP-Growth algorithm defined by Pei *et al.*[23] is as follows:

- For each frequent item, construct its projected database, and then its projected FP-tree
- Repeat the process on the newly created projected FP-tree until the resulting FP-tree is empty, or contains only one path.

Next, we define two special types of frequent patterns: the closed frequent patterns and frequent generators.

Definition 3.2.1. (Closed Pattern and Generator) An itemset X is *closed* in dataset $\mathcal{D}$ if there exists no proper super-itemset Y such that $X \subset Y$ and $support(X) = support(Y)$ in $\mathcal{D}$. X is a *closed frequent pattern* in $\mathcal{D}$ if it is both closed and frequent in $\mathcal{D}$ [11].

An itemset Z is a *generator* in $\mathcal{D}$ if there exists no proper sub-itemset Z' such that $Z' \subseteq Z$ and $support(Z') = support(Z)$ [18].

An equivalent class EC is "a set of itemsets that always occur together in some transactions of dataset $\mathcal{D}$". That is, for all $X \in EC$ and $Y \in EC$, $f_D(X) = f_D(Y)$, where $f_D(Z) = \{T \in D | Z \subseteq T\}$ [15]. Frequent equivalence classes can be uniquely represented by a set of generators $\mathcal{G}$ and their associated closed frequent patterns $\mathcal{C}$, in the form of $EC = [\mathcal{G}, \mathcal{C}]$.

3.3 DPMiner Algorithm

DPMiner stands for Discriminative Pattern Miner. It finds closed frequent patterns and frequent generators simultaneously to form equivalent classes. Given a dataset $\mathcal{D}$, suppose $\mathcal{D}$ can be divided into various disjoint classes, denoted by $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup ... \cup \mathcal{D}_n$. Let δ be a small number (usually 1 or 2), θ be the minimum support, EC be a frequent equivalence class of D, and C be a closed pattern of EC. An equivalent class EC is a *δ-discriminative* equivalent class provided that its closed pattern $\mathcal{C}$'s support is greater than θ in $\mathcal{D}_i$ but less than δ in $\mathcal{D} - \mathcal{D}_i$ where $i \leq n$. Furthermore, EC is a *non-redundant δ-discriminative* equivalent class if and only if (1) it is δ-discriminative, (2) there exists no $\widehat{EC}$ such that $\widehat{\mathcal{C}} \subseteq \mathcal{C}$, where $\widehat{\mathcal{C}}$ and $\mathcal{C}$ are the closed patterns of $\widehat{EC}$ and EC respectively.

Data Structures and Computational Steps of The DPMiner

DPMiner uses a modified FP-tree structure. Normal FP-Tree stores all frequent items sorted by descending frequency in the header table. DPMiner however does not store any frequent items with a *full support*. That is, those items that appear in every transaction of the original database or conditional projected databases must be removed from the header table of FP-trees. The anti-monotonic property of generators states that for X as a frequent itemset of $\mathcal{D}$, X is a generator if and only if $sup(X, \mathcal{D}) < sup(Y, \mathcal{D})$ for every immediate proper subset Y of X. Thus the reason for such a FP-tree modification is that those items and the itemsets containing them are not generators due to the anti-monotonic property of generators. This modification often leads to much smaller tree structures [15].

Given k non-empty classes of transactions $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k$, a minimal support threshold θ and a maximal threshold δ, the method to discover equivalence classes for the k classes of transactions consists of the following 5 steps:

1. Let $\mathcal{D} = \bigcup_{i=1}^{k} \mathcal{D}_i$.

2. Construct a FP-tree based on $\mathcal{D}$ and run a depth-first search on the tree to find frequent generators and closed patterns simultaneously. For each search path along the tree, the search terminates whenever a δ-discriminative equivalent class is reached.

3. For every frequent closed pattern X, determine the class label distribution. That is, find the class where a closed pattern has the highest support. This step is necessary because patterns are not mined separately for each $\mathcal{D}_i$ $(1 \leq i \leq k)$, but rather on the entire $\mathcal{D}$.

4. Pair generators and closed frequent patterns to form δ-discriminative equivalent classes.

5. Output the non-redundant δ-discriminative equivalent classes where $EC = [\mathcal{G}, \mathcal{C}]$.

3.4 Implementation

The following is the pseudocode for the baseline algorithm [20]. It carries out a depth-first search to build data cubes and mine δ-discriminative patterns simultaneously.

Algorithm

```
Procedure ButtomUpCubeWithDPMiner(data, dim, theta, delta)
Inputs:
  data:  the dataset upon which we build our integrated model.
  dim:   number of standard dimensions in input data.
  theta: the minimal support threshold of candidate patterns
         in the target class.
  delta: the maximal support threshold of candidate patterns
         in the background class.
Outputs:
  cells with their measures (patterns)
Method:
  1: if (data.count == 1) then
  2:   writeAncestors(data, dim);
  3:   return;
  4: endif
  5: for each dimension d (from 0 to (dim - 1)) do
  6:   C := cardinality(d);
```

```
 7:   newData := partition(data, d); // counting sort.
 8:   for each partition i (from 0 to (C - 1)) do
 9:     cell := createEmptyCell();
10:     posData := newData.gatherPositiveTransactions();
11:     negData := newData.gatherNegativeTransactions();
12:     isDuplicate := determineCoverage(posData, negData);
13:     if (!isDuplicate) then
14:       cell.measure := DPMiner(posData, negData, theta, delta);
15:       writeOutput(cell);
16:       subData := newData.getPartition(i);
17:       ButtomUpCubeWithDPMiner(subData, d+1, theta, delta);
18:     endif
19:   endfor
20: endfor
```

The Execution Flow

Let us illustrate the execution flow using a simple example on a multidimensional database MDB with dimensions A, B, C and D. Suppose dimension A contains four distinct values, a_1, a_2, a_3, a_4; dimension B contains four distinct values, b_1, b_2, b_3, b_4; dimension C contains two distinct values, c_1, c_2, and dimension D contains two distinct values d_1, d_2. Lines 1-4 are optimizations and we will discuss them later in this section.

To begin, we loop through each of the dimensions of MDB (line 5). We then determine the cardinality of the dimension we are currently traversing (line 6). In the first iteration we will determine the cardinality of dimension A to be four. We then group-by on dimension A to produce the four aggregate cells $(a_1, *, *, *)$, $(a_2, *, *, *)$, $(a_3, *, *, *)$ and $(a_4, *, *, *)$ (line 7). Each group-by is sorted linearly using the counting sort algorithm. It was found that the use of CountingSort to be an important optimization to BUC [4].

The algorithm then iterates over each of the aggregate cells to mine patterns for the tuples belonging to the cell (line 8). It starts with cell $(a_1, *, *, *)$ and gathers the tuples with a_1 on dimension A as the Positive Class (PC) (line 10) and collects the remaining in a Negative Class (NC) (line 11). Line 12 is another optimization which we will discuss later. We then pass both the PC and NC classes to the DPMiner algorithm (line 14) which finds the δ-discriminative patterns.

BUC is called recursively on the current partition to materialize cells, mine patterns and

output them. The algorithm further group-bys on dimension B of the tuples belonging to the cell $(a_1, *, *, *)$ to materialize and mine patterns from the descendant cell $(a_1, b_1, *, *)$. The algorithm continues and further recurses on the descendant cells $(a_1, b_1, c_1, *)$ and (a_1, b_1, c_1, d_1). Upon reaching the base cells, the algorithm backtracks to the nearest ancestor cell, $(a_1, b_1, c_2, *)$. It continues in the same processing order as per Figure 3.1.

Optimizations

Line 12 is an optimization aimed at avoiding producing cells with identical tuples and patterns. For example, the cells $(a_2, *, *, *)$ and $(a_2, b_2, *, *)$ may contain the exact same aggregated tuples. Since we have already computed the ancestor cell $(a_2, *, *, *)$ which contains the exact same tuples and therefore the exact same PC and NC classes. It is redundant to process the descendant cell $(a_2, b_2, *, *)$ and to pass on to DPMiner to find the same δ-discriminative patterns as cell $(a_2, *, *, *)$. Thus the duplicate checking removes these kinds of redundant work to find the exact same patterns.

The above duplicate checking function generalizes the original BUC [4] optimization called *writeAncestors* (lines 1- 4). The above duplicate checking also includes *writeAncestors* with slight modifications, as a special case of the duplicate checking. Suppose we are processing the cell $(a_2, *, *, *)$ and this cell contains only one tuple. As per the discussion above, the descendant cells $(a_2, b_2, *, *)$, $(a_2, b_2, c_2, *)$, and (a_2, b_2, c_2, d_2) will also contain exactly the same tuple as the ancestor cell $(a_2, *, *, *)$ and hence the same patterns. These four cells actually form an equivalent class. The baseline algorithm outputs the lower bound cell $(a_2, *, *, *)$ together with the upper bound cell (a_2, b_2, c_2, d_2) and skips all intermediate cells in this equivalent class.

Both optimization techniques shorten the running time and reduce the number of output cells. Experiments conducted in [4] found out that in real-life data warehouses, about 20% of the aggregates contain only one tuple.

In addition to the optimizations made in the baseline algorithm [20], we also altered the algorithm such that only one output file was written consisting of all the cells and the patterns associated to them rather than one file per cell.

3.5 Adapting DPMiner for Distinct Patterns

Given the dynamic set of PC and NC classes for each cell, DPMiner will carry out the pattern mining task. It mines frequent generators and closed patterns for both classes by executing the computational steps in Section 3.3.

In DPMiner, the *class label distribution* of an itemset X in dataset D_i is denoted $n_i(X) = |f_D(X) \cap D_i|$ with $i = 1, 2, ..., k$, where for transaction T in dataset D, $f_D(Z) = \{T \in D | Z \subseteq T\}$ [15]. As we can see, this is the frequency of the itemset X in the dataset D_i. The following is the definition of a δ-discriminative equivalence class as defined by [15].

Definition 3.5.1. Let $D = \bigcup_{i=1}^{k} D_i$. Let EC be a frequent equivalence class of D, and C be the closed pattern in EC. Let $n_i(C), i = 1, 2, ..., k$, be the class label distribution information of C. Then EC is a δ-discriminative equivalence class if there is $i \in \{1, 2, ..., k\}$ such that $\sum_{j \neq i} n_j(C) \leq \delta$, where δ is usually a small integer number 1 or 2.

Recall that an itemset X is a distinct pattern of cell c if:

$$\frac{Sup_c(X)}{Sup_{\overline{c}}(X)} \geq \rho$$

To eliminate trivial cases, we will set the lower bound on $Sup_c(X)$ such that $Sup_c(X) > \rho$. We will also set an upper bound δ on $Sup_{\overline{c}}(X)$. We define this using the following theorem.

Theorem 3. *Consider a multidimensional database MDB and $\rho \geq 0$. For any itemset X and cell c, if X is a distinct pattern of cell c, that is:*

$$\frac{Sup_c(X)}{Sup_{\overline{c}}(X)} \geq \rho$$

then,

$$Sup_{\overline{c}}(X) \leq \Delta,$$

where

$$\Delta = \frac{|MDB|}{1 + \rho}$$

Proof. Assume X is a distinct pattern of cell c. It is trivial to see that if $Sup_{\overline{c}}(X) < \Delta$, then clearly the upper bound on $Sup_{\overline{c}}(X)$ is met. We will show: (1) if we take the maximum of $Sup_c(X)$ when $Sup_{\overline{c}}(X)$ reaches Δ, then X is still a distinct pattern and (2) if for any $\lambda > 0$, if $Sup_{\overline{c}}(X) = \Delta + \lambda$ then X is not a distinct pattern of cell c.
(1) The maximum that $Sup_c(X)$ can be is the entire set of tuples belonging to the cell (i.e.

the entire PC class). Since $Sup_{\overline{c}}(X) = \Delta$ (as maximum as possible) and since we know from Property 2.2.1 that $c.NC = \overline{c}.PC$, and $c.PC = \overline{c}.NC$, the maximum $Sup_c(X)$ be is $Sup_c(X) = |MDB| - \Delta = |MDB| - \frac{|MDB|}{1+\rho}$. Thus we have:

$$\frac{|MDB| - \frac{|MDB|}{1+\rho}}{\frac{|MDB|}{1+\rho}} \geq \rho$$

$$\frac{\frac{|MDB|+\rho|MDB|-|MDB|}{1+\rho}}{\frac{|MDB|}{1+\rho}} \geq \rho$$

$$|MDB| + \rho|MDB| - |MDB| \geq \rho|MDB|$$

$$0 \geq 0$$

(2) If for any $\lambda > 0$, if $Sup_{\overline{c}}(X) = \Delta + \lambda$ then X is not a distinct pattern of cell c. We will show this by contradiction. Assume X is a distinct pattern of c. Let $Sup_c(X)$ be as maximum as possible, that is $Sup_c(X) = |MDB| - \frac{|MDB|}{1+\rho}$ and $Sup_{\overline{c}}(X) = \Delta + \lambda$, for any $\lambda > 0$ that is $Sup_{\overline{c}}(X) = \frac{|MDB|}{1+\rho} + \lambda$. Then we have:

$$\frac{|MDB| - \frac{|MDB|}{1+\rho}}{\frac{|MDB|}{1+\rho} + \lambda} \geq \rho$$

$$\frac{\frac{|MDB|+\rho|MDB|-|MDB|}{1+\rho}}{\frac{|MDB|+\lambda+\lambda\rho}{1+\rho}} \geq \rho$$

$$|MDB| + \rho|MDB| - |MDB| \geq \rho|MDB| + \rho\lambda + \rho^2\lambda$$

$$0 \geq \rho\lambda(1+\rho)$$

As we can see the only case that this holds is if $\rho = 0$. However, from the Theorem 2 we know if $\rho = 0$ then the concentrate threshold $\alpha = 100\%$. Which means that $Sup_{\overline{c}}(X) \not> \Delta$. In particular, if $\rho = 0$ then $Sup_{\overline{c}}(X) = 0$ as 100% means the negative class cannot contain any itemset X. Contradiction. □

The following example illustrates the bounds on the supports.

Example 3.5.1. Suppose we are given a multidimensional database MDB, an itemset X that is a distinct pattern of cell c given $\rho = 3$. If we let $|MDB| = 100$ then $Sup_c(X)$ must be greater than 3 and $Sup_{\overline{c}}(X)$ must be less than 25. Suppose $Sup_{\overline{c}}(X) = 25$. We can see that even if $Sup_c(X)$ is at its maximum such that $Sup_c(X) = 75$, the distinctiveness ratio still holds where $\frac{75}{25} \geq 3$. However, if we set $Sup_{\overline{c}}(X) = 26$, we can see that $\frac{74}{26} \not\geq 3$.

For each aggregate cell, we only need to keep track of the positive class transactions in a bitmap, where each bit represents if the transaction is in the positive class or not. At the worst case, the bitmap would contain k bits where k is the total number of transactions in the multidimensional database. Since the recursive depth of BUC is at most n where n is the total number of dimensions, it can be seen that the maximum number of bits we would need to keep would be $k*n$. In addition, in reality the number of tuples in the positive class would decrease at each level, as not all tuples are expected to be in only one cell. Thus, as the dataset grows sparse, the total number of tuples at each level in the BUC processing tree decreases. Some bitmap compression algorithm can be applied at each level in the BUC processing tree.

After the mining process, the most general closed patterns (i.e., the ones that have the shortest length among others in its equivalent class) are determined as the so-called non-redundant δ-discriminative patterns. Thus using the frequency of itemsets in the PC and NC classes, we can check if $DR(X) \geq \rho$ and thus determine if X is a distinct pattern of cell c.

Mining disjunctive association rules can be a byproduct of our algorithm. An example of a disjunctive association rule is if an itemset X is bought by the customer group of males from Toronto or Vancouver. We can implement this with some minor revision. Sort all those cities according to the ratio of their support. Then you accumulate the total support of the cities and if they pass the threshold than it is a distinct pattern.

3.6 Summary

In this chapter, we discussed the Bottom Up Cubing process in Section 3.1. Then in Section 3.2, we introduced Frequent Pattern Mining and in Section 3.3 the DPMiner algorithm. Then in Section 3.4, we showed and discussed the execution of the baseline algorithm [20]. Finally, we concluded the chapter by showing how we use DPMiner to extract distinct patterns.

Chapter 4

Pruning Techniques

In this chapter we present several effective pruning techniques. We name this advanced algorithm MDPM (Multidimensional Distinct Pattern Miner). In Section 4.1 we demonstrate how we can use an ancestor cell's distinct patterns to derive the descendant cell's distinct patterns. In Section 4.2 we discuss how we can use the support of a parent cell to prune out descendant cells. In Section 4.3 we show how we can reduce the number of transactions under consideration by turning off the transactions that no longer belong to an aggregate cell. In Section 4.4 we find that by limiting the distinct patterns to finding the least generalized cell, we realize an optimization. Finally in Section 4.5 we summarize the findings.

4.1 Calculating Distinct Patterns in Descendants from Ancestors

In the problem formulation (Section 2.5) we discussed the use of the antimonotonic property to prune descendant cells based on ancestor cells. We can use a similar notion to derive the descendant cell's distinct patterns based on the ancestor cell's distinct patterns. From BUC, given two cells c_1 and c_2 such that $c_1 \succ c_2$ we know that $c_2.PC \subseteq c_1.PC$. Since our target cell c_2 is based on only a selected set of tuples from c_1, it can be seen that the distinct patterns of c_2 must also be a subset of distinct patterns of c_1. We formally state this in the following theorem.

Theorem 4. *Given cells c_1 and c_2 such that $c_1 \succ c_2$. If X is a distinct pattern in c_2, then X is also a distinct pattern in c_1.*

Proof. Let c_1 be the cell based on the set of tuples aggregated on dimension A. Let c_2 be a child cell of c_1 based on the set of tuples aggregated on dimensions A and B. From the definition of BUC, we can see:

$$c_2.PC \subseteq c_1.PC \tag{4.1}$$

For contradiction, assume that itemset Y be a DP in c_2 and not in c_1. From Equation 4.1 we can state:

$$\begin{aligned} Sup_{c_2}(Y) &\leq Sup_{c_1}(Y) \\ Sup_{\overline{c_2}}(Y) &\geq Sup_{\overline{c_1}}(Y) \end{aligned} \tag{4.2}$$

We can define the Distinctiveness Ratio of both cells as follows:

$$\begin{aligned} DR_{c_2}(Y) &= Sup_{c_2}(Y)/Sup_{\overline{c_2}}(Y) \\ DR_{c_1}(Y) &= Sup_{c_1}(Y)/Sup_{\overline{c_1}}(Y) \end{aligned}$$

From Equation 4.2, we can state:

$$DR_{c_2}(Y) \leq DR_{c_1}(Y) \tag{4.3}$$

From the definition of DP, in order for Y to be an DP of cell c_2, we must have DR_{c_2}(Y) $\geq \rho$. If this holds then based on equation 4.3, DR_{c_1}(Y) $\geq \rho$. Thus Y is also a distinct pattern of c_1. Contradiction.

□

Corollary 1. Let Y be the set of DPs in the cell c_1. Let X be the set of DPs in a child cell b_1. From Theorem 4, we can also state that $X \subseteq$ Y

The following example illustrates Theorem 4.

Example 4.1.1. Using Table 2.2. Suppose cell $c_1 = (a_1, *, *, *)$ and cell $c_2 = (a_1, *, *, l_1)$. We can see that $c_1 \succ c_2$, $c_1.PC = \{\{t_1, t_2, t_3, t_4\}, \{t_2, t_3, t_4\}, \{t_1, t_3, t_4\}, \{t_2, t_3, t_4\}, \{t_2, t_4, t_5\}\}$ and $c_2.PC = \{\{t_1, t_2, t_3, t_4\}, \{t_2, t_3, t_4\}, \{t_2, t_4, t_5\}\}$. Thus $c_1.PC \subseteq c_2.PC$. Given $\rho = 1$, consider the itemset $X = \{t_2, t_3, t_4\}$. The distinctiveness ratio of X in c_1 is $3/1 \geq 1$ and thus X is a distinct pattern of c_1. The distinctiveness ratio of X in c_2 is $2/2 \geq 1$ and thus X is a distinct pattern of c_2. Now consider an itemset that is a DP in c_1 but not a DP in c_2. Given $\rho = 1$, consider the itemset $X' = \{t_1, t_3, t_4\}$. The distinctiveness ratio of X' in c_1 is $2/2 \geq 1$ and thus X' is a distinct pattern of c_1. However, the distinctiveness ratio of X in

c_2 is $1/3 \not\geq 1$ and thus X' is not a distinct pattern of c_2. We cannot find any itemset that is a distinct pattern in c_2 and not a distinct pattern in c_1. Thus distinct patterns of a child cell is a subset of its parent cell. Figure 4.1 shows a snapshot of the data structures based on this example.

Implementation

We create two structures, `FirstLevelDistinctPatterns` and `PSubData`. `FirstLevelDistinctPatterns` stores the distinct patterns of the 1-D aggregate cells and `PSubData` stores all the transactions of the 1-D group-bys. Recall the pseudo-code for the baseline algorithm in Section 3.4. Line 14 now computes all the distinct patterns of only 1-D aggregate cells. Upon discovery of a distinct pattern X, an object of type `FirstLevelDistinctPatterns` F is created and is inserted into the list of `FirstLevelDistinctPatterns` objects. We then find all the transactions T_i from `PSubData` such that $X \subseteq T_i$ and make a pointer from F to all the transactions T_i.

Now we have all the distinct patterns of 1-D cells and the transactions associated to them. Based on Theorem 4, we can derive a descendant cell's distinct patterns from the ancestor cell. The descendant cell contains only the transactions belonging to the group by on that particular dimension. We add a boolean variable called `turned_off` to `PSubData`. The `turned_off` flag is set to true if a transaction is *not* part of the current group by. Thus, while doing the group by operation, we set the `turned_off` flag to true or false depending on whether a transaction is in this group by or not. Thus, while traversing through the list of distinct patterns from `FirstLevelDistinctPatterns`, we can check the transactions associated with each of the distinct patterns to see if that particular transaction is turned off or not. If it is turned off, then it is part of the negative class and thus reduces the support of this distinct pattern by 1. We can now calculate the supports of the distinct patterns in the descendant cell's negative class and positive class. Thus we can compute the distinctiveness ratio and test if a particular distinct pattern from the list of `FirstLevelDistinctPatterns` is also a distinct pattern of a descendant cell. Figure 4.1 shows a snapshot of the data structures from Example 4.1.1.

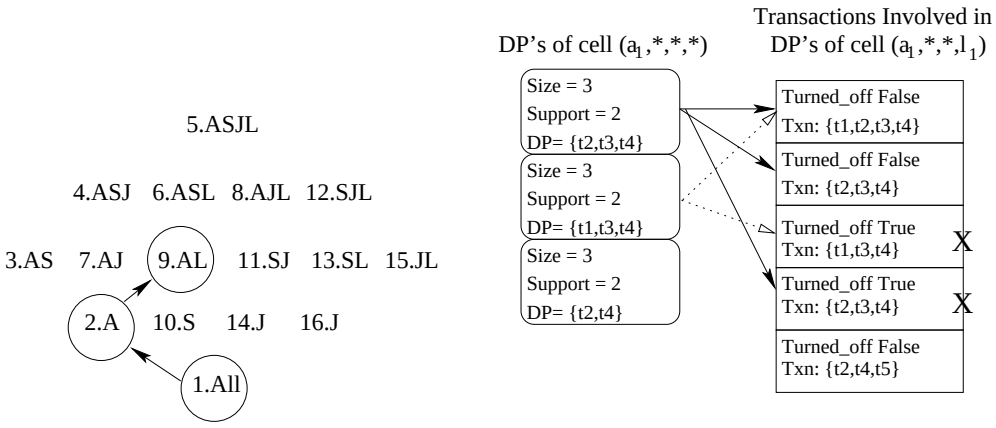

Figure 4.1: Distinct Patterns of descendant cells using ancestor cells, based on Example 4.1.1.

4.2 Pruning Child Cells Based on the Support of the Parent Cell

This optimization is also based on the fact that the transactions of a descendant cell is a subset of its parent cell. However, in this optimization we will look at the transactions of descendant cell as contributing to the support of the itemset in the parent cell. That is, the parent's transactions are made up of the sum of all descendant's transactions. Using this fact, we can prune some of the child cells of the parent. As we traverse the child cells of a parent, if we traverse a particular child at which point we know the rest of the cells cannot hold the distinctiveness ratio, then we can stop traversing the remaining cells. We formalize this in Theorem 5.

Definition 4.2.1. (Child Support Ratio). Given cells c_1 and b_i for $i = 1, ..., n$ where n is the total number of child cells of c_1 and $c_1 \succ b_i$. If X is a distinct pattern of c_1. Then for any child cell b_i, the Child Support Ratio (CSR) is defined as:

$$CSR(X) = \frac{Sup_{c_1}(\mathrm{X})}{Sup_{\overline{c_1}}(X) + \sum_{k=1}^{i} Sup_{b_k}(\mathrm{X})}$$

Using the following theorem, we show how we can use the child support ratio to prune out some of the child cells.

Theorem 5. *Given cells c_1 and b_i for $i = 1, ..., n$ where $c_1 \succ b_i$ and n is the total number of child cells of c_1. Let X be a distinct pattern of c_1. If for child cell b_i, $CSR(X) < \rho$, then for all child cell b_j where $j \geq$ i+1,...,n :*

$$\rho < \frac{Sup_{b_j}(X)}{Sup_{\overline{b_j}}(X)}$$

Proof. Based on the definition of DP we know that:

$$\frac{Sup_{c_1}(X)}{Sup_{\overline{c_1}}(X)} \geq \rho$$

It can be see that:

$$Sup_{\overline{c_1}}(X) + \sum_{k=1}^{i+1} Sup_{b_k}(X) \geq Sup_{\overline{c_1}}(X) + \sum_{k=1}^{i} Sup_{b_k}(X) \tag{4.4}$$

We also know the total support of X is $Sup_{c_1}(\mathrm{X}) + Sup_{\overline{c_1}}(\mathrm{X})$. Using Equation 4.4, if:

$$\frac{Sup_{c_1}(\mathrm{X})}{Sup_{\overline{c_1}}(X) + \sum_{k=1}^{i} Sup_{b_k}(\mathrm{X})} < \rho,$$

There will be no more child cells that can satisfy:

$$\frac{Sup_{b_j}(\mathrm{X})}{Sup_{\overline{b_j}}(\mathrm{X})} \geq \rho$$

Where $j = i + 1, ..., n$. Thus we can stop checking the remaining child cells of c_1 to see if itemset X is a DP of the child cell b_j for j ≥ i+1. □

We will illustrate this theorem using Example 4.2.1 based on the simple Table 4.1.

Example 4.2.1. Using Table 4.1, let itemset $X = \{t_1, t_2, t_3\}$ and $\rho = 3$. Suppose we are given cell $c_1 = \{a_1, *\}$ and cells $s_i = \{a_1, b_i\}$ where $i = 1, ..., n$ and $c \succ s_i$. Now using Theorem 5, as we traverse the child cells we will check the Child Support Ratio and see whether the following holds:

$$\frac{Sup_{c_1}(\mathrm{X})}{Sup_{\overline{c_1}}(X) + \sum_{k=1}^{i} Sup_{s_k}(\mathrm{X})} < \rho$$

For child cell $s_1 = \{a_1, b_1\}$, the Child Support Ratio is $\frac{8}{0+1} \not< \rho$ and so we continue to the next child. For child cell $s_2 = \{a_1, b_2\}$, the Child Support Ratio is $\frac{8}{0+(1+1)} \not< \rho$ and so we continue to the next child. However, for child cell $s_3 = \{a_1, b_3\}$, the Child Support Ratio is $\frac{8}{0+(1+1+1)} < \rho$ and so we can stop checking the remaining child cells. There will not be

A	B	*Transaction*
a_1	b_1	$\{t_1, t_2, t_3\}$
a_1	b_2	$\{t_1, t_2, t_3\}$
a_1	b_3	$\{t_1, t_2, t_3\}$
a_1	b_4	$\{t_1, t_2, t_3\}$
a_1	b_4	$\{t_1, t_2, t_3\}$
a_1	b_4	$\{t_1, t_2, t_3\}$
a_1	b_4	$\{t_1, t_2, t_3\}$
a_1	b_5	$\{t_1, t_2, t_3\}$

Table 4.1: Sample table to demonstrate Theorem 5

another child that holds the distinctiveness ratio threshold even though cell $s_4 = \{a_1, b_4\}$ contains the majority of support for X. For cell s_4, the $DR(X) = \frac{4}{3} < \rho$ and so X is not a DP of s_4. Figure 4.2 shows a snapshot of the `FirstLevelDistinctPatterns` after the processing of cell s_3.

Implementation

Given an m-D cell, we define the `level` of that cell to be m. We modify the `FirstLevelDistinctPatterns` structure such that each distinct pattern will contain an array of `child_supports` of size equal to the total number of dimensions in MDB. We keep a running total of the supports of the descendants in the `level` position of the array. Each time we traverse the `FirstLevelDistinctPatterns`, if for distinct pattern X, $CSR(X) < \rho$, then we continue to the next DP and stop checking the remaining child cells for X. Figure 4.2 shows the `child_supports` array within the `FirstLevelDistinctPatterns` object. As can be seen, since the dimension is two, the array size is two. This figure is a snapshot after the `child_supports` was updated during the processing of cell s_3 from Example 4.2.1. As we can see, 2.6 is less than ρ which is 3 and we can stop processing the remaining cells for this distinct pattern in the AB cuboid.

4.3 Sorting by Turned off Transactions

This optimization is also based on Theorem 4. Normally, as we group by we collect the set of transactions belonging to the PC value of the cell. However, as we group by on 2-D

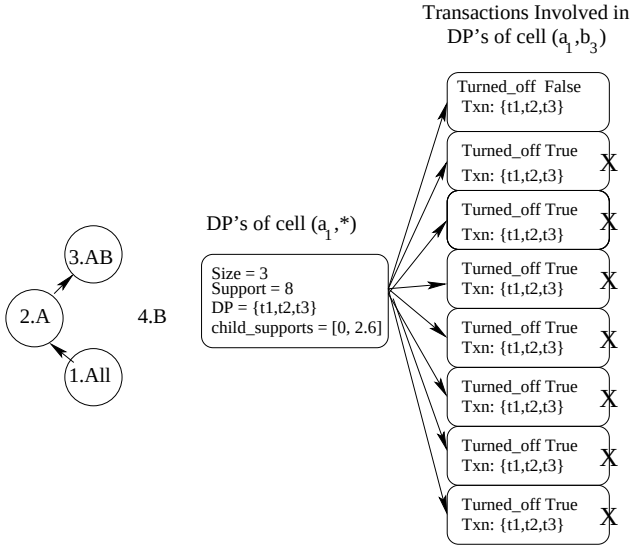

Figure 4.2: Shows a snapshot of the `FirstLevelDistinctPatterns` after `child_supports` has been updated during the processing of cell s_3 from Example 4.2.1

cells and their descendants, we determine whether or not a transaction in this particular cell was turned off when its parent was processed. If we sort the transactions using the turned off flag such that all non turned off transactions are at the top of the list and turned off transactions are at the bottom of the list. Then, as we traverse the list of transactions of a distinct pattern, at the first point when a turned off transaction is seen, we can stop checking the remaining transactions. The following example illustrates this optimization.

Example 4.3.1. Using Table 4.1. Let itemset $X = \{t_1, t_2, t_3\}$. Suppose we are performing a group-by on dimension B of the tuples belonging to cell $c_1 = \{a_1, *\}$. As we get to the cell $s_5 = \{a_1, b_5\}$, under normal circumstances the transaction belonging to this cell's PC will be at the very bottom. However, since we sort by the turned off transactions during the group-by, the transaction belonging to this cell's PC are at the very top of the list. Thus as we traverse through the list of distinct patterns, we need only to check two transactions, the first and the second. Upon checking the second, we find it is turned off and thus all remaining transactions are also turned off. At this point we can stop checking for the support of that particular DP in this cell. Figure 4.3 shows a snapshot of the list before and after the sort.

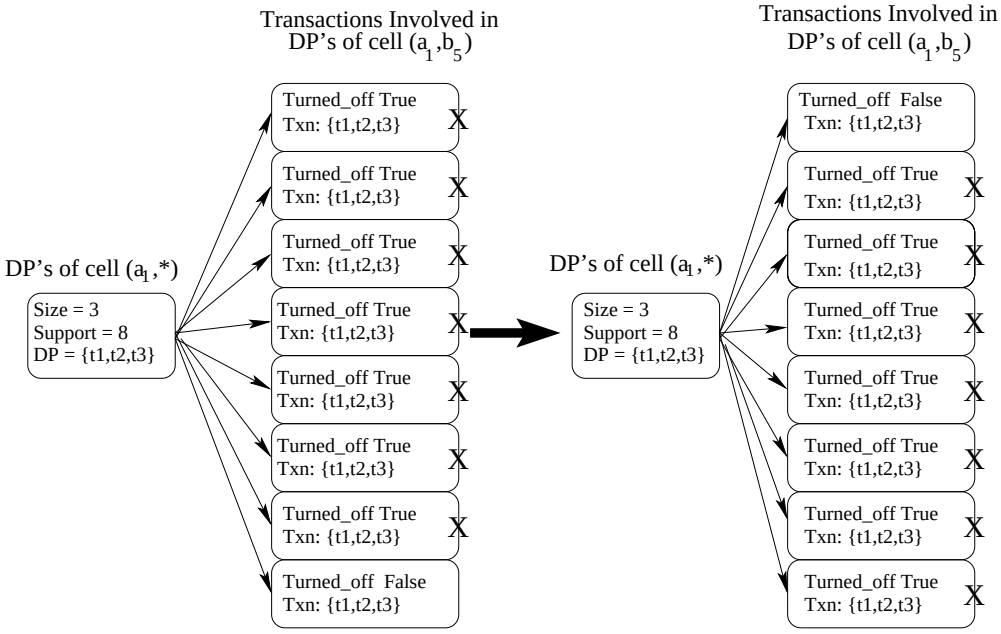

Figure 4.3: Sort by turned off transactions brings the non turned off transactions to the top of the list.

Implementation

As we group by on a descendant cell of a 1-D cell, we traverse each transaction belonging to that group by. `PSubData` contains the entire set of transactions from its parent 1-D cell. When a transaction in the descendant group-by is not a part of `PSubData`, we set the `turned_off` flag to true. We then use counting sort over a range of $\{0, 1\}$ where $0 = false$ and $1 = true$ which means the temp array in counting sort contains 2 elements. After the counting sort, the transactions that are set to turned off are at the bottom of the list and the transactions that are not turned off are at the top. As we traverse each child, we only pass the total number of transactions of the immediate parent cell as the size of the array to count sort. We know that the remaining transactions were set to turned off by the parent cell prior to materializing the current cell.

4.4 Max Group-by of Distinct Patterns

In this section we discuss an optimization that not only reduces redundant output of distinct patterns but also reduces the number of distinct patterns to traverse when finding distinct patterns of child cells of 1-D cells. We wish to output the least generalized group by that satisfies the distinctiveness ratio. That is, given two cells c_1 and c_2 such that $c_1 \succ c_2$ and an itemset X that is a distinct pattern of both c_1 and c_2. We wish to output only c_2 as the cell for distinct pattern X, provided that X is not a distinct pattern of a descendant cell of c_2. The following example illustrates these ideas.

Example 4.4.1. Using Table 2.2. Suppose we have the following cells $c_1 = (a_1, s_1, *, *)$, $c_2 = (a_1, *, j_1, *)$ and $c_3 = (a_1, s_1, j_1, *)$ such that $c_3 \prec c_1$ and $c_3 \prec c_2$. Given $\rho = 1$, itemset $X = \{t_1, t_3, t_4\}$ is a distinct pattern in c_1, c_2, and c_3. We wish to output c_3 as the least generalized cell that still satisfies the distinctiveness ratio.

We realize another optimization when we consider that for an m-D cell where $m > 1$, there is more than one parent for each cell. This can be seen by referring to Figure 2.1. Based on Corollary 4.1, if X is a DP in a cell c_1 where $c_1 \prec p_i$ where p_i is the parent cell of c_1 for $1 \leq i \leq m$, then we can ignore the traversal of this distinct pattern X in the parent cells p_i for $2 \leq i \leq m$. That is, we ignore the traversal of this DP from the list of `FirstLevelDistinctPatterns` if the current cell is traversed after c_1 and is a parent of cell c_1.

Example 4.4.2. Using Figure 4.4 and referring back to Example 4.4.1. We can see that cuboid AJ is processed after cuboid ASJ. Thus when looking at cell c_2, since X is a distinct pattern of cell c_3 and $c_3 \prec c_2$, thus X must also be a DP in c_2 using Corollary 4.1. Hence we can stop traversing this distinct pattern X in cell c_2. Figure 4.4 shows the `level` of the distinct pattern X is 3 and since we are traversing a cell that is only level 2 we can ignore any distinct patterns which have a `level` higher than the current level.

Implementation

Given an itemset X such that X is a distinct pattern of a particular cell, we initially set the level of each distinct patterns of `FirstLevelDistinctPatterns` to be the level of the 1-D cell which is 1. As we traverse the descendant cells of the 1-D cell, we update the level of

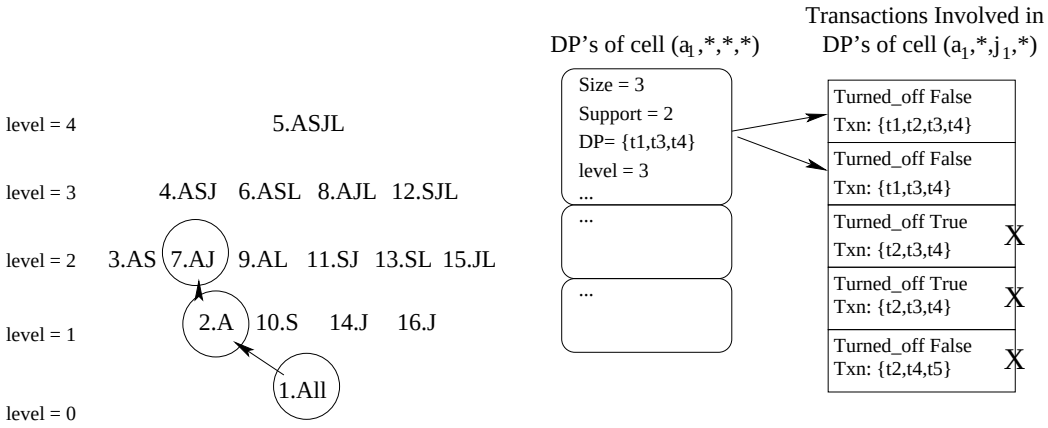

Figure 4.4: Showing a snapshot of `FirstLevelDistinctPatterns` after processing the cells from Example 4.4.1

each distinct pattern. We set the level of the distinct pattern to the level of the current cell, as long as the distinctiveness ratio holds and the level of the current cell is not less than the level of the distinct pattern. Thus, looking back at the previous example, although itemset X is a distinct pattern of cell c_2, the level of c_2 is less than the level of c_3. Referring back to the BUC group-by ordering in Figure 3.2, we would have materialized the cell c_3 before c_2. Thus we do not need to consider this distinct pattern for c_2 and so we ignore it when traversing the list of distinct patterns from `FirstLevelDistinctPatterns`.

We also exploit the antimonotonicity property here. As we traverse the 1-D cell's distinct patterns in an m-D cell, if the m minus the level of the distinct pattern is greater than one, then we can ignore this distinct pattern and continue to the next. That is, we had considered this distinct pattern in $(m-1)$-D cells and had concluded that they do not meet the distinctiveness ratio and so we didn't update the level to $m-1$. Thus, we can say that by the antimonotonicity property of BUC that this distinct pattern will not be a DP in any m-D cell.

4.5 Summary

In this chapter we discussed the several interesting pruning techniques to the baseline algorithm. We started off with the major optimization where we reduce the calls to DPMiner

by realizing that child cells distinct patterns are a subset of the parents distinct patterns. We then used the support of the distinct pattern in the parent cell and the supports of the distinct pattern in the child cells of the that particular parent to eliminate the traversal of some of the child cells. Next we sorted based on the transactions of the group by, whether it was turned off in that particular group by or not. Finally we discussed the need to reduce the redundant output of distinct patterns. That is, we know that a distinct pattern of a child cell must also be a distinct pattern of a parent cell. Thus we only output the least generalized group-by for a distinct pattern. In doing so, we realized that we can also avoid traversing some distinct patterns of the parent cells based on Corollary 4.1.

Chapter 5

Experimental Results and Performance Study

In this chapter, we present the empirical evaluation comparing the baseline algorithm and MDPM. All experiments were conducted on a PC running Ubuntu 10.04 with an Intel Core 2 Duo CPU, 3.0 GB memory and a 400 GB hard disk. The programs were implemented in C++ using Eclipse 3.5 with CDT. The DPMiner source code was kindly given to us by Dr. Guimei Liu, one of the authors of [15].

5.1 The Dataset

To the best of our knowledge, there are no widely accepted benchmark datasets that consist of a multidimensional set of attributes to transactions. Thus we created synthetic multidimensional datasets using five real benchmark datasets. We created the dimensions using a generator based on Zipf distribution. The transaction data are five widely-used benchmark datasets from the Frequent Itemset Mining Implementations (FIMI) Repository. These datasets from the UCI repository and PUMSB were prepared by Roberto Bayardo. The characteristics of the datasets are shown in Table 5.1.

We synthesized the datasets by appending the dimensions with the bench mark transactions.

Data Set	*Number of Tuples*	*Number of Items*	*Average Transaction Length*
Chess	3,196	76	37
Mushroom	8,124	120	23
Pumsb	49,046	2,113	74
Connect	67,557	150	43
Accidents	340,184	572	45

Table 5.1: Characteristics of the benchmark datasets

5.2 Comparative Performance Study and Analysis

For the default test case we used 2 as the distinctiveness threshold on a four dimensional MDB where the cardinality of each dimension was four and was generated using two as the Zipf distribution parameter. We ran five types of test cases on the synthesized datasets:

- varying the number of tuples
- varying the distinctiveness threshold
- varying the number of dimensions
- varying the cardinality of the dimensions
- varying the distribution of the dimensions

In addition, we also ran the datasets against just the BUC algorithm. Such that upon materializing the cell, it does not pass the aggregate measure of the cell to DPMiner to discover any distinct patterns. This is to test the performance of our BUC implementation. We decided not to test DPMiner by itself as it has been extensively tested in [15] using the benchmark datasets against many well known frequent pattern mining algorithms such as LCM3 [25] and FPClose [9].

5.2.1 Evaluating the BUC Implementation

To test BUC, we created a set of datasets, varying the number of tuples based on the accidents data. As can be seen from Figure 5.1, the running time of the BUC algorithm is approximately proportional to the number of tuples. That is, the running time of BUC grows linearly as the size of the input data increases. The use of the linear counting sort algorithm is mainly attributed to the linearity in the running time of BUC. It was shown that

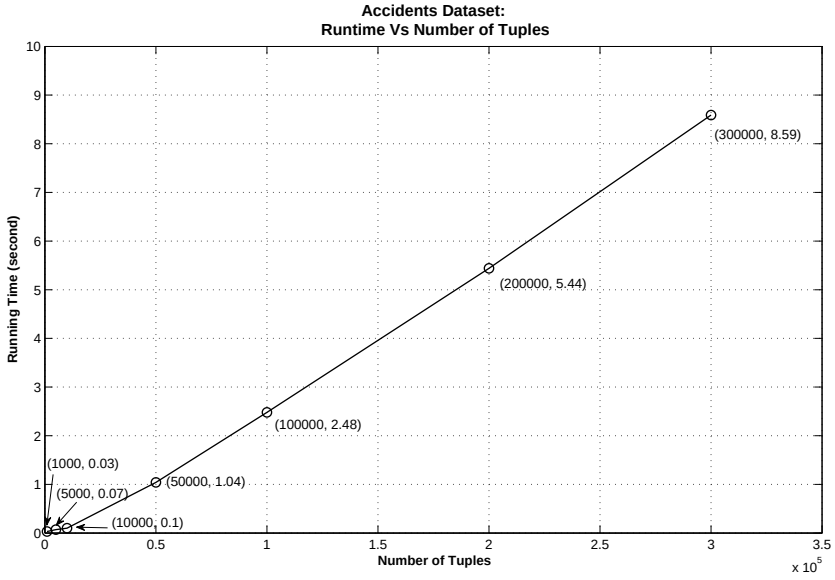

Figure 5.1: Runtime of the BUC implementation.

partitioning and sorting data were the most time consuming steps in a cubing process [4]. It is also notable that the heuristics such as the *writeAncestors* becomes of little significance as the number of tuples grows. This is because as the number of tuples increase, the MDB becomes less sparse. Thus, at each point a descendant cell will most of the time contain tuples and thus we cannot simply use *writeAncestors* as the descendant's measure might differ.

5.2.2 Varying the number of tuples

The default test case would test the varying number of tuples scenario. Each dataset from Table 5.1 contains different number, of tuples with accidents being the largest with 340,184 tuples. The results of the experiments are shown in Table 5.2. As can be seen, MDPM is considerably faster than the baseline algorithm. When considering the connect dataset, our algorithm is roughly 3 times faster than the baseline algorithm. When considering the Pumsb dataset, which contains a very large number of distinct itemsets, our algorithm is at least 20 times faster. The reason for such a behavior is that MDPM does not call DPMiner for each cell that is materialized. In a worst case scenario without any concept hierarchies

Data Set	*MDPM Runtime (s)*	*No. of DP*	*No. of Cells*	*Baseline Runtime (s)*
Chess	1.46	4	3	27.88
Mushroom	3.82	9	4	39.77
Pumsb	285.95	353	56	6464
Connect	290.98	6	4	978.35
Accidents	8396	36	12	N/A

Table 5.2: Runtime comparison between the Baseline and MDPM algorithm for each of the benchmark dataset

in the cube, the baseline algorithm would call DPMiner 2^n times where n is the number of dimensions plus the number of cells in each of the group-bys. However, MDPM would only call DPMiner on the 1-D cells. That is, MDPM would call DPMiner $\sum_{i=1}^{n} Cardinality(i)$ times. For the accidents test case, the baseline algorithm terminated with an out of memory error for the accidents dataset where as MDPM completed the run in 8,396 seconds.

5.2.3 Varying the support threshold

In this test case, we varied the distinctiveness threshold rate and kept the remaining default parameters the same. That is, dimension = 4, cardinality = 4 and distribution = 2. We ran five test cases where ρ was set to 1, 2, 3, 4 and 5. As can be seen in Figure 5.2, MDPM was far superior to the baseline. It was at least 10 times faster for the chess, mushroom and pumsb datasets and 3 times faster for the connect dataset. The baseline algorithm failed to complete for the accidents dataset. It is also noticeable that as we set the threshold lower, we produce much more distinct patterns. That is when $\rho = 1$, we want to find itemsets such that 50% of the itemset occur in a particular cell. That is, by setting the threshold lower, we are looking at a much larger set of frequent itemsets.

5.2.4 Varying the number of dimensions

Next we tested by varying the number of dimensions and keeping the rest of the parameters the same, with the distinctiveness threshold = 2, cardinality = 4 and distribution = 2. We again ran five test cases where the number of dimensions were set to 2, 3, 4, 5 and 6. As can be seen from Figure 5.3, MDPM is still considerably faster than the baseline algorithm. The most notable point in this experiment was when we set the number of dimensions low. Then the baseline algorithm was comparable to, if not better than MDPM. This is because of the

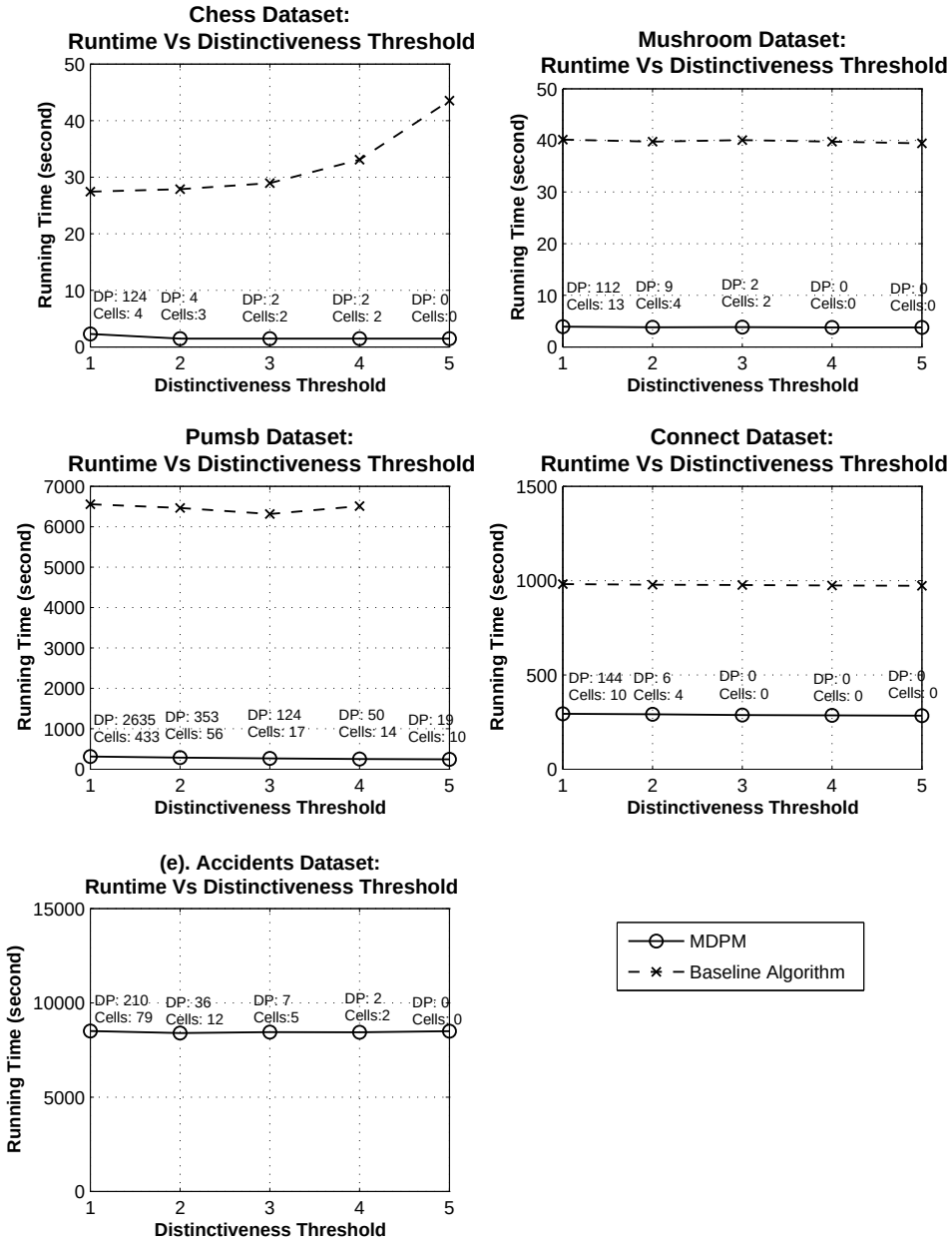

Figure 5.2: Runtime comparison between the Baseline and MDPM algorithm by varying support threshold

exponential nature of the baseline algorithm. MDPM does have some overhead in keeping the first level distinct patterns and ordering the transactions etc. Thus if you look at Figure 5.3(e), you can see that as the dimensions are set low, at 2 or 3, the baseline is faster than MDPM. However, as the number of dimensions increase, the baseline runtime increases exponentially. It can also be seen that this behavior only happens when the number of tuples is large. If you look at Figure 5.3(a-d), the optimal algorithm is still considerably faster than the baseline algorithm.

5.2.5 Varying the cardinality of the dimensions

In this test case, we varied the cardinality of the dimensions, with the distinctiveness threshold = 2, dimension = 4, and distribution = 2. The cardinality of each of the dimensions were set to 2, 4, 6, 8 and 10. Figure 5.4 shows the results of the experiments. For MDPM, as the cardinality increased, the run time decreased. This is attributable to the fact that when the cardinality of the dimensions is low, the MDB becomes more dense. Thus there are more frequent patterns that are mined and the overhead of keeping the first level distinct patterns increases. Even so, MDPM was still faster than the baseline in most cases. For the accidents dataset, the baseline algorithm failed to return for all but the first test case where cardinality = 2 where it was faster than MDPM. The baseline algorithm runtime increases as the cardinality increases.

5.2.6 Varying the distribution of the dimensions

Finally, we varied the distribution of the dimensions with the distinctiveness threshold = 2, dimension = 4, and cardinality = 4. The distributions were set to 1, 1.5, 2, 2.5, and 3. The goal of this experiment was to see how the runtime and the number of distinct patterns change as the dimensions become evenly distributed amongst the cardinality. Figure 5.5 shows the results of the experiments. The runtime for both the optimal and baseline algorithm were the same as varying the tuples test case. However, the number of distinct patterns returned by the optimal algorithm increased as the distribution increased. This is the logical result as the distribution increases, the dimensions become evenly distributed. Thus, for a lot of the frequent itemsets, the distinctiveness ratio would hold. That is, the transactions would be sparsely distributed in many cells and thus would not hold the apriori property for that cell.

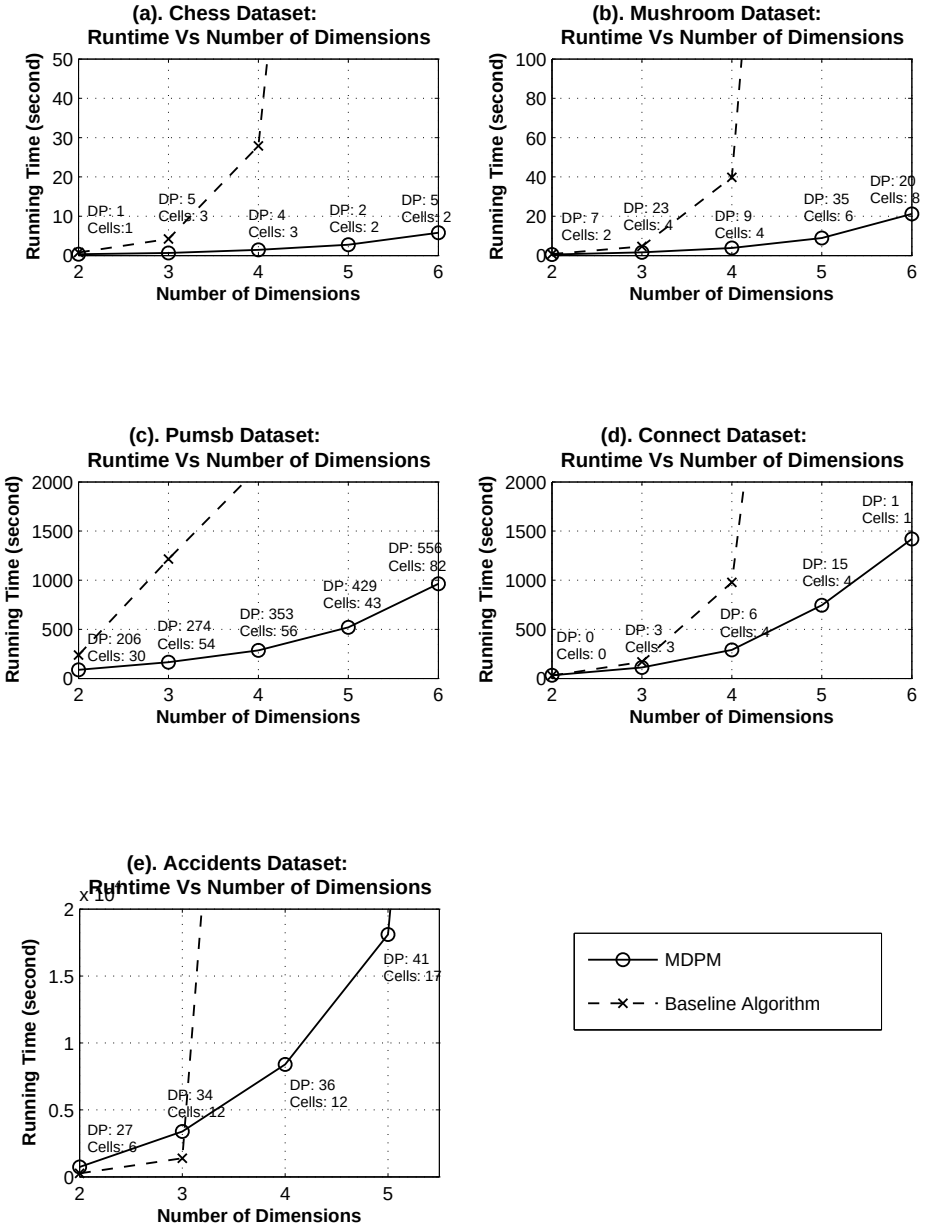

Figure 5.3: Runtime comparison between the Baseline and MDPM algorithm by varying the number of dimensions of MDB on the five benchmark data sets

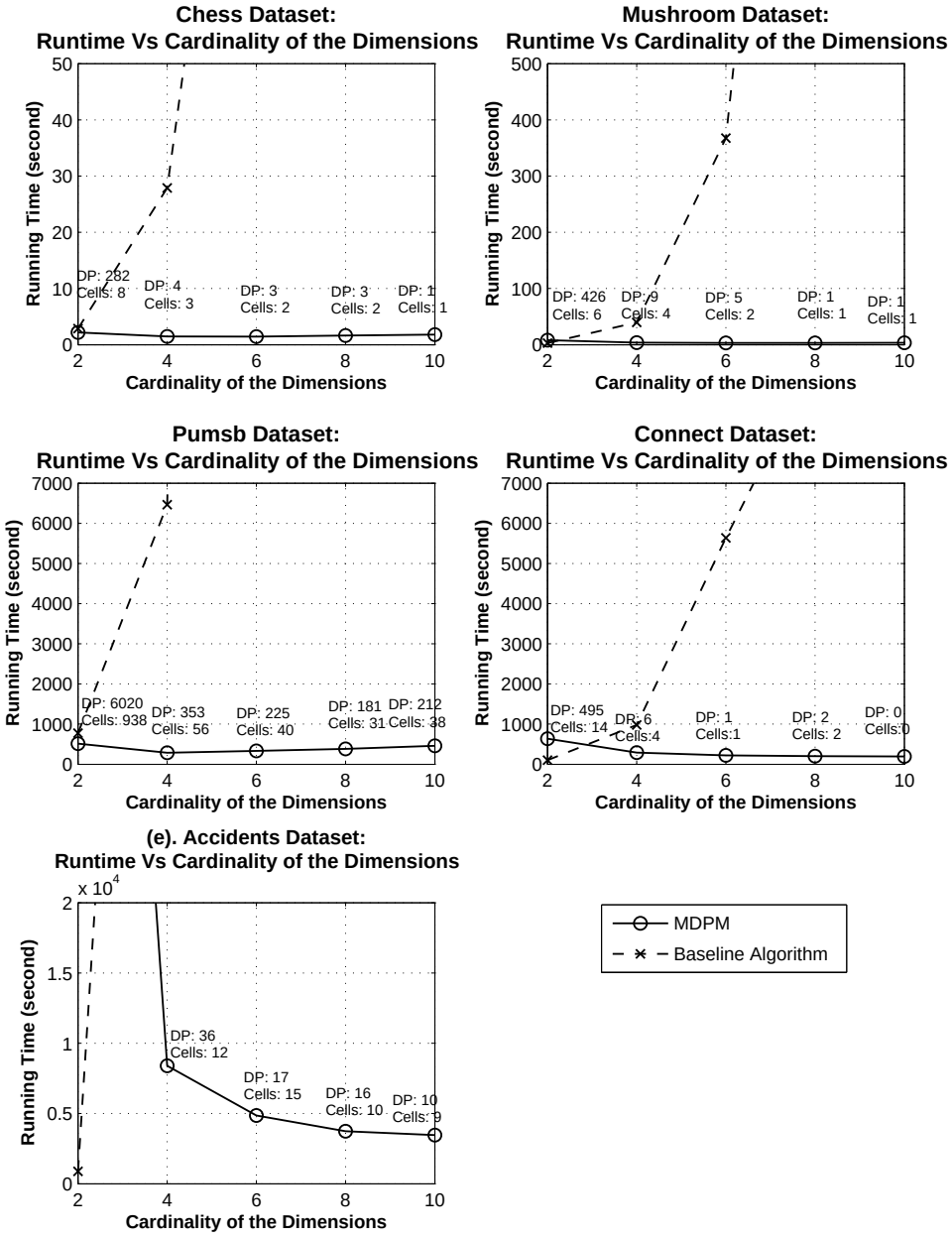

Figure 5.4: Runtime comparison between the Baseline and MDPM algorithm by varying the cardinality of the dimensions of MDB on the five benchmark data sets

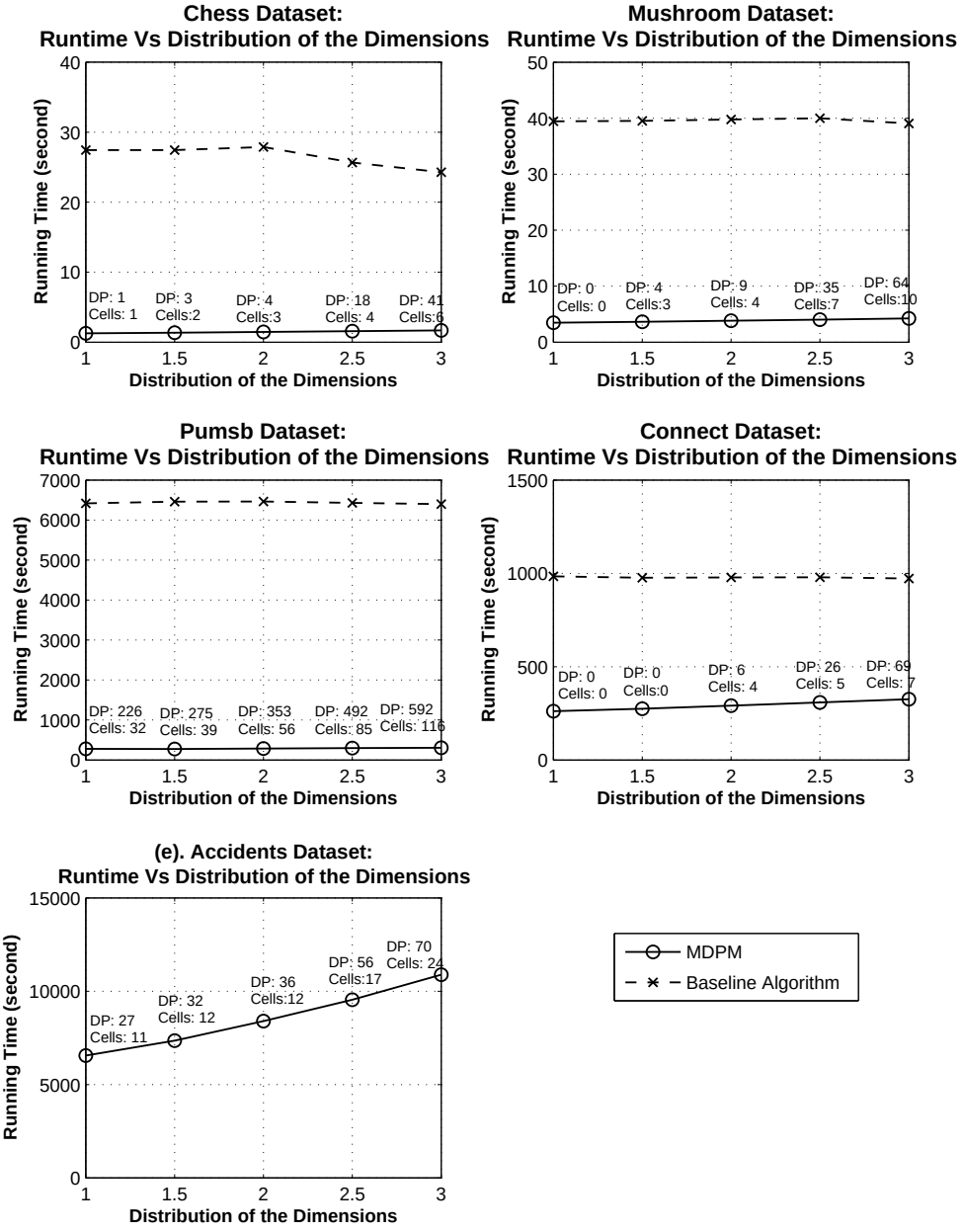

Figure 5.5: Runtime comparison between the Baseline and MDPM algorithm by varying the distribution of the dimensions of MDB on the five benchmark data sets

5.3 Summary

In light of the above experiments, it can be seen by the series of extensive comparative experiments that our proposed algorithm is indeed efficient compared to the baseline algorithm. There are very few cases where the baseline algorithm is faster than MDPM, but that occurs when we move towards a single dimensional database with a large number of tuples. However, the problem we are trying to solve is for multidimensional databases and in those cases our algorithm is much faster than the baseline algorithm.

Chapter 6

Conclusions

6.1 Summary of the Thesis

In this thesis, we defined a "distinct pattern" as a frequent itemset in which the absolute support ratio is greater than a given distinctiveness threshold. Given a multidimensional database, we presented a cube based algorithm to mine such distinct patterns. The idea was based on emerging patterns [6]. An "emerging pattern" has the ratio of the relative supports greater than a given threshold. The difference between distinct pattern and emerging pattern is that distinct pattern uses the absolute support ratio of itemset X from D_2 to D_1 where as emerging pattern uses the increase in growth rate of itemset X from D_1 to D_2. Cubing was not considered in the original emerging pattern problem.

We use BUC as our cubing algorithm. BUC traverses each cuboid from the apex towards the base. The reason for the choice of BUC is that it allows pruning during the construction of the cubes using the Apriori property. As can be seen from the experimental results of the BUC algorithm, it is quite a fast algorithm even when dealing with large datasets such as the accidents dataset [8].

For frequent pattern mining, we used a well known algorithm called DPMiner [15]. This algorithm was compared with LCM3 and FPClose and proved to be better than both [15]. DPMiner uses a modified version of FP-Trees to discover δ-discriminative equivalence classes along with their associated class label distributions.

A baseline approach to mining a multidimensional database was presented which mined every cell for distinct patterns. In a worst case scenario, this would call DPMiner 2^n times plus the number of cells in each of the group-bys. This exponential algorithm quickly

becomes impractical as the number of dimensions or tuples increases. Thus, we proposed another algorithm where we used the the Apriori property to prove that distinct patterns of descendants must be a subset of their ancestors. Based on this, we drastically reduced the number of calls to DPMiner to $\sum_{i=1}^{n} Cardinality(i)$. In Chapter 4, we presented other ideas where we used the ancestors properties to prune out descendant cells.

Next, we conducted extensive experiments using synthetic data based on the following five well known benchmark datasets from FIMI, chess, mushroom, pumsb, connect and accidents. We varied the number of tuples, the support threshold, number of dimensions, the cardinality of the dimensions and the distribution of the dimensions. As can be seen from the empirical study, MDPM surpassed the baseline algorithm in all types of multidimensional test cases.

6.2 Future Work

We have explored the BUC cubing and pruning based on Apriori. As future work, we can investigate other types of cubing strategies to see if better alternatives exist. For example, Star-Cubing [27] is known for its shared dimensions advantage. It combines the strengths of BUC as well as MultiWay to perform the cubing. Although BUC is considered the most efficient cubing algorithm for computing iceberg cubes, it would still be interesting to see if other cubing algorithms present more pruning opportunities.

Although we explored the direct line of ancestor to descendant relationship, it might be possible to explore, if we can construct sibling cell's partial distinct patterns using a 1-D cell. If such a case is possible, then we could produce the distinct patterns of other 1-D cells using only the first dimension 1-D cell. This type of pruning might be possible using Star-cubing.

Bibliography

[1] Rakesh Agrawal, Tomasz Imielinski, 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.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, pages 487–499, Santiago, Chile, September 1994.

[3] James Bailey, Thomas Manoukian, and Kotagiri Ramamohanarao. Fast Algorithms for Mining Emerging Patterns. In *PKDD '02: Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 39–50, London, UK, 2002. Springer-Verlag.

[4] Kevin Beyer and Raghu Ramakrishnan. Bottom-up Computation of Sparse and Iceberg CUBE. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 359–370, New York, NY, USA, 1999. ACM.

[5] Boling Ding et al. TopCells: Keyword-based Search of Top-k Aggregated Documents in Text Cube. In *ICDE '10: Proceedings of the 26th International Conference on Data Engineering*, Long Beach, CA, USA, 2010. IEEE.

[6] Guozhu Dong and Jinyan Li. Efficient Mining of Emerging Patterns: Discovering Trends and Differences. In *KDD '99: Proceedings of the fifth ACM SIGKDD International Conference on Knowledge discovery and data mining*, pages 43–52, New York, NY, USA, 1999. ACM.

[7] Guozhu Dong and Jinyan Li. Mining Border Descriptions of Emerging Patterns from Dataset Pairs. *Knowledge Information System*, 8(2):178–202, 2005.

[8] Bart Goethals et al. Frequent itemset mining implementations repository. Website, 2003. http://fimi.cs.helsinki.fi/data/.

[9] Gosta Grahne and Jianfei Zhu. Fast algorithms for frequent itemset mining using fp-trees. *IEEE Trans. on Knowl. and Data Eng.*, 17(10):1347–1362, 2005.

[10] Jim Gray et al. Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-tab, and Sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[11] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2nd edition, 2006.

[12] Jiawei Han, Jian Pei, and Yiwen Yin. Mining Frequent Patterns Without Candidate Generation. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 1–12, New York, NY, USA, 2000. ACM.

[13] William Inmon. What Is A Data Warehouse, 1995.

[14] Laks V. S. Lakshmanan, Jian Pei, and Yan Zhao. QC-Trees: An Efficient Summary Structure for Semantic OLAP. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 64–75, New York, NY, USA, 2003. ACM.

[15] Jinyan Li, Guimei Liu, and Limsoon Wong. Mining Statistically Important Equivalence Classes and Delta-discriminative Emerging Patterns. In *KDD '07: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge discovery and data mining*, pages 430–439, New York, NY, USA, 2007. ACM.

[16] Cindy Xide Lin et al. Text Cube: Computing IR Measures for Multidimensional Text Database Analysis. In *ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 905–910, Washington, DC, USA, 2008. IEEE Computer Society.

[17] Zhenhua Lin. Mining discriminative items in multiple data streams. Master's thesis, School of Computing Science, Simon Fraser University, January 2010.

[18] Guimei Liu, Jinyan Li, and Limsoon Wong. A New Concise Representation of Frequent Itemsets Using Generators and A Positive Border. *Knowledge and Information Systems*, 17(1):35–56, 2008.

[19] Elsa Loekito, James Bailey, and Jian Pei. A Binary Decision Diagram Based Approach for Mining Frequent Subsequences. *Knowledge and Information Systems*, 24(2):235–268, 2010.

[20] Wei Lu. Integrating data cube computation and emerging pattern mining for multidimensional data analysis. Technical report, DDP Capstone Report, School of Computing Science, Simon Fraser University, April 2010.

[21] Christopher Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[22] Sébastien Nedjar, Alain Casali, Rosine Cicchetti, and Lotfi Lakhal. Emerging Cubes for Trends Analysis in OLAP Databases. *Lecture Notes in Computer Science*, 4654:135–144, 2007.

[23] Jian Pei. *Pattern-Growth Methods for Frequent Pattern Mining.* PhD thesis, Simon Fraser University, 2002.

[24] Jian Pei. Cmpt 843 lecture notes. Website, 2009. http://www.cs.sfu.ca/CC/843/jpei/.

[25] Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. Lcm ver.3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In *OSDM '05: Proceedings of the 1st international workshop on open source data mining*, pages 77–86, New York, NY, USA, 2005. ACM.

[26] Lusheng Wang, Hao Zhao, Guozhu Dong, and Jianping Li. On the complexity of finding emerging patterns. *Theoretical Computer Science*, 335(1):15–27, 2005.

[27] Dong Xin et al. Star-Cubing: Computing Iceberg Cubes by Top-down and Bottom-up Integration. In *VLDB '2003: Proceedings of the 29th International Conference on Very Large Data Bases*, pages 476–487. VLDB Endowment, 2003.

[28] Duo Zhang et al. Topic Modeling for OLAP on Multidimensional Text Databases: Topic Cube and Its Applications. *Stat. Anal. Data Min.*, 2(56):378–395, 2009.

[29] Yan Zhao. Quotient Cube and QC-Tree: Efficient Summarizations for Semantic OLAP. Master's thesis, The University of British Columbia, 2003.

[30] Yihong Zhao, Prasad M. Deshpande, and Jeffrey F. Naughton. An Array-based Algorithm for Simultaneous Multidimensional Aggregates. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 159–170, New York, NY, USA, 1997. ACM.

Index