Spring 2011

Association Rule Mining -- Geometry and Parallel Computing Approach

Dongyi Jia
San Jose State University

Association Rule Mining

------ Geometry and Parallel Computing Approach

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Dongyi Jia

May 2011

Approved by: Department of Computer Science
College of Science
San Jose State University
San Jose, CA

_____

Dr. Tsau Young Lin

_____

Dr. Soon Tee Teoh

_____

Dr. Howard Ho, IBM Almaden Research Center

Acknowledgements

I would like to sincerely thank my advisor Dr. Tsau Young Lin for his great guidance and support to my project. The project would not be possible without Dr. Lin's thoughtful enlightening and patiently encouragement. I would also like to thank Dr. Soon Tee Teoh and Dr. Howard Ho for their valuable suggestions to my project.

I would further like to thank my families to give me endless love and encouragement throughout my graduate studies.

Abstract

Mining association rules is a very important aspect in data mining fields. The process to mine association rules not only take much time, but also take huge computing source. How to fast and efficiently find the large itemsets is a crucial point in the association rule algorithms. This paper will focus on two algorithms research and implementation in parallel computing environments. One is Bitmap Combination algorithm, the other is Bitmap FP-Growth algorithm. Compared to Apriori algorithm, both Bitmap Combination and Bitmap FP-Growth algorithms don't need generate candidate items, avoids costly database scans. Both algorithms need to translate the original database to Bitmap format, analyze bit distribution to reduce database size and apply high-speed bit calculation to improve the algorithms. The divide-and-conquer replace generation-and-test idea as the basic strategy. Bitmap Combination Algorithm shows the quick combination skills between any two, three, four and more rows, then screening the qualified itemsets. Bitmap FP-Growth Algorithm apply special bit calculation to recursively mine association rules. Based on the experimental results in this paper, both algorithms greatly improve the efficiency and performance of mining association rules, especially provide the possibility to mine association rules in highly parallel computing environments.

Keywords: Data mining, Association Rule, Bitmap Combination, Bitmap FP-Growth, Algorithms

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

In recent years, the researches on data mining have become a focused topic in databases fields with large datasets coming up. Finding interesting trends or patterns hidden in the huge data is aiming to guide decisions about future activities. Research on association rule is important part of data mining research. In a transaction database, an association rule is an expression to show the relationships of two sets of items. For example, {X} -> {Y}, we can explain the rules as follows: if x is purchased in a transaction, it is possible that y is also purchased in the same transaction. The possibility of X and Y happen together in the future transaction will be forecasted. There are two kinds of association rules: support and confidence. The problem of mining association rules is to generate all the association rules whose support and confidence are more than the user-specific minimum value.

Generally, the problem of data mining can be divided into two sub-problems: (a). search all the items whose transaction support is more than the minimum value, combine all the items. (b). generate the association rules from larger item sets. A lot of algorithms were belongs to these two categories. The classic Apriori algorithm takes method (a), mainly by candidate-generation-test method. The usual pattern-growth approach need to scan transaction database, is one of methods belong to method (b).

In my paper, I research two new data mining algorithms research in highly parallel computing environments. One is Bitmap-Combination algorithm; the other is Bitmap FP-Growth approach. Bitmap-Combination and Bitmap FP-Growth tree algorithm are different from most traditional data mining algorithms which usually use candidate-generation-and-test approach. Bitmap-Combination algorithm is mainly based on bitmap computing, in the meaning time, applies granular computing approach. And Bitmap FP-Growth tree algorithm is applying Bitmap technology to improve the traditional pattern-growth approach. Both algorithms avoid full database tables scan and multiple passes of all the itemsets to search association rules from very huge database. Bit operation is a kind of fast and effective operation to reduce the time-consuming multiple scans. It is more like granular computing between database itemsets. By bitmap method, we greatly improve the efficiency of association rule mining approaches, and create the improved association rule algorithms.

The structure of this paper is: we first generally introduce frequent itemsets and association rule, then present bitmap technology which applies to improve association rule algorithms. Then we introduce Bitmap-Combination and Bitmap

FP-Growth tree technologies in the second part. The third part we discuss the implementation of the Bitmap-Combination algorithm. The fourth part we discuss what the classic FP-Growth tree algorithm is and understand its kernel. The fifth part we discuss the implementation of Bitmap FP-growth tree algorithm. In the meaning time, we detailed analyze how and why Bitmap technologies can prune FP-growth tree and greatly optimize FP-growth algorithm. The sixth part is our experiments result and analysis. The last part is the resolution and the prospect for the future research work.

2. Frequent Itemsets and Association Rule Mining

The statistic of frequent itemsets intends to calculate purchase relation. For example, when we check the transaction sets in purchases tables, we often observe how likely a pen and a sharper are purchased together, or more items, such as ink, paper. An itemset is such a set of items: pen, sharper, ink, paper. We always try to find the biggest itemset with most high frequent items.

The support of an itemset is “a measure of what fraction of the population satisfies both the antecedent and the consequent of the true” [1]. Confidence is “a measure of how often the consequent is true when the antecedent is true” [1]. The calculation formula is written as follows:

“Let I = $\{i_1, i_2, \ldots i_m\}$ be a total set of items, D is a set of transactions, d is one

transaction consists of a set of items d $\subseteq$ **I,** Association rule X -> Y where X $\subset$ I ,Y $\subset$ I and X $\cap$ Y = $\varnothing$, support=(#of transactions contain X $\cup$ Y) /D; confidence = (#of transactions contain X $\cup$ Y) / #of transactions contain X. “ [1]

In the above example, we continue observe itemset {pen, sharper}, If we see that the support of this itemset was 60 percent in purchase, we will conclude that pen and sharper are often purchased at the same time. If we see the support is only 10 percent. We will make conclusion that pen and sharper are not purchased together frequently. Generally, the percent of sets of items is relatively small, especially when the size of the itemsets is very huge. If all itemsets whose support are higher than user-specified minimum support, we call the itemsets as frequent itemsets. For example, if the minimum support is 20 percent, the frequent itemsets in this example are {pen}, {sharper}, {ink}, {pen, ink}, and {pen, sharper}.

transid	item	qty
1	pen	2
1	sharper	1
1	ink	3
1	paper	6
2	pen	1
2	sharper	1
2	ink	1
3	pen	1
3	sharper	1
4	pen	2
4	sharper	2
4	paper	4

Table 1 - Purchase Relation (Same Transid, Same Purchase)

Just like the purchase relation shown in Table1. A set of tuples presents one transaction. When we retrieve the values in the item column, we get the items purchased in that transaction. Thus, the sequence of transactions naturally corresponds to the sequence of itemsets which are purchased by the customer. And a subsequence of itemsets is also a sequence which qualifies this condition. We can obtain more frequent itemsets by listing its subsequence combination.

Just as “a sequence $\{a_1, \ldots . a_m\}$ is contained in another sequence S if S has a subsequence $\{b_1, \ldots . b_m\}$ such that $a_i \subseteq b_i$, for $1 \leq i \leq m$.”[2] For example, {pen}, {ink, sharper}, {pen, paper} can be contained from the following sequent itemsets: {pen, ink}, {paper, ink, sharper}, {paper, pen, ink, sharper}. Itemsets determine sequent pattern, no matter the order of items. “The problem of identifying sequential patterns is to find all sequences that have a user-specified minimum support. “[2]

Mining association rule is a scientific method for prediction, but this prediction must be based on related background knowledge and additional analysis. On the contrary, improperly applying association rules will mislead people. So we must take it seriously. For example, the rule {pen}->{paper}, the confidence associated rule extracted from this database maybe is based on the conditional probability that a customer purchased a pen together with the purchase of a box of paper. Only based on the similar situation, this rule might be used to guide future sales promotions. Store can offer a discount on pens in order to increase the sales of paper or put the two merchandises nearby to make purchase easier. The

association relation is a good indicator for future customer transactions. And this cause link between two purchases is justified. Of course, most purchase relation between two merchandises has not clear cause link, mining association rule will do great help to identify the kind of relation.

3. Bitmap-Combination Algorithm

3.1 Bitmap Indexes Application

After accumulating the experience of using large scale database, we may have some thinking about index. Index will directly influence the database access efficiency. A simple index will shorten the running time of the same program greatly. Current most databases apply B-tree as index method. Traditional B-tree technology cannot greatly improve the access efficiency, so we consider bitmap index. Bitmap technique has been widely used since it was proposed in 1960's. Bitmap index is a special kind of technique which applied bitmap to database. Bulk of data can be expressed and stored as bit arrays and most database operation can be replaced by bitwise logical operations. Bitmap technique has more advantages than traditional B-Tree structure. 0 and 1 replace true or false to answer questions and are showed on the point of intersection of row and column in the table. So any distinct values in database can be simply expressed by 0 and 1. For example, an office store has five different kinds of printers, A, B, C, D, E. F.

A	B	C	D	E	F
1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	0	1	0	0
1	0	0	0	0	0

Table 2 - Bitmap Indexes on printers in store

When building bitmap index, we should firstly scan the whole table to create a bit stream. Every bit of stream corresponds to an intersection of single row and column. Based on the column number and properties to decide how many bitmap indexes will be built. Bitmap indexes do not repeat or repeat a relatively low number of times. In contrast, the bitmap index is designed for cases where number of distinct values is low, in other words, the values repeat very frequently. For example, when we represent “gender” in a staff database, only three values can be the answer: male, female or unknown. For such variables, the bitmap index can have a significant space and performance advantage over the commonly used trees.

Bitmap indexes are a structure derived from original tables. The first step to build bitmap index is to determine the exact quantities of those discrete values in the table. The following rules should be followed: if some rows include a special value, then the value on this bit is 1, else it is 0.

When the quantities of discrete value are low, bitmap can be represented very smaller. We can use the an example to illustrate: Suppose there are one hundred million rows in office product sale table, and each index in B-tree is ten bytes, then a B-tree index will take 100MB, but a bitmap index only takes 6.25MB(one million row * one bit per row * 5 bitmap)/ 8 bit per byte. Bitmap not only save memory, the read and write speed is much faster than B-tree. By the logical operation (such as AND, OR, NOT, XOR), one bitmap can easily calculate with other bitmap tables in the related groups. We also see the above example, suppose there are 20 local office stores, we build a Store_ID bitmap. And we also have a product “A” bitmap. We “AND” product ‘A’ and store_ID, then count ‘1’ in it. We can easily know how many ‘A’ products have been sold in store 15. Because computer has great advantages on calculating logical AND, OR and NOT, bitmap can greatly improve information retrieval efficiency.

3.2 Bitmap-Combination Algorithms

3.2.1 Conversion from Original Database to Bitmap Tables

The following table is the original data format. Each attribute has different values. It is very difficult to calculate numbers of distinct values in a huge datasets with thousands of rows and columns, and these tables will take up huge memory storage.

Item#	Fruits	Package Size
F1	Apple	Big
F2	Pear	Small
F3	Peach	Medium

F4	Peach	Big
F5	Banana	Medium
F6	Orange	Big

Table 3 - Original Input Data Format

Based on the description of the part of 3.1, we can convert the original data format to bitmap pattern. Not only save memory storage space, but also easily improve the calculation speed. Bit pattern is to decide how to use 0's or 1's to represent the original database and make data to bit-streams.

	Fruits					**Package Size**		
LINE#	Apple	Pear	Peach	Banana	Orange	Big	Small	Medium
1	1	0	0	0	0	1	0	0
2	0	1	0	0	0	0	1	0
3	0	0	1	0	0	0	0	1
4	0	0	1	0	0	1	0	0
5	0	0	0	1	0	0	0	1
6	0	0	0	0	1	1	0	0

Table 4 - Change Original Input Data to Bitmap

After knowing the bit pattern for individual value in columns we want to make the bit-stream for the consistent. So we convert bitmap Table 90 degree to make the data more clear. After converting, we can count 1's number to get the sum. This method is very simple, especially when the database is very huge. This preprocessing make the counting speed improve greatly.

	1	2	3	4	5	6
(Fruits, Apple)	1	0	0	0	0	0
(Fruits, Pear)	0	1	0	0	0	0
(Fruits, Peach)	0	0	1	1	0	0
(Fruits, Banana)	0	0	0	0	1	0
(Fruits, Orange)	0	0	0	0	0	1
(Package Size, Big)	1	0	0	1	0	1
(Package Size, Small)	0	1	0	0	0	0
(Package Size, Medium)	0	0	1	0	1	0

Table 5 - Bitmap Table 90 degree conversion

The preprocessing makes it easier to calculate the common itemset numbers. This number is very useful in the tree construction. For example, when we calculate how many common itemsets in transaction A and transaction B, we just did AND and count operation for transaction A and B. Bitmap operation greatly improve the calculating speed.

3.2.2 Logical Operation(AND, OR, NOT, SHIFT) and Count on Bitmap Tables

Now it is time to compare the two rows of bits sequentially. If the same positions at different rows are both 1, it means there is associate relationship between the two attributes. When do two rows AND operation, bit presents in bit-stream style. If the result equals to 1, we will increase the counter 1. After scanning the two rows, the counter will tell us the total associated item numbers. This quotient of counter divided by columns number is the certain attribute's association rule. If the quotient is very high, it means these two rows have high relative for each other. We can apply the same reason to three tuples, four tuples and so on, after ANDing, 1's quantities are more, the rows have higher association rules.

So, the aim behind the comparison and AND operation was to get the Association Rule between the attributes of data table. If there are more number of match after AND operation, more the chances for the rows that have high Association Rule.

3.2.3 Bitmap-Combination Basic Algorithm Description

From mathematics views, this computational process is a combination. Each combination contains two attributes values: one is from the first attribute; the other is from the second. There are C(n, k) possible ways to choose k attributes from n attributes.[3] If two rows do AND, k is 2, n is the column numbers. If we calculate many attributes relations, the number of combinations can be very huge. Especially, when the dataset is huge and support threshold is very low, the result is so huge. So the key part to improve the program running speed is how to create k-combinations using the k-1 combinations results.

Combination is the basic mathematics theory in this algorithm. How many times of combinations decides the running speed. At the same of combination, the counted result will be get. So AND, SHIFT and COUNT is the very frequent action. For bitmap, we do not need to worry the speed of bit operation. They are very fast. Some combination can be ignored if the potential combination result will not create a huge frequent pattern.

Suppose a List is L= {a, b, c, d, e, f, g , h, i, j, k, l, m}, if d, f, k are not in any 3-itemsets, they must no show in 4-itemsets. So d, f, k can be deleted from the List L, any other values' 4-combinations make up the 4-itemsets[3]. This is the process of creating k-itemsets from k-1itemsets. Same reason, 3-itemsets can be created by 2-itemsets. Using this method, much calculation work can be saved.

3.3 Bitmap-Combination Algorithms Implementation

3.3.1 Overview

First, I do some preprocessing work. Because the dataset is translated from other file formats. There are some unnecessary comma and sum between 0 and 1. It is not pure bitmap expression. And the dataset is very huge, so I have to divide the file into hundreds of small files. Data of each row in original file is stored into a separate text file. I remove all the commas and unnecessary attribute number between data. Only 1s and 0s are shown in each file. And I transfer the 0 and 1 txt file to .bin file. When program directly use binary file format, its running time also is shortened.

Functions of Read_file(), bit_and(), bit_count(), read_all_files() implement some basic operation during processing. Bit_and() realizes two bits from different files doing "AND" operation; bit_count() realizes counting 1's number for the whole row. Core functions are count1(), count2(), count3(), count4(). Please see the appendix to check the detail of functions.

For the "AND" function, I first spit the bitmap file into several files and then read them into main memory separately. The iteration of the AND loop use these files to reduce the time for the disk I/O's. All operations related to reading and writing files, we all use binary mode to reduce unnecessary conversion between binary and decimal expression.

For the "COUNT" function, I use the parallel bit count algorithm to increase the efficiency of the bit count. It separates the input bit stream into some smaller blocks, and count these block simultaneously. In this way, we can count millions of bits in a few seconds.

The databset is like huge table of data that contained 136 columns and 150432 rows, so we can say that it is in the form of 136 * 150432 matrixes of data. Although the dataset is very huge, all the running time of any combinations of 2-tuple, 3-tuple and 4-tuple are very good. For example, 2-tuple, set support threshold to 0.1, calculating all the qualified pairs results only take a few seconds; 3-tuple, set support threshold to 0.1, calculating all the qualified pairs results only

take 9 minutes and 24 second; 3-tuple, set support threshold to 0.7, calculating all the qualified pairs results only take 12 minutes and 17 second. Of course, all the support thresholds are set very low. If improve the threshold value, the program running time must be much shorter. Generally 3-tuple and 4-tuple combinations usually take a few hours. Comparing with this regular running time data of similar scale datasets, our experience result is very good.

In my program, I store the entire Bitmap in to a huge array of characters, and then do loops to begin calculation. It is not necessary to open and close the file million times. Just pick one row with any other row from the big file, then do AND operation. It will firstly compare the first row with every other row in bitmap table except the first one. So it will take total numbers of rows minus one times iterations. Then in outer level of loops, change the first row to all the other rows, repeat the same process. After ANDing, all the 1s will be counted. The quotient of this number divided by the whole bit number in one row is the ratio of support threshold.

Calculating 3-tuple combination based on 2-tuple results can save much time. 3-tuple combinations must come from the combinations which are greater than 2–tuple threshold. Same reason, 4-tuple calculation should based on 3-tuple‘s results.

3.3.2 Executable and Output

We run “AND” and “COUNT” on all possible pairs of rows of bitmap files. Set threshold as 50% for association rule, and selected all pairs that satisfy this rule. And based on the result of the result, we run “AND” and “COUNT” on all possible 3-tuple, using 50% to select triples. And then run “AND” and “COUNT” on all possible 4-tuple, using 80% to select 4-tupls. When execute these programs, you should input exe command, source file directory, file number, tuple number and support threshold.

The output of the calculation in the form as:

For pair: (file_name1_line# vs. file_name2 line#) = count_result (percentage)
For triples: (file_name1_ line# vs. file_name2_ line# vs. file_name3_ line#) = count_result (percentage)
For 4-tuple: (file_name1_ line# vs. file_name2_ line# vs. file_name3_ line# vs. file_name4_ line#) = count_result (percentage)

The result of previous step will be maintained in main memory, and the following steps will use this result to complete the calculation.

Bitmap-Combination Implementation Output files (part).

Pairs:
(40, 14 vs. 112, 1) = 101018 (67.1519%)
(40, 14 vs. 113, 1) = 101086 (67.1971%)
(40, 14 vs. 114, 1) = 101869 (67.7176%)
(40, 14 vs. 115, 1) = 101704 (67.6080%)
(40, 14 vs. 116, 1) = 99058 (65.8490%)
(40, 14 vs. 117, 9) = 99050 (65.8437%)
(40, 14 vs. 118, 4) = 92314 (61.3659%)
(40, 14 vs. 119, 1) = 98639 (65.5705%)
(41, 1 vs. 42, 2) = 94811 (63.0258%)
(41, 1 vs. 43, 1) = 93009 (61.8279%)
(41, 1 vs. 44, 2) = 92997 (61.8200%)
(41, 1 vs. 45, 1) = 93890 (62.4136%)
(41, 1 vs. 46, 22) = 93879 (62.4063%)
(41, 1 vs. 47, 2) = 93370 (62.0679%)

Triples:
(59, 1 vs. 100, 4 vs. 103, 1) = 124824 (82.9770%)
(59, 1 vs. 100, 4 vs. 104, 1) = 126068 (83.8040%)
(59, 1 vs. 100, 4 vs. 105, 1) = 125631 (83.5135%)
(59, 1 vs. 100, 4 vs. 106, 1) = 125492 (83.4211%)
(59, 1 vs. 100, 4 vs. 107, 1) = 123286 (81.9546%)
(59, 1 vs. 100, 4 vs. 108, 1) = 122765 (81.6083%)
(59, 1 vs. 100, 4 vs. 112, 1) = 124007 (82.4339%)
(59, 1 vs. 100, 4 vs. 113, 1) = 123951 (82.3967%)
(59, 1 vs. 100, 4 vs. 114, 1) = 126158 (83.8638%)
(59, 1 vs. 100, 4 vs. 115, 1) = 125765 (83.6026%)
(59, 1 vs. 100, 4 vs. 119, 1) = 122184 (81.2221%)
(59, 1 vs. 100, 4 vs. 126, 13) = 122740 (81.5917%)
(59, 1 vs. 100, 4 vs. 128, 6) = 127218 (84.5684%)

4-tuple:
(72, 1 vs. 77, 1 vs. 114, 1 vs. 119, 1) = 133273 (88.5935%)
(72, 1 vs. 77, 1 vs. 114, 1 vs. 126, 13) = 134108 (89.1486%)
(72, 1 vs. 77, 1 vs. 114, 1 vs. 128, 6) = 139068 (92.4458%)

(72, 1 vs. 77, 1 vs. 114, 1 vs. 133, 2) = 132005 (87.7506%)
(72, 1 vs. 77, 1 vs. 115, 1 vs. 116, 1) = 127114 (84.4993%)
(72, 1 vs. 77, 1 vs. 115, 1 vs. 117, 9) = 127101 (84.4907%)
(72, 1 vs. 77, 1 vs. 115, 1 vs. 119, 1) = 132841 (88.3063%)
(72, 1 vs. 77, 1 vs. 115, 1 vs. 126, 13) = 133719 (88.8900%)
(72, 1 vs. 77, 1 vs. 115, 1 vs. 128, 6) = 138595 (92.1313%)
(72, 1 vs. 77, 1 vs. 115, 1 vs. 133, 2) = 131614 (87.4907%)
(72, 1 vs. 77, 1 vs. 116, 1 vs. 117, 9) = 128644 (85.5164%)
(72, 1 vs. 77, 1 vs. 116, 1 vs. 119, 1) = 123627 (82.1813%)
(72, 1 vs. 77, 1 vs. 116, 1 vs. 126, 13) = 124681 (82.8820%)
(72, 1 vs. 77, 1 vs. 116, 1 vs. 128, 6) = 128317 (85.2990%)

Of course, we can continue to calculate 5-tuples, 6 tuples and so on base on the above algorithms. The combinations with the count greater than the minimal support are association rules[3]. So we get all the 4-combinations and counted all of their bitmaps. All the 4-combinations bitmap counters are greater or equal to the minimum support becomes a 4 large itemset, and all the other 4-candidates will be deleted.

3.3.3 Results Explanations and Analysis

Bitmap-Combination algorithm is much faster than traditional Apriori algorithm. From the theory, traditional Apriori algorithm is based on the counting of attributes in different tuples. The comparisons between attributes take much time. A dataset with m candidates and n tuples need m*n comparisons to get all the counts of m candidates, then deleted the candidates whose support value is less than thresholds. For bitmap-Combination algorithm, bit value has presented the combination of attributes and tuples, the number of AND operation decides the algorithm running time. When the dataset is become bigger, the running time difference of Bitmap-Combination and Apriori is much bigger, even include printing association rule result time. Bitmap-Combination algorithm is much faster than Apriori algorithm.

Of course, Bitmap-Combination algorithm need some time to do preprocessing work, such as use bitmap to represent the combination of attributes and tuples, and bitmap 90 degree transpose. But once finishing the preprocess work, the AND operation is significant faster than Apriori algorithm.

Bitmap-Combination algorithm doesn’t need to generate a huge number of candidate sets. It belongs to divide-and-conquer methods. From the bitmap technology, bit operation is a kind of high efficiency operation. Bitmap-

Combination algorithm doesn't need to repeatedly scan the database. Parallel bit operation between blocks greatly shorten running time. Bitmap also have great advantage on storage. Frequent items are stored as bitmap format. This format greatly reduces the dataset scale.

4. FP-Growth Tree Algorithm

Currently most data mining methods focus on generating frequent patterns, Apriori method is a famous one. But Apriori algorithm has the following disadvantages. Frist, Apriori algorithm will generate a huge candidate sets during the calculating process. Second, Apriori algorithm will scan the database repeatedly. These disadvantages decided Aprioir's processing speed is very low. A novel data mining method called frequent pattern tree proposed by Dr. Jiawei Han created to solve these problems[4]. FP-growth algorithm has many advantages than traditional Apriori algorithm: a condensed tree structure is developed to represent a large dataset; avoid dataset multi-scans; no need to create and maintain huge candidate sets; use divide-and-conquer method to do data mining in partitioned smaller datasets. These advantages make FP-growth algorithm achieve better performance than Apriori algorithm.

FP-growth tree take a set of prefix trees as the children of root. The root node is null. A related item-frequency table works as an assistant table. Every node is composed by three parts: itemname, quantities and pointer—itemname represents which item in the database, quantities are the number of transitions related to the items, pointer points to the next node in the tree structure.

There are 2 very important parts using FP-growth tree algorithm to mining frequent pattern. The first part is frequent pattern tree construction; the second part is mining complete set of frequent patterns by the constructed FP-tree in the first step. But FP-growth tree algorithm also has some problems during implementation. Because FP-tree is a tree structure; it is difficult to code. And FP-growth tree is an algorithm based on main memory mining. If the dataset is very huge or the threshold is very low, it is almost impossible to put the whole datasets into main memory. So many disk-based optimization strategies were created to solve main memory problem: divide the whole dataset into smaller ones and do FP-tree mining in each smaller datasets; prune FP-growth tree to reduce the size, remove less than threshold items to build sub-datasets; optimize single path of FP-tree to reduce the calculation time and so on. Although there are optimization strategies to help solve FP–tree main memory problems, FP-growth algorithm still has the limitation to mining huge scale datasets and its tree structure also is the source to difficultly implement on actual code. So we

propose bitmap technology to solve these problems. Bitmap FP-growth algorithm use bitmap to express dataset, apply FP-growth tree's thoughts but not use tree as data structure, utilize bit operation to shorten calculation time. And during the implementation of Bitmap FP-growth tree algorithm, I also apply some necessary optimization strategies with bit operation traits to shorten running time.

Let us first introduce the FP-growth algorithm and its tree construction.

4.1 Frequent Pattern Tree Construction

Frequent Pattern Tree is a compact data structure.

1) The transaction database will be scanned from the beginning to the end. Calculate and record the frequent pattern itemsets and their support values. The frequent itemsets will be sorted by support values in descending order.
2) The itemset of each transaction will be stored in compact data structure – an extended prefix tree structure. The function of this informative tree structure will replace the whole dataset to be the mining object.
3) Merging the common sets can save multiple time and space costs. When much transactions share a set of frequent set, this advantage is more obvious. And all the frequent items are sorted in descending order, the more prefix strings will be shared.

Following is an example of the processing to construct FP-growth Tree [4]. The 10 transactions are picked from multiple data sets. And minimum support value is 3.

ID	Items in Basket	(Ordered) frequent items
1	**5**,**10**,14,16,**17**,**19**, 34,**38**,39	5, 19, 38, 17,10
2	**5,7,** 23, 26, 31, **40**, 47, 59, 63	5, 7, 40
3	**5,13,19**,23,26,29,36, **40**,75	5,19,13,40
4	**5,19,**27,29,30,33,36,**38**,54	5,19,38
5	**5, 7, 10**, 12,**13,** 20, 54, 59, 90	5, 7,13,10
6	**5,13,**14,24,27,30,32,43,48	5,13

7	6, **7**,14, **17,19,21**,34,**38,46**	7,19, 38,17,21,46
8	6**,10,21**,22,24,28,29,35,39	21,10
9	**7,**16,**17,21,40,**43,**46**,53,70	7,17,21,40,46
10	**7,13,**34,35,**38,46**,48,54,68	7,13,38,46

Table 6 - Transaction Examples

From above figure, we get the head table about frequency count. Table7 shows the items in descending order.

Item ID	Frequency Count
5	6
7	5
19	4
13	4
38	4
17	3
21	3
40	3
46	3
10	3

Table 7 - Frequency Count of Items

The following is the whole process to gradually construct FP-tree based on the above tables: ((5:6), (7:5), (19:4), (13:4), (38:4), (17:3), (21,3), (40:3),(46:3), (10: 3)) The number after colon is support and support >=minimum. All the branches of tree follow the same order - descending order. The root is set to “NULL”. By scanning the dataset second time, we can build the each branch one by one.

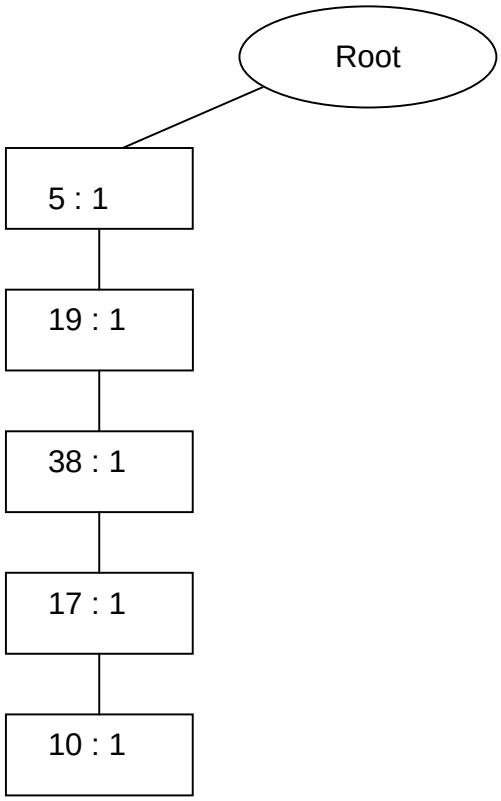

Figure 1 - Frequent Pattern Tree Construct Process 1

For 2nd transaction, it also follows the descending order, (5, 7, 40) share same prefix 2 with (5, 19, 38, 17, 10). So the second branch will extend from the node (5), and the same time, the count after 2 is increased by 1, (5:1) change to (5 : 2).

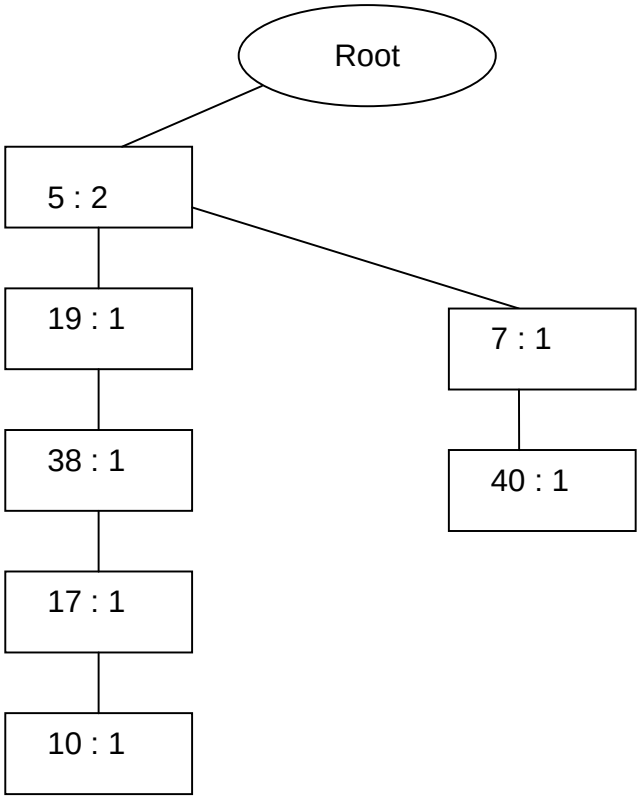

Figure 2 - Frequent Pattern Tree Construct Process 2

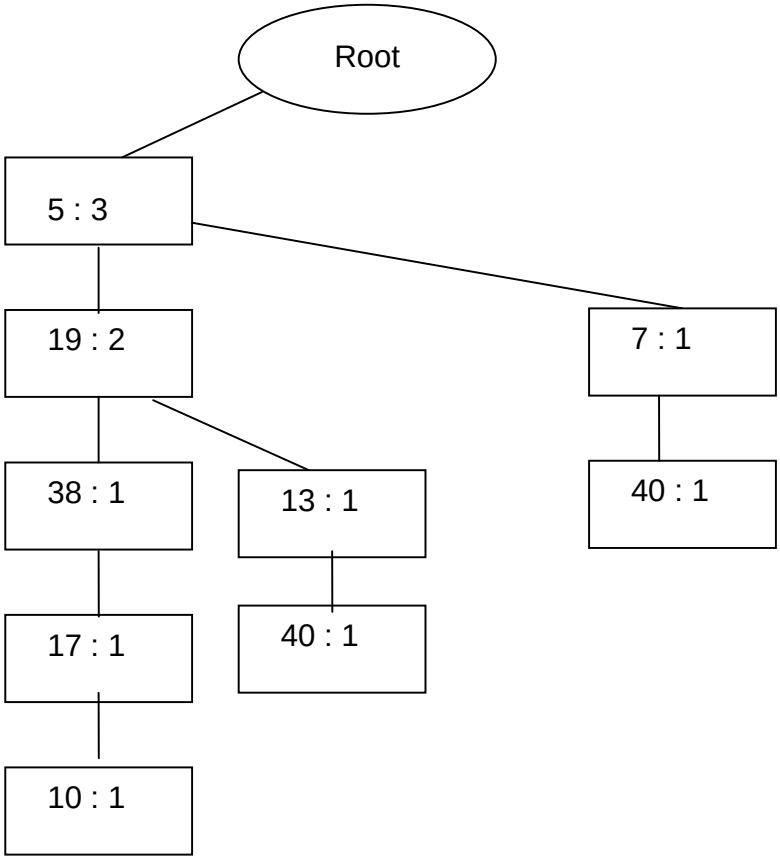

Figure 3 - Frequent Pattern Tree Construct Process 3

For 3rd transaction, (5, 19, 13, 40) share the common prefix (5,19) with the 1st transaction, so a branch created from the first branch, and both 5 and 19 counts are increased by 1. Figure 3 shows the tree structure.

With the same method, after scanning the fourth transaction, we construct the fourth branch, but the fourth transaction (5,19,38) is exactly one part of first transaction(5, 19, 38, 17,10), so the tree will not increase one branch, all the count of items in (5, 19, 38) are increased by 1. The figure4 is formed.

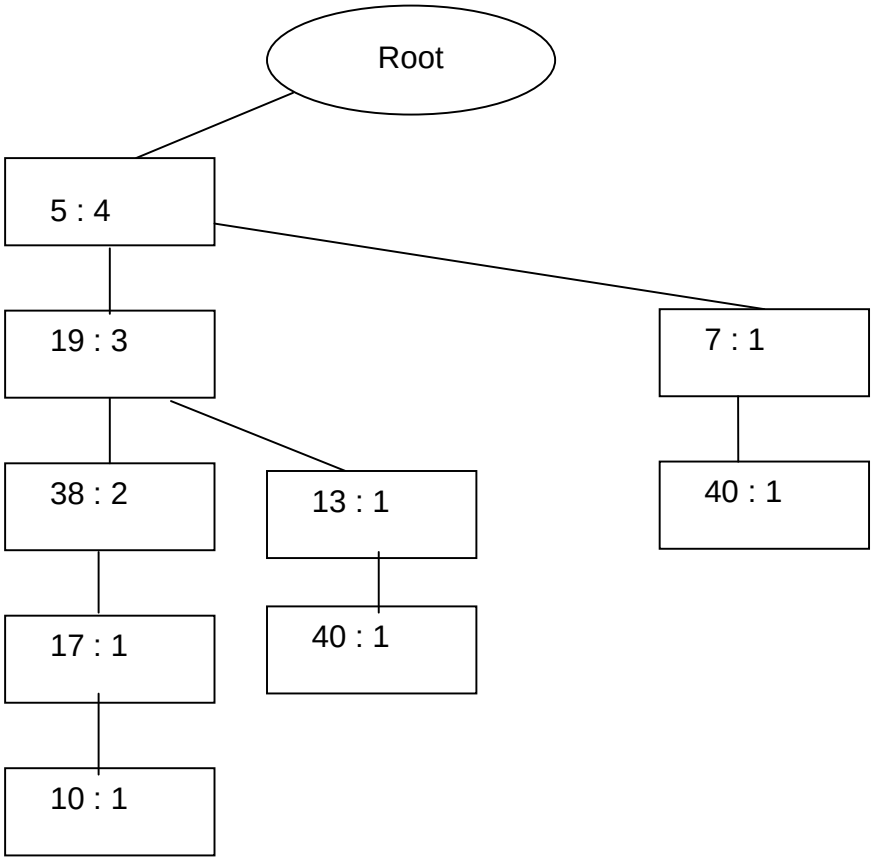

Figure 4 - Frequent Pattern Tree Construct Process 4

In the following, we continue to construct the transaction 5, transaction 6 and so on. We get the part of FP-growth tree (one step in the constructing process, Figure 5) and complete FP-tree (final step in constructing FP-tree, Figure 6)

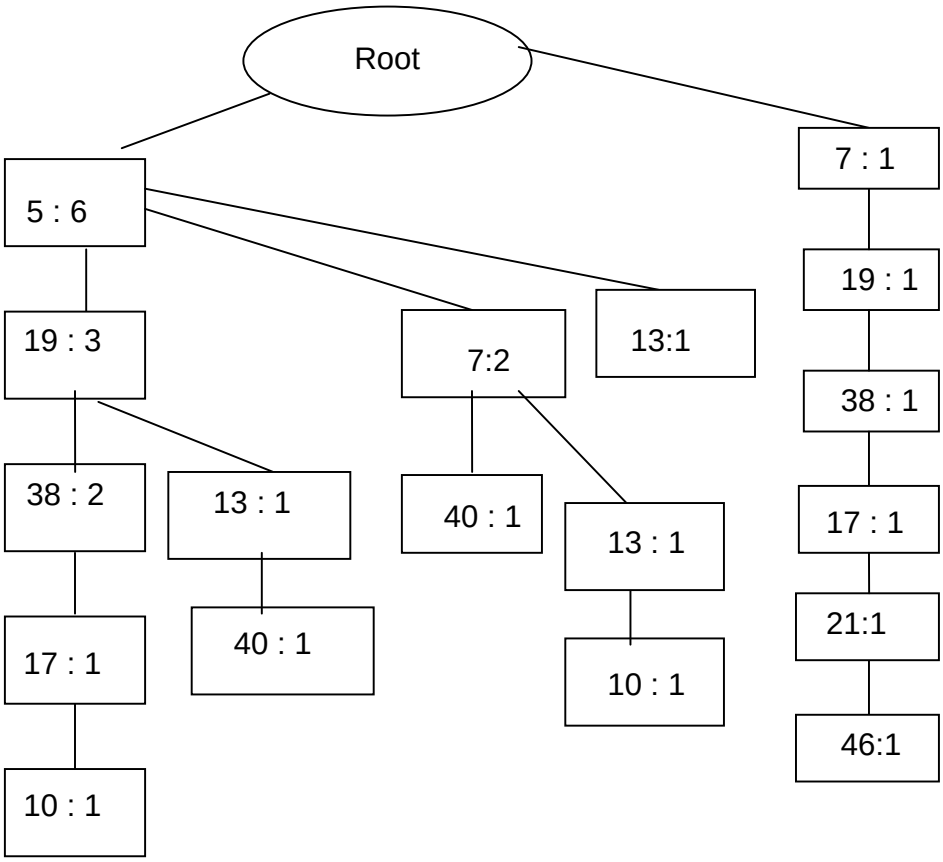

Figure 5 - Frequent Pattern Construct Process 5

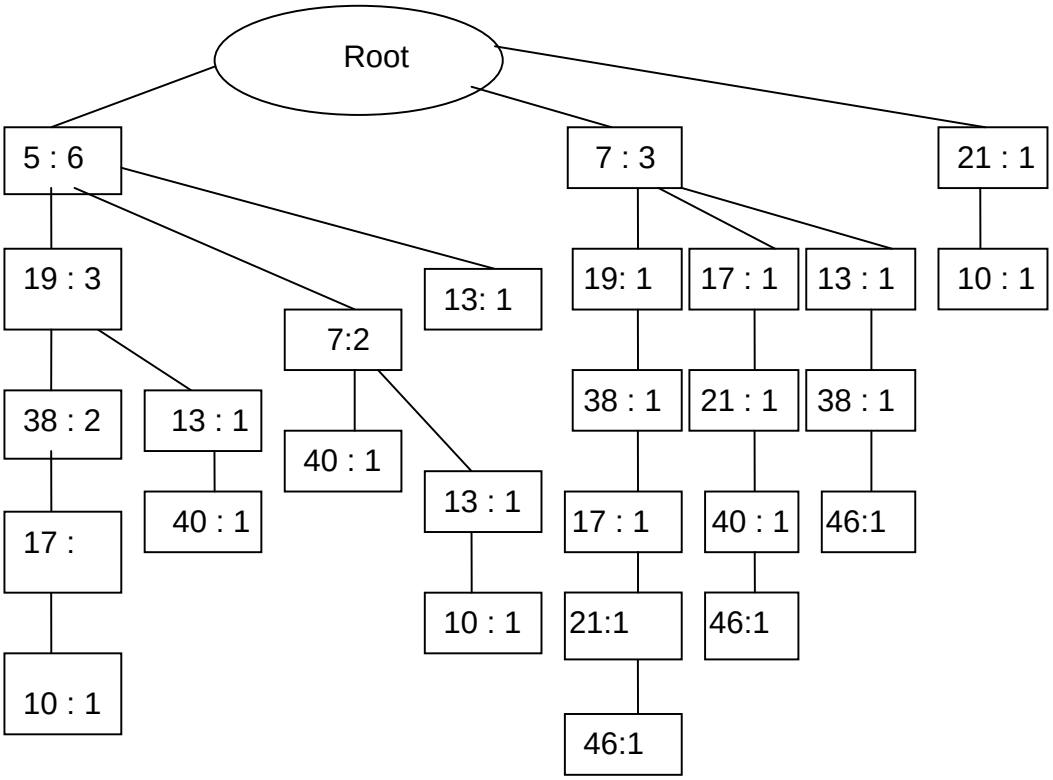

Figure 6 - Complete FP-TREE

FP-growth tree has great advantages to mining data. First, there are only 2 times scans during FP-growth Tree Construction. First scan gets all the qualified transaction items, second scan sorts items, inserts and creates FP-growth Tree. Second, this data structure of FP-growth Tree greatly reduces the size of datasets. Let we analyze the relationship between FP-tree and transaction datasets. Every piece of transaction at most produces one branch of FP-growth Tree. Many transactions will share common items with other transactions, so this kind of prefix tree structure reduces the width of tree. And branch depth is also not unlimited. It is decided by the item numbers in one transaction. FP-growth tree is a kind of highly compact data structure to store and retrieve data than the original transaction sets.

4.2 Get Frequent Pattern by FP-growth Tree

4.2.1 The definition of Conditional FP Base and FP-Tree

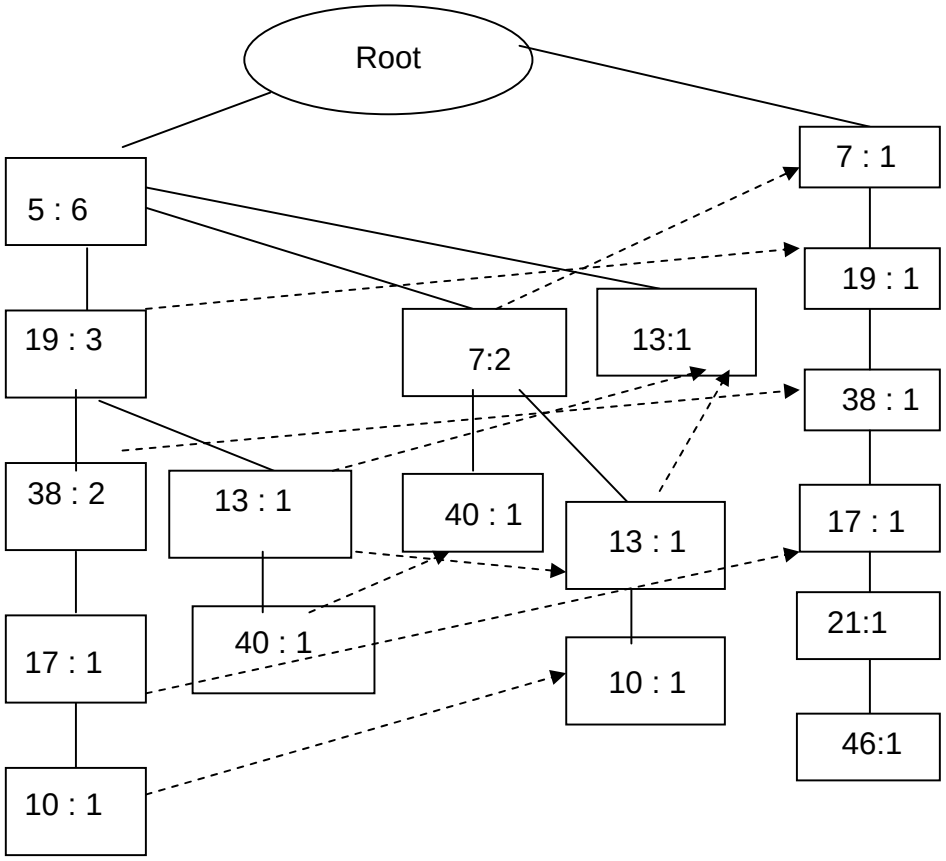

Figure 7 - Node_Link

There is a special point, which is helpful to mine frequent pattern by FP-growth Tree, in the constructing process. In FP- tree, each node includes item-name, count and node-link. Item-name and count have clear explanations in the 3.1 part. Node-link is a special pointer points the other node in the same FP-tree with the same item-name. If no same item-name in the other branches, the pointer is set to null. This node-link pointer builds a width-relationship between same items. For example, the dotted line from (19 : 3) to (19 : 1) indicate the node-link between them because they have same item name: 19. The concept of Conditional Frequent Pattern based on Node-link relationships.

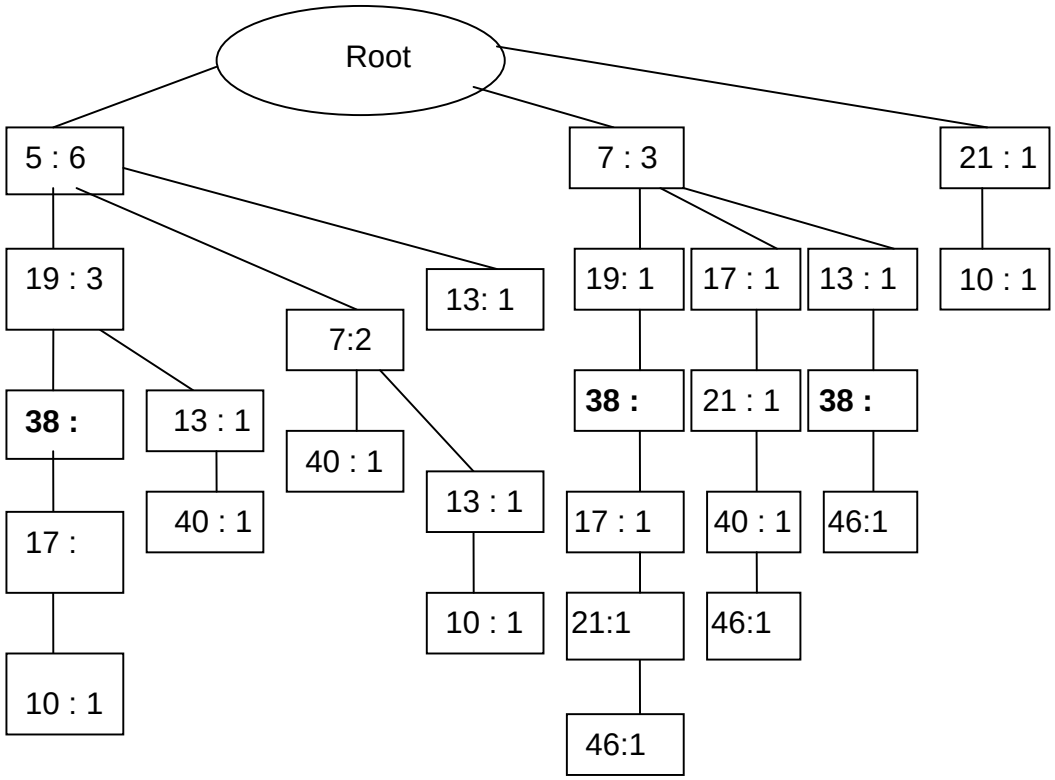

Figure 8 - Construct Conditional FP-tree

There are 3 paths including item 38. on path (5, 19, 38, 17, 10), path (7, 19, 38, 17, 21, 46) and (7, 13, 38, 46); from 38's prefix and 38's count, we know the string (5: 2, 19 : 2) have relationship with 38, write it as (5 19: 2), same reason, from path (7:1, 19:1) has relationship with 38, write it as(7 19:1), and (7:1, 13:1) has relationship with 38, write it as(7 13:1), so 38's conditional pattern base is {5 : 2, 19 : 2}, {7 :1, 19 : 1}, {7 :1,13 :1} or write it as (5 19 : 2, 7 19 :1, 7 13 :1). Based on FP-tree construction algorithm, after getting conditional pattern base, we continue to construct its conditional FP-tree. We get the following sub-tree structure:

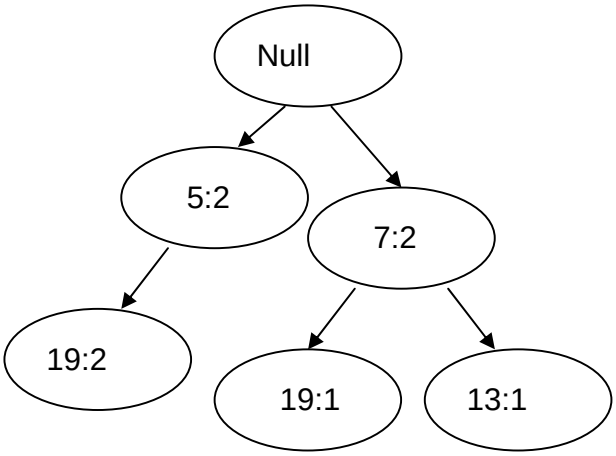

Figure 9 - 38’s Tree Branch of Conditional Frequent Pattern

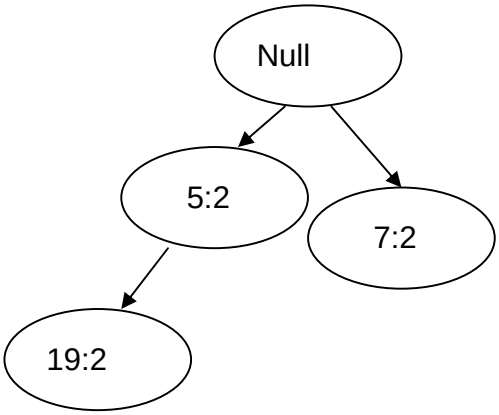

Figure 10 - Simplify 38’s conditional FP-Tree

We take 2 as the threshold. 5,19 and 7’s count bigger than support value(2), so 38’s conditional FP-tree is (5:2,19:2,7:2), because 19:2 and 19:1 are in different branches, we can’t simply add them to 19:3. We write it as {(5 : 2, 19 : 2), (7:2)} | 38 to express its 38’s conditional FP-tree. From figure 10, we generate frequent pattern: 2 35:2, 16 35:2, 4 35:2, 2 16 35:2. Two numbers in same branch can do combination; two numbers in different branches cannot do combination. Mining FP-tree to get frequent patterns is a recursive process. This is the key point of mining algorithm.

38-conditional FP base : {5 : 2,19 : 2}, {7:1, 19:1}, {7:1,13:1}
38-conditional FP tree : {(5 : 2, 19 : 2), (7:2)} | 38
38-Frequent pattern generated : 2 35:2, 16 35:2, 4 35:2, 2 16 35:2

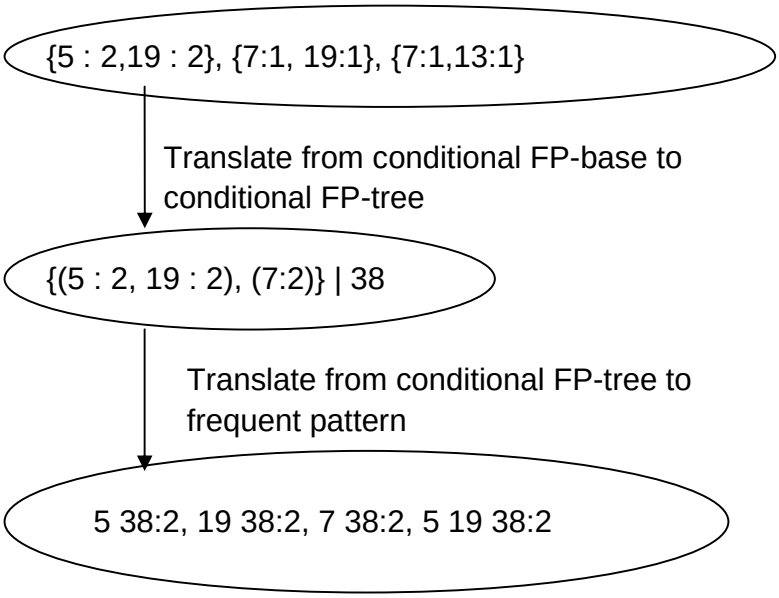

Figure 11 - Translate conditional FP-base to conditional FP-tree

Item	Conditional pattern-base	Conditonal FP-tree	Frequent patterns results
38	{5 : 2,19 : 2}, {7:1, 19:1}, {7:1,13:1}	{(5 : 2, 19 : 2), (7:2)} | 38	5 38:2, 19 38:2, 7 38:2, 5 19 38:2

Table 8 - Conditional FP-tree Table

From the above description, we know “the set of transformed prefix paths of a forms a small database of patterns which co-occur with a. Such a database of pattern occurring with is called a’s conditional pattern-base, and is denoted as ‘pattern_base | a’. Then one can computer all the frequent patterns associated with a in this a-conditional pattern by creating a small FP-tree, called a’s conditional FP-tree and denoted as ‘FP-tree | a’. Subsequent mining can be performed on this small conditional FP-tree. The process is performed recursively [4].

The sequence of getting conditional pattern base is from the number with low frequency to the number with higher one. Base on the Table7 Frequency count of items by ID, we should get the conditional pattern base from 10 to 5.

4.2.2 The Algorithm to Mining Frequency Pattern by FP-Tree

Algorithm [5]:
Procedure FP-growth(Tree, α){

For each header a_i , in the head of Tree do {

Generate pattern $\beta = a_i \cup a$ with

support= a_i .support;

Construct β 's conditional pattern base and

Then β 's conditional FP-tree Tree β

If Tree? <> null

Then call FP-growth (Tree β , β)}

}

There are 4 steps to mining FP-tree. In the above paragraphs, we know what the conditional frequent pattern tree is. We will continue use Figure8-complete FP growth tree structure to explain the mining process.

1) Constructing Frequent Pattern
The last item in the Table7 Frequency count of items by ID is 10. We should get the conditional pattern base from low frequency number to high frequency number, so we first try 10, then go up till 5. There are 3 branches include item 10. The 3 branches are
(5:6, 19:3, 38:2, 17:1, 10:1), (5:6, 7:2, 13:1,10:1) and (21:1, 10:1)

2) Analyze and build conditional FP Base
Now we see 10's prefix part in the three branches. In the first branch, 10's count is 1, so the prefix related to 10 is (5:1, 19:1, 38:1, 17:1). In the second branch, 10's count is also 1, so the prefix in this branch related to 10 is (5; 1, 7:1, 13:1). Same reason, the prefix in third branch related to 10 is (21:1). Write them as follows: (5:1, 19:1, 38:1, 17:1), (5:1, 7:1, 13:1), (21:1). They are 10's conditional FP base.

3) Construct Conditional Frequent Pattern Tree
Now we add all the same item count together from above 3 branches. We get (5:2, 19:1, 38:1, 17:1, 7:1, 13:1, 21:1). Because except 5, all the other items only

show once, and less than support value 2, so only (5:2) is kept. The conditional frequent pattern is {(5:2)}| 10, the frequent pattern is (5, 10:2).

We use the same method to get 46's frequent pattern. In the Figur8 complete FP growth tree structure, there are 3 46s in this tree. (7:1, 19:1, 38:1, 17:1, 21:1, 46:1), (7:1, 17:1, 21:1, 40:1, 46:1) and (7:1, 13:1, 38:1, 46:1). Consider 46's prefix and the item count related to 46, the three branches turn to (7:1, 19:1, 38:1, 17:1, 21:1), (7:1, 17:1, 21:1, 40:1), (7:1, 13:1, 38:1). We calculate the common item counts and delete those items whose counts are less than support value. 19, 40 and 13 are deleted. The three branches turn to (7:1, 38:1, 17:1, 21:1), (7:1, 17:1, 21:1), (7:1, 38:1). We construct new Frequent Pattern Tree based on these 3 branches, it shows on the below:

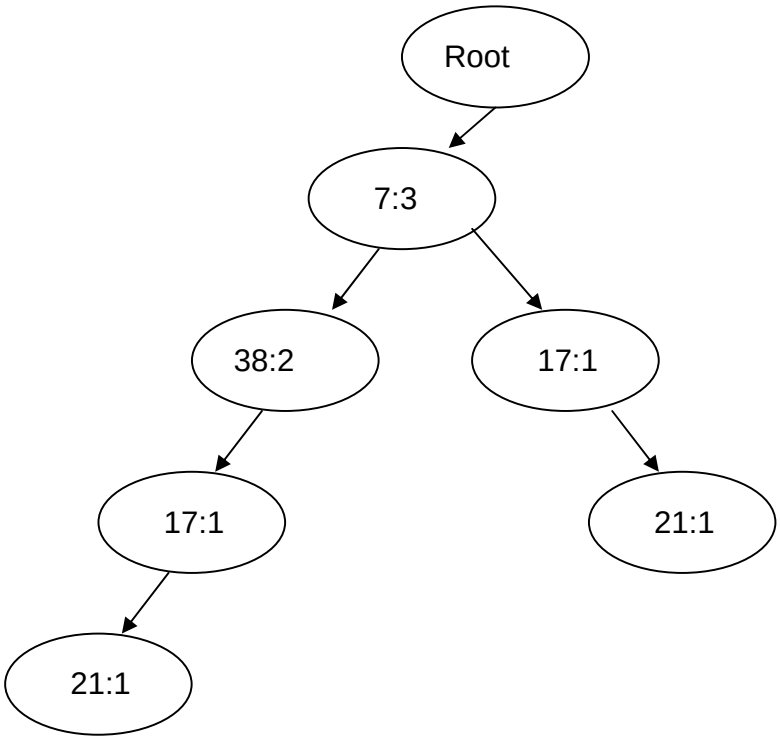

Figure 12 - 46's Tree Branch of Conditional Frequent Pattern Tree

Item	support
7	3
38	2
17	2
21	2

Table 9 - Frequent Patterns Header Table

The following steps are that we use recursive method to mine frequent patterns, produce all combinations. Because all the numbers in two branches are more than threshold 2, so the tree does not need to be simplified. And write them as 7 46:3, 38 46:2, 17 46:2, 21 46:2, 7 38 46:2, 7 17 46:2, 7 21 46:2, 17 21 46:2, 7 17 21 46:2. Generated frequent patterns are the final results we want to get.

ITEM		**Conditional pattern base**	**Conditional FP-Tree**	**frequent patterns generated**
10		{(5:1, 19:1, 38:1, 17:1), (5:1, 7:1, 13:1), (21:1)}	{(5:2)}| 10	5 10:2
46		{(7:1,19:1,38:1,17:1,21:1), (7:1,17:1,21:1,40:1), (7:1,13:1,38:1)}	{(7:1,38:1,17:1,21:1), (7:1,17:1,21:1), (7:1, 38:1)} | 46	7 46: 3, 38 46:2, 17 46:2, 21 46:2, 7 38 46:2, 7 17 46:2, 7 21 46:2, 17 21 46:2, 7 17 21 46:2
40		{5:1,19:1,13:1}, {5:1,7:1}, {7:1,17:1,21:1}	{(5:2, 7:1), (7:1)} |40	5 40:2, 7 40:2
21		{7:1,19:1,38:1,17:1}, {7:1,17:1}	{(7:2,17:2)}|21	7 21:2,17 21:2,7 17 21:2
17		{5:1,19:1,18:1}, {7:1,19:1, 38:1}, {7:1}	{(7:2)} |17	7 17:2, 7,17:2, 19,17:2, 38,17:2 19,38,17:2,
38		{5 : 2,19 : 2}, {7:1, 19:1}, {7:1,13:1}	{(5 : 2, 19 : 2), (7:2)} | 38	5 38:2, 19 38:3, 7 38:2, 5 19 38:2
13		{5:1,19:1},{5:1,7:1},{7:1}	{(5:2,7:1), (7:1)}|13	5 13:2, 7 13:2

19		{5:3,{7:1}}	{(5:3)}|19	5 19:3
7		{5:2}	{5:2}| 7	7 5:2
5		0	0	--------

Table 10 - Final Result of Using Conditional Frequent Pattern Table to Mine Frequent Pattern

5. Bitmap FP-Growth Algorithm

5.1 Overview

Applying bitmap technology to optimize FP-growth algorithm is a novel technology. Refining and mining database through very fast bit operation greatly improve the efficiency of FP-growth algorithm. Bitmap table is like a rectangular matrix with great adaptive character to do the further bit calculation. After applying bitmap technology and analyzing database specialty, we can largely shrink the size of original database and speed the mining computation.

Although FP-growth is an order of magnitude faster than Aprior, but it has the following disadvantages:1) FP-growth tree algorithm is based on sequential computing and is not fit for parallel computing. Every step is based on its previous results. 2) FP-growth tree algorithm is a complicated algorithm. Building and mining FP-growth tree are complex processes. And it is difficult to implement FP-Tree on actual coding because of its tree structure. 3)FP-tree is expensive to build. Many pointers are used to record the linkage between nodes, and some extra storage units are used to record the counters.

We use Bitmap technology to improve FP-Growth Algorithm - Bitmap FP-Growth Algorithm. This algorithm is of the following advantages compared to the original FP-Growth tree algorithm: 1. The mining part of new algorithm can work in parallel environments. A single computer computing resource is limited. If many computers can work on a task simultaneously, the improved efficiency will be ten, even hundreds times of the original one, which depends on the numbers of paralleled computers. 2. The new Bitmap FP-growth algorithm cuts tree building steps of the old algorithm, directly mine frequent patterns. In fact, the relationship of items expressed by tree is hidden in Bitmap table. We can say that the simplified table is a well-built tree. 3. Bit calculation has much specialties. We can improve computing speed by bitmap specialties. For example, we divide one

column into many groups and each group is a 32-bit integer. We use integer to calculate. We simplify frequent itemsets calculation by bitmap specialities. When we mine “c a f e”, only AND all the itemsets of “caf” and “cae”. The clustered bitmap is adequate to mine the corresponding frequent itemsets. We reorder and group the dataset. Reordering rows will not change the bit 1’s total number. Dividing database into groups may make easily mine. 4. Bitmap FP-Growth algorithm can get the same correct results with FP-Growth tree algorithm. We have confirmed this by programming results. The program of Bitmap FP-growth algorithm submitted with the report is a C++ code. Now, let us explain the detailed Bitmap FP-Growth Algorithm.

5.2 Bitmap FP-growth Algorithm Description

Most database design is irregular, the compare and marching from any two transactions is very tedious. Bitmap representation is more standard and well-organized, which fits for large scale scientific computing. After the database translating to bitmap, all the logical operations at the bit level (AND, NOT, OR, SHIFT…) with their constant execution time has the specific application.

The merge sorting in the FP-growth program can be replaced by logical AND operation. Bit operation need less memory and calculate very fast. So we use intensive bit operation to simulate FP-growth process. Using a bit array to replace the behavior of an itemset can easily produce the possibility of combining itemsets[8]. Any logical operator becomes possible to represent association rules, for example, paper => pen $\vee$ eraser. The relation between pen and eraser can be connected by OR. Similarly, NOT is also important in representing association rules. So using bitmap to represent and mine association rule is a natural consideration.

The whole programs obey the following rules:
Bitmap FP-growth Tree Algorithm:
Sort items by their frequency decreasing;

Call FP-growth-Bitmap(TS, ϕ) TS: transaction set

```
FP-Growth-Bitmap(TS, IS){
  If TS is φ then return IS;
    I = { # items in TS >= Threshold}
  foreach i in I:
    call FP-Growth-Bitmap (STS, IS U { i })
    where STS = subsets of TS and item i contained
}
```

5	7	19	13	38	17	21	40	46	10
1	0	1	0	1	1	0	0	0	1
1	1	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	1	0	0
1	0	1	0	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0
0	1	1	0	1	1	1	0	1	0
0	0	0	0	0	0	1	0	0	1
0	1	0	0	0	1	1	1	1	0
0	1	0	1	1	0	0	0	1	0

Table 11 - Bitmap Table of All Items

But we have many innovations based on bitmap characters. Base on the example on last chapter, we write down the bitmap table of all items(Table11). The table can be looked as the well-built tree in the FP-Growth algorithm. It is very necessary to get the longest branch firstly, then get shorter one, then get more shorter ones. The process is mining frequent patterns. During bitmap implementation, it is useful to put the longest branch here. All the other items did “AND” operation with the longest one by one. From bottom (FP-tree lowest leaves) to up(higher item), item itself AND any other higher item. Use lowest frequent item as the first one can cut most unqualified results

I conclude the Bitmap FP-growth tree implementation process includes the following steps:

1. read items file
2. run from bottom to up
3. for each condition (a bitmap) and equivalent class set,
   1) compute AND of itself and any other higher item bitmap (only one from one equivalent item class)
   2) Count 1's in each AND result (by 16-bit -> 1 to speed it up)
   3) Drop underwater items by the count
   4) Sort the counts
   5) Find equal counts as candidates to check/predict AND bitmap equivalent classes
   6) Consider one equivalent class as one item
   7) For each item, add it to the condition, then use its AND bitmap and equivalent classes to repeat 3

The implementation can be simply expressed as the following table(table-12).

5	7	19	13	38	17	21	40	46
1	1	0	1	0	0	0	0	0
1	0	1	0	1	1	0	0	0
0	0	0	0	0	0	1	0	0
5	7	19	13	38	17	21	40	
0	1	1	0	1	1	1	0	
0	1	0	1	1	0	0	0	
0	1	0	0	0	1	1	1	
5	7	19	13	38	17	21		
1	1	0	0	0	0	0		
1	0	1	1	0	0	0		
0	1	0	0	0	1	1		
5	7	19	13	38	17			
0	1	1	0	1	1			
0	1	0	0	0	1			
5	7	19	13	38				
1	0	1	0	1				
0	1	1	0	1				
0	1	0	0	0				
5	7	19	13					
1	0	1	0					
1	0	1	0					
0	1	1	0					
0	1	0	1					
5	7	19						
1	1	0						
1	0	1						
1	0	0						
0	1	0						
5	7							
1	0							
1	0							
1	0							
5								
1								

Table 12 - Whole Structure of Bitmap FP-Growth Algorithm

Each branches can individually compute their results. If the data set is very huge, many calculations based on different branches can be carried out simultaneously on different computers. Operating on the large problem can be divided into smaller ones. The bitmap FP-Growth algorithm can realize mining association

rule in parallel computing environments. Computation efficiency will be 10 times, even 100 hundred times of the original one, which depends on how much computers join the computations.

5.3 Bitmap FP-growth Algorithm Implementation

The following c++ files are main part of implementation: main.cpp, util.cpp, bitmap.cpp, fpgrowther.cpp. The following header files are also important: bitmap.h, fpgrowther.h, utils.h

bitmap.cpp does bit calculation, including the following functions:
bool Set(int i)
Bitmap* AND(const Bitmap* src)
int Count()
float Frequency()

util.cpp reads and parses initial file, including the following functions:
ParseDescFile(const char* filename, vector<ItemInfo>* items, vector<vector<int> >* transactions);
PrintItemInfo(const char* outfile, const vector<ItemInfo>& items);
void PrintTransactions(const char* outfile, const vector<vector<int> >& transactions)

fpgrowther.cpp is a very important part. It realizes the core part of Bitmap FP-Growth Algorithm.
bool Init(const char* filename);
bool Run(const float threshold) { return Run(threshold, NULL); }
void Compute(vector<ItemInfo>* candidate_items);
void FilterAndSortItems(vector<ItemInfo>* candidate_items, vector<ItemInfo>* selected_items);
void PrintOneResult(FILE *fp, const int i);
void PrintConditionItems(const int candidate_size);
void PrintItemPick(const ItemInfo& item, const bool last_candidate);
void PrintFrequencyAndAction(const ItemInfo& item, const char* action);
void PrintSortedSelectedCandidates(const vector<ItemInfo>& selected_items);

bool Set() will set the ith bit.

Init() will call this function to filter out unqualified single items and sort the qualified items by their frequency.

ParseDescFile() will get all the initial items from the original file. The original file includes plenty of mess information. This function will pick up the item information from original files.

void SelectEligibleItems(const unsigned long* bitmap, const vector<int>& items, vector<int>* eligible_items) choose eligible items from item file. Only frequency equal to or more than threshold items will be kept.

sort(eligible_items_count.begin(), eligible_items_count.end(), CompAsc), and ReadSortedItems() realize sorted these eligible items by their decreasing frequency, built new file to store eligible items and read sorted items to the program.

void BitmapAND(const unsigned long* a, const unsigned long* b, unsigned long* c) and void CountAllOnes(const int itemno) realize doing AND operation between two bitmap lines and count 1's quantities. After AND operation, recalculate 1's counts. Not qualified items will be deleted.

Void PrepareBitmaps(map<int, vector<int> >* eligible_items), void CreateBitmaps() realize create bitmap files. The original database is not bitmap file, so I need to do plenty of work to translate the old file to bitmap files. And because the dataset is so huge, I divide them into a few hundreds files and label them.

void StartFPGrowthBitmap() and void FpGrowthBitmap(const bool heuristic_selection, FILE* fp, const unsigned long* bitmap, const vector<int>& items, const vector<int>& condition, int* single_path) are the key functions. They realize FP_Growth algorithm using bitmap idea. The whole mining process is based on FP_Growth technology.

Also there are other functions realize many very necessary work. Such as, void GetAllOneBitmap(unsigned long* bitmap), int GetFirstNumber(char* str), void CountInputAll() and so on.

void CsvStringSplit(char* str, vector<int>* a), int StripLine(char* str, char* stripped_line, int* ones) realize single prefix branch speed up process.

5.4 Skills of Speeding up Mining by Bitmap

During bitmap computation, I also apply some skills to speed up mining according to bitmap and database specialty. For example, I removed most sparse bits which cannot reach the threshold and rearrange the data to

rectangular integer matrices. This 2-dimensional matrix can easily operated using bit computing. And if we represent each column by dividing them into many groups and each group is a 32-bit integer. Using integer to calculate is more fast and convenient.

Here is another good rule to simply frequent itemsets calculation. When we mine “c a f e”, we can only AND all the itemsets of “caf” and “cae”. “The clustered bitmap is adequate to mine the corresponding frequent itemsets. If we collect all such clusters into a cluster tree, the join of frequent itemsets from a parent cluster to a child directly maps to the bit-wise AND of the corresponding bits”[7]. So clustering corresponding frequent itemsets is a good way to save computation. And I also apply some other skills to save computation. If we reorder rows, the bit 1’s total number will not be changed. And the total number of bit 1 corresponds to the support value. So we can divide database into many groups to make the group easily mine. For example, we put all the empty rows into one group, so that the group can be easily erased. Reordering and grouping bitmap are good ways to improve bitmap calculation efficiency. For some special case, a few bit 1s in one row, though these 1 will contribute support value, after AND operation, these rows will turn into empty rows. So it is not necessary to do AND operation for these special rows, just recording 1’s counts and erase these rows.

6. Experiment Results and Analysis

6.1 Program Running Results

After applying bitmap technology to improve FP-growth algorithm, our algorithm is shown to be more efficient and scalable than the original FP-growth algorithm. Especially, the new bitmap FP-Growth algorithm provides parallel computing solution for mining association rule. Our experiments are performed on Intel Core2 Quad CPU Q6600 2.4GHz Desktop machine with 3GB main memory. The machine operating system is Microsoft WindowsXP. Program running environments are Microsoft/Visual C++6.0. We do not choose a database with special attributes, such as spanning too sparse or dense, or there are some requirements on size.

Program execution order: "fpgrowth.exe infile(input file name) threshold(between 0 and 1)". When printing log file to see the detailed process, use order "fpgrowth.exe infile threshold print". idx and .trans file will be produced no matter if inputting "print".

filename. dat: Input file
filename. out: Output file
filename. log: print detailed middle steps
filename. idx: print item ID, description and translated bitmap expression
filename. trans: translate item name to item ID
Programming solution will print out item quantities, transaction quantities, frequency pattern quantities and computing time. Printing log file will influence computation speed.

I did experiments to confirm the correctness of program and algorithm, and also measured its performance. Experiment1 is a short example. Original table only has 7 items and 10 transaction. The result is 19 association rules. This solution is easily to be confirmed. The running time of computing part(no printing log file) is nearly immediate.

Input data(example1):

a b
b c d
a c d e f
a d e
a b c
a b c d
a g
a b c g
a b d
b c e

output data(example1):

[g]: 0.200
[g, a]: 0.200
[e]: 0.300
[e, a]: 0.200
[e, a, d]: 0.200
[e, c]: 0.200
[e, d]: 0.200
[d]: 0.500
[d, b]: 0.300
[d, b, a]: 0.200
[d, b, c]: 0.200

[d, c]: 0.300
[d, c, a]: 0.200
[d, a]: 0.400
[c]: 0.600
[c, a]: 0.400
[c, a, b]: 0.300
[c, b]: 0.500
[b]: 0.700
[b, a]: 0.500
[a]: 0.800

The experiment2 data is comes from Jiawei Han’s presentation. This example is not a perfect example to confirm the correctness of our program, because in that presentation report, Dr. Han use 2 different thresholds in FP-Growth tree building and mining parts in order to limit the size of results. In fact, only one threshold in the whole algorithm is more reasonable. No matter we use either 2 or 3 as our threshold, we will get different results. So we have to use the ordered and selected items as input file and use 2 as threshold to confirm the rightness of mining part. In fact, experiment1 has confirmed the correctness of the whole program. The running time of computing part(no printing log file) is also nearly immediate.

Input data(example2):

5 19 38 17 10
5 7 40
5 19 13 40
5 19 38
5 7 13 10
5 13
7 19 38 17 21 46
21 10
7 17 21 40 46
7 13 38 46

output data(example2, not include single item):

[17, 7]: 0.200
[17, 7, 21]: 0.200
[17, 7, 21, 46]: 0.200
[17, 7, 46]: 0.200

[17, 38]: 0.200
[17, 38, 19]: 0.200
[17, 19]: 0.200
[17, 46]: 0.200
[17, 46, 21]: 0.200
[17, 21]: 0.200
[10, 5]: 0.200
[40, 5]: 0.200
[40, 7]: 0.200
[21, 7]: 0.200
[21, 7, 46]: 0.200
[21, 46]: 0.200
[46, 38]: 0.200
[46, 38, 7]: 0.200
[46, 7]: 0.300
[19, 5]: 0.300
[19, 5, 38]: 0.200
[19, 38]: 0.300
[38, 5]: 0.200
[38, 7]: 0.200
[13, 7]: 0.200
[13, 5]: 0.300
[7, 5]: 0.200

The input file of Example3 is a standard database that coming from the UCI ML repository with 48823 records, around 13.1MB. The running time of computing part(no printing log file) is only 48 millisecond. The original database, running result and log file are partly listed in the appendix.

6.2 Result Analysis

During the experiments, I set different support thresholds from 90% to 10% to measure the program running time, including database preprocessing, computing part and so on. If printing results, program running will take much time. Larger database, lower efficiency FP-growth shows; sparser database, lower efficiency FP-growth shows; lower support threshold, lower efficiency FP-growth shows. In conclusion, Bitmap FP-growth algorithm shows great advantage on more huge, sparse database, especially when calculating low support threshold. The main reason is that bit calculation time is super short.

There are the listed reasons to explain Bitmap FP-growth algorithm‘s good result: 1) After original database changing to bitmap format, it is more easily to master the rule of data, then reduce a large database into a set of smaller ones. And the database is pruned substantially. After unnecessary data are deleted and rearranged, the smaller database usually is 1/5 to 1/100 of the size of original database. 2) During the calculation, the algorithm cut off much of unqualified data, which ensure the candidate set is close to the real result and reduce the later amount of computation. Although FP – growth algorithm also has this process, bitmap format is more easily to carry out this step. 3) Bit calculation saves much time in computing. AND, OR, NOT and SHIFT operations replace the regular counting to do calculation. Longer the row is, the effect is more obvious. Although in the beginning Bitmap FP-Growth takes extra time to transfer the original file to bitmap file, later bit operations are significant faster than FP-growth algorithm. 4)Bitmap technology make association rule mining in parallel computing environments possible. In Bitmap FP-Growth Algorithm, each branches can individually compute their results. If the data set is very huge, many calculations based on different branches can be carried out simultaneously on different computers.

Applying bitmap to FP-growth algorithm is not only a general concept, the key part is how to organize the bitmap, decide which bitmap representation is most effective, how to utilize bitmap computing characters, how to optimize bitmap matrices and mine frequent items.We cannot delete the useful information, and cannot waste computing resource to useless information once and again. Reducing the frequency to open and close database is another detail. We can accumulate many actions to do one-time database load, such as bitmap translation, Bitmap table construction, transaction sorting etc. We should try to reducing the database scanning times, no matter the original or bitmap one. Any huge database scan will take very long time. I/O issue should always be paid more attention. We follow FP-Growth algorithm to avoid many Apriori’s problems, such as multiple database scans, data size of exponentially growing in the intermediate steps.

7. Conclusion and Future Work

In this paper, I first implemented Bitmap Combination Algorithm to mining association rule, then applied bitmap technology to FP-Growth algorithm, creating Bitmap FP-Growth Algorithm and achieve very good experiment results. Translating the original database to the bitmap format, analyzing bit distribution, reducing database size, and using quick bit computing are a few key steps. Bitmap Combination algorithm shows the quick combination skills between any

two, three, four and more rows, then screening the qualified itemsets; Bitmap FP-Growth algorithm is a new algorithm to mine association rule correctly and quickly, and provide the possibility to parallel mining. The two algorithms research and implementation indicate that applying Bitmap technology to compute association rule is a very promising research area. Our experiments results shows great successful in applying bitmap technology in association rule mining working in parallel computing environments.

Bitmap technology has many other special characters to speed up the data mining algorithms, such as bit compression. Although I did not try bit compression/decompression in my experiments, I believe it is a good direction to store and mine database. Some new method will be considered to compress/decompress bitmap database when applying bitmap to do association rule computing. Bitmap technology enables large scale problems solved by the independent, parallel smaller size problems. Bitmap technology will get more widely application in the future association rule mining.

8. Reference

[1]. Silberschatz, Korth, Sudarshan, “Database System Concepts”, 5th Edition, Mc Graw Hill, 2005

[2]. Raghu Ramakrishnan, Johannes Gehrke, “Database Management Systems”, 2nd Edition, Mc Graw Hill, 2002

[3]. T.Y.Lin, Xiaohua Hu, Eric Louie, “A Fast Association Rule Algorithm Based on Bitmap and Granular Computing”, Journal of Applied Intelligence, Oct.2000

[4] Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao, “Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach”, Data mining and knowledge discovery, 8,53-87,2004

[5] Jiawei Han, Jian Pei and Yiwen Yin Presentation Summery, “Mining Frequent Patterns without Candidate Generation”

[6] Christian Borgelt, “An Implementation of the FP-growth Algorithm”

[7] Jianwei Li, Alok Choudhary, Nan Jiang, Wei-keng Liao, “Mining Frequent Patterns by Differential Refinement of Clustered Bitmaps”

[8] Georges Gardarin, Philippe Pucheral, Fei wu, “Bitmap Based Algorithms for Mining Association Rules”

[9] Usama M.”Advances in Knowledge Discovery and Data Mining”

[10] T.Y.Lin, “Data Mining and Machine Oriented Modeling: A Granular Computing Apporach”

[11] Agarwal, R. Aggarwal C., “A Tree Projection Algorithm for Generation of Frequent Itemsets”, Journal of Parallel and Distributed Computing, 2002

[12] Ryszad S., “Machine Learning and Data Mining: Methods and Applications”

[13] Peter C. “Discovering Data Mining: From Concept to Implementation”

[14] Zaki, M.J. and Hsiao, C.J. CHARM: An efficient algorithm for closed itemset mining.” 2002

[15] Pei, J. Han, H.Lu, “Hyper-structure Mining of Frequent Patterns in Large Databases”, The 2001 IEEE Int’s Conference on Data Mining

9. Appendix- Running Result of Bitmap FP-Growth Algorithm

9.1 Example3. dat(part)

age=young workclass=Private education=Bachelors edu_num=13 marital=Married-civ-spouse occupation=Prof-specialty relationship=Wife race=Black sex=Female gain=none loss=none hours=full-time country=Cuba salary<=50K
age=middle-aged workclass=Private education=Masters edu_num=14 marital=Married-civ-spouse occupation=Exec-managerial relationship=Wife race=White sex=Female gain=none loss=none hours=full-time country=United-States salary<=50K
age=middle-aged workclass=Private education=9th edu_num=5 marital=Married-spouse-absent occupation=Other-service relationship=Not-in-family race=Black sex=Female gain=none loss=none hours=half-time country=Jamaica salary<=50K
age=senior workclass=Self-emp-not-inc education=HS-grad edu_num=9 marital=Married-civ-spouse occupation=Exec-managerial relationship=Husband race=White sex=Male gain=none loss=none hours=overtime country=United-States salary>50K
age=middle-aged workclass=Private education=Masters edu_num=14 marital=Never-married occupation=Prof-specialty relationship=Not-in-family race=White sex=Female gain=high loss=none hours=overtime country=United-States salary>50K

age=middle-aged workclass=Private education=Bachelors edu_num=13 marital=Married-civ-spouse occupation=Exec-managerial relationship=Husband race=White sex=Male gain=high loss=none hours=full-time country=United-States salary>50K
age=middle-aged workclass=Private education=Some-college edu_num=10 marital=Married-civ-spouse occupation=Exec-managerial relationship=Husband race=Black sex=Male gain=none loss=none hours=too-many country=United-States salary>50K
age=middle-aged workclass=State-gov education=Bachelors edu_num=13 marital=Married-civ-spouse occupation=Prof-specialty relationship=Husband race=Asian-Pac-Islander sex=Male gain=none loss=none hours=full-time country=India salary>50K
age=young workclass=Private education=Bachelors edu_num=13 marital=Never-married occupation=Adm-clerical relationship=Own-child race=White sex=Female gain=none loss=none hours=full-time country=United-States salary<=50K
age=middle-aged workclass=Private education=Assoc-acdm edu_num=12 marital=Never-married occupation=Sales relationship=Not-in-family race=Black sex=Male gain=none loss=none hours=overtime country=United-States salary<=50K
age=middle-aged workclass=Private education=Assoc-voc edu_num=11 marital=Married-civ-spouse occupation=Craft-repair relationship=Husband race=Asian-Pac-Islander sex=Male gain=none loss=none hours=full-time country=? salary>50K
age=middle-aged workclass=Private education=7th-8th edu_num=4 marital=Married-civ-spouse occupation=Transport-moving relationship=Husband race=Amer-Indian-Eskimo sex=Male gain=none loss=none hours=overtime country=Mexico salary<=50K
age=young workclass=Self-emp-not-inc education=HS-grad edu_num=9 marital=Never-married occupation=Farming-fishing relationship=Own-child race=White sex=Male gain=none loss=none hours=full-time country=United-States salary<=50K
age=middle-aged workclass=Private education=HS-grad edu_num=9 marital=Never-married occupation=Machine-op-inspct relationship=Unmarried race=White sex=Male gain=none loss=none hours=full-time country=United-States salary<=50K
age=middle-aged workclass=Private education=11th edu_num=7 marital=Married-civ-spouse occupation=Sales relationship=Husband race=White sex=Male gain=none loss=none hours=overtime country=United-States salary<=50K
age=middle-aged workclass=Self-emp-not-inc education=Masters edu_num=14 marital=Divorced occupation=Exec-managerial relationship=Unmarried race=White sex=Female gain=none loss=none hours=overtime country=United-States salary>50K
age=middle-aged workclass=Private education=Doctorate edu_num=16 marital=Married-civ-spouse occupation=Prof-specialty relationship=Husband race=White sex=Male gain=none loss=none hours=overtime country=United-States salary>50K
age=senior workclass=Private education=HS-grad edu_num=9 marital=Separated occupation=Other-service relationship=Unmarried race=Black sex=Female gain=none loss=none hours=half-time country=United-States salary<=50K

age=middle-aged workclass=Federal-gov education=9th edu_num=5 marital=Married-civ-spouse occupation=Farming-fishing relationship=Husband race=Black sex=Male gain=none loss=none hours=full-time country=United-States salary<=50K
age=middle-aged workclass=Private education=11th edu_num=7 marital=Married-civ-spouse occupation=Transport-moving relationship=Husband race=White sex=Male gain=none loss=medium hours=full-time country=United-States salary<=50K
age=senior workclass=Private education=HS-grad edu_num=9 marital=Divorced occupation=Tech-support relationship=Unmarried race=White sex=Female gain=none loss=none hours=full-time country=United-States salary<=50K
age=senior workclass=Local-gov education=Bachelors edu_num=13 marital=Married-civ-spouse occupation=Tech-support relationship=Husband race=White sex=Male gain=none loss=none hours=full-time country=United-States salary>50K

9.2 Example3. out(part)

[edu_num=9, salary<=50K]: 0.272
[edu_num=9, salary<=50K, race=White]: 0.227
[edu_num=9, salary<=50K, race=White, country=United-States]: 0.213
[edu_num=9, salary<=50K, race=White, country=United-States, gain=none]: 0.203
[edu_num=9, salary<=50K, race=White, country=United-States, loss=none]: 0.206
[edu_num=9, salary<=50K, race=White, gain=none]: 0.217
[edu_num=9, salary<=50K, race=White, gain=none, loss=none]: 0.210
[edu_num=9, salary<=50K, race=White, loss=none]: 0.220
[edu_num=9, salary<=50K, country=United-States]: 0.250
[edu_num=9, salary<=50K, country=United-States, gain=none]: 0.239
[edu_num=9, salary<=50K, country=United-States, gain=none, loss=none]: 0.232
[edu_num=9, salary<=50K, country=United-States, loss=none]: 0.243
[edu_num=9, salary<=50K, gain=none]: 0.260
[edu_num=9, salary<=50K, gain=none, loss=none]: 0.252
[edu_num=9, salary<=50K, loss=none]: 0.264
[edu_num=9, race=White]: 0.274
[edu_num=9, race=White, gain=none]: 0.255
[edu_num=9, race=White, gain=none, country=United-States]: 0.240
[edu_num=9, race=White, gain=none, country=United-States, loss=none]: 0.230
[edu_num=9, race=White, gain=none, loss=none]: 0.245
[edu_num=9, race=White, country=United-States]: 0.258
[edu_num=9, race=White, country=United-States, loss=none]: 0.248
[edu_num=9, race=White, loss=none]: 0.264
[edu_num=9, country=United-States]: 0.298
[edu_num=9, country=United-States, gain=none]: 0.279
[edu_num=9, country=United-States, gain=none, loss=none]: 0.267
[edu_num=9, country=United-States, loss=none]: 0.287

[edu_num=9, gain=none]: 0.302
[edu_num=9, gain=none, loss=none]: 0.290
[edu_num=9, loss=none]: 0.311
[marital=Never-married]: 0.330
[marital=Never-married, workclass=Private]: 0.251
[marital=Never-married, workclass=Private, race=White]: 0.207
[marital=Never-married, workclass=Private, race=White, loss=none]: 0.202
[marital=Never-married, workclass=Private, country=United-States]: 0.223
[marital=Never-married, workclass=Private, country=United-States, gain=none]: 0.214
[marital=Never-married, workclass=Private, country=United-States, gain=none, loss=none]: 0.208
[marital=Never-married, workclass=Private, country=United-States, gain=none, loss=none, salary<=50K]: 0.202
[marital=Never-married, workclass=Private, country=United-States, gain=none, salary<=50K]: 0.208
[marital=Never-married, workclass=Private, country=United-States, salary<=50K]: 0.214
[marital=Never-married, workclass=Private, country=United-States, salary<=50K, loss=none]: 0.209
[marital=Never-married, workclass=Private, country=United-States, loss=none]: 0.217
[marital=Never-married, workclass=Private, gain=none]: 0.241
[marital=Never-married, workclass=Private, gain=none, salary<=50K]: 0.233
[marital=Never-married, workclass=Private, gain=none, salary<=50K, loss=none]: 0.227
[marital=Never-married, workclass=Private, gain=none, loss=none]: 0.234
[marital=Never-married, workclass=Private, salary<=50K]: 0.241
[marital=Never-married, workclass=Private, salary<=50K, loss=none]: 0.235
[marital=Never-married, workclass=Private, loss=none]: 0.244
[marital=Never-married, race=White]: 0.271
[marital=Never-married, race=White, country=United-States]: 0.249
[marital=Never-married, race=White, country=United-States, salary<=50K]: 0.237
[marital=Never-married, race=White, country=United-States, salary<=50K, gain=none]: 0.230
[marital=Never-married, race=White, country=United-States, salary<=50K, gain=none, loss=none]: 0.223
[marital=Never-married, race=White, country=United-States, salary<=50K, loss=none]: 0.230
[marital=Never-married, race=White, country=United-States, gain=none]: 0.239
[marital=Never-married, race=White, country=United-States, gain=none, loss=none]: 0.231
[marital=Never-married, race=White, country=United-States, loss=none]: 0.241
[marital=Never-married, race=White, salary<=50K]: 0.257
[marital=Never-married, race=White, salary<=50K, gain=none]: 0.250
[marital=Never-married, race=White, salary<=50K, gain=none, loss=none]: 0.243
[marital=Never-married, race=White, salary<=50K, loss=none]: 0.250
[marital=Never-married, race=White, gain=none]: 0.259
[marital=Never-married, race=White, gain=none, loss=none]: 0.251
[marital=Never-married, race=White, loss=none]: 0.262
[marital=Never-married, country=United-States]: 0.296

[marital=Never-married, country=United-States, salary<=50K]: 0.283
[marital=Never-married, country=United-States, salary<=50K, gain=none]: 0.274
[marital=Never-married, country=United-States, salary<=50K, gain=none, loss=none]: 0.266
[marital=Never-married, country=United-States, salary<=50K, loss=none]: 0.275
[marital=Never-married, country=United-States, gain=none]: 0.284
[marital=Never-married, country=United-States, gain=none, loss=none]: 0.275
[marital=Never-married, country=United-States, loss=none]: 0.287
[marital=Never-married, salary<=50K]: 0.315
[marital=Never-married, salary<=50K, gain=none]: 0.306
[marital=Never-married, salary<=50K, gain=none, loss=none]: 0.297
[marital=Never-married, salary<=50K, loss=none]: 0.306
[marital=Never-married, gain=none]: 0.316
[marital=Never-married, gain=none, loss=none]: 0.306
[marital=Never-married, loss=none]: 0.320
[sex=Female]: 0.332
[sex=Female, hours=full-time]: 0.213
[sex=Female, hours=full-time, gain=none]: 0.201
[sex=Female, hours=full-time, loss=none]: 0.206
[sex=Female, workclass=Private]: 0.237
[sex=Female, workclass=Private, country=United-States]: 0.213
[sex=Female, workclass=Private, country=United-States, gain=none]: 0.202
[sex=Female, workclass=Private, country=United-States, loss=none]: 0.206
[sex=Female, workclass=Private, salary<=50K]: 0.215
[sex=Female, workclass=Private, salary<=50K, gain=none]: 0.208
[sex=Female, workclass=Private, salary<=50K, gain=none, loss=none]: 0.202
[sex=Female, workclass=Private, salary<=50K, loss=none]: 0.209
[sex=Female, workclass=Private, gain=none]: 0.225
[sex=Female, workclass=Private, gain=none, loss=none]: 0.218
[sex=Female, workclass=Private, loss=none]: 0.230
[sex=Female, race=White]: 0.267
[sex=Female, race=White, salary<=50K]: 0.235
[sex=Female, race=White, salary<=50K, country=United-States]: 0.217
[sex=Female, race=White, salary<=50K, country=United-States, gain=none]: 0.209
[sex=Female, race=White, salary<=50K, country=United-States, gain=none, loss=none]: 0.203
[sex=Female, race=White, salary<=50K, country=United-States, loss=none]: 0.211
[sex=Female, race=White, salary<=50K, gain=none]: 0.226
[sex=Female, race=White, salary<=50K, gain=none, loss=none]: 0.220
[sex=Female, race=White, salary<=50K, loss=none]: 0.229
[sex=Female, race=White, country=United-States]: 0.247
[sex=Female, race=White, country=United-States, gain=none]: 0.231
[sex=Female, race=White, country=United-States, gain=none, loss=none]: 0.222

[sex=Female, race=White, country=United-States, loss=none]: 0.238
[sex=Female, race=White, gain=none]: 0.250
[sex=Female, race=White, gain=none, loss=none]: 0.241
[sex=Female, race=White, loss=none]: 0.257
[sex=Female, salary<=50K]: 0.295
[sex=Female, salary<=50K, country=United-States]: 0.266
[sex=Female, salary<=50K, country=United-States, gain=none]: 0.256
[sex=Female, salary<=50K, country=United-States, gain=none, loss=none]: 0.249
[sex=Female, salary<=50K, country=United-States, loss=none]: 0.259
[sex=Female, salary<=50K, gain=none]: 0.285
[sex=Female, salary<=50K, gain=none, loss=none]: 0.276
[sex=Female, salary<=50K, loss=none]: 0.287
[sex=Female, country=United-States]: 0.299
[sex=Female, country=United-States, gain=none]: 0.281
[sex=Female, country=United-States, gain=none, loss=none]: 0.271
[sex=Female, country=United-States, loss=none]: 0.289
[sex=Female, gain=none]: 0.312
[sex=Female, gain=none, loss=none]: 0.301
[sex=Female, loss=none]: 0.320
[relationship=Husband]: 0.404
[relationship=Husband, hours=full-time]: 0.215
[relationship=Husband, hours=full-time, loss=none]: 0.203
[relationship=Husband, hours=full-time, loss=none, marital=Married-civ-spouse]: 0.203
[relationship=Husband, hours=full-time, loss=none, marital=Married-civ-spouse, sex=Male]: 0.203
[relationship=Husband, hours=full-time, loss=none, sex=Male]: 0.203
[relationship=Husband, hours=full-time, marital=Married-civ-spouse]: 0.215
[relationship=Husband, hours=full-time, marital=Married-civ-spouse, sex=Male]: 0.215
[relationship=Husband, hours=full-time, sex=Male]: 0.215
[relationship=Husband, salary<=50K]: 0.223
[relationship=Husband, salary<=50K, gain=none]: 0.210
[relationship=Husband, salary<=50K, gain=none, loss=none]: 0.203

9.3 Example3. log(part)

Pick loss=none (Frequency 0.110).
New Result [occupation=Craft-repair, gain=none, loss=none]: 0.110
Tried all candidates. Go back to a higher level.

Pick sex=Male (Frequency 0.119).
New Result [occupation=Craft-repair, sex=Male]: 0.119
Create new candidate items:
  loss=none: Frequency 0.113, Selected.

Sort 1 selected candidate items:
  loss=none: Frequency 0.113

Pick loss=none (Frequency 0.113).
New Result [occupation=Craft-repair, sex=Male, loss=none]: 0.113
Tried all candidates. Go back to a higher level.

Pick loss=none (Frequency 0.120).
New Result [occupation=Craft-repair, loss=none]: 0.120
Tried all candidates. Go back to a higher level.

Pick occupation=Prof-specialty (Frequency 0.126).
New Result [occupation=Prof-specialty]: 0.126
Create new candidate items:
  loss=none: Frequency 0.117, Selected.
  gain=none: Frequency 0.109, Selected.
  country=United-States: Frequency 0.113, Selected.
  race=White: Frequency 0.111, Selected.
  salary<=50K: Frequency 0.069, Filtered.
  workclass=Private: Frequency 0.070, Filtered.
  sex=Male: Frequency 0.080, Filtered.
  hours=full-time: Frequency 0.066, Filtered.
  age=middle-aged: Frequency 0.073, Filtered.
  marital=Married-civ-spouse: Frequency 0.065, Filtered.
  relationship=Husband: Frequency 0.055, Filtered.
  sex=Female: Frequency 0.046, Filtered.
  marital=Never-married: Frequency 0.038, Filtered.
  edu_num=9: Frequency 0.007, Filtered.
  education=HS-grad: Frequency 0.007, Filtered.
  age=young: Frequency 0.025, Filtered.
  hours=overtime: Frequency 0.044, Filtered.
  relationship=Not-in-family: Frequency 0.039, Filtered.
  salary>50K: Frequency 0.057, Filtered.
  edu_num=10: Frequency 0.013, Filtered.
  education=Some-college: Frequency 0.013, Filtered.
  age=senior: Frequency 0.026, Filtered.
  edu_num=13: Frequency 0.046, Filtered.
  education=Bachelors: Frequency 0.046, Filtered.
  relationship=Own-child: Frequency 0.010, Filtered.
  marital=Divorced: Frequency 0.016, Filtered.

Sort 4 selected candidate items:
  loss=none: Frequency 0.117
  country=United-States: Frequency 0.113
  race=White: Frequency 0.111
  gain=none: Frequency 0.109

Condition Items: occupation=Prof-specialty
4 candidate items under this condition(threshold 0.10)

Pick gain=none (Frequency 0.109).
New Result [occupation=Prof-specialty, gain=none]: 0.109
Create new candidate items:
  loss=none: Frequency 0.100, Selected.
  country=United-States: Frequency 0.097, Filtered.
  race=White: Frequency 0.096, Filtered.

Sort 1 selected candidate items:
  loss=none: Frequency 0.100

Pick loss=none (Frequency 0.100).
New Result [occupation=Prof-specialty, gain=none, loss=none]: 0.100
Tried all candidates. Go back to a higher level.

Pick race=White (Frequency 0.111).
New Result [occupation=Prof-specialty, race=White]: 0.111
Create new candidate items:
  loss=none: Frequency 0.103, Selected.
  country=United-States: Frequency 0.103, Selected.

9.4 Example3. idx(part)

Id, GroupId, Desc, Bitmap
100, 2, education=Preschool,
00000000000000000000000000000000000000000000000000000000000000000000000000000000100000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000001000000000000000000000000000000000000000000000000000000000000000000000100000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000100000000000000000
00000000000000000000000000000000000000000000000010000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000010000000000000000000
00000000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000100000000000000000000
00000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000010000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000001000000000000000000000000000000000000000000000000000000000000000000000000010000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000
00000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000001000000000000000000000000000000000000000000000000000000000100000000000000000000000000000
00000000000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000
00000000010000000000000000000000000000100000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000100000000000100000000000000000000000000000000000000000000000000000000000000000
00000000000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000
00000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000

9.5 Example3. trans(part)

0 21 36 37 38 39 6 28 32 19 10 20 40 13
14 15 22 23 16 17 18 7 8 19 10 41 12 42
0 21 34 35 4 30 6 7 32 43 10 41 12 42
0 21 2 3 16 17 18 7 8 43 10 11 12 42
0 21 44 45 16 17 18 28 8 19 10 46 12 42
0 1 2 3 16 30 18 47 8 19 10 11 48 42
29 21 2 3 4 5 49 7 32 19 10 11 12 13
0 21 50 51 4 52 6 28 8 19 10 41 12 13
0 21 53 54 16 55 18 47 8 19 10 11 56 42
0 21 57 58 16 59 18 60 8 19 10 41 61 13
29 15 22 23 4 62 49 7 8 19 10 11 12 13

0 21 22 23 4 63 64 7 8 19 10 11 12 13
0 21 26 27 16 52 18 7 8 19 10 41 12 13
0 15 34 35 24 17 64 7 32 19 10 41 12 42
0 21 65 66 16 30 18 7 8 19 10 41 12 42
14 21 22 23 67 39 64 28 32 19 10 20 12 13
0 68 36 37 16 62 18 28 8 19 10 11 12 13
0 21 26 27 16 59 18 7 8 19 69 11 12 13
14 21 22 23 24 70 64 7 32 19 10 11 12 13
14 71 2 3 16 70 18 7 8 19 10 11 12 42
29 21 22 23 4 55 49 7 8 19 10 11 12 13
14 72 44 45 16 73 18 47 8 19 10 41 74 42
0 21 22 23 24 17 6 7 8 19 10 46 12 13
0 21 22 23 16 55 18 7 8 19 10 11 12 13
29 71 50 51 4 75 6 7 8 19 10 41 12 13
29 21 44 45 4 52 49 28 8 19 10 41 12 13
0 21 2 3 24 17 49 7 8 19 76 11 12 13
0 68 44 45 16 5 49 7 8 19 10 11 12 13
29 1 44 45 16 39 18 28 8 19 10 20 12 13
0 21 26 27 4 63 64 7 8 19 10 11 77 13
29 21 44 45 4 63 49 7 8 19 10 11 12 13
29 21 22 23 78 5 31 7 32 19 10 20 12 13
0 21 44 45 16 52 18 7 8 19 10 11 56 42
0 15 50 51 16 30 18 7 8 19 10 11 12 13
0 21 36 37 16 63 18 7 8 19 10 41 12 13