TAXONOMY-BASED PRUNING IN GENERALIZED FREQUENT ITEMSETS MINING

LinLin Ma

A Thesis

In

The Department

Of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of

Master of Computer Science at

Concordia University

Montréal, Québec, Canada

March 2012

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: LinLin Ma

Entitled: Taxonomy-based Pruning in Generalized Frequent Itemsets Mining And

submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science Degree

Complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________________ Chair
Dr. B. C. Desai

______________________________ Examiner
Dr. N. Shiri

______________________________ Examiner
Dr. Y. Yan

______________________________ Supervisor
Dr. Gosta Grahne

Approved by ______________________________
Chair of Department or Graduate Program Director

____
Dr. Robin A. L. Drew, Dean
Faculty of Engineering and Computer Science

Date ______________________________

ABSTRACT

The orignal purpose of data mining is for analysis of supermarket transaction data. Now with the rapid development in business, industry and science, data mining is used in lots of domains, so mining interesting information from large database becomes more important. Data mining includes two main parts: frequent itemsets mining and association rules mining. And frequent itemsets mining plays an essential role between them.

Our thesis is focused on frequent itemsets mining. Previous studies on frequent itemsets mining is at single or multiple concept level, however, mining frequent itemsets at flexible multiple concept level may help finding more specific and useful information from huge data. In this thesis, four methods are introduced for mining frequent itemsets at flexible multiple level by extension of Apriori and Eclat algorithms. We also implement two algorithms for frequent pairs mining. We draw some conclusions about which method is suitable for which distributions of data.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr. Gosta Grahne, whose precious guidance, support and encouragement were pivotal in establishing my self-confidence in this endeavor. I would also thank my fellow researcher Dr.Adrain Onet for his help and encouragement. He spent a lot of time on my research and gave me lots of good ideas. He contributed substantially to the completion of my thesis.

I would also like to thank my friends who give my lots of support and help.

Finally, I wish to express my gratitude to my family members - my parents, my parents-in-law, my husband and my son for their love and support.

Contents

List of Tables

List of Figures

LIST OF ACRONYMS

DB	Database
TID	Transcation ID
ET-apriori	Expanded Trascation Apriori Mining
LP-apriori	Level Pruning Apriori Mining
ET-Elcat	Expanded Trascation Elcat Mining
LP- Elcat	Level Pruning Elcat Mining
ET-Pairs	Expanded Trascation Pairs Mining
LP- Pairs	Level Pruning Pairs Mining
FIM	Frequent Itemset Mining
TDB	Transaction Database
Min-sup	Minimum Support

Chapter 1

Introduction

1.1 Description of problem

The discovery of frequent itemsets is at the core of many data mining tasks, such as association rules, correlations, classifiers, clusters, etc. An item can be a consumer product, a medical symptom, a word in a document, a webpage, etc. Our prototypical application is the "market basket", in which the items are consumer products that are bought by a customer in a transaction. The task is then to find all sets of items that are frequent, i.e. they occur in at least a given fraction σ of the transactions. The quantity σ can also be given as an absolute number, which is called the *minimum support*, or *min-sup*. Since its inception in Agrawal et al. [6], mining frequent itemsets has been the focus of intense research that has resulted in countless algorithms and publications.

In many applications, the items can be aggregated into categories, which can be further aggregated until a taxonomy suitable for the domain is obtained. For example, 'skim-milk' belongs to the category of 'milk', which belongs to the category of 'food'. The word 'wonder bread' belongs to the category of 'bread', the word 'bread' belongs to the category of 'food', and so on. It is clear that having a taxonomy available will allow for the discovery of more fine-grained sets of items. For instance, it can be more valuable to know that {white bread, milk} is frequent, than simply knowing that "higher-level" itemset {bread, milk} is frequent. Single-level frequent itemset mining will generate one concept level itemset like {bread, milk}. Multi-level frequent itemset mining will generate itemsets whose items are at the same level, e.g. {white bread, 2% milk} or {bread, milk}.

In this paper we study the problem of mining mix-level frequent itemsets. Since, for instance, there are naturally more 'milk' items than 'Lactaid Milk' items, the *min-sup* σ might depend on the level of the taxonomy. Furhter-more, some measures could also be used to determine the "interestingness" of a mix-level frequent itemset, allowing us, for example, to discard the itemset {white bread, milk} in favor of the more "interesting" itemset {bread, Lactaid Milk}, or vice versa. There is, however, no universally agreed interestingness measure (for a survey, see [15]). We, therefore, make the simplified assumption that the *min-sup* σ is the same for each level, and consider the problem of mining all frequent mix-level itemsets.

Related work: Frequent itemsets mining is to discover the useful patterns from the databases. It is an important and progressive topic in the field of Data Mining. It was first presented in [4]. Agrawal's Apriori [4], Han's FP-Growth [21] and Mohammed J. Zaki's Eclat [40] are considered as three of the most significant contributions in data mining.

Later lots of research based on these three basic algorithms are proposed. [11; 28; 33; 37] use the whole structures and procedures of Apriori. Zaki[39] gets some pruning on Eclat. MAFIA [12] and SPAM [9] use vertical bit-vectors for fast itemset and sequence mining respectively, are also considered a vertical format. [2; 3; 17; 21] are based on FP-Growth. For more detail, some work on association rules maintenance [8; 13; 14; 36], episode mining [25], mining sequential patterns [7; 35], discovering functional and approximate dependencies [22; 23] Previous work has been focused on single concept level [4; 6; 10; 16; 21; 39; 40] or multiple concept levels frequent itemsets mining [18; 20; 26; 34]. Little work has been done in the flexible multiple concept levels. For example, if we have a taxonomy for the product like {Category, Brand, Content}, the former research can generate symmetrical frequent itemsets of 70% customers that buy {bread, milk} or {white bread,2% milk},all the items in a itemset are in the same level. But now one may be interested in finding frequent itemsets with alternative, multiple hierarchies. So we give the applications which finding frequent itemsets at flexible concept level. For example, they can generate asymmetric frequent itemsets of 70% customers that buy {Pom white bread,Quebon milk}. R. Srikan has introduced an algorithm [34] which can generate the flexible concept level frequent itemsets based on Apriori like our ET-apriori. Later, Runying Mao has introduced a method that can generate the flexible concept level frequent itemsets base on FP-Growth [26]. But it is limited by the concept level due to the FP-Growth's data structure. That means it is only suitable for small concept level and small number of items. To restrict the frequent itemsets, we would introduce the concept of minimum support (*min-sup* hereafter)which has been defined in Agrawal's paper [4]. Informally, the support of a pattern A in a set of transactions S is the probability that pattern A occurs in S.

TID	Items
T1	111,212 ,112,222,312
T2	312,113,231
T3	111,212,312,121,232
T4	212,211,311
T5	111, 212,312,221,321
T6	111, 312,322,412

Table 1.1: Encoded Original Transaction table

1.2 Taxonomy of the Product Schema

Most of time, products are organized as hierarchies of their attributes. A simple hierarchy example is: ‘milk’, ‘Quebon milk’ and ‘2% Quebon milk’. In the presence of hierarchies, we denote ‘milk’ $\succeq$ ‘Quebon milk’ $\succeq$ ‘2% Quebon milk’. In other words, the count of ‘milk’ in transaction dataset is greater than ‘Quebon milk’ , and ‘Quebon milk’ is greater than ‘2% Quebon milk’. We show this hierarchy relation in a drill-down process in Figure 1.1

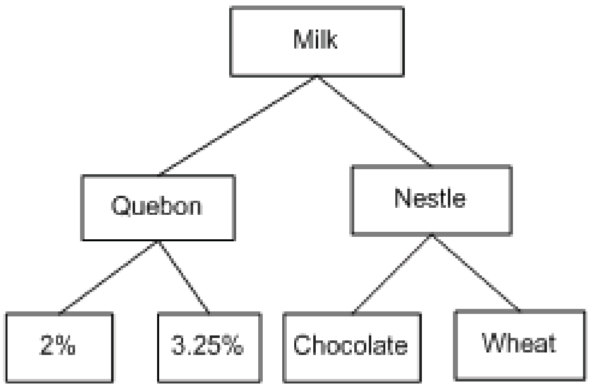


Figure 1.1: Hierarchy of Milk attributes

1.3 Encoded mode

First of all, we need to construct the taxonomy of the Product, and each position in this product hierarchy should be given a unique encoded digit which requires fewer bits than the corresponding food-identifier. We assume that one shopping transaction database contain three parts: 1) One item data set which contains the description of each product item in $\mathcal{I}$ in the form of $(P_i,\ Name)$, where encoded digit $P_i \in \mathcal{I}$, 2) a customer transaction table like Table 1.1, which consists of a set of $(TID_i, \{P_x,\ \ldots,\ P_y\})$, where TID_i is a transaction identifier and $P_i \in \mathcal{I}$ (for $i = x,\ \ldots,\ y$), 3) the notation of $'*'$ represents a class of object in an encoded digit P_i. To clarify above, an abstract example is illustrated below.

Example 1 *An instance that shows the taxonomy information of food schema with hierarchy is shown in Figure 1.2. Let "Category" represent the first-level concept, "Brand" for the second level, and "Content" for the third level. Thus we can represent a product by one unique encoded digit. For example,'Ground Café of Van' will be encoded as '311' in which the first digit, '3' represents 'Café' at level1, the second digit '1' for the 'Van' at level2, the last '1' for the content 'Ground' at level3. Then by using this taxonomy tree, we can convert the customers shopping transaction database to the encoded a customer transaction table as Table 1.1.*

In our research, we focus on the flexible multiple level frequent item sets mining, which releases the restriction of mining among the concepts at the same level of a hierarchy. It may generate frequent set like '2% Quebon milk', 'Pom bread' ({111, 22*}) in which the two items are at different levels of a hierarchy.

In Figure 1.3 we show a simple example for 'milk' and 'Café' hierarchy relation and their encoding.

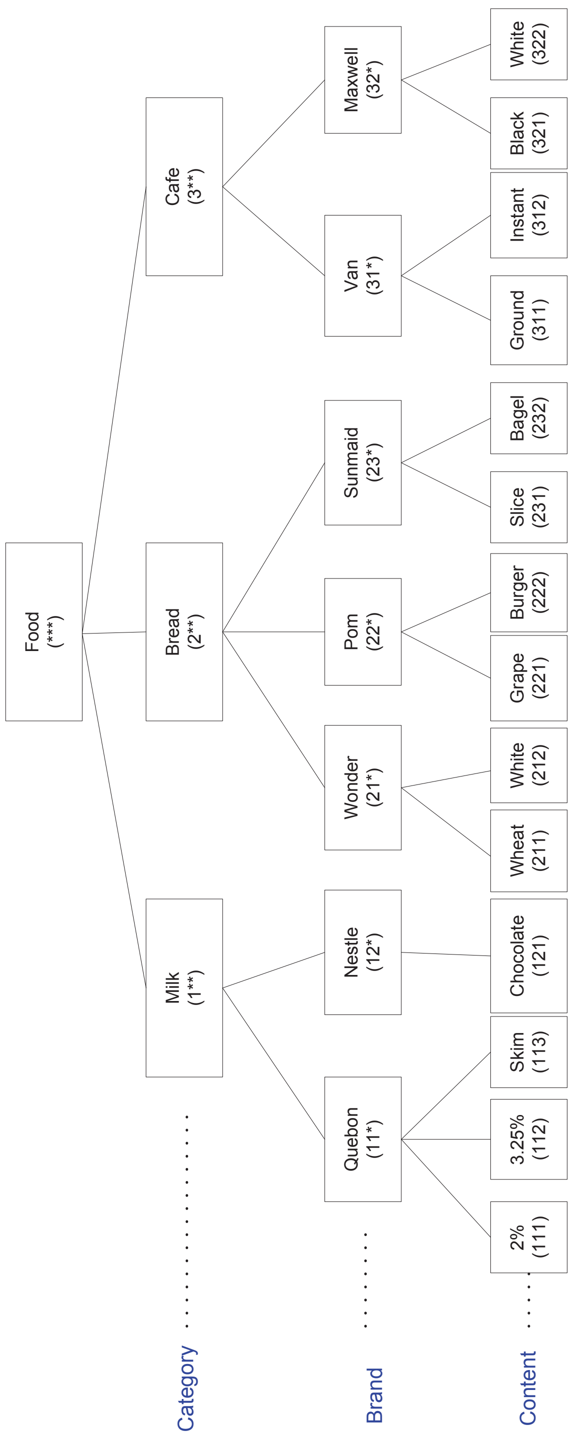


Figure 1.2: An example of a product schema

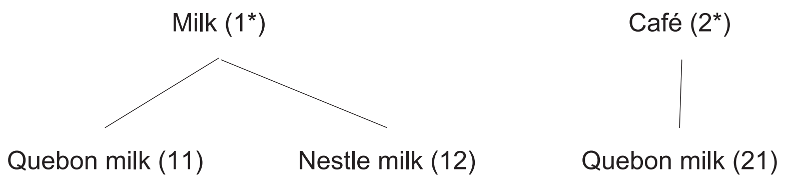


Figure 1.3: A hierarchy relation of Milk and Café with their encoding

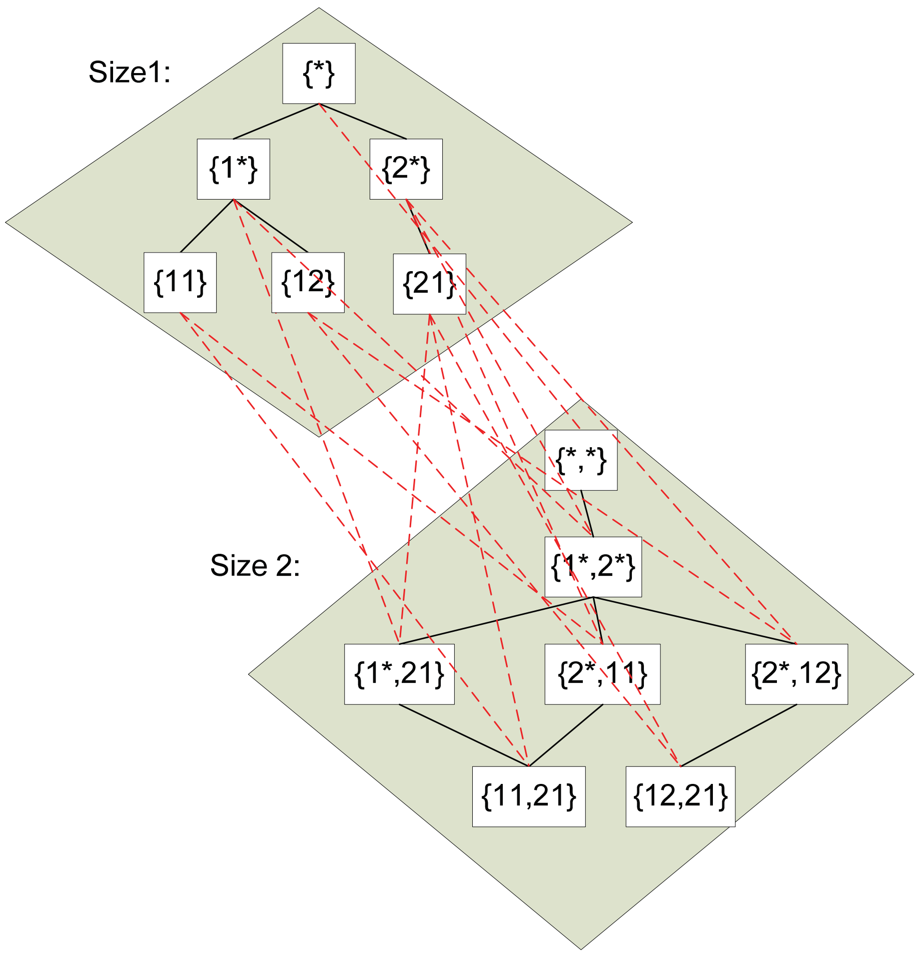


Figure 1.4: The candidates' composite lattices

Then we give the size one and size two candidates' composite lattices for multiple, hierarchical dimensions as shown in Figure 1.4 using the example in Figure 1.3.

1.4 Motivations

Even though more than a decade of study over frequent itemsets mining, it has been noticed that traditional mining methods can not meet today's needs. We need more detailed information and more efficient mining methods. First,it is more interesting to mine the flexible multiple concept levels frequent patterns. For example, besides that finding 60% customers buy 'milk' and 'bread' together, it will be more desirable that 50% customers buy 'Quebon 2% milk' and 'whole wheat bread' together. Second, some of the algorithms concentrate on multiple concept levels, which can only find symmetrical information. For example, it can find 'Quebon milk' and 'pom Bread'. Last, few algorithm can mine flexible multiple concept levels frequent itemset based on Apriori and FP-growth. For the former, it just extended transcations with low efficiency. And for the second, since FP-growth has a complicated data structure, it has a very low efficiency for mining high level data structure.

1.5 Our contribution

In this study, we extend the itemset mining from single level, muiltiple level to flexible multiple level. And we analyze some previous research work. Since less work was done in flexible muiltiple levels itemset mining, we try lots of methods to tracle this problem, and finally four algorithms ET-apriori, LP-apriori, ET-Eclat and LP-eclat are developed and examined. All of them can solve the multiple level frequent itemsets mining problem. The experimental results indicate that certain algorithms could be fastest for certain kinds of data distributions. We also implemented two fast algorithms for mining frequent pairs. One of them using hierarchy structure is shown to be a faster one in most of the cases.

1.6 Thesis Organization

This document is organized as follows. In chapter 1, the problem is proposed. In chapter 2, we introduce the related work. In chapter 3 we introduce some definitions and the properties of the item or itemsets. In chapter 4, four algorithms of flexible multiple level frequent itemsets mining are studied. And two algorithms of mining frequent pairs are impelemented. In chapter 5, experimental results show the performance of four algorithms which can help us choose the relevant algorithm for different distribution of the data. The future work and the conclusion are presented in chapter 6 and 7.

Chapter 2

Preliminaries

2.1 Single level frequent itemset mining

In the recent years many algorithms on mining single-level frequent itemset have been proposed. Each of them has its distinctive merits. Practically, all of them can be classified into three different basic algorithms: Apriori, FP-growth and Eclat, which will be introduced in the following subsections.

2.1.1 Apriori

Apriori is considered as the first FIM algorithm[4] proposed by Rakesh Agrawal and Ramakrishnan Srikant from IBM Almaden Research Center.
A detailed description of Apriori can refer to [5]. Briefly, it uses a bottom-up strategy for the traversal of the search space, i.e., beginning from an empty set and then generate $k-itemsets$ (there are k items in it and k from 1). It performs an iterative approach to find $k+1-itemsets$ using $k-itemsets$. First it scans the database and generates the frequent $1-itemset$ and then it generates candidates

TID	**Items**
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

Table 2.1: Encoded Transaction table

of $k - itemsets$ and scans the database to find frequent k-itemsets until the candidates of $k+1-itemsets$ is NULL. The total number of the full scan of the database is $k+1$ times.

Example 2 *This example illustrates the Apriori Algorithm. A transaction database TDB is given in Table 2.1.* $< 10, \{A,\ C,\ D\} >$ *is a transaction, in which 10 is the transaction identifier, and* $\{A,\ C,\ D\}$ *is a set of items, which can also be denoted as* ACD.

Given an absolute min-sup equals to 2. Figure 2.1 lists the steps how frequent patterns in TDB are generated using the Apriori algorithm.

Then the frequent patterns in TDB we can get are:

$\{A\}$*:2 ;* $\{B\}$*:3 ;* $\{C\}$*:3 ;* $\{E\}$*:3 ;*

$\{A,\ C\}$*:2;* $\{B,\ C\}$*:2;* $\{B,\ E\}$*:3;* $\{C,\ E\}$*:2;*

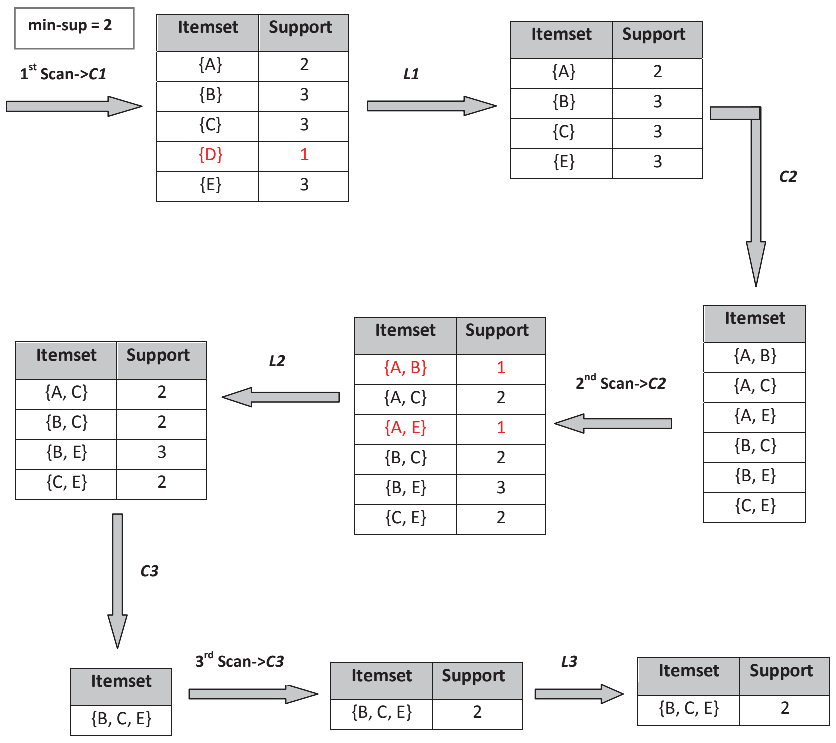
$\{B,\ C,\ E\}$*:2.*

The total number of the database scan is 3.

The pseudo-code for the Apriori algorithm is given as below in Table 2.2.

The algorithm mainly includes three steps: join step, prune step and stop step.

- The join step:



Itemset	Support
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

Itemset	Support
{A}	2
{B}	3
{C}	3
{E}	3

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

Itemset	Support
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

Itemset	Support
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

Itemset
{B, C, E}

Itemset	Support
{B, C, E}	2

Itemset	Support
{B, C, E}	2

Figure 2.1: Frequent patterns in TDB generated by Apriori

Apriori Algorithm:

```
1: $\mathcal{C}_k$ : Candidate itemset of size $k$
2: $\mathcal{F}_k$ : frequent itemset of size $k$
3: $\mathcal{F}_1$ : = {frequent items}
4: for $(k = 1;\ \mathcal{F}_k \neq \emptyset; k++)$ do begin
5:     $\mathcal{C}_{k+1}$ = candidates generated from $\mathcal{F}_k$;
6:     for each transaction $t$ in database do
7:         increase the count of all candidates in $\mathcal{C}_{k+1}$ that are contained in $t$
8:     $\mathcal{F}_{k+1}$= candidates in $\mathcal{C}_{k+1}$ which $\mathcal{C}_{k+1}.suppot \geq min - sup$
9: end
10: Return $\bigcup_k \mathcal{F}_k$
```

Table 2.2: Pseudo code of Apriori algorithm

  - Find $\mathcal{F}_k$ in line 5, the set of candidate of $k - itemsets$, join $\mathcal{F}_{k-1}$ with itself.
  - Rules for joining: 1.Order the items first so you can compare item by item 2.The join of $\mathcal{F}_{k-1}$ is possible only if its first $(k-2)$ items are in common

- The Prune step:
  - The "join" step will produce all $k - itemsets$, but not all of them are frequent.
  - Scan DB in line 6 to see which itemsets are indeed frequent and discard the others.

- Stop when "join" step produces empty set in line 4.

Based on the Apriori algorithm, several optimized algorithms are proposed to improve the performance by adding more specific techniques while keeping the same candidates generation structure.

AprioriTid, AprioriHybrid

Along with the basic Apriori Algorithm, Agrawal et al.[4] proposed two other algorithms, AprioriTid and AprioriHybrid. AprioriTid replaces all the transactions in the database by the set of candidate itemsets that occur in that transaction. It repeats this process in every iteration k as detailed in [6]. AprioriTid is much faster in the later iterations when there are less frequent patterns, it can convert the dataset much smaller, but it is much slower than Apriori in the early iterations. For improvement, another algorithm AprioriHybrid was proposed by Agrawal et al.[4]. It combines Apriori and AprioriTid, and lets the algorithm decide to switch between Apriori and AprioriTid.

Combining passes

Another enhancement, tries to scan as many iterations as possible in once when only few candidate patterns can be generated at the higher iterations. This combination technique was mentioned in [6].

Sampling

Since Apriori Algorithm relies on multiple database scans, the Sampling algorithm, proposed by Toivonen [37], uses at most two scans through the database by selecting a random sample from the database, and then uses this sample to predict all the possible frequent patterns of the whole database, and verifies the results with the rest of the database. However, the performance of the Sampling algorithm highly depends on the sample extracted from the database. Actually transactions databases are seldom uniform distributed, hence itemsets that are frequently appeared in the sample might turn to be infrequent in the whole database.

2.1.2 FP-growth

Later, a depth-first algorithm, FP-growth, was proposed by Han [21]. FP-growth uses a trie structure to store the database in the main memory. It uses a compressed representation of the database by an FP-tree. Once an FP-tree is constructed, it uses a recursive divide-and-conquer approach to mine the frequent itemsets.

- First step: it scans data to determine the count of each item. Then it sorts items in the database in ascending order.
- Second step: it makes a second pass over the data to construct the FP-tree. It creates the "nul" root node of the tree. For each transaction in the ordered database, a branch is added for each transaction. Each node in the FP-tree also stores a counter which keeps track of the number of transactions that share that node. When it adds a branch to the FP-tree, it follows the rules that if there already exists a common prefix, it increases the count of the common node and adds new nodes to the FP-tree. At the same time, maintaining a header table that each node on the tree has a link via the same node to the header table.

  Additionally, the header table also has a support record for each item. Why we store the transaction items in support of descending order? Because

TID	**Items**
10	C, A
20	B, C, E
30	B, C, E, A
40	B, E

Table 2.3: Ordered Transaction table

we can save the database using smallest space since the more frequently occurring items are arranged closer to the root of the FP-tree and thus are more likely to be shared.

- Third step: mining frequent item pattern uses a partition-based, divide-and conquer method rather than Apriori-like bottom-up generation of frequent patterns combinations.

Inherently, it converts the problem of finding long frequent patterns to look for shorter ones and then join with the suffix.
Thus, FP-growth only needs two database scans. One is to find frequent one items and order the transactions according to the frequent one item support and the other is to build FP-tree. The rest operation is to recursively mine frequent items on the FP-tree using FP-growth.

Example 3 *We will illustrates the FP-growth Algorithm in following example. Given an absolute min-sup equals to 2. Continue with Table 2.1.*

- *Step 1: After the first scan we get a new ordered table as Table 2.3 and frequent one itemsets:* $\{A\}$*:2 ;* $\{B\}$*:3 ;* $\{C\}$*:3 ;* $\{E\}$*:3 .*

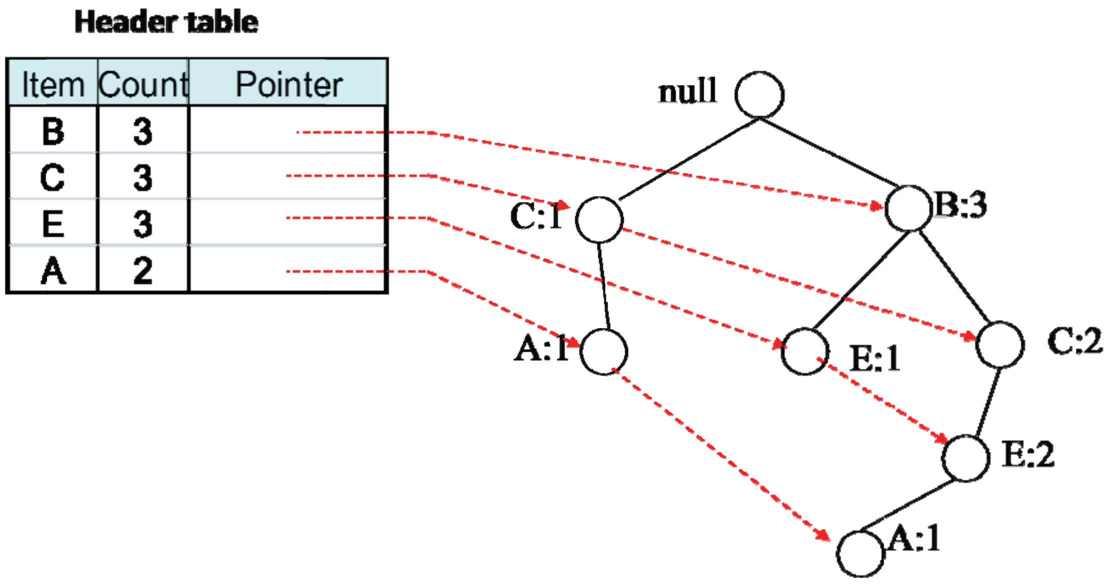


Figure 2.2: An example of FP-tree

- *Setp 2: Scan the transaction database TDB the second time. For each transaction, insert a branch to construct a FP-tree. The result is shown as Figure 2.2.*
- *Step 3: Use FP-growth to recursively mine frequent item patterns from FP-tree. We describe the FP-growth algorithm as below.*

We mine FP-tree from bottom to top. Starting from A, for each frequent 1-item, we construct its conditional pattern base. A conditional pattern base for an item/itemset contains the transactions that end with that item/itemset. We then treat the conditional pattern base the same as a transaction database and build the conditional FP-tree. The FP-growth algorithm is recursively performed on such conditional FP-trees. Item A's conditional pattern base is: $P = \{(C : 1), (B : 1, C : 1, E : 1)\}$ .The conditional tree for A is shown as

Figure 2.3.

Count for A is 2, so $\{A\}$ is frequent itemset. Then Recursively apply FP-growth on P. The conditional pattern base for E within conditional base for A is $P = \{(B : 1, C : 1)\}$. Count for E is 1. So $\{A, E\}$ is not frequent. The conditional pattern base for C within conditional base for A is $P = \{(C : 2)\}$. Count for C is 2. So $\{A, C\}$ is frequent. The conditional pattern base for B within conditional base for A is $P = \{(B : 1)\}$. Count for B is 1. So $\{A, B\}$ is not frequent.

After we mine item A, we use this algorithm on E. Item E's conditional pattern base is: $P = \{(B : 2, C : 2)\}$ .The conditional tree for E is shown as Figure 2.4.

Count for E is 3, so $\{E\}$ is frequent itemset. Then recursively apply FP-growth on P. Count for $\{C, E\}$ is 2, count for $\{B, E\}$ is 3, count for $\{B, C, E\}$ is 2. So $\{C,\ E\}$, $\{B,\ E\}$, $\{B,\ C,\ E\}$ is frequent itemsets.

We continue use the algorithm on C. Count for C is 3, and count for $\{B, C\}$ is 2. So they are frequent itemsets. Last, we use the algorithm on B, count for B is 3, so $\{B\}$ is frequent itemset.

Combined with all the frequent 1-items generated during the first database scan, we get the same set of frequent patterns:

{A}:2; {B}:3; {C}:3; {E}:3;

{A, C}:2; {B, C}:2; {B, E}:3; {C, E}:2;

{B, C, E}:2.

FP-growth only needs two database scans. For some database that generate lots of frequent item patterns, FP-growth may require more time when compared with Apriori, but for only a small portion of the candidate sets will survive to

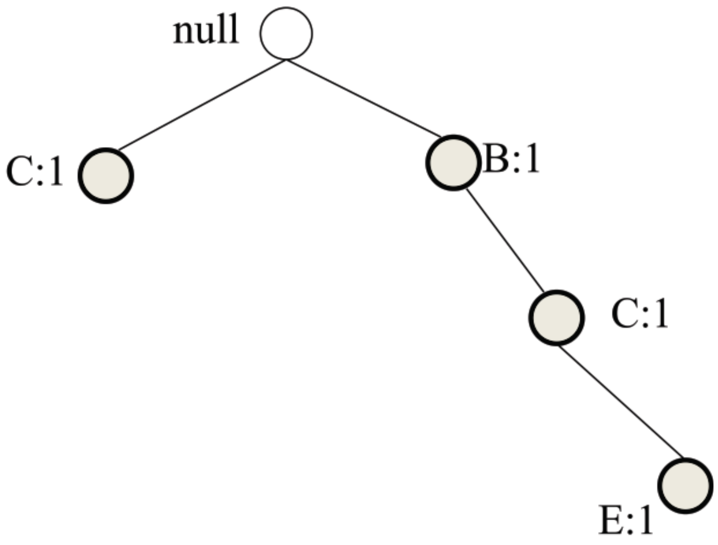


Figure 2.3: Conditional tree for A

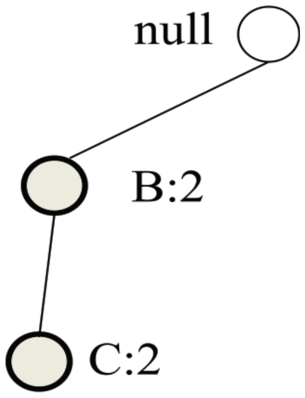


Figure 2.4: Conditional tree for E

FP-growth Algorithm:

```
Input: $\mathcal{D}$, $\sigma$, $I \subseteq \mathcal{I}$
Output: $\mathcal{F}[I](\mathcal{D}, \sigma)$
1: $\mathcal{F}[I] = \{\}$
2: for all $i \in \mathcal{I}$ occurring in $\mathcal{D}$ do
3:     $\mathcal{F}[I] = \mathcal{F}[I] \cup \{I \cup \{i\}\}$
       // Create $\mathcal{D}^i$
4:     $\mathcal{D}^i = \{\}$
5:     $H = \{\}$
6:     for all $j \in \mathcal{I}$ occurring in $\mathcal{D}$ such that $j > i$ do
7:         if support $(I \cup \{i, j\}) \geq \sigma$ then
8:             $H = H \cup \{j\}$
9:         end if
10:    end for
11:    for all $(tid, X) \in \mathcal{D}$ with $i \in X$ do
12:        $\mathcal{D}^i = \mathcal{D}^i \cup \{(tid, X \cap H)\}$
13:    end for
       //Depth-first recursion
14:    Compute $\mathcal{F}[I \cup \{i\}](\mathcal{D}^i, \sigma)$
15:    $\mathcal{F}[i] = \mathcal{F}[i] \cup \mathcal{F}[I \cup \{i\}]$
16: end for
```

Table 2.4: Pseudo code of FP-growth algorithm

become frequent patterns, Apriori loses drastically as well due to the costly candidate generation.

The pseudo-code for the FP-growth algorithm is given as below in Table 2.4.

2.1.3 Eclat

Another method for Frequent Itemset Generation is Eclat [40]. It uses the vertical database layout and the Tid-list intersection based approach to compute the support of an itemset. Before introducing the algorithm, we need to know the definition of "Cover". The cover of an itemset X in $\mathcal{D}$ is a list of the transaction ID (tid) in which X is included in those transactions. Let us see an example. We still use Table 2.1 as the original transaction database given an absolute *min-sup* equals to 2.

- Step 1: After the first scan, we convert the original database to the vertical data layout as shown in Figure 2.5. Item D's support is 1, less than the *min-sup*, we eliminate it. Then we get the frequent one itemsets: {A}, {B}, {C}, {D}.
- We use depth-first recursion to generate frequent itemset candidates, and then for each candidate set, intersect their Tid-list to get their support. Determine support of any $k - itemset$ by intersecting tid-lists of two of its $(k - 1)$ subsets. For example: Figure 2.6 illustrates the result of itemset $\{B,\ C\}$.

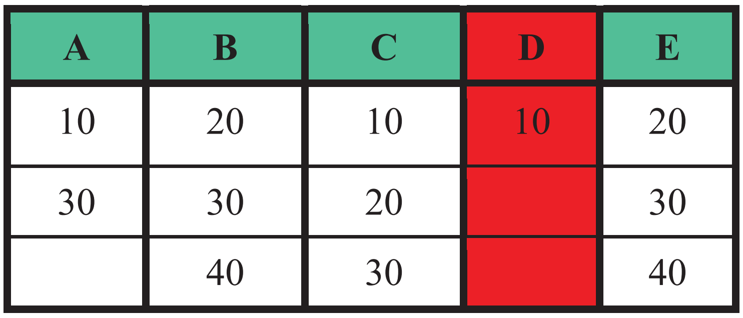
  From $\{B,\ C\}$ tid-list, it is easy to get their support by $|cover\{B,\ C\}|$ to be equal to 2. So $\{B,\ C\}$ is frequent. The same way, we can get all the frequent itemsets as below:

  {A}:2; {B}:3; {C}:3; {E}:3;

  {A, C}:2; {B, C}:2; {B, E}:3; {C, E}:2;

  {B, C, E}:2.

The pseudo-code for the Eclat algorithm is given as below in table 2.5. The advantage of Eclat is very fast support counting. We just need count the itemset's absolute value of its cover. The disadvantage is intermediate tid-lists may become too large for memory usage. If the tid-list is too long, the intersect



A	B	C	D	E
10	20	10	10	20
30	30	20		30
	40	30		40

Figure 2.5: Vertical data layout

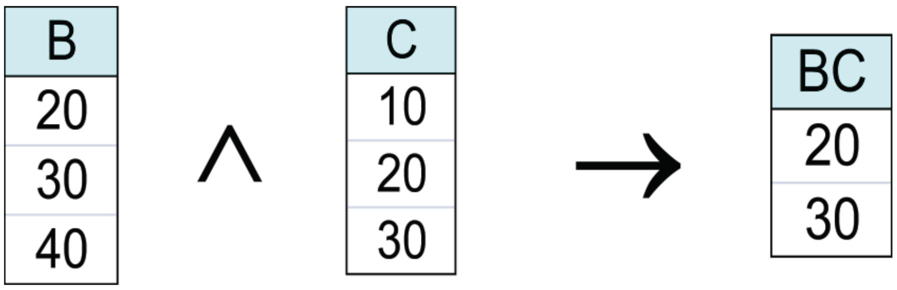


Figure 2.6: Intersecting tid-lists of B and C

Eclat Algorithm:

```
Bottom-Up(S):
1: for all atoms $A_i \in S$ do
2:     $T_i = \emptyset$
3:     for all atoms $A_j \in S$, with $j > i$ do
4:         $R = A_i \cup A_j$
5:         $\mathcal{L}(R) = \mathcal{L}(A_i) \cap \mathcal{L}(A_j)$
6:         if $\sigma(R) \geq min - sup$ then
7:             $T_i = T_i \cup \{R\}; \mathcal{F}_{|R|} = \mathcal{F}_{|R|} \cup \{R\}$
8:     end
9: end
10: for all $T_i \neq \emptyset$ do Bottom-Up($T_i$)
```

Table 2.5: Pseudo code of Eclat algorithm

operator also costs time.

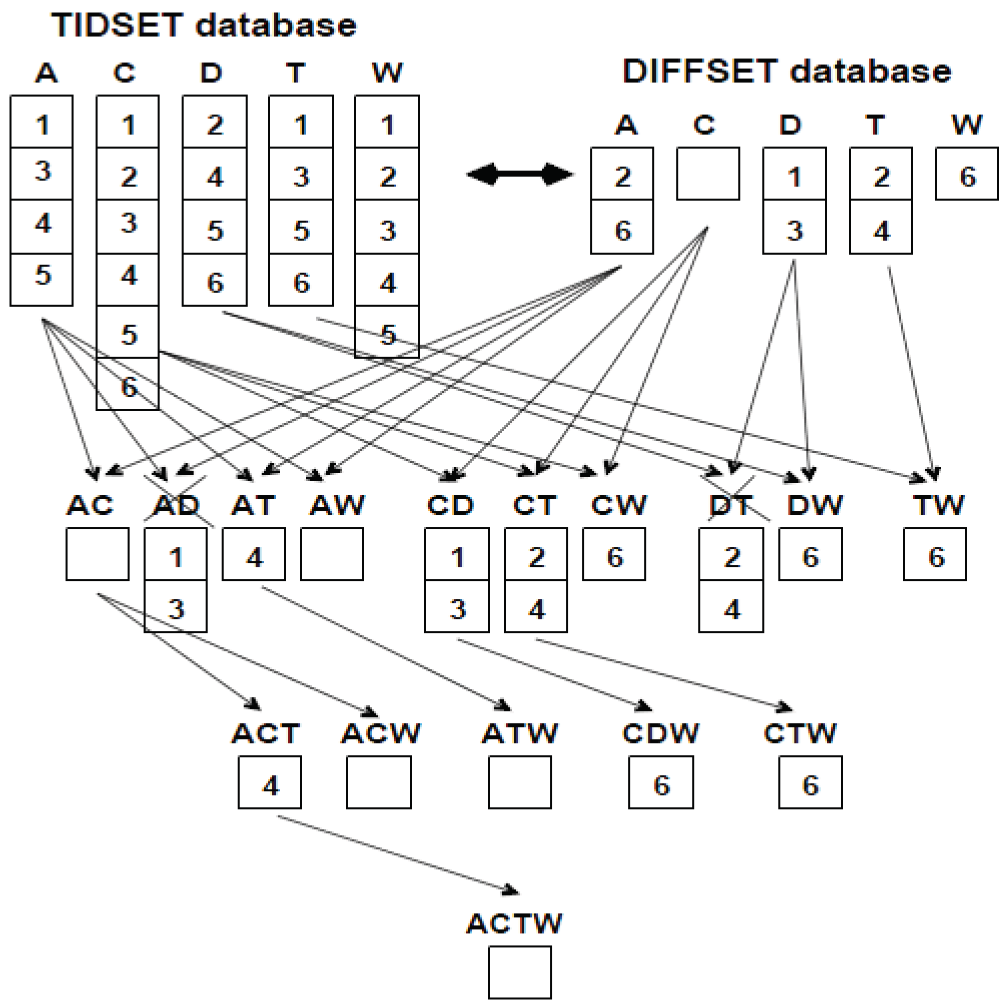
Diffsets The vertical format gains a lot on the TIDs intersection operations. The problem of these methods is that when the TID lists become too large, it will take more time on calculations. So Jaki proprose a novel vertical data representation called Diffset [39]. Instead of storing the entire tidset of each member of a class, the diffsets only keep track of the differences in the TIDs between each itemset and their prefix itemset. This method is efficient for dense datasets. Check Figure 2.7, it shows the diffsets algorithm and how it works.

2.1.4 Closed pattern mining

We will generate a large number of frequent itemsets according to the density of the database and *min-sup*. Among them, users have to do lots of analysis to find useful patterns. So Pasquier et al. [29] proposed to mine only closed set of frequent itemsets instead of the complete set. For example, the set of frequent patterns {(A:3), (B:3), (AB:3)} can be represented by $\{(AB:3)\}$. Thus we give the definition of frequent closed itemset as below.

Definition 1 *A frequent closed itemset is either a maximal frequent itemset, or a frequent itemset whose support is higher or equal to the supports of all its proper supersets.*

Pasquier developed an Apriori-based algorithm A-Close [30]. A-close is a breath first search method to find frequent closed itemset. Later, Pei proposed the algorithm CLOSET [31] based on FP-Growth. In 2002, Zaki and Hsiao present CHARM [41] algorithm which is proved to be the most efficient one among the



```
dEclat([P]):
for all $X_i \in [P]$ do
    for all $X_j \in [P]$, with $j > i$ do
        $R = X_i \cup X_j$;
        $d(R) = d(X_j) - d(X_i)$;
        if $\sigma(R) \geq min\_sup$ then
            $T_i = T_i \cup \{R\}$; //$T_i$ initially empty
    if $T_i \neq \emptyset$ then dEclat($T_i$);
```

Pseudo-code for dEclat

Figure 2.7: Diffsets for Pattern Counting

published. Below we will show an example of A-close to explain what closed pattern mining is.

Example 4 *We still use Table 2.1 as the original transaction database, given an absolute min-sup equals to 2. First we generate the frequent itemsets. We then compare the support between itemsets and their proper supersets.*

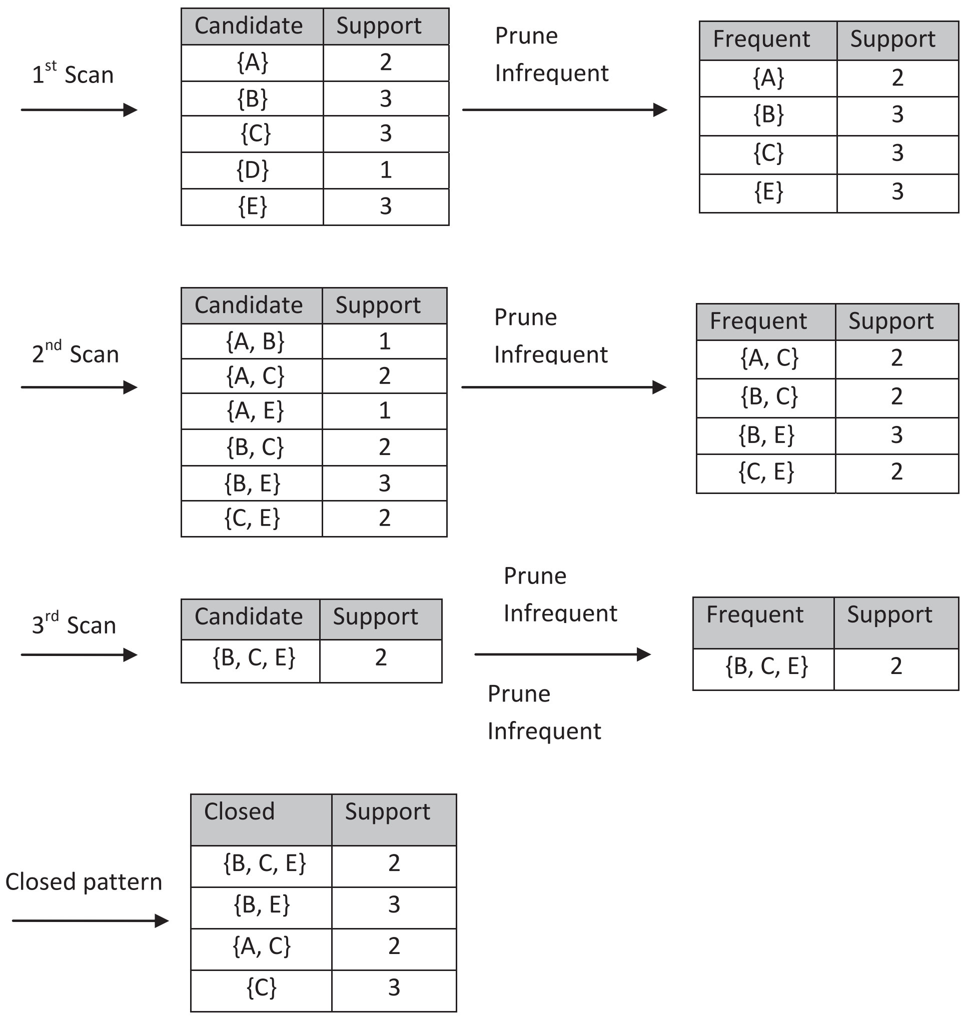
The mining process is as Figure 2.8
An instance for 2-itemset and 3-itemset $sup(\{B,\ C\}) = sup(\{C,\ E\}) = sup(\{B,\ C,\ E\}) = sup(\{B,\ E\})$. This means every transaction containing $\{B,\ C,\ E\}$ must also have $\{B,\ C\}$ and $\{C,\ E\}$. So the closed set is $\{B,\ C,\ E\}$ and $\{B,\ E\}$.

2.2 Multi-level or Cross-level frequent itemset mining

In this section, we will introduce Han's work [20] for multiple-level frequent pattern mining. Although we have mentioned some related work in section 1.1, Han's work is first one which introduces the multiple-level frequent pattern mining.

2.2.1 Multi-level frequent itemset mining

Most of the time, the real transaction database appears with hierarchy information as shown in Figure 1.2. When customer buys the 'Quebon 2% milk', the transaction database will save as $\{milk,\ Quebon,\ 2\%\}$ by the format $\{Category, Brand, Content\}$. The highest level represents the product index, i.e., *milk* or *bread*. In the second level, products are classified by the "Brand", i.e., *milk* is divided into two subclass based on their brand - $Quebon\ and\ Nestle$. The lowest level represents the



Candidate	Support
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

Frequent	Support
{A}	2
{B}	3
{C}	3
{E}	3

Candidate	Support
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

Frequent	Support
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

Candidate	Support
{B, C, E}	2

Frequent	Support
{B, C, E}	2

Closed	Support
{B, C, E}	2
{B, E}	3
{A, C}	2
{C}	3

Figure 2.8: A-closed pattern mining

content. Mining on different level will lead to different layer of the information hierarchy.

Generating multi-level frequent itemsets can help us get more detailed information. Unlike single-level frequent itemsets mining, it will generate a large amount of information, which takes more time to find interested ones. We can see that few people buy 'Quebon 2% milk' and 'Wheat Wonder Bread' together.

Thus, compared with single-level frequent itemsets mining, multi-level frequent itemsets mining is fine-grained to generate interesting frequent itemsets. It matches the business needs much better.

Algorithms:

Han [20] introduces four algorithms for multi-level frequent itemset mining: ML_T2L1, ML_T1LA, ML_TML1 and ML_T2LA. Since they are all based on Apriori with similar data structures and little different pruning methods, we therefore only introduce ML_T2L11 as an example.

[ML_T2L1]

Input: a hierarchy-information-encoded dataset and *min-sup* for each level.

Output: Multi-level frequent itemsets.

Method: A top-down method which scans the database one times for each level and each size of itemsets.

Starting at level 1, derive each level l, the frequent k-itemsets, $\mathcal{L}[l, k]$, for each k, and the frequent itemset, $ll[l]$ (for all k's), as Table 2.6.

Below we will give an example in Example 5 to illusion [ML_T2L1]. According to the product hierarchy of 1.2, we encode the supermarket transaction database to three digit format as Table 2.7. Each digit represents one concept level of

```
$ML\_T2L1:$
1: for $(l = 1; \mathcal{L}[l,1] \neq \emptyset \; and \; l < max\_level; l++)$ do
2:     if$(l == 1)$ then
3:         $\mathcal{L}[l,1] = get\_large\_itemsets[T[1], l)$
4:         $T[2] = get\_filteed\_table(T[l], \mathcal{L}[l,1]$
5:     else
6:         $\mathcal{L}[l,1] = get\_large\_1\_itemsets[T[2], l)$
7:     endif
8:     for $(k = 2; \mathcal{L}[l, k-1] \neq 0; k++)$ do
9:         $C_k = get\_candidate\_set(\mathcal{L}[l, k-1])$
10:          for each transaction $t \in T[2]$ do
11:              $C_t = get\_subsets(C_k, t)$
12:              for each candidate $t \in T[2]$ do $c.support++$ end for
13:          endfor
14:          $\mathcal{L}[l,k] = \{c \in C_k | c.support \geq min - sup[l]\}$
15:      endfor
16:      $\mathcal{LL}[l] = \bigcup_k \mathcal{L}[l,k]$
17: endfor
```

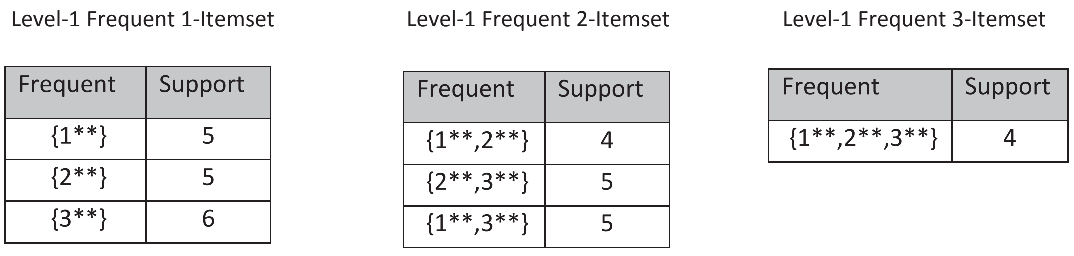
Table 2.6: Pseudo code of Eclat algorithm

TID	Items
T1	111,212,312
T2	312,113,231
T3	111,212,312,121
T4	212,311
T5	111,212,312,221
T6	111,312,412

Table 2.7: Encode Transaction Table for ML_T2L1

hierarchy. For example: '111' represent '2% Quebon Milk'. We suppose the *min-sup* for each level is: 4, 3, 3.

Example 5 1. *Step 1: Find frequent itemsets for level-1 for each size with three scans and min-sup equals to* 4. *The result is as shown in Figure* 2.9. *Then, using Level-1 Frequent 1-itemset we can get the filtered transaction table shown in Table* 2.8.



Level-1 Frequent 1-Itemset

Frequent	Support
{1**}	5
{2**}	5
{3**}	6

Level-1 Frequent 2-Itemset

Frequent	Support
{1**,2**}	4
{2**,3**}	5
{1**,3**}	5

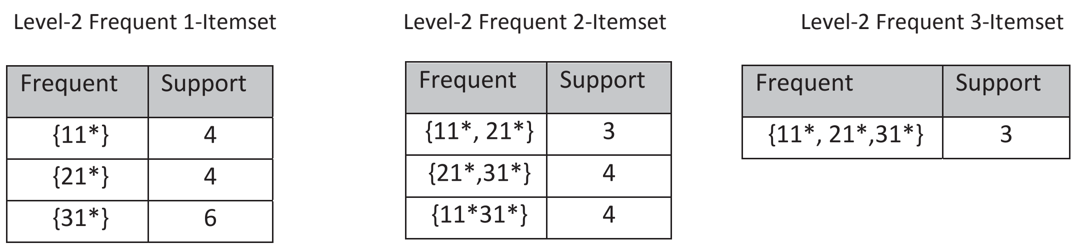
Level-1 Frequent 3-Itemset

Frequent	Support
{1**,2**,3**}	4

Figure 2.9: Frequent itemsets for level-1

2. *Step 2: find frequent itemsets for level-2 using filtered transaction Table* 2.8 *with min-sup equal to* 3. *The result is as Figure* 2.10.

3. *Step 3: find frequent itemsets for level-3 using filtered transaction Table* 2.8 *with min-sup equal to* 3. *The result is as Figure* 2.11.



Level-2 Frequent 1-Itemset

Frequent	Support
{11*}	4
{21*}	4
{31*}	6

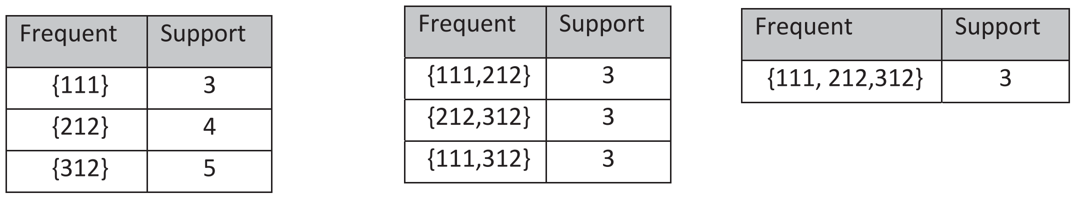
Level-2 Frequent 2-Itemset

Frequent	Support
{11*, 21*}	3
{21*,31*}	4
{11*31*}	4

Level-2 Frequent 3-Itemset

Frequent	Support
{11*, 21*,31*}	3

Figure 2.10: Frequent itemsets for level-2



Frequent	Support
{111}	3
{212}	4
{312}	5

Frequent	Support
{111,212}	3
{212,312}	3
{111,312}	3

Frequent	Support
{111, 212,312}	3

Figure 2.11: Frequent itemsets for level-3

Therefore, ML_T2L1 algorithm generates the multi-level frequent itemset as follow:

Frequent itemset at level-1:

{{1**}, {2**}, {3**}, {1**, 2**}, {2**, 3**}, {1**, 3**}, {1**, 2**, 3**}};

Frequent itemset at level-2:

TID	**Items**
T1	111,212,312
T2	312,113,231
T3	111,212,312,121
T4	212,311
T5	111,212,312,221
T6	111,312

Table 2.8: Filtered Transaction Table for ML_T2L1

{{11*}, {21*}, {31*}, {11*, 21*}, {21*, 31*}, {11*, 31*}, {11*, 21*, 31*}};
Frequent itemset at level-3:
{{111}, {212}, {312}, {111 ,212}, {212, 312}, {111, 312}, {111, 212, 312}}.

2.2.2 Cross-level frequent itemset mining

Several studies on Cross-level frequent itemsets mining are also proposed. R. Srikan has introduced an algorithm [6] which can generate flexible concept level frequent itemsets based on Apriori like the ET-apriori proposed in this thesis. However, our method has more pruning on the filtered database. Runying Mao introduced a method that can generate flexible concept level frequent itemsets base on FP-Growth [26], which focus more on multi-dimension. Due to the complexity of the structure of FP-Growth, this algorithm is suitable for low level dataset. That is,it may be suitable for three-level data, But if usedthe total data level is 6, the FP-Growth tree will became bigger and more complicate. Venkata [32] proposed a algorithm based on Han's Discovery of Multiple-Level ssociation Rules from Large Databases. They just add cross-level items to the algorithms. So they are low efficient since it scans the database more times.

Chapter 3

Item definitions and Properties

Products are frequently organized as so called isA hierarchies, or taxonomies. A taxonomy $\mathcal{T}$ is a tree structure of classifications for a given set of items. At the root of this tree is a single class consisting of all items. Nodes below are more specific classes, each corresponding to a subset of the items. The leaves of the taxonomy $\mathcal{T}$ are the items. See Figure 1.2 for an example of a taxonomy. It illustrates that 'Quebon 2% milk' isA 'Quebon milk' isA 'Milk' isA 'Food', and so on. In the following we present the encoding schema [20] used for the items in the taxonomy. Each item is identified by a unique list of 'n' digit encoding. For example 'Quebon 2% milk' is encoded as '112', that is this item is represented by 3 digits. The encoding of the items in an 'n' level taxonomy is defined recursively as follows:

- The root is encoded with n '*'s.
- The child on level $k+1$ is encoded with the same first k digits as its parent, followed by an additional digit and $n-k-1$ '*'s.

TID	**Items**
T1	111,212,112
T2	111,222,231
T3	111,222,312,121

Table 3.1: Encoded Transaction table using the taxonomy

Thus, the leaves are encoded with n digits and no '*'s representing the original encoding of the items. Actually in our implementation, we use binary representation for these items, which can help using binary operators to check two items are parent each other. we will introduce it in section 4.1.

Example 6 *Figure 1.2 shows an example of such an encoding. The first-level concept is represented by <u>Category</u>, the second level is represented by <u>Brand</u> and the last level is represented by the <u>Content</u>. Each product is represented by a unique digit encoding. For example, 'Ground Café of Van' is encoded as '311' in which the first digit, '3' represents the category 'Cafe', the second digit '1' representing the content 'Van' the last digit '1' represents the content 'Ground'. Using this encoding we can encode the transaction data as represented in Table 3.1.*

In this paper we focus on a mixed multilevel itemsets mining, based on the taxonomy hierarchy. Our approach can generate frequent itemsets as $\{2\%\ Quebon\ milk, Pom\ bread\}$, with encoding ($\{112,\ 22*\}$), that is with items located on different levels in the taxonomy tree. In Figure 3.1 are displayed frequent size 2 itemsets from Table 3.1, using *min-sup* 0.5. For two same size itemsets X and Y, we denote X is a descendant of Y if for any item i in X, there exists an item j in Y that $i \preceq j$ and for any item j in Y,there exists an item i in X that $j \succeq i$. The taxonomy based pruning means that if an node on the tree is infrequent, then all the descendants of that node are also infrequent. For instance of Figure 3.1, if the node ($\{11*,\ 22*\}$) is infrequent, all of its leaves in the tree are also infrequent.

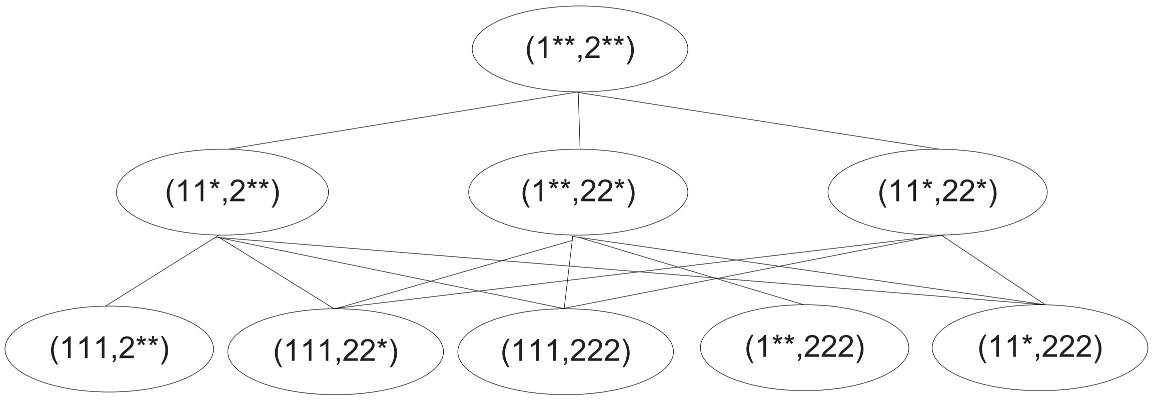


Figure 3.1: Frequent size two itemset

3.1 Properties of itemsets

Single-Level Definitions and Properties

Frequent itemset mining is aimed to find frequent patterns from transaction databases. In this section, we review some concepts for single-level frequent itemsets mining which will be useful for multiple-level data mining. Let $\mathcal{I}$ be a set of items from a transaction database $\mathcal{D}$. The k-itemset is a set of size k with elements from $\mathcal{I}$. Each transaction from the set $\mathcal{D}$ is identified by a unique identifier *tid*, named transaction id. Let I be a function that for each transaction id *tid* returns the set of elements part of that transaction.

Next, let us introduce the definition of the support. The *support* of an itemset X in a transaction database $\mathcal{D}$, is percentage of transactions in $\mathcal{D}$ that contains

X. The support of X in $\mathcal{D}$ is denoted as $\sigma(X, \mathcal{D})$

Proposition 1 ***(Support monotonicity)*** *[4] Let $X, Y \subseteq \mathcal{I}$ be two itemsets, then,*

$$X \subseteq Y \Rightarrow \sigma(Y, \mathcal{D}) \leq \sigma(X, \mathcal{D})$$

$\square$

This means that the support of an itemset is less than or equal to the support of its subsets.

An itemset is said to be frequent if its support is greater or equal than a threshold support given, in general, given by the users, and it is denoted by *min-sup*. The frequent itemset mining problem is to find all frequent itemsets in the transaction database given by a threshold support *min-sup*. The Apriori algorithm is based on this proposition.

Definition 2 *Let $\mathcal{I}$ be the set of items for transaction database $\mathcal{D}$, and min-sup be the threshold support. The collection of frequent itemsets in $\mathcal{D}$ is denoted by*

$$\mathcal{F}(\mathcal{D}, \sigma) = \{X \subseteq \mathcal{I} \mid (\sigma(X, \mathcal{D})) \geq \textit{min-sup}\}$$

Mutil-Level Definitions and Properties

Let us now introduce some properties of multiple level itemsets. According to the item taxonomy, we denote $a \preceq b$ if a is a descendant of item b or b is an ancestor of a. For example: $11{*} \preceq 1{*}{*}$. For two same size itemsets X and Y, we denote $X \preceq Y$ if for any item i in X, there exists an item j in Y that $i \preceq j$ and for any item j in Y,there exists an item i in X that $j \succeq i$. We will need the following notations:

Definition 3

$$X \sqcup Y =^{def} \{a \in X : \forall b \in Y,\ a \not\prec b\} \cup \{b \in Y : \forall a \in X,\ b \not\prec a\}$$

Example 7 *Consider the itemset* $X = \{111,\ 21*,\ 311\}$ *and itemset* $Y = \{11*,\ 3**,\ 4**\}$. *Then* $X \sqcup Y = \{11*,\ 21*,\ 3**,\ 4**\}$.

Definition 4

$$X \sqcap Y =^{def} \{a \in X : \forall b \in Y,\ a \not\succ b\} \cup \{b \in Y : \forall a \in X,\ b \not\succ a\}$$

Example 8 *Consider the itemset* $X = \{111,\ 21*,\ 311\}$ *and itemset* $Y = \{11*,\ 3**,\ 4**\}$. *Then* $X \sqcap Y = \{111,\ 21*\ ,311,\ 4**\}$.

Definition 5

$$max(X) =^{def} \{a \in X :\ \forall b \in X,\ a \not\prec b\}$$

Example 9 *Consider the itemset* $X = \{111,\ 21*,\ 2**,\ 311\}$. *Then* $max(X) = \{111,\ 2**,\ 311\}$.

Definition 6

$$min(X) =^{def} \{a \in X :\ \forall b \in X,\ a \not\succ b\}$$

Example 10 *Consider the itemset* $X = \{111,\ 21*,\ 2**,\ 311\}$. *Then* $min(X) = \{111,\ 21*,\ 311\}$.

We now have the following propositions:

Proposition 2 $X \sqcup Y = max(X \cup Y)$

Proof: Let $a \in X \sqcup Y$, follows that either $a \in \max(X)$ or $a \in \max(Y)$ that is $a \in \max(X \cup Y)$. $\square$

Example 11 *Consider the itemset* $X = \{111,\ 21*,\ 311\}$ *and itemset* $Y = \{11*,\ 3**,\ 4**\}$. *We have* $X \sqcup Y = \{11*,\ 21*,\ 3**,\ 4**\}$

Proposition 3 $X \sqcap Y = min(X \cup Y)$

Proof: Let $a \in X \sqcap Y$, it follows that either $a \in \min(X)$ or $a \in \min(Y)$. Thus $a \in \min(X \cap Y)$ $\square$

Example 12 *Consider the itemset* $X = \{111,\ 21*,\ 311\}$ *and itemset* $Y = \{11*,\ 3**,\ 4**\}$. *We have* $X \sqcap Y = \{111,\ 21*,\ 311,\ 4**\}$

Proposition 4 *If* $X \preceq Y$ *then* $X \sqcup Y = Y$.

Proof: Let $a \in X \sqcup Y$, that is $a \in \max(X \cup Y)$. case 1: $a \in \max(X)$, since $X \preceq Y$, it follows that $a \in Y$. case 2: $a \in \max(Y)$, it follows that $a \in Y$, so $X \sqcup Y \subseteq Y$.
Let $a \in Y$, thus $a \in X \sqcup Y$, so $Y \subseteq X \sqcup Y$. From above, we can get if $X \preceq Y$ then $X \sqcup Y = Y$. $\square$

Example 13 *Consider the itemset* $X = \{111,\ 21*,\ 311\}$ *and itemset* $Y = \{11*,\ 2**,\ 3**\}$. *We have* $111 \prec 11*$, $21* \prec 2**$ *and* $311 \prec 3**$. *In this case we compute* $X \sqcup Y = max(111,\ 21*,\ 311, 11*,\ 2**,\ 3**)$ *that is* $X \sqcup Y = \{11*,\ 2**,\ 3**\} = Y$

Proposition 5 *If* $X \preceq Y$ *then* $X \sqcap Y = X$.

Proof: similar to the proof of Proposition 4. $\square$

Example 14 *Consider itemset* $X = \{111,\ 21*,\ 311\}$ *and itemset* $Y = \{11*,\ 2**,\ 3**\}$. *We have* $111 \prec 11*$, $21* \prec 2**$ *and* $311 \prec 3**$ *we can now compute:*

$$X \sqcap Y = min(\{111,\ 21*,\ 311,\ 11*,\ 2**,\ 3**\})$$

that is $X \sqcap Y = \{111,\ 21*,\ 311\} = X$.

The following proposition is used in the LP-Apriori algorithm in order to check if the set of parent items for an itemset is frequent then the itemset can be a candidate, otherwise we eliminate it.

Proposition 6 *Let* $X \cup Y \subseteq \mathcal{I}$, *and let* $X \preceq Y$, *then* $\sigma(X, \mathcal{D}) \leq \sigma(Y, \mathcal{D})$.

$\square$

Example 15 *Consider itemset* $X = \{111,\ 21*,\ 311\}$ *and itemset* $Y = \{11*,\ 2**,\ 3**\}$. $\sigma(\{111,\ 21*,\ 311\}) \leq \sigma(\{11*,\ 2**,\ 3**\})$

Chapter 4

The Algorithms

Throughout the past decade, lots of implementations were developed for frequent itemset mining. The most general and popular of these algrithms are Apriori [4], Eclat [40] and FP-growth [21]. For mixed multiple level frequent itemset mining, we need to generate more single items, which makes the data structure of FP-growth more complicated. For example, if we mine for a six level itemset, the FP-growth tree will become six times bigger. It will be inefficient to mine from large information tree. So we chose Apriori and Eclat as our basic algorithms to develop ET-apriori,LP-apriori, ET-eclat and LP-eclat in this study. And among the dozens of Apriori implementations, our algorithms extend A Fast APRIORI[10], which is proved to be the fastest one in most cases. It uses a trie stucture which is first introduced in [24] to store and retrieve words of a dictionary. The implementation uses an array [27] to support this trie stucture. The essential difference between ET-apriori, LP-apriori and A Fast APRIORI is A Fast APRIORI can only generate single-level fequent itemsets, and ours can generate cross-level fequent itemsets. We also make a significant research on fequent

TID	Items
T1	111,212 ,112,222,312
T2	312,113,231
T3	111,212,312,121,232
T4	212,211,311
T5	111, 212,312,221,321
T6	111, 312,322,412

Table 4.1: Encoded Original Transaction table

pairs mining, we will introduce them in Section 4.5 and Section 4.6. Due to the fast development of memory, we store our transaction table in the main memory in all the presented algorithms.

4.1 ET-apriori

The ET-apriori algorithm, it is similar to [34], and is an extension of A Fast APRIORI [10]. The algorithm uses a breadth-first search technique through candidate itemsets. The ET-apriori expand each item with all its ancestors in the transaction table creating a new extended transaction table. There is a little difference between ET-apriori and the algorithm presented by Srikant, Agrawal in [34]. The main difference is that after the first scan, ET-apriori removes from the original transaction database all items that are not frequent decreasing the size of the database. This step is not done in the Srikant's algorithm. Also, Srikant's algorithm uses the original dataset for all scans. This extra step done by our algorithm helps by making the original transaction database smaller, and increasing the performance of the rest of the scans on the cost of one extra scan.

The ET-apriori algorithm can generate multi-level frequent itemsets. The

```
Algorithm ET-apriori:
Input: $\mathcal{D}, \sigma$
Output: $\mathcal{F}(\mathcal{D}, \sigma)$
1: Extend $\mathcal{D}$ by including in each transaction all the higher level items;
2: Let $\mathcal{C}_1 := \{\{i\} \ : \ i \in \mathcal{I}(\mathcal{D})\}$
3: Let $\mathcal{F}_1 = \text{scan}(\mathcal{D}, \mathcal{C}_1)$
4: $\mathcal{D}'$ be the database obtained from $\mathcal{D}$ by removing items such that
        $\mathcal{I}(\mathcal{D}') = \mathcal{I}(\mathcal{D}) \bigcap (\bigcup_{X \in \mathcal{F}_1} X)$
4: Let $\mathcal{C}_2 := \mathcal{F}_1 \times \mathcal{F}_1$
5: Let $k := 2$
6: while $\mathcal{C}_k \neq \emptyset$ do
7:     Let $\mathcal{F}_k = \text{scan}(\mathcal{D}', \mathcal{C}_k)$
8:     Let $k := k + 1$
9:     Let $\mathcal{C}_k := \mathcal{F}_{k-1} \times \mathcal{F}_{k-1}$
10: end while
11: $\mathcal{F}(\mathcal{D}, \sigma) = \bigcup_{i=1}^{k} \mathcal{F}_i$
```

Table 4.2: ET-apriori Algorithm

ET-apriori algorithm that generates mixed-level frequent itemsets is described in Table 4.2. The input of the algorithm is the encoded transaction database and the support. The algorithm outputs the set of mixed-level frequent itemsets.

Where the scan function computes the frequent itemsets for a database $\mathcal{D}$ and a set of candidate itemsets $\mathcal{C}_k$, for some k. This computation is done in the usual way, by scanning each transaction step by step and increasing the support for each itemset that is a subset of the transaction.

Note that the computation of the set $\mathcal{F}_k \times \mathcal{F}_k$, for some k, involves an expensive step that checks if for two items X, Y is it that $X \preceq Y$ or $Y \preceq X$. For a better performance we optimized this step as follows: convert the encoded item to binary representation by expanding each digit to its four bits binary encoding. As a special case digit $*$ is replaced by binary encoding '1111'. For instance item 112 is encoded with the following binary vector '000100010010', item 11$*$ is represented by binary vector ‘000100011111‘. Thus, to check if $112 \preceq 11*$ is enough to check if $112 \wedge 11* = 112$ in their binary representation, where $\wedge$ represents the logical "and" operator. In our example we have that ‘000100010010‘ $\wedge$ ‘000100011111‘ = ‘000100010010‘, meaning that $112 \preceq 11*$. This is indeed an optimization as it is well known that binary operators are executed faster than any other language specific operators.

Intuitively, the ET-apriori algorithm follows these steps:

1. expand each transaction by adding all ancestors for each item in the transaction;
2. to compute size 1 frequent itemsets we simply scan once the extended database;
3. remove from the extended database all the items that are not found frequent. This can be done by scanning once the extended transaction database;
4. for any $k > 1$, we repeat the following steps until there are no more candidate itemsets:
   (a) Compute the candidate set $\mathcal{C}_k$ as $\mathcal{F}_{k-1} \times \mathcal{F}_{k-1}$ by using apriori candidate generation algorithm [4].
   (b) In one scan of the transaction database find the frequent itemsets of size k based on the candidates $\mathcal{C}_k$.
5. Finally output is the union of all computed sets $\mathcal{F}_k$.

As an optimization in our implementation, we use a trie structure to save frequent itemsets. For more details about the data structure, the reader should refer to [10].

TID	Items
T1	111,112,11*,1**,212,21*,222,22*,2**,312,31*,3**
T2	113,11*,1**,231,23*,2**,312,31*,3**
T3	111, 112,11*,1**,212,21*,232,2**,312,31*,3**
T4	211,212,21*,2**,311,31*,3**
T5	111, 11*,1**,212,21*, 221,22*,2**,312,31*,321,32*,3**
T6	111, 11*,1**, 312,31*,322,32*,3**,4**,41*,412

Table 4.3: Extended Transaction table using the taxonomy

TID	Items
T1	111,~~112~~,11*,1**,212,21*,~~222,22*,2**~~,312,31*,3**
T2	~~113~~,11*,1**,~~231,23*~~,2**,312,31*,3**
T3	111, ~~112~~,11*,1**,212,21*,~~232~~,2**,312,31*,3**
T4	~~211~~,212,21*,2**,~~311~~,31*,3**
T5	111, 11*,1**,212,21*,~~221,22*~~,2**,312,31*,~~321,32*~~,3**
T6	111, 11*,1**, 312,31*,~~322,32*~~,3**,~~4**,41*,412~~

Table 4.4: Filtered Transaction table

Example 16 1. *Let us suppose Table 4.1 to be our original dataset. Next we convert Table 4.1 in Table 4.3 by extending each transaction by adding for each item its ancestors.*

2. *After first scan Table 4.3, we generate the following size-1 frequent itemsets* $\{111,\ 11*,\ 1**,\ 212,\ 21*,\ 2**,\ 312, 31*, 3**\}$ *with min-sup equal to 3. Then we use this set to filter Table 4.4. The resulted table is represented in Table 4.4.*

3. *We continue mining size-k itemsets and get the final output frequent itemset trie as represented in Figure 4.1.*

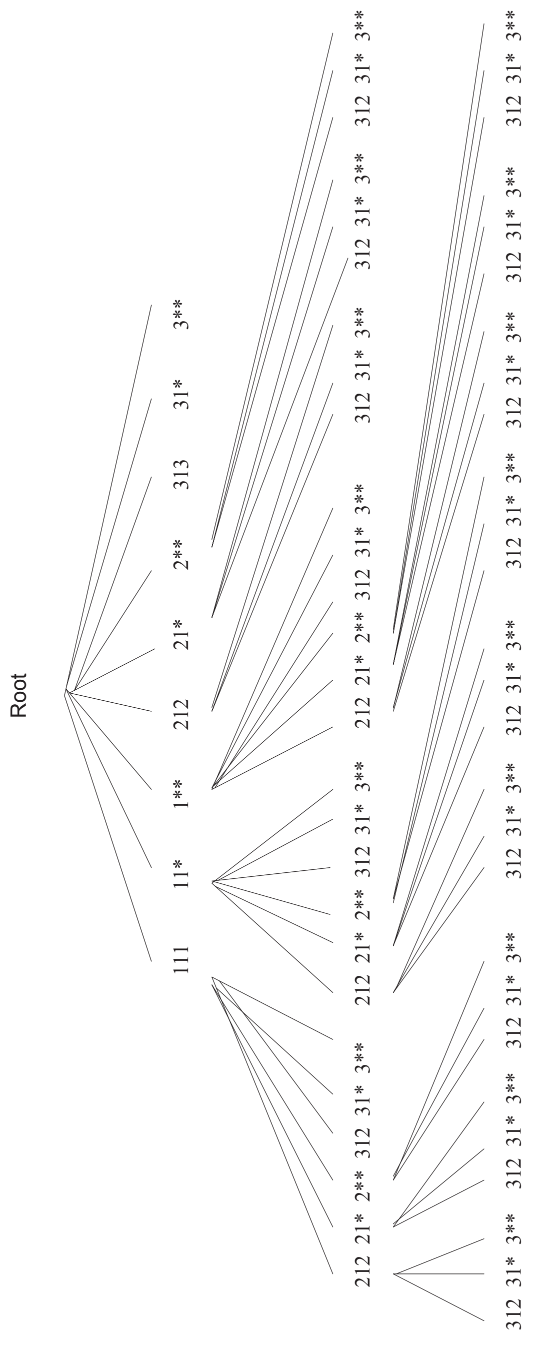


Figure 4.1: Frequent Itemsets Tree for ET-apriori from input of Table 4.1 with *min-sup* is 0.5

```
Algorithm LP-apriori:
Input: $\mathcal{D}, \sigma$
Output: $\mathcal{F}(\mathcal{D}, \sigma)$
1: Let $\mathcal{C}_1 := \{\{i\} \ : \ i \in \mathcal{I}(\mathcal{D})\}$
2: Let $\mathcal{F}_1 = \text{scan}(\mathcal{D}, \mathcal{C}_1)$
3: Consider $\mathcal{F}_1 = \bigcup_{1 \le l \le n} \mathcal{F}_1^l$, where $n$ represents the levels
4: Let $\mathcal{D}'$ be the database obtained from $\mathcal{D}$ by removing items such that
            $\mathcal{I}(\mathcal{D}') = \mathcal{I}(\mathcal{D}) \bigcap (\bigcup_{X \in \mathcal{F}_1} X)$
5: for $l = 1$ to $n$ do //$n$ represents the number of levels, where level 1 is ***
6: $\mathcal{C}_2^l = \mathcal{F}_1^l \times (\bigcup_{1 \le j \le l} \mathcal{F}_1^j)$
7: Let $k := 2$
8: while $\mathcal{C}_k^l \neq \emptyset$ do
9:         Let $\mathcal{F}_k^l = \text{scan}(\mathcal{D}', \mathcal{C}_k^l)$
10:        Let $\mathcal{C}_{k+1}^l = \mathcal{F}_k^l \times (\bigcup_{1 \le j \le l} \mathcal{F}_k^j)$
// The level pruning
11:          eliminate from $\mathcal{C}_{k+1}^l$ all sets $X$ such that there exists $Y \in \mathcal{P}(\mathcal{I}^*)_{k+1}$ with $X \prec Y$.
12:     $k++$
13: end while
14: end for
15: $\mathcal{F}(\mathcal{D}, \sigma) = \bigcup_{l=1}^{\ell} \bigcup_{i=1}^{k} \mathcal{F}_i^l$
```

Table 4.5: LP-apriori Algorithm

4.2 LP-apriori

The LP-apriori algorithm also extends A Fast APRIORI algorithm [10]. We construct trie based structures for each level. The algorithm uses the breadth-first search technique to search candidate itemsets.

Table 4.5 represents the LP-apriori algorithm. Similarly to the ET-apriori algorithm, the input is the transaction database and the minimum support used to find frequent itemsts. The main difference between ET-apriori and LP-apriori it that the later does not expand the transactions in the database. The performance comparing is done in the next section.

The intuition behind the algorithm is as follows:

1. First compute the frequent itemsets of size 1 for all levels. This can be done in only one scan of the transaction database;
2. remove from the extended database all the items that are not found frequent. This can be done by scanning once the extended transaction database;
3. For all levels, starting with the lowest level (***), for each size $k \geq 2$ compute the candidate itemsets
4. From the computed candidate set of itemsets eliminate those itemsets for which there is an ancestor set such that it is not in the already computed frequent itemsets. (Step 11)
5. Last, the output is the union of $\mathcal{F}_k^l$ for all the k's.

Example 17 *1. After the first scan we get frequent itemsets of level 1 and size-1* $\{1**,\ 2**,\ 3**\}$ *with min-sup equal to* 3.

2. Use, frequent one size one itemset $\{1**,\ 2**,\ 3**\}$ *to filter the Table 4.1 and get Table 4.6. For an instance,* $4**$ *is infrequent, we eliminate* 412 *from the Table 4.1.*

3. We generate candidates for each level. For an instance, after we generate candidate $\{11*,\ 21*,\ 31*\}$*, we check its parent* $\{1**,\ 2**,\ 3**\}$ *from its first level. since* $\{1**,\ 2**, 3**\}$ *is frequent,* $\{11*,\ 21*,\ 31*\}$ *could be a candidate. We than scan the dataset to calculate the support for the candidate itemsets and eliminate the ones that are infrequent. For each level we will have a frequent itemset trie to save the frequent itemsets as Figure 4.2.*

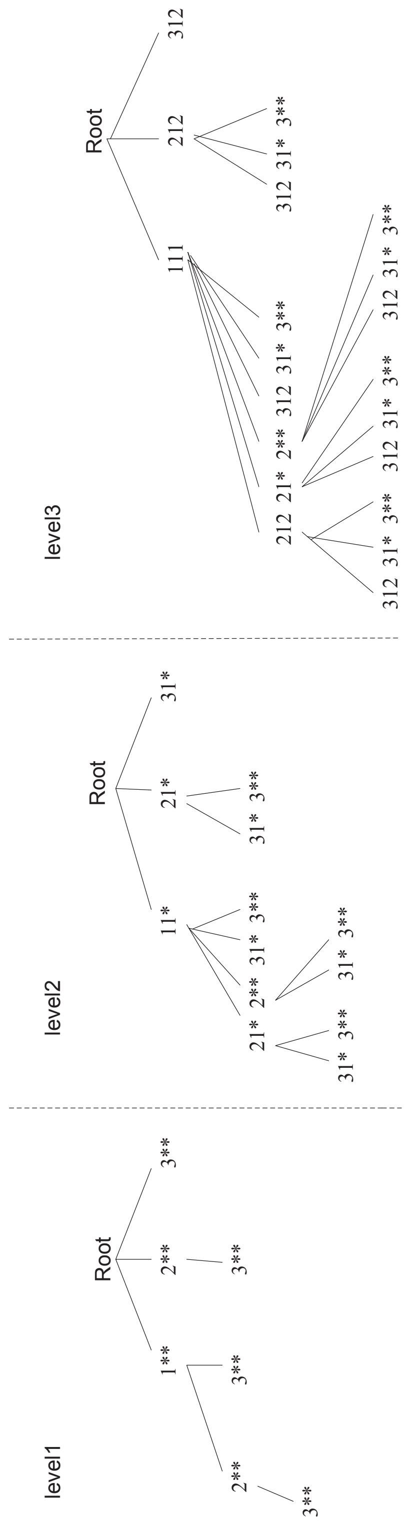


Figure 4.2: Frequent Itemsets Tree for Each Level

TID	Items
T1	111,212 ,112,222,312
T2	312,113,231
T3	111,212,312,121,232
T4	212,211,311
T5	111, 212,312,221,321
T6	111, 312,322, ~~412~~

Table 4.6: Filtered table by using Frequent level-1 size-1 itemset for LP-apriori

4.3 ET-eclat

The ET-eclat algorithm is based on An efficient algorithm for closed itemset mining presented in [40]. The algorithm uses a Depth-First search technique through search candidate itemsets. The database is converted into a vertical bit-map format and use the intersection based approach to calculate an itemset's support. ET-eclat recursively generates the frequent itemsets $\mathcal{F}[i](\mathcal{D}, \sigma)$ for each item $i \in \mathcal{I}$ which $\mathcal{I}$ is the expanded set of all the items. The frequent itemsets mining of ET-eclat algorithm is given in Table 4.7.

The intuition behind the algorithm is as follows:

1. First, we expand the each transaction with different level in digital number with numeric order, the notation '*' is replaced by '9' and the result is shown in Table 4.3.

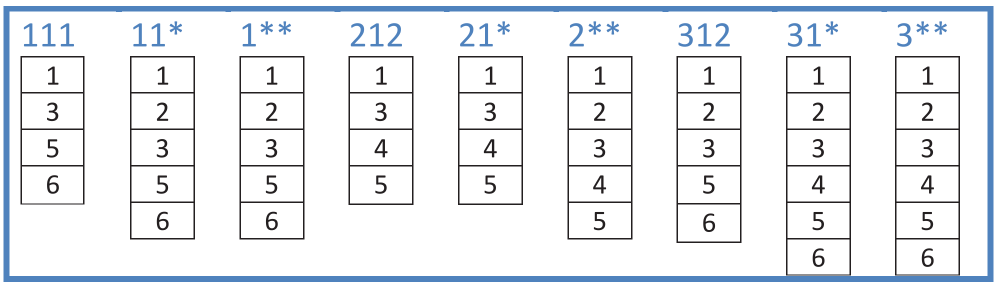
2. Second, we convert the expanded database to a vertical tid-list database format where we associate with each itemset a list of transactions in which it occurs. The result is shown as Figure 4.3.

3. Third, for each item i we use depth-first approach and do the intersection with $j(i < j)$ recursively where i and j have no parents pairs. The support

Algorithm ET-eclat:

```
Input: $\mathcal{D}, \sigma, \mathcal{L}, I \subseteq \mathcal{I}$
Output: $\mathcal{F}[I](\mathcal{D}, \sigma)$
1:$D = convertTtoMultiLevel(D)$ \\for each item generate all its parents as new item
2:$\mathcal{F}[I] := \{\}$
3:for all $i \in \mathcal{I}$ occurring in $\mathcal{D}$ do
4:    $\mathcal{F}[I] := \mathcal{F}[I] \cup \{I \cup \{i\}\}$
5:    Create $\mathcal{D}^i$
6:    $\mathcal{D}^i := \{\}$
7:    for all $j \in \mathcal{I}$ occurring in $\mathcal{D}$ such that
        $j > i$ and NotParentEachOther$(i, j)$ do
8:        $C := cover(\{i\}) \cap cover(\{j\})$
9:        if $|C| \geq \sigma$ then
10:           $\mathcal{D}^i := \mathcal{D}^i \cup \{(j, C)\}$
11:       end if
12:   end for
13:   \\Depth-first recursion
14:   Compute $\mathcal{F}[I \cup \{i\}](\mathcal{D}^i, \sigma)$
15:   $\mathcal{F}[I] := \{\mathcal{F}\}[I] \cup \mathcal{F}[I \cup \{i\}]$
16:end for
```

Table 4.7: ET-eclat Algorithm



111	11*	1**	212	21*	2**	312	31*	3**
1	1	1	1	1	1	1	1	1
3	2	2	3	3	2	2	2	2
5	3	3	4	4	3	3	3	3
6	5	5	5	5	4	5	4	4
	6	6			5	6	5	5
							6	6

Figure 4.3: Tid-List for frequent one item

for the itemsets is the number of results of the intersection, for example: for itemset {111,21*}, the intersection result is {1,3,5}, so the support is 3. We give an example for item '111' as Figure 4.4.

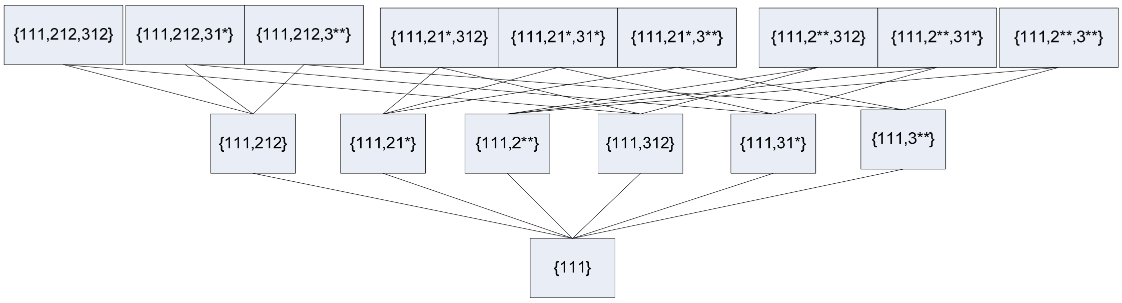


Figure 4.4: Frequent itemsets Result for item '111'

4. Last, the output is the union of $\mathcal{F}[i]$ for each $i \in \mathcal{I}$. We'll get the same results with ET-apriori and LP-apriori.

4.4 LP-eclat

The LP-eclat algorithm is the same with ET-eclat. We add a frequent prefix tree for the first level to filter some itemsets before count their support. This algorithm is fast for the two-digit represent data format. The algorithm still uses a Depth-First search technique through search candidate itemsets. We convert the database to a vertical bit-map format and use the intersection based approach to calculate an itemset's support. LP-eclat recursively generates the frequent itemsets $\mathcal{F}^l[i](\mathcal{D}, \sigma)$ for each item $i \in \mathcal{I}^l$ which $\mathcal{I}^l$ is the expanded set of all the items for each level. The frequent itemsets mining of LP-eclat algorithm is given in Table 4.8.

Algorithm LP-eclat:

```
Input: $\mathcal{D}, \sigma, \mathcal{L}, I \subseteq \mathcal{I}$
Output: $\mathcal{F}[I](\mathcal{D}, \sigma)$
1:for each level $l \in \mathcal{L}$ do
2:    $\mathcal{F}^l[I] := \{\}$
3:    for all $i^l \in \mathcal{I}^l$ occurring in $\mathcal{D}$ do
4:        $\mathcal{F}^l[I] := \mathcal{F}^l[I] \cup \{I \cup \{i^l\}\}$
        \\Create $\mathcal{D}^{i^l}$
5:        $\mathcal{D}^{i^l} := \{\}$
6:        for all $j^{l' \leq l} \in \mathcal{I}$ occurring in $\mathcal{D}$ such that
          $j^l > i^l$ and NotParentEachOther $(i^l, j^{l'})$ do
7:            if ParentSetIsFrequnet($i^l \cup j^{l'}$) then
            \\check its parent in level1
8:                $C := cover(\{i^l\}) \cap cover(\{j^{l'}\})$
9:                if $|C| \geq \sigma$ then
10:                   $\mathcal{D}^{i^l} := \mathcal{D}^{i^l} \cup \{(j^{l'}, C)\}$
11:               end if
12:           end if
13:       end for
        \\Depth-first recursion
14:       Compute $\mathcal{F}^l[I \cup \{i^l\}](\mathcal{D}^{i^l}, \sigma)$
15:       $\mathcal{F}^l[I] := \mathcal{F}^l[I] \cup \mathcal{F}^l[I \cup \{i^l\}]$
16:   end for
17:end for
```

Table 4.8: LP-eclat Algorithm

The intuition behind the algorithm is as follows:

1. we scan the encoded transaction and convert the database to a vertical tid-list database format for each level. The result is shown as Figure 4.5.

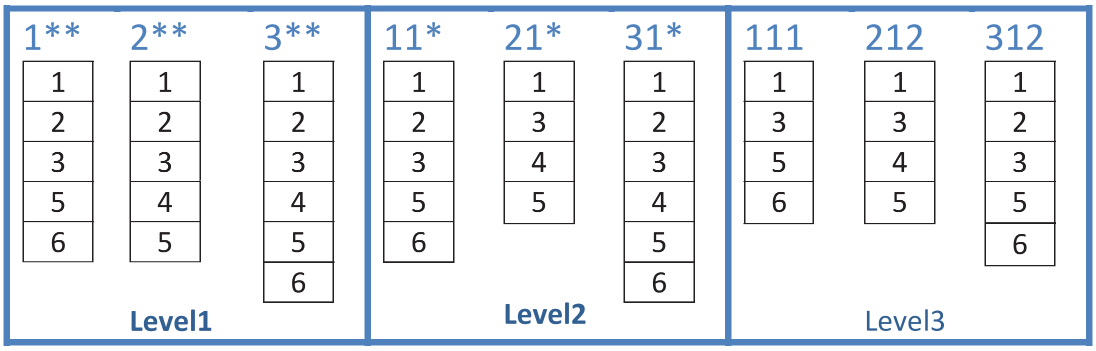


Figure 4.5: Tid-List for frequent one items for each level

2. for each item i of level 1 we use depth-first approach and do the intersection with $j(i < j)$ recursively. The support for the itemsets is the number of results of the intersection. At the same time we construct a prefix tree of all frequent itemsets of first level. The result is shown as Figure 4.6.

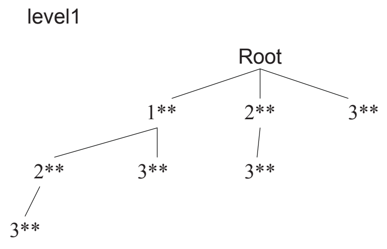


Figure 4.6: Prefix tree of frequent itemsets of first level

3. for each item $i^l(l > 1)$ we use depth-first approach to generate candidate itemset with all its upper levels frequent one items $j^{l' \leq l}(i < j)$ and check if

Table 4.9: Encoded original transaction table for pairs

TID	**Items**
T1	111,112,311
T2	111,112,211
T3	211,221,311
T4	211,221,412
T5	221

its parent of the first level with the prefix tree that we generate in the second step. If its parent is frequent then we make the intersection where i and j have no parents-pairs. The support for the itemsets is the number of results of the intersection, for example: for itemset {111, 21*}, the intersection result is {1, 3, 5}, so the support is 3. We do the step 3 recursively for each item of each level.

4. the output is the union of $\mathcal{F}^l[i]$ for each $i \in \mathcal{I}$, $l \in \mathcal{L}$.

4.5 ET-Pairs

The ET-Pairs pair algorithm uses a breadth-first search technique through candidate itemsets. The ET-Pairs expand each item with all its ancestors in the transaction table creating a new extended transaction table.

The ET-Pairs algorithm can generate multi-level frequent pairs. The ET-Pairs algorithm that generates mixed-level frequent itemsets is described in Table 4.11. The input of the algorithm is the encoded transaction database and the support. The algorithm outputs the set of mixed-level frequent pairs. Note that the frequent items are stored as structured types, that is each item t stores the

Algorithm ET-Pairs:
Input: $\mathcal{D}, \sigma, \mathcal{L}, I \subseteq \mathcal{I}$
Output: $\mathcal{F}_2[I](\mathcal{D}, \sigma)$

```
1:  Let $\mathcal{F}_2 := \emptyset$;
2:  $\mathcal{D}$ := convertTtoMultiLevel ($\mathcal{D}$) \\for each item generate all its parents as new item;
3:  Generate frequent 1 itemset $\mathcal{F}_1$ by single scan of $\mathcal{D}$;
4:   for $i := 1$ to $n$ do
5:   begin
6:      $j := i + 1$;
7:      while ($j \leq n$) do
8:      begin
9:         if ($\mathcal{F}_1[i].id, \mathcal{F}_1[j].id$ are parent each other) then
10:           $j + +$;
11:         else if ($\mathcal{F}_1[i].tid \cap \mathcal{F}_1[j].tid > \sigma$) then
12:              begin
13:                 $\mathcal{F}_2 := \mathcal{F}_2 \cup \{\mathcal{F}_1[i].id, \mathcal{F}_1[j].id\}$;
14:                 $j + +$;
15:              end
16:           end
17:      end
18:  end
```

Table 4.10: ET-Pairs Algorithm

item encoding ($t.id$) and the bitmap corresponding with the transactions where the item appeared in the database ($t.tid$).

Intuitively, the algorithm computes the frequent 1 itemsets for a database $\mathcal{D}$ and then uses $\mathcal{F}_1 \times \mathcal{F}_1$ to compute the candidate set $\mathcal{C}_2$. This computation is done in the usual way, by intersecting the bit vector of each item.

Note that the computation of the set $\mathcal{F}_1 \times \mathcal{F}_1$ involves an expensive step that checks if for two items X, Y is it that $X \preceq Y$ or $Y \preceq X$. For a better performance we optimized this step as follows: convert the encoded item to binary representation by expanding each digit to its four bits binary encoding. As a special case digit $*$ is replaced by binary encoding '1111'. For instance item 112 is encoded with the following binary vector '000100010010', item $11*$ is represented by binary vector '000100011111'. Thus, to check if $112 \preceq 11*$ is enough to check if $112 \wedge 11* = 112$ in their binary representation, where $\wedge$ represents the logical "and" operator. In our example we have that '000100010010' $\wedge$ '000100011111' $=$ '000100010010', meaning that $112 \preceq 11*$. This is indeed an optimization as it is well known that binary operators are executed faster than any other language specific operators.

The ET-Pairs algorithm follows these steps:

1. expand each transaction by adding all ancestors for each item in the transaction;
2. to compute size 1 frequent itemsets we simply scan once the extended database;
3. convert the expanded database to a vertical tid-list database format where we associate with each itemset a list of transactions in which it occurs.
4. Compute the candidate set $\mathcal{C}_2$ as $\mathcal{F}_1 \times \mathcal{F}_1$ by intersecting the tid-list.
5. Finally output is the union of all computed pairs $\mathcal{F}_2$.

Example 18 *1. Let us suppose Table 4.9 to be our original dataset. First we convert the database, represented by Table 4.9, in the one represented by Table 4.11. That is extending each transaction by adding for each item its ancestors.*

2. After the first scan of the database, we generate the following frequent itemsets of size one $\{111,\ 112,\ 11*,\ 1**,\ 211,\ 21*,\ 221,\ 22*,\ 2**,\ 311, 31*, 3**\}$ *with min-sup equal to 2. Each of them has a bit vector list as Figure 4.7.*

Table 4.11: Extended Transaction table using the taxonomy

TID	Items
T1	111,112,11*,1**,311,31*,3**
T2	111,112,11*,1**,211,21*,2**
T3	211,21*,221,22*,2**,311,31*,3**
T4	211,21*,221,22*,2**,4**,41*,412
T5	221,22*,2**

Table 4.12: Frequent Pairs Result

Pairs	Support
111,112	2
211,221	2
211,22*	2
21*,22*	2

3. *We continue mining size-2 itemsets by computing $\mathcal{F}_1 \times \mathcal{F}_1$ and get the final output frequent itemset pair as represented in Table 4.12.*

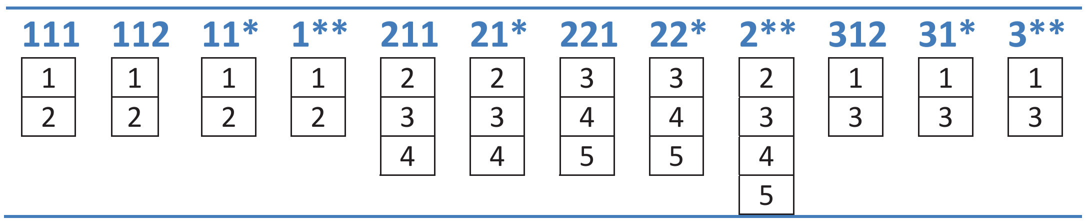


Figure 4.7: Frequent Size-1 items Tid-List

4.6 LP-Pairs

The LP-Pairs pair algorithm uses a hierarchy list structure to store the Frequent size-1 items.

Data Structure: Each node include item encoding (id), item bitmap (tid), item sibling. All the nodes are saved into an array.

Table 4.13 gives a pseudo-code definition for the LP-Pairs algorithm. Similarly to the ET-Pairs algorithm, the input is the transaction database and the minimum support. The difference between ET-Pairs and LP-Pairs it that the later does not expand the transactions in the database and organize the frequent size-1 items within a hierarchical structure. Thus, ET-Pairs may get pruned using the observation from Proposition 6. The performance comparison between these two algorithms is presented in the next section.

Here is the intuition behind this algorithm:

1. First compute the frequent itemsets of size 1 for all taxonomy levels and construct a hierarchy tree $\mathcal{T}$. Where $\mathcal{T}$ is a parent prefix tree and the parent of each node is the prefix of the child. All nodes with the same parent are sibling and are considered ordered. Then, convert the tree structure $\mathcal{T}$ to a list (L) structure using a single scan of the transaction database;

2. The size-2 itemsets are computed from $L \times L$. For optimization, when $\sup((X \in L) \times (Y \in L))$ is less than *min-sup*, the algorithm will stop calculating the support the children nodes of X and Y.

3. Finally, the algorithm outputs the union of all pairs in $\mathcal{F}_2$.

Example 19 *1. After the first scan of the database the algorithm computes*

Algorithm LP-Pairs:

Input: $\mathcal{D}, \sigma, \mathcal{L}, I \subseteq \mathcal{I}$

Output: $\mathcal{F}_2[I](\mathcal{D}, \sigma)$

```
1:  Let $\mathcal{F}_2 := \emptyset$;
2:  Generate frequent 1 itemset listing $L(\mathcal{F}_1)$ using a single scan of $\mathcal{D}$;
3:  for $(i = 1\ to\ n)$ do
4:  begin
5:      $j := i + 1$;
6:      while $(j \leq n)$ do
7:      begin
8:          if $((L[i].id, L[j].id\ are\ parent\ each\ other))$ then
9:              $j++$;
10:         else if $(L[i].tid \cap L[j].tid < \sigma)$ then
11:             $j = L[j].sibling$;
12:         else
13:         begin
14:             $\mathcal{F}_2 := \mathcal{F}_2 \cup \{L[i].id, L[j].id\}$;
15:             $j++$;
16:         end
17:     end
18: end
```

Table 4.13: LP-Pairs Algorithm

the frequent itemsets of level 1 and builds a hierarchy tree $\mathfrak{T}$ with a bit vector list for each node as shown in Figure 4.8 (the min-sup used is 2). The $\mathfrak{T}$ is converted in a list representation as shown in Figure 4.9

2. *Compute $L \times L$ on Figure 4.9. After it calculates sup($1 * * \times 2 * *$) is less than min-sup 2, it will not calculate anymore sup($1 * * \times 21*$). Next it will continue calculating sup($1 * * \times 3 * *$) for the item $3 * *$, sibling of $2 * *$. At this point by pruning from the list structure, it does not need to compute the children nodes if their parent are not frequent pairs.*

3. *Finally, the algorithm outputs the frequent itemset pairs as presented in Table 4.12.*

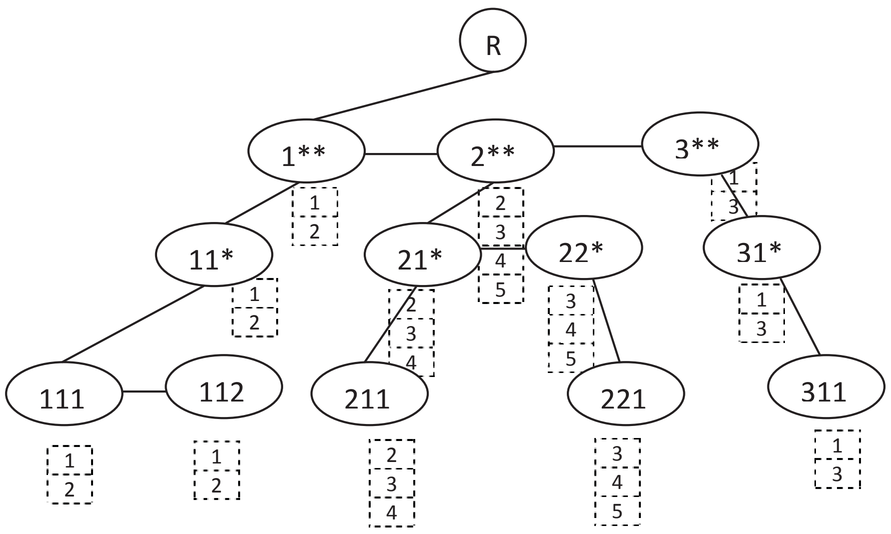


Figure 4.8: Hierarchy Tree for Frequent Size-1 items with Tid-List

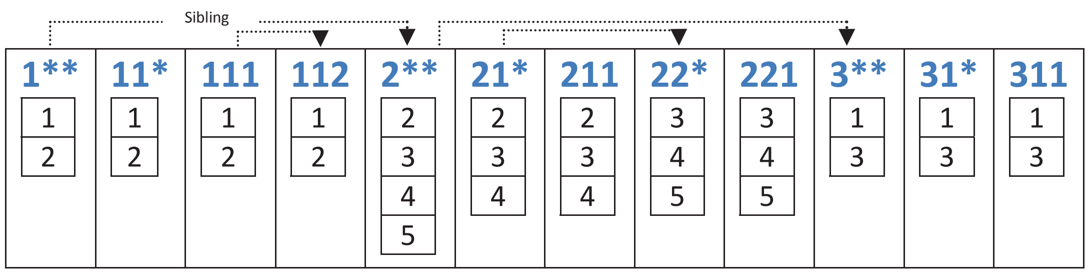


Figure 4.9: Listing Structure for Frequent Size-1 items with Tid-List

Chapter 5

Experimental Result

The algorithms were implemented in C++. All experiments were run on a 32-bit, 2.4GHz machine with 4GB RAM Windows 7 OS. We used various synthetic datasets that were generated with our own data generator. Since there is no data generator served for multiple level datasets, we implement an small tool to generate multiple level datasets. Next, we will describe our data generator used to produce the data for used in our experimental results.

5.1 Data generator

Each item is encoded as integers with digits ranging between 1 and 8. The number of digits is given by the number of levels in the taxonomy, for example 2456 represents an item in a 4 level taxonomy and number 63467 represents an item in a 5 level taxonomy. Figure 5.1 represents a three level and fan-out 3 taxonomy for leaf items 111, 112, 113, 121, 122, 123, 131, 211, 212 . . .

The algorithm proposed, controls the height and width of the taxonomy tree by

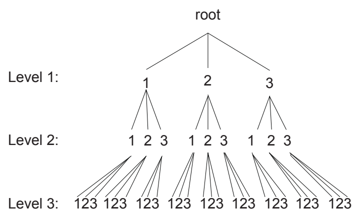


Figure 5.1: Item Represent Tree

input parameters. Also, by input, are specified the number of items generated and the data distribution of these items in the generated transaction set.

The data generator algorithm has the following input parameters:

- l: the number of levels, acceptable values are from 1 to 6;
- i: the total number of items generated. This number should be less than 8^l;
- f: represents the taxonomy tree fanout. This parameter determines the width of the tree;
- T: represents the number of transactions;
- k_1: this number represents the size of the items candidate for frequent itemsets (see description for k_2);
- k_2: gives the number of items considered for the frequent itemsets of size k_1, that is for each transaction k_1 items are selected from the fixed k_2 items. The rest of the items for the transaction are choose form the set of items except the fixed k_2 items. T is the total number of transactions. Thus, the

probability of exists k_1 size frequent itemsets is:

$$\frac{T}{\binom{k_2}{k_1}}$$

- h: represents the homogeneity, that is a high value of h gives us more items related to a lower level ancestor in the taxonomy tree. A low value of h will give items related by a higher level ancestor in the taxonomy tree. The value for h should range between 1 and i;

  Example 20 *From Figure 5.1 we can get the total number of item i is 27. If we set h to 3, then we will get 111,112,113, which have the same parent with each other. If we set h to 27, then we will get 111,113, ...,332,333, which will generate items have less relation among them. So h can help us control generate sparse or dense dataset.*

With this input the algorithm will construct the transaction set. In order to delimit between sets generated with different parameters we conveniently named the files to include the input parameters as well. The file name has the format as "6-200000-10-1020-200000$5000" which means
"levels-Total number of items-transaction size- number of items in the file-from where to select items $ transaction No.". To obtain a dataset similar to the one depicted in Table 4.1, the generator should be executed with the following parameters: $l = 3$; $i \approx 64$; $f = 3$; $T = 6$; $k_1 = 3$; $k_2 \approx 5$; $h \geq 50$

Data generator:
Input: $l, i, f, k_1, k_2, h, T, s$
Output: The output file is a binary representation dataset. It will help us use binary operators to determine two itemsets are parent each other, and save time.

Table 5.1: Data Generator Description

5.2 Frequent itemset Performance Study

In this section we present an experimental comparative performance study, for the two algorithms introduced. The performance is measured based on the type of the input data and the minimum support value. Here, by type of the data, we refer to the size of the data, size of the transactions but also the shape of its taxonomy tree. For the taxonomy tree, we consider two parameters that may vary the width of the tree and the depth of the tree.

For the experiments we considered seven distinct type datasets. The type of the data sets considered was given by our data generator tool based on different input parameters as described in the previous section. We run our algorithms against each such datasets and retain the execution time.

In Figure 5.2, we used deep taxonomy tree (6 level), transactions with a large number of items (10 items in each transaction) and large datasets. For this dataset we started with a low *min-sup* and slowly increase, in order to see the behavior for each of our four algorithms. As it can be seen, for small *min-sup*, the best choice is the LP-apriori algorithm as it filters more data during the candidate generation

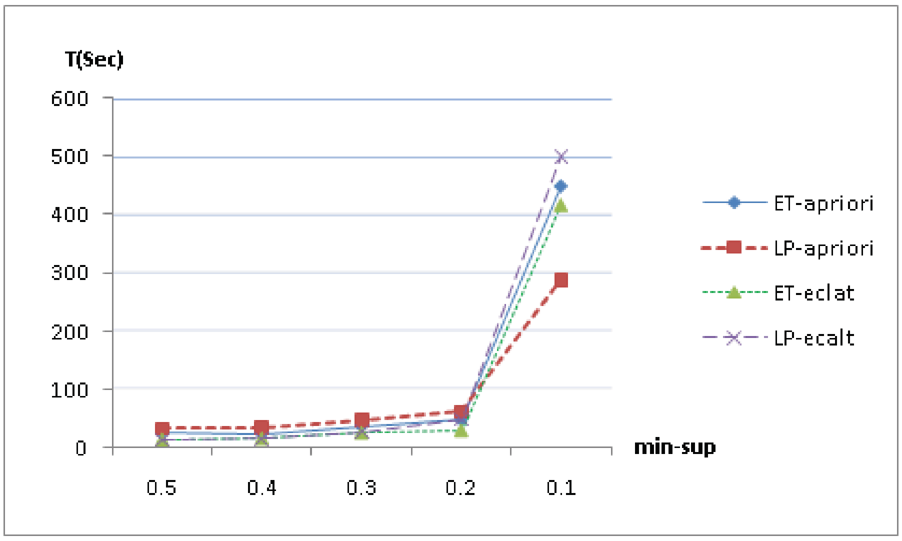


Figure 5.2: 6-200000-10-50-200000$20000

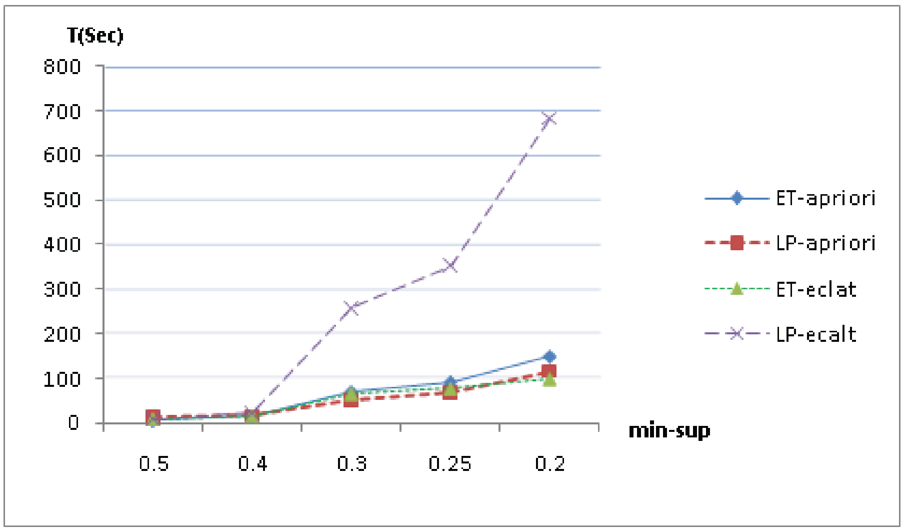


Figure 5.3: 6-200000-15-30-200000$3000

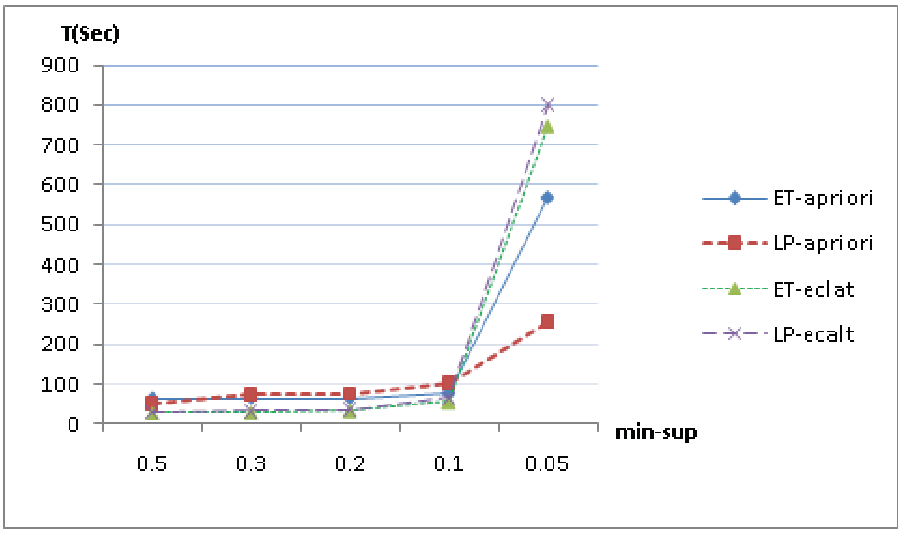


Figure 5.4: 6-200000-5-50-200000$100000

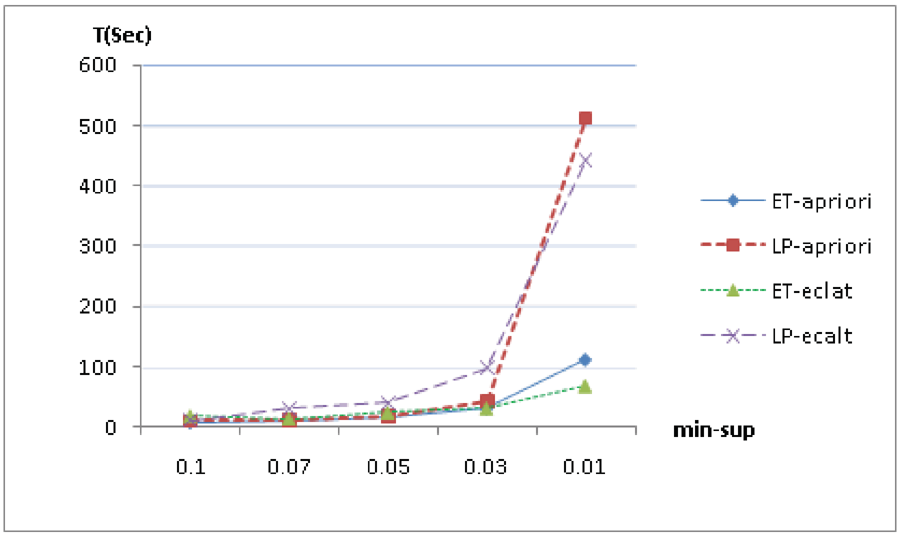


Figure 5.5: 6-100000-3-30-100000$2000

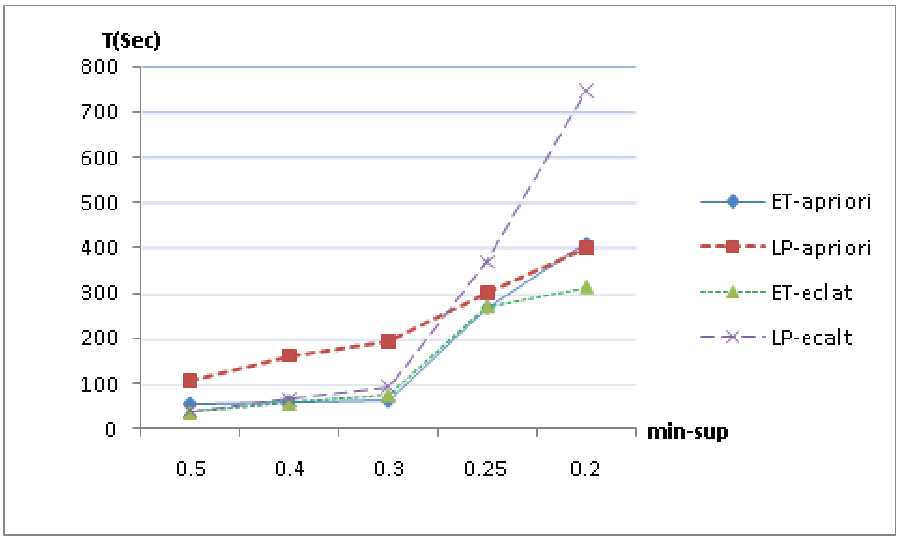


Figure 5.6: 3-500-10-30-500$100000

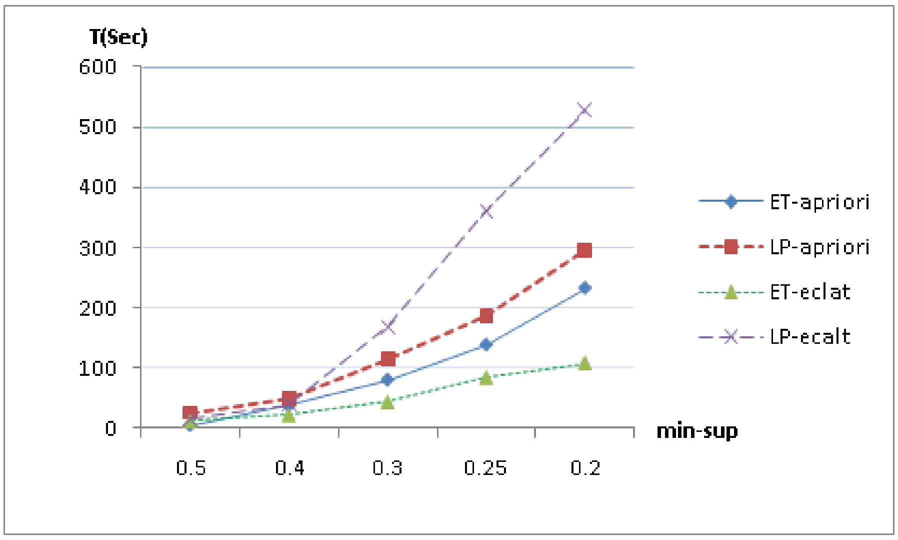


Figure 5.7: 3-500-15-30-500$3000

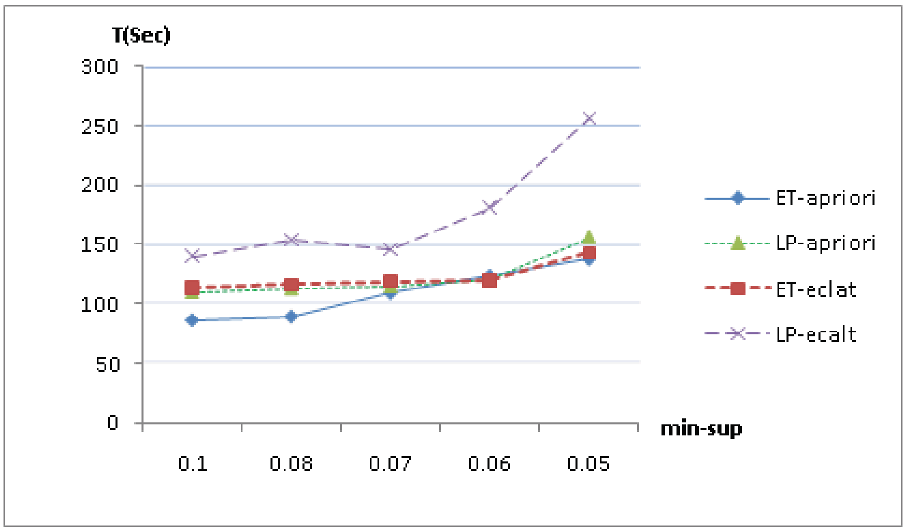


Figure 5.8: 3-500-5-30-500$100000

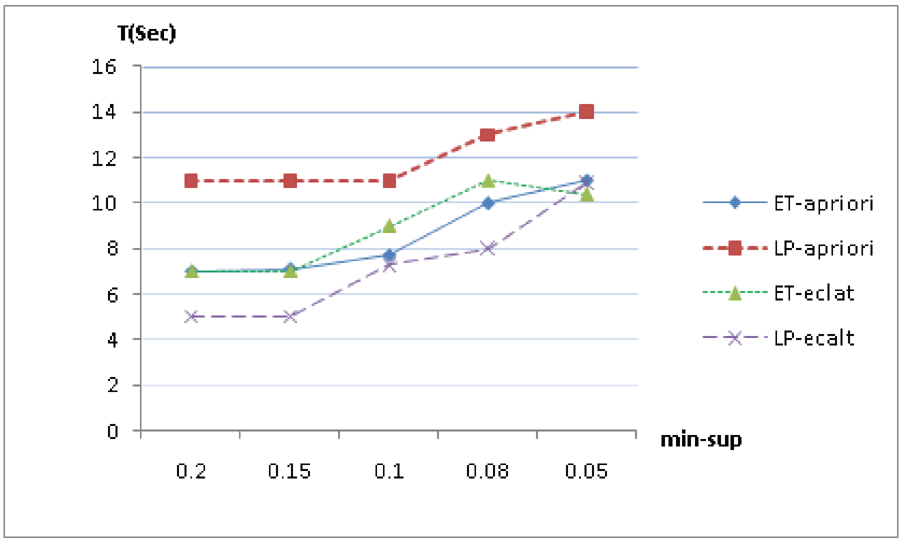


Figure 5.9: 6-200000-5-5000-200000$10000

phase for sparse datasets and also it is faster than ET-apriori algorithm, as it does not extend the already large transactions.

Next, we used we changed the previous dataset by decreasing the size of the dataset (3000 transactions). In this case, as it can be seen in figure 5.3, ET-eclat is faster for small *min-sup*, but as we increase the *min-sup* LP-apriori runs faster on the same dataset. This is the expected behavior as LP-apriori can filter more candidates during the pruning phase. On the other hand, ET-eclat runs faster for lower *min-sup*.

In the following experiment we go back for larger datasets, but using smaller transaction size (5 items in each transaction). The result of this experiment is shown in Figure 5.4. As it can be seen, LP-apriori runs faster on small values for the *min-sup*. This is mainly because when the transaction number is large and small *min-sup*, the tid list used in ET-eclat is large, and makes ET-eclat slower. Also, ET-apriori does not filter many items in its pruning phase, made him slower than LP-apriori.

In the next scenario,as Figure 5.5, we considered both the transaction size and the size of the dataset as small. In this case ET-Eclat is faster when using small values for the *min-sup*, as the size of the tid-list is smaller and it is processed faster. As the *min-sup* is small the apriori based algorithms are slower as there is not much filtering during candidate pruning.

Figure 5.6, represents the experiment run on a dataset with a wide taxonomy tree , large transaction size and a large number of transactions (100000 transactions), that is a sparse distribution of data in the dataset. In this case, as there are a lot of candidates considered, ET-Eclat runs faster than the other algorithms independent on the *min-sup* value. This is the expected behavior as in the case of ET-Apriori the transaction size gets very large by the transaction extension process.

The next dataset considered, Figure 5.7, used a wide taxonomy with long transaction size but with a small number of transactions. For this dataset, we obtain similar results as for the previous case, making the ET-Eclat the favorite algorithm for this type of dataset too.

In the following scenario, we kept the wide taxonomy tree, but we considered small transaction sizes and with a large number of transactions. This gives a sparse distribution of the items in the dataset. In this scenario, Figure 5.8, ET-Apriori is the fastest one. This is explained by the dataset small transaction size. By extending these small transactions ET-Apriori doesn't add to much workload on the algorithm. ET-Eclat is slower again because of its large tid-list that is due to the large number of transactions.

In the last scenario considered, Figure 5.9, for the dataset creation we used a deep taxonomy tree, short transaction size, medium number of transactions and also we partition the items in the dataset in 2 disjoint sets such that each transaction, excepting a small number, contains items only from one of these two partition. Again, this case gives us a sparse data distribution in the dataset. For large *min-sup* values LP-Eclat runs faster. This is because LP-Eclat filters much of the candidates even before calculating their support. On the other hand, for small *min-sup* values, most of the one level item sets becomes frequent, slowing down LP-Eclat. In this case, ET-eclat is the runs faster.

From the experiment result, we can conclude the following: ET-apriori is general good for small size of transaction items in large dataset; LP-apriori is faster for small *min-sup* and large transaction items in sparse datasets; ET-eclat is faster for dense and small dataset; LP-eclat is faster for skewed distribution database.

5.3 Frequent Pairs Performance Study

In this section we compare the performance of the two algorithms introduced in Section 4.5 and 4.6. The performance is measured based on the shape of the input data and the minimum support value. Here, by shape of the data, we refer to the size of the data, size of the transactions and also the shape of its taxonomy tree, thus the input parameters for the data generator algorithm. For the taxonomy tree, we consider two parameters that may vary: (i) the width of the tree and (ii) the depth of the tree.

For the experiments we considered six distinct shape of datasets and one real world GROCERIES dataset [19]. The shape of the data sets considered were given by our data generator tool based on different input parameters as described in the previous subsection.

In Figure 5.10 (top left), we used deep taxonomy tree (6 level), transactions with a large number of items (10 items in each transaction) and a large dataset. For this dataset we started with a low *min-sup* and slowly increase it, in order to see the behavior for each of our two algorithms. As it can be seen, for small *min-sup*, LP-Pairs algorithm discovers the frequent mixed pairs faster as it needs to compute less data during the candidate generation phase for these dense datasets.

Next, we changed the previous dataset by increasing the size of each transaction (15 items in each transaction). In this case, as it can be seen in Figure 5.10 (top right), LP-Pairs is faster. This is an expected result as LP-Pairs will skip

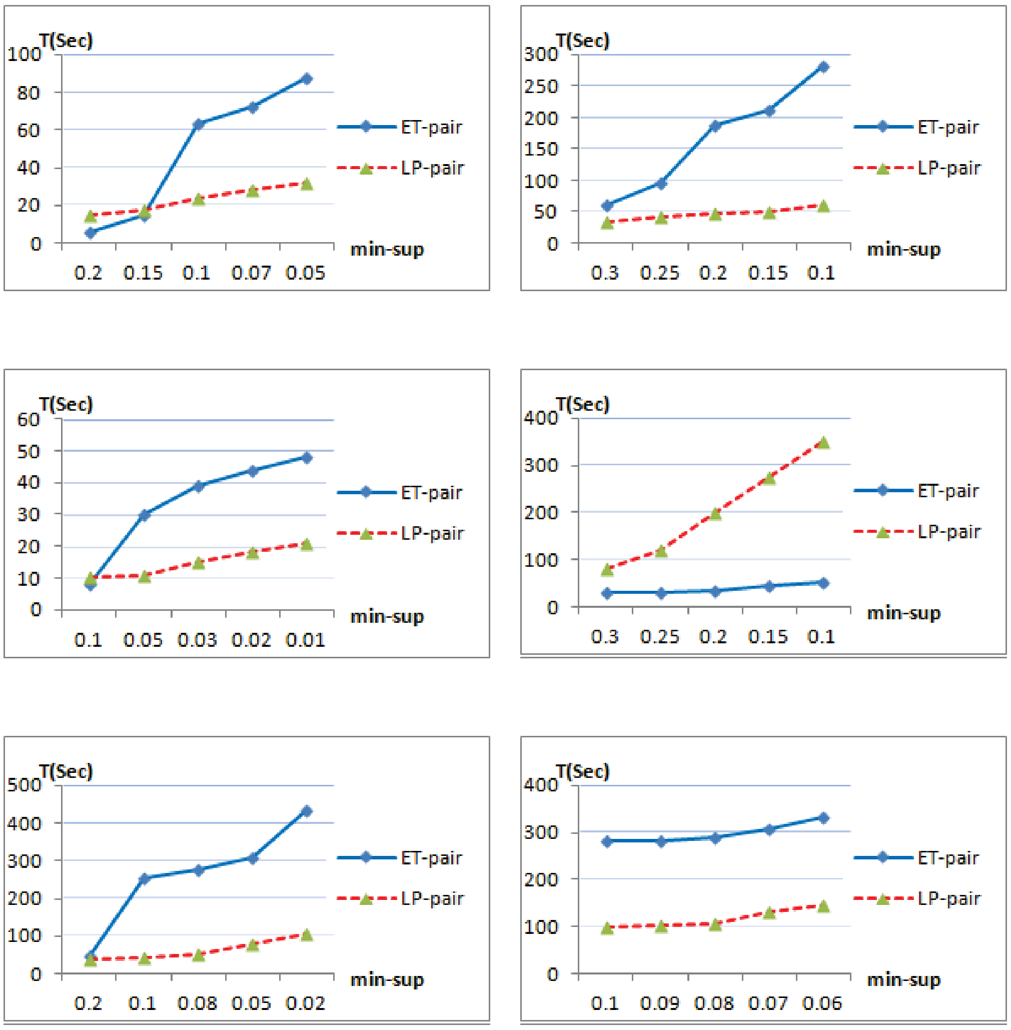


Figure 5.10: **Top left:** 6-200000-10-50-200000$3000; **top right:** 6-200000-15-30-200000$3000; **center left:** 6-200000-5-50-200000$3000; **center right:** 6-200000-5-50-200000$100000; **bottom left:** 3-500-15-30-500$3000; **bottom right:** 3-500-5-30-500$100000.

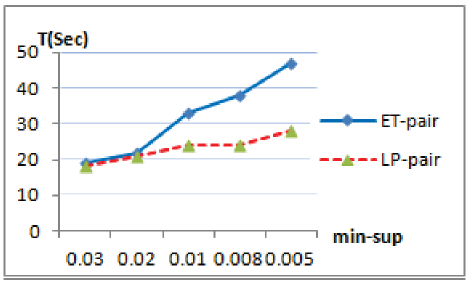


Figure 5.11: Real World Groceries

more items during the pruning phase.

In the third experiment we used again small datasets, but in this case using smaller transaction size (5 items in each transaction). The result of this experiment is shown in Figure 5.10 (center left). As it can be seen, LP-Pairs algorithm runs faster on small values for the *min-sup*. This is mainly because when the transaction number is small and small *min-sup*. More frequent 1 items are generated. LP-Pairs computes less than ET-Pairs.

In the next scenario, as Figure 5.10 (center right), we considered the transaction size as small(5 items in each transaction) and the size of the dataset as big . ET-Pairs is fast when the *min-sup* is getting smaller. This is because the transaction size is big and number of frequent 1 items are small and we can't gain much by pruning from our hierarchy structure. In this case the total number of frequent pairs is less than 100.

Figure 5.10 (bottom left), represents the result for running the experiment on a dataset with a wide taxonomy tree and with long transaction size but with a small number of transactions, small number of levels in the taxonomy, thus, a sparse distribution of data in the dataset. Since it will generate more frequent 1 items when the *min-sup* is small. LP-Pairs will compute less data than ET-Pairs.

The next dataset considered, Figure 5.10 (bottom right), uses a dataset with a wide taxonomy tree, small transaction size and a large number of transactions (we used 100,000 transactions), still using a taxonomy with only a few levels, thus, a sparse distribution of data in the dataset. For this dataset, we obtain similar results as for the previous case, making the LP-Pairs the favorite algorithm for this type of dataset too.

In the next scenario, we used a real world database. The GROCERIES dataset [19] includes 1-month of the point-of-sale transactions in a local grocery store. The taxonomy of items is provided and it represents item categorization used in this store. The dataset contains 9,835 transactions, it has three levels of abstraction and 170 different items. Comparing our two algorithms against this data, Figure 5.11, LP-Pairs was fastest than ET-Pairs algorithm. This is because as the *min-sup* decreases, more pairs are generated, so LP-Pairs will compute less data than ET-Pairs.

From the experiment result, we can conclude the following:

1. ET-Pairs is in general a good choice for small size of items in large dataset with high *min-sup*.

2. If the *min-sup* is high, we will get a small number of frequent 1 item sets, thus, the calculation is not to high, so there is no or small pruning from the hierarchy structure.

3. LP-Pairs is faster for small *min-sup* and large transaction items in large number of items datasets.

4. In general, no matter of the width and height of the taxonomy tree, most of the time, LP-Pairs is faster than ET-Pairs.

Chapter 6

Future Work

All of our six methods assume that the input file can fit in main memory, and we use the uniform *min-sup* for all the levels. Therefore, the future work can be improved by processing large data files from the disk, and using different *min-sup* for different levels or special items. We examine the applications of finding frequent itemset at flexible concept level, which means that we can find frequent itemset patterns like {11*, 2**}, but we cannot find frequent itemsest at mix concept level, for example:{1*1, 2**}. This is important to note for the future studies, and not difficult to implement, but because of data complexity, the running time will become a big issue. We also implement two algorithms to find frequent pairs at mix-level since it's more useful. Finding frequent itemsets of size greater than 2 will introduce new and interesting aspects that need to be taken into account. Although several algorithms are proposed, none of them use the hierarchy pruning. Therefore the challenge for researchers is how to efficiently organize the data structure, and how to use hierarchy pruning to mine size k frequent itemset.

Chapter 7

Conclusions

Throughout the past decade, mining frequent itemsets has been developed from single concept level to multiple concept level. A lot of data mining methods were introduced, and based on these methods, we have selected the fastest ones to develop flexible multiple level frequent itemset mining which will help make data mining reach a higher level. In this paper, we have developed two algorithms: ET-apriori and LP-apriori. Our experimental results show that different algorithms will have higher performance for different distributions of data. Generally, ET-apriori is good for small transaction items in large datasets; LP-apriori is fast for small *min-sup* and large transaction items in sparse datasets. Mining flexible multiple level frequent itemsets may lead to the discovery of more detailed information from data. We also developed two algorithms for pairs: ET-Pairs and LP-Pairs. Our experimental results show that ET-Pairs is usually better for small transaction items in large datasets, while LP-Pairs tends to perform better for small *min-sup* and large transaction items in sparse datasets. Mining mix-level frequent itemsets can lead to the discovery of more detailed information from data.

References

[1] *Proceedings of the Fourteenth International Conference on Data Engineering, February 23-27, 1998, Orlando, Florida, USA*. IEEE Computer Society, 1998. 80

[2] Ramesh C. Agarwal, Charu C. Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. In *KDD*, pages 108–118, 2000. 3

[3] Ramesh C. Agarwal, Charu C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for generation of frequent item sets. *J. Parallel Distrib. Comput.*, 61(3):350–371, 2001. 3

[4] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD Conference*, pages 207–216, 1993. 2, 3, 10, 14, 35, 39, 42

[5] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996. 10

[6] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining

association rules in large databases. In *VLDB*, pages 487–499, 1994. 1, 3, 14, 31

[7] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee L. P. Chen, editors, *ICDE*, pages 3–14. IEEE Computer Society, 1995. 3

[8] Necip Fazil Ayan, Abdullah Uz Tansel, and M. Erol Arkun. An efficient algorithm to update large itemsets with early pruning. In *KDD*, pages 287–291, 1999. 3

[9] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential pattern mining using a bitmap representation. In *KDD*, pages 429–435. ACM, 2002. 3

[10] Ferenc Bodon. A fast apriori implementation. In *In Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, 2003. 3, 39, 40, 42, 45

[11] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In Joan Peckham, editor, *SIGMOD Conference*, pages 255–264. ACM Press, 1997. 2

[12] Douglas Burdick, Manuel Calimlim, and Johannes Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *ICDE*, pages 443–452. IEEE Computer Society, 2001. 3

[13] David Wai-Lok Cheung, Jiawei Han, Vincent T. Y. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremen-

tal updating technique. In Stanley Y. W. Su, editor, *ICDE*, pages 106–114. IEEE Computer Society, 1996. 3

[14] David Wai-Lok Cheung, Sau Dan Lee, and Ben Kao. A general incremental technique for maintaining discovered association rules. In Rodney W. Topor and Katsumi Tanaka, editors, *DASFAA*, volume 6 of *Advanced Database Research and Development Series*, pages 185–194. World Scientific, 1997. 3

[15] B. Goethals. Survey on frequent pattern mining. Manuscript, 2003. 2

[16] Gösta Grahne, Laks V. S. Lakshmanan, Xiaohong Wang, and Ming Hao Xie. On dual mining: From patterns to circumstances, and back. In *ICDE*, pages 195–204, 2001. 3

[17] Gösta Grahne and Jianfei Zhu. Fast algorithms for frequent itemset mining using fp-trees. *IEEE Trans. Knowl. Data Eng.*, 17(10):1347–1362, 2005. 3

[18] Robert Györödi, Cornelia Györödi, Mirela Pater, Ovidiu Boc, and Zoltan David. Afopt algorithm for multi-level databases. In *SYNASC*, pages 129–133, 2005. 3

[19] Michael Hahsler, Kurt Hornik, and Thomas Reutterer. Implications of probabilistic data modeling for mining association rules. In Myra Spiliopoulou, Rudolf Kruse, Christian Borgelt, Andreas Nürnberger, and Wolfgang Gaul, editors, *GfKl*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 598–605. Springer, 2005. 71, 74

[20] Jiawei Han and Yongjian Fu. Discovery of multiple-level association rules from large databases. In *VLDB*, pages 420–431, 1995. 3, 25, 27, 32

[21] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, 2004. 2, 3, 15, 39

[22] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In *ICDE* [1], pages 392–401. 3

[23] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999. 3

[24] Donald E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998. 39

[25] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovering frequent episodes in sequences. In *KDD*, pages 210–215, 1995. 3

[26] Runying Mao. Adaptive-fp: An efficient and effective method for multi-level multi-dimensional frequent pattern mining, 2001. 3, 31

[27] Banu Özden, Sridhar Ramaswamy, and Abraham Silberschatz. Cyclic association rules. In *Proceedings of the Fourteenth International Conference on Data Engineering, February 23-27, 1998, Orlando, Florida, USA* [1], pages 412–421. 39

[28] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. An effective hash based algorithm for mining association rules. In Michael J. Carey and Donovan A.

Schneider, editors, *SIGMOD Conference*, pages 175–186. ACM Press, 1995. 2

[29] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Pruning closed itemset lattices for associations rules. In Mokrane Bouzeghoub, editor, *BDA*, 1998. 23

[30] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In Catriel Beeri and Peter Buneman, editors, *ICDT*, volume 1540 of *Lecture Notes in Computer Science*, pages 398–416. Springer, 1999. 23

[31] Jian Pei, Jiawei Han, and Runying Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30, 2000. 23

[32] V. Venkata Ramana, M V Rathnamma, and A. Rama Mohan Reddy. Article: Methods for mining cross level association rule in taxonomy data structures. *International Journal of Computer Applications*, 7(3):28–35, September 2010. Published By Foundation of Computer Science. 31

[33] Ashok Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *VLDB*, pages 432–444. Morgan Kaufmann, 1995. 2

[34] Ramakrishnan Srikant and Rakesh Agrawal. Mining generalized association rules. In *VLDB*, pages 407–419, 1995. 3, 40

[35] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin, editors, *EDBT*, volume 1057 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 1996. 3

[36] Shiby Thomas, Sreenath Bodagala, Khaled Alsabti, and Sanjay Ranka. An efficient algorithm for the incremental updation of association rules in large databases. In *KDD*, pages 263–266, 1997. 3

[37] Hannu Toivonen. Sampling large databases for association rules. In Vijayaraman et al. [38], pages 134–145. 2, 14

[38] T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors. *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India.* Morgan Kaufmann, 1996. 82

[39] Mohammed J. Zaki and Karam Gouda. Fast vertical mining using diffsets. In *Proc. of the 9th ACM SIGKDD Intl. Conf. on Knowledge discovery and data mining*, KDD '03, pages 326–335. ACM, 2003. 2, 3, 23

[40] Mohammed Javeed Zaki. Scalable algorithms for association mining. *IEEE Trans. Knowl. Data Eng.*, 12(3):372–390, 2000. 2, 3, 20, 39, 48

[41] Mohammed Javeed Zaki and Ching-Jiu Hsiao. Charm: An efficient algorithm for closed itemset mining. In Robert L. Grossman, Jiawei Han, Vipin Kumar, Heikki Mannila, and Rajeev Motwani, editors, *SDM*. SIAM, 2002. 23