CHAPTER 3

Discovery of Frequent Superset

3.1 Introduction

In this thesis, to discover the frequent superset, we propose three algorithms, Apriori-C, Eclat-C, and data complement technique (DCT), which are all based on the set complement view of point. We do not find frequent supersets directly, but the complement of frequent supersets, and then obtain frequent supersets by taking the complement of those patterns. Apriori and Eclat are two representatives for breadth-first and depth-first search strategy in the problem of frequent subset mining, respectively. They are also the most efficient algorithm in their categories [9]. Therefore, we choose these two algorithms as the bases to design our new algorithms for this novel mining task.

First of all, we give some related definitions, and describe the Baseline method for comparing performance with our three algorithms in the experimental studies.

Definition 3.1 Let $I=\{i_1, i_2, \ldots, i_n\}$ denote the set of all items, $X=\{j_1, j_2, \ldots, j_m\}$ be an itemset, where $j_p \in I$, $\forall$ p, p=1~m. An itemset $X'=I \backslash X=\{i_q \mid \forall q=1\sim n,\ i_q \notin X\}$ is defined as the complement of X, denoted as X'. Moreover, given a database $D=\{T_1, T_2, \ldots, T_k\}$, the complement of D is defined as $D'=\{T_1', T_2', \ldots, T_k'\}$.

Example 3.1 Let I={1, 2, 3} be the set of all items. Suppose we have two itemsets X_1={1, 3}, X_2={2, 3} and a database D={X_1, X_2}, the complement of X_1, X_2 and D are X_1'={2}, X_2'={1}, and D'={{2}, {1}}, respectively.

3.2 Baseline Method

The brute force way to look for frequent superset is to check all the candidate superset. Given a database with n items, there will be (2^n -1) candidate supersets. For example, given the following database with five items, shown in Figure 3.1, the set of all items is I={1, 2, 3, 4, 5}, and all candidate superset are listed in Table 3.1. We can obtain the number of superset to be ($2^{|I|}$ -1) = (2^5 -1) = 31, where $|I|$ is the number of different items.

TID	Items
100	1, 3, 4
200	3, 5
300	1, 5
400	5
500	2, 4

Figure 3.1: A database with 5 transactions and the set of all items is I={1, 2, 3, 4, 5}.

Table 3.1: The list of candidate supersets, and their supports.

Itemset	Support	Itemset	Support	Itemset	Support	Itemset	Support
{1}	0	{1, 5}	2	{1, 2, 4}	1	{3, 4, 5}	2
{2}	0	{2, 3}	0	{1, 2, 5}	2	{1, 2, 3, 4}	2
{3}	0	{2, 4}	1	{1, 3, 4}	1	{1, 2, 3, 5}	3
{4}	0	{2, 5}	1	{1, 3, 5}	3	{1, 2, 4, 5}	3
{5}	1	{3, 4}	0	{1, 4, 5}	2	{1, 3, 4, 5}	4
{1, 2}	0	{3, 5}	2	{2, 3, 4}	1	{2, 3, 4, 5}	3
{1, 3}	0	{4, 5}	1	{2, 3, 5}	1	{1, 2, 3, 4, 5}	5
{1, 4}	0	{1, 2, 3}	0	{2, 4, 5}	2		

For each candidate superset, we have to scan database 31 times in the above example to determine the support values of each superset and check if they are frequent. This is time consuming and not efficient.

To imitate the breadth-first search and counting occurrence strategy of Apriori algorithm, we design the Baseline method by making a little modification from the original Apriori algorithm.

1) C_1 = { all candidate 1-superset }
2) L_1 = { frequent superset with size of 1 };
3) **for** ($k = 2$; $L_{k-1} \neq \phi$; k++) **do begin**
4) C_k = apriori-gen(C_{k-1});
5) **forall** transactions $t \in D$ **do**
6) superset(C_k ,t,0); //count support of items in C_k
7) $L_k = \{c \in C_k | c.\text{count} \geqq \text{minsup}\}$;
8) **end**
9) Answer = $\bigcup_k L_k$;

Figure 3.2: The algorithm of the Baseline method.

First, Baseline starts from 1-superset, and counting their support values. Next, it generates candidate (k+1)-superset from the infrequent k-supersets, and stores them in a prefix tree. Because if a superset is not frequent, the supersets of it may be frequent, we still need checking the supersets of infrequent superset. Baseline improves the brute force method, and reduces the number of times of database scan from $2^{|I|}$-1 to the maximum length of frequent superset. In worse case, the maximum length of superset is $|I|$, where $|I|$ is the number of different items. Figure 3.2 is the algorithm of Baseline method and it can be seen that the principal difference is in the sixth line. We check how many transactions are contained in the candidate superset, rather than how many transactions contain the candidate

itemset in original Apriori algorithm. Figure 3.3 is the pseudo code of the superset function.

```
Procedure superset (Node:n,Transaction:t,Int:i)
1)  Match=false;
2)  if(n is internal node) then begin
3)      if(i=t.size) then begin
4)          Match=true;
5)          i--;
6)      end
7)      forall children c of n do begin
8)          if(c < t[i]) then begin
9)              call superset(c, t, i);
10)         else if(c > t[i]) then
11)             if(Match) then begin
12)                 call superset(c, t, i+1);
13)             else break;
14)         else
15)             call superset(c, t, i+1);
16)         end
17)     else
18) else
19)     n.counter++;
20) end
```

Figure 3.3: The pseudo code for counting the support values of candidate supersets.

Since the candidate supersets are stored in a prefix tree, the mission of superset function is to find the paths which contain the certain transaction.

Example 3.2 Considering the transaction database depicted in Figure 3.1, and given the minimum support is three transactions, the frequent superset discovery process is illustrated in Figure 3.4. The answer is shown in Figure 3.5 which joins L_3, L_4 and L_5. The total number of database scan is four.

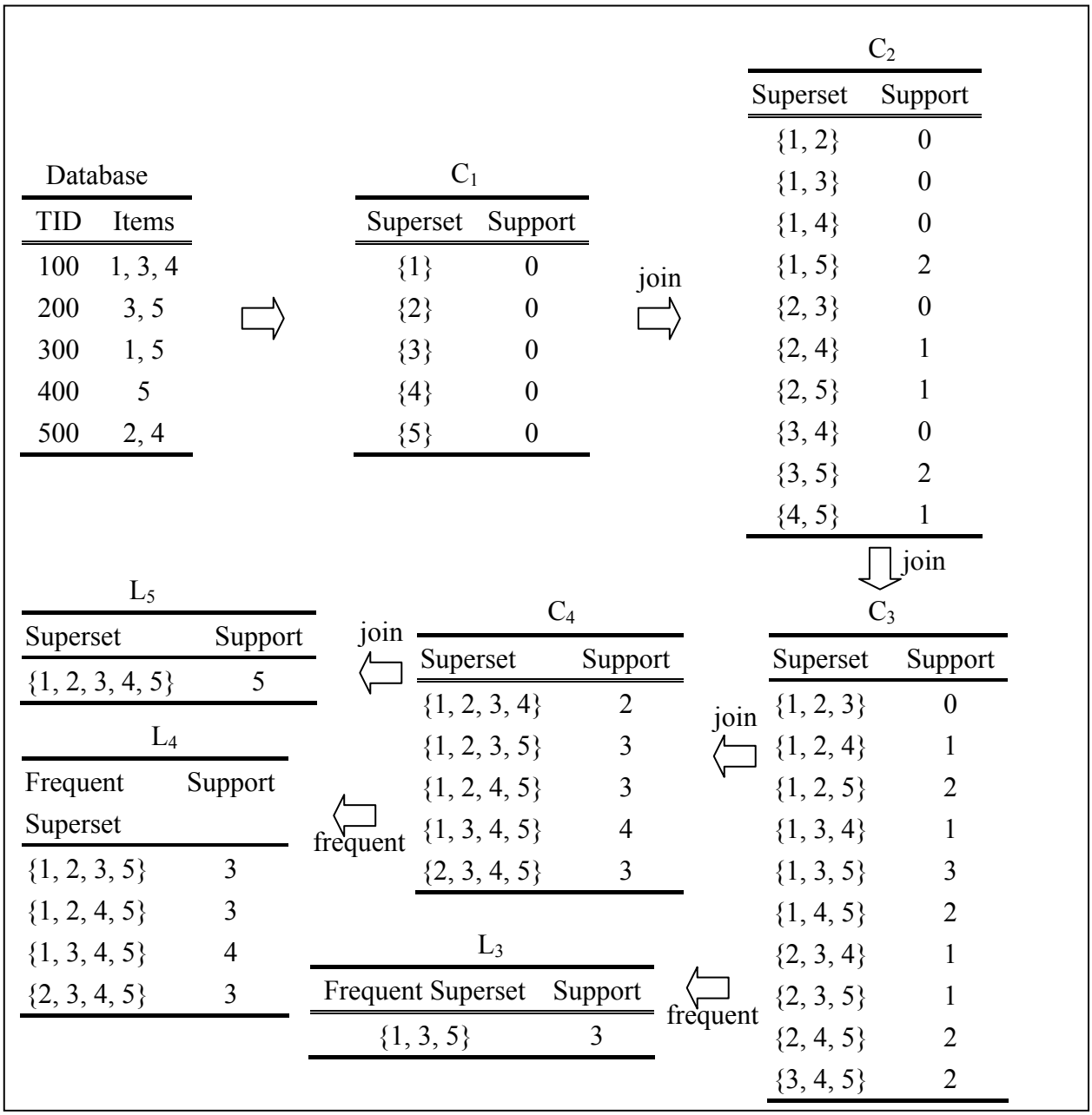

Figure 3.4: The Baseline method.

Answer	
Frequent Superset	Support
{1, 3, 5}	3
{1, 2, 3, 5}	3
{1, 2, 4, 5}	3
{1, 3, 4, 5}	4
{2, 3, 4, 5}	3
{1, 2, 3, 4, 5}	5

Figure 3.5: The result of the Baseline method.

3.3 Algorithm Apriori-C

Baseline method takes the advantage of breadth-first search strategy to reduce the number of database scan successfully. However, it can not lessen the number of candidate supersets in each level, especially the candidate 2-supersets in the second level which is the bottleneck of Apriori-base algorithm. The main reason that causes the non-efficiency candidate reduction of the Baseline method is that it does not inherit the Apriori property: all non-empty superset of a frequent subset must be frequent [8].

Algorithm Apriori-C is an Apriori-based method, which is the level-wise discovery process starting from 1-itemset, 2-itemset, and so on. Each level is also a join-and-prune process. Apriori-C adopts the spirit of Apriori property, and can lessen the number of candidate supersets in each level greatly. We first define some terms and properties before describing the algorithm.

Definition 3.2 Consider two itemsets X_1 and X_2, if the complement of X_1 contains X_2, we say that X_1 complement-contains X_2, denote this situation as $X_1 \supseteq_c X_2$.

Example 3.2 Let X_1={1, 3} and X_2={2, 4}. First, we obtain the set of all items I={1, 2, 3, 4} from X_1 and X_2. Because the complement of X_1 is {2, 4} that contains X_2, we say that X_1 complement-contains X_2, represented as $X_1 \supseteq_c X_2$.

Definition 3.3 Let D={T_1, T_2, …, T_k} be a transaction database with k transactions, and h be the minimum support. If an itemset X, $X \supseteq_c T_{i_j}$, where $1 \leqq i_j \leqq k$, for each j=1, 2, …, m, we say that m is the complement-support of X. If $m \geqq h$, we define X to be a complement-frequent superset of D.

Example 3.3 Given the database in Example 1, suppose that the minimum support is three, the complement-frequent supersets are {1}, {2}, {3}, {4}, and {2, 4}, and their complement-support are 3, 4, 3, 3, and 3, respectively.

Lemma 3.1 Given two itemsets X and T. If $X \supseteq_c T$, then $\forall P \in \text{Pow}(X)$, the powerset of X, $P \supseteq_c$ T.

Proof Suppose the set of all items is I. Because $\forall P \in \text{Pow}(X)$, we can obtain $X \supseteq P$, $I \backslash P \supseteq I \backslash X$, and then $P' \supseteq X'$. In addition, $X \supseteq_c T$, that is $X' \supseteq T$, then $P' \supseteq X' \supseteq$ T, $P \supseteq_c T$.

Example 3.4 Suppose we have a transaction T={2, 4} and an itemset X={1, 3}. Let I={1, 2, 3, 4, 5}, we can obtain the powerset of X, Pow(X)={{1}, {3}, {1, 3}} and the complement is {{2, 3, 4, 5}, {1, 2, 4, 5}, {2, 4, 5}}. Because $X \supseteq_c T$, all of {1}, {3}, and {1, 3} complement-contain T.

By lemma 3.1, Apriori-C states that all subset of a complement-frequent superset must also be complement-frequent. In other words, if not all of the subsets of an itemset are

complement-frequent, the itemset must not be complement-frequent. This is important for reducing the candidate generation.

There are two steps consisted in each pass of proposed algorithm Apriori-C. At the *k*-th pass, first, the candidates with size of *k* are generated from complement-frequent superset with size of (*k*-1) using the original aprior-gen method which is discussed in chapter 2. Next, scan database once and count the complement-support of candidates to determine the complement-frequent supersets. Figure 3.6 gives the algorithm of Apriori-C. The CompSupp procedure in the fifth line is to count the complement-support value of candidate superset.

1) L_1 = {frequent complement-superset with size of 1}
2) **for** ($k = 2$; $L_{k-1} \neq \phi$; k++) **do begin**
3) C_k = apriori-gen(L_{k-1});
4) **forall** transactions $t \in D$ **do**
5) compSupp(root,*t*,*0*); //count complement-support
6) $L_k = \{c \in C_k \mid c.\text{count} \geqq \text{minsup}\}$
7) **end**
8) Answer = $\bigcup_k L_k'$;

Figure 3.6: Algorithm Apriori-C.

Lemma 3.2 Given two itemsets X_1 and X_2. $X_1 \supseteq_c X_2$, if and only if $X_1 \cap X_2 = \phi$.

In order to count complement-support efficiently, we do not take the complement of candidate complement-superset and check if it contains a certain transaction. According to lemma 3.2, we use the property that $X \cap T = \phi$. In other words, if all of the items in an itemset *X* do not exist in a certain transaction *T*, then we can say that *X* complement-contains

T. We build a prefix tree, which is modified from [15] to record the candidates and design an algorithm shown in Figure 3.7 for counting complement-support using the prefix tree for each transaction. Therefore, the mission of CompSupp function is to look for prefix paths which do not exist in a certain transaction.

Up to now, we have obtained all of the complement-frequent superset, and we must take the complement of the complement-frequent superset to derive the frequent supersets.

```
Procedure compSupp (Node:n,Transaction:t,Int:i)
1)  if(n is internal node) then begin
2)     forall children c of n do begin
3)        if(c < t[i]) then begin
4)           call compSupp(c, t, i);
5)        else if(c > t[i]) then
6)           if(exist t[i+1]) then
7)              i++;
8)           else
9)              forall leaf node f large then c do
10)                f.counter++;
11)       else
12)          if( exist t[i+1]) then
13)             i++;
14)       end
15)    end
16) else
17)    n.counter++;
18) end
```

Figure 3.7: An algorithm for counting complement-support.

Example 3.5 Consider the database in Figure 3.8 and assume that minimum support is three. At the first pass, we determine the complement-frequent superset with size of one showed in

L_1. At the second pass, there are six complement-supersets generated, but only {2, 4} is frequent. There are three transactions, {3, 5}, {1, 5}, and {5}, which complement-contain {2, 4}. At last, we obtain the frequent supersets by taking the complement of each frequent-complement superset, shown in the table Answer.

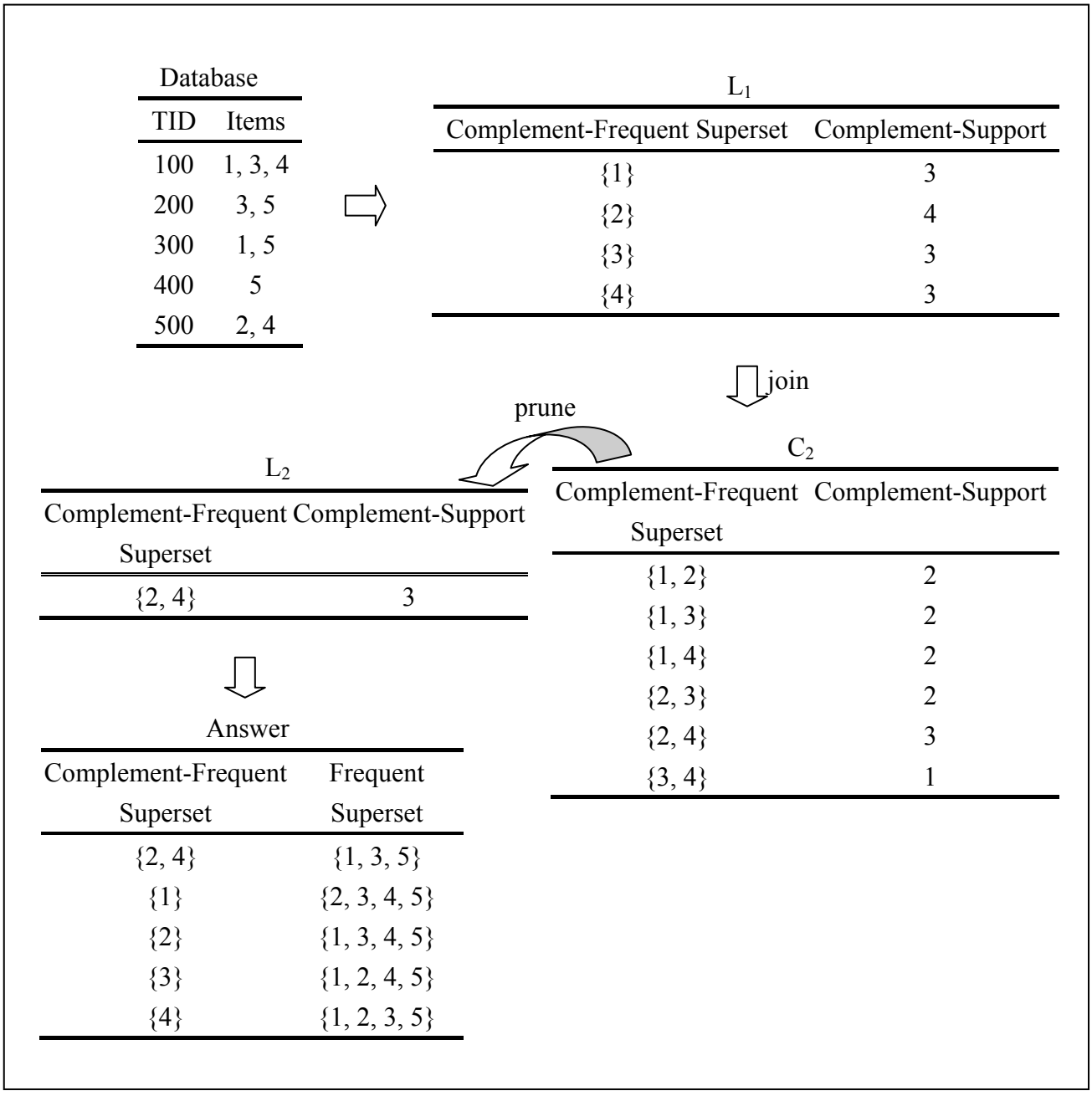

Figure 3.8: An example of Apriori-C.

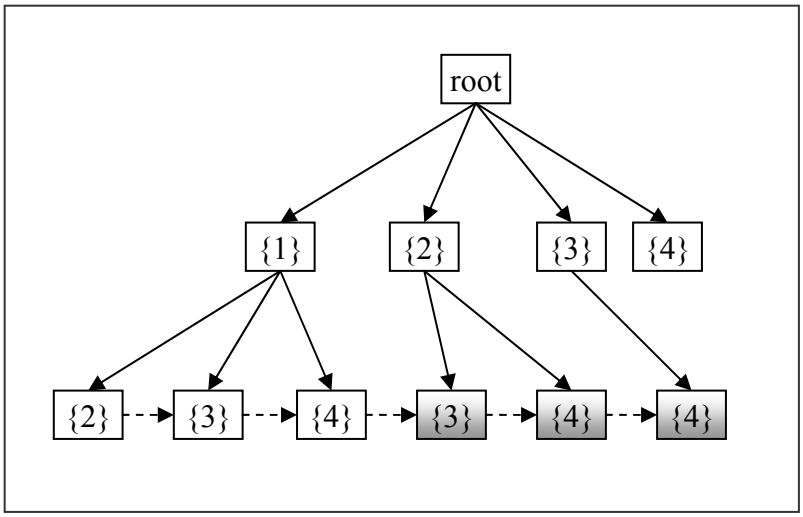

Figure 3.9: The prefix tree when counting the complement-support of C_2 in the example of Figure 3.8.

Figure 3.9 is the prefix tree which stores the candidate complement 2-superset in the example of Figure 3.8. The dashed links are used for speeding up the search. Suppose we have a transaction {1}, and we want to know which candidates complement-contain this transaction. First, the node {1} is matched. Next, because there are no other items that are larger than {1} in the transaction, we can obtain that the paths with the first node after {1} in level one are the answers which are marked in darker color. In this situation, the dashed links can help us to find out the next node without traversing the tree, and this is important when the tree is huge.

Although Baseline method and Apriori-C algorithm use the breadth-first search and counting occurrence strategy, Apriori-C has better efficiency. Because Apriori-C adopts the Apriori property based on the following observation: if an itemset X is not a

complement-frequent superset, the superset of X is not a complement-frequent superset either. It needn't generate all candidate complement supersets, but only the ones that their subsets are all complement-frequent.

3.4 Algorithm Eclat-C

According to the characteristics of complement-frequent superset, the shorter the frequent supersets are, the longer the complement-frequent supersets are. For Apriori-C, in order to find shorter frequent supersets, we have to go more passes to obtain longer complement-frequent supersets. It both increases the times of database scan and the candidate generation. Eclat is an algorithm which is beneficial for mining frequent subset when the patterns are long [9], owing to its depth-first search and transaction-ID list intersecting mechanisms [4][11]. Eclat maintains the transaction-ID list for each frequent itemset. Each transaction-ID list records the set of transaction-ID corresponding to the itemset. When generating 2-itemset, Eclat intersects two transaction-ID lists of 1-itemsets.

Definition 3.4 Let D be a transaction database, and F(D) denote the set of frequent superset of D. For a frequent superset $X \in$ F(D), if $\forall Y \in$ F(D), $Y \not\subset X$, where $Y \neq X$, we say that X is the minimum frequent superset.

Example 3.6 Consider the Answer table in Figure 3.8. The frequent superset {1, 3, 5}, {2, 3, 4, 5}, and {1, 2, 4, 5} are minimum frequent superset.

Difinition 3.5 Let D be a transaction database, and F'(D) denote the set of complement-frequnet superset of D. For a complement-frequent superset $X \in$ F'(D), if $\forall Y \in$ F'(D), $X \not\subset Y$, where $Y \neq X$, we say that X is the maximum complement-frequent superset,

and *X'* must be a minimum frequent superset of *D*.

Example 3.7 Consider the Answer table in Figure 3.8. The complement-frequent superset {2, 4}, {1}, and {3} are maximum complement-frequent superset.

In this thesis, we design an Eclat-based algorithm, named Eclat-C, which takes the advantages of Eclat. The depth-first search strategy lets it find the **maximum complement-frequent superset** directly, while the transaction-ID list intersection strategy lets it possible to scan database only once.

To find frequent superset through Eclat-C, we maintain transaction-ID list to record the ID of transactions that are complement-contained in the candidate complement supersets.

Database

TID	Items
100	1, 3, 4
200	3, 5
300	1, 5
400	5
500	2, 4

Transaction-ID List for Each Item

Items	TID List	Complement Support
1	200, 400, 500	3
2	100, 200, 300, 400	4
3	300, 400, 500	3
4	200, 300, 400	3
5	100, 500	2

The Depth First Search Path for {2}

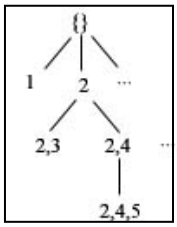

Figure 3.10: An example for Eclat-C.

Example 3.8 Consider the database in Figure 3.10, and let the minimum support be three transactions. The transaction-ID list records the ID of transactions that are complement-contained in the 1-itemset. For example, the itemset {2} complement-contains the transactions, 100, 200, 300, and 400. The complement-support of {2} is four, which is larger than the minimum support. Because {2} is complement-frequent, the complement of the

itemset {2}, that is {1, 3, 4, 5}, is a frequent superset. In the right of Figure 3.10 shows the search path for generation of supersets with the prefix {2}. The final results are shown in the second column of Table 3.2.

Table 3.2: The answer for the example in Figure 3.10.

Complement-Frequent Superset	Frequent Superset	TID List	Complement-Support
{1}	{2, 3, 4, 5}	200, 400, 500	3
{2}	{1, 3, 4, 5}	100, 200, 300, 400	4
{2, 4}	{1, 3, 5}	200, 300, 400	3
{3}	{1, 2, 4, 5}	300, 400, 500	3
{4}	{1, 2, 3, 5}	200, 300, 400	3

3.5 Data Complement Technique (DCT)

In the previous sections, the two approaches Apriori-C, and Eclat-C is proposed to discover the frequent superset. The other proposed method in this thesis is called **Data Complement Technique** (DCT). DCT utilizes the original frequent itemset mining algorithms to mine frequent superset. Figure 3.11 depicts a diagram of DCT. All of Apriori [2], FP-growth [3], Eclat [4], Tree-Projection [5], H-Mine [6], DHP [7], or other frequent itemset mining algorithms can directly be adopted as the black box of our algorithm without any modification. We can adopt the well-developed frequent itemset mining system without developing any new data mining system to discover the frequent superset.

The reason causes DCT to transform the frequent subset to frequent superset is stated in the following lemma.

Lemma 2 If X is a frequent subset of database D with minimum support h, X' must be a frequent superset of database D' with the same minimum support h.

Proof If X is a frequent subset of database D with minimum support h, then there must be k transactions, $T_1, T_2, \ldots, T_k$, $k \geqq h$, that $T_i \supseteq X$, for all $i = 1, 2, \ldots, k$. We can get $I \setminus T_i \subseteq I \setminus X$, that is $T_i' \subseteq X'$, for all $i = 1, 2, \ldots, k$, and $k \geqq h$. This means X' is a superset of each T_i, where $i = 1, 2, \ldots, k$, and $k \geqq h$. In other words, X' is a frequent superset of D'.

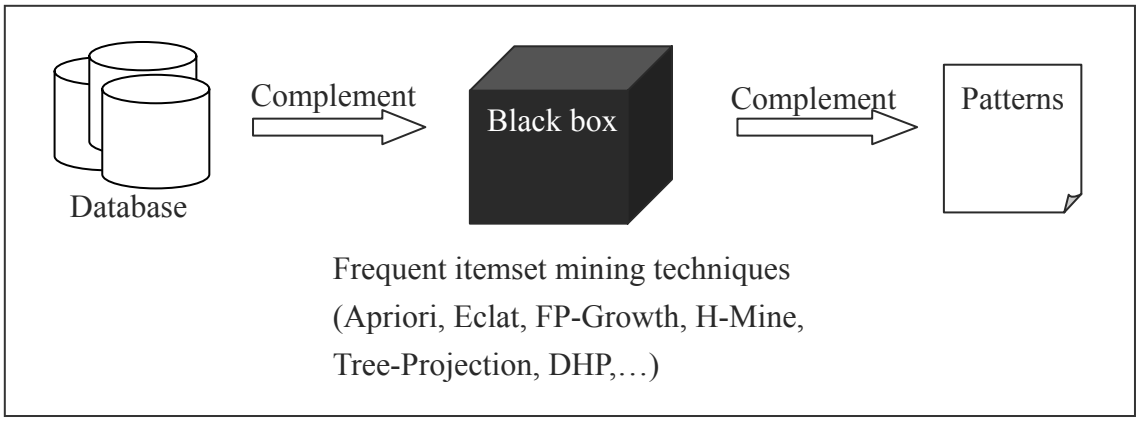

Figure 3.11: A diagram of Data complement technique (DCT).

According to this lemma, we can first transfer the database D into its complement, D', and then explore the frequent subsets of D'. Finally, the frequent supersets are obtained by taking the complement of those explored frequent subsets.

Example 3.9 Figure 3.12 gives an example for illustration of DCT-Apriori. Consider the database D and assume the minimum support is 3 transactions, where CompData is the complement of database D. L_1 and L_2 are frequent itemset of CompData, and also frequent complement-superset of original database (See the previous example of Apriori-C). The frequent supersets are shown in the table Answer.

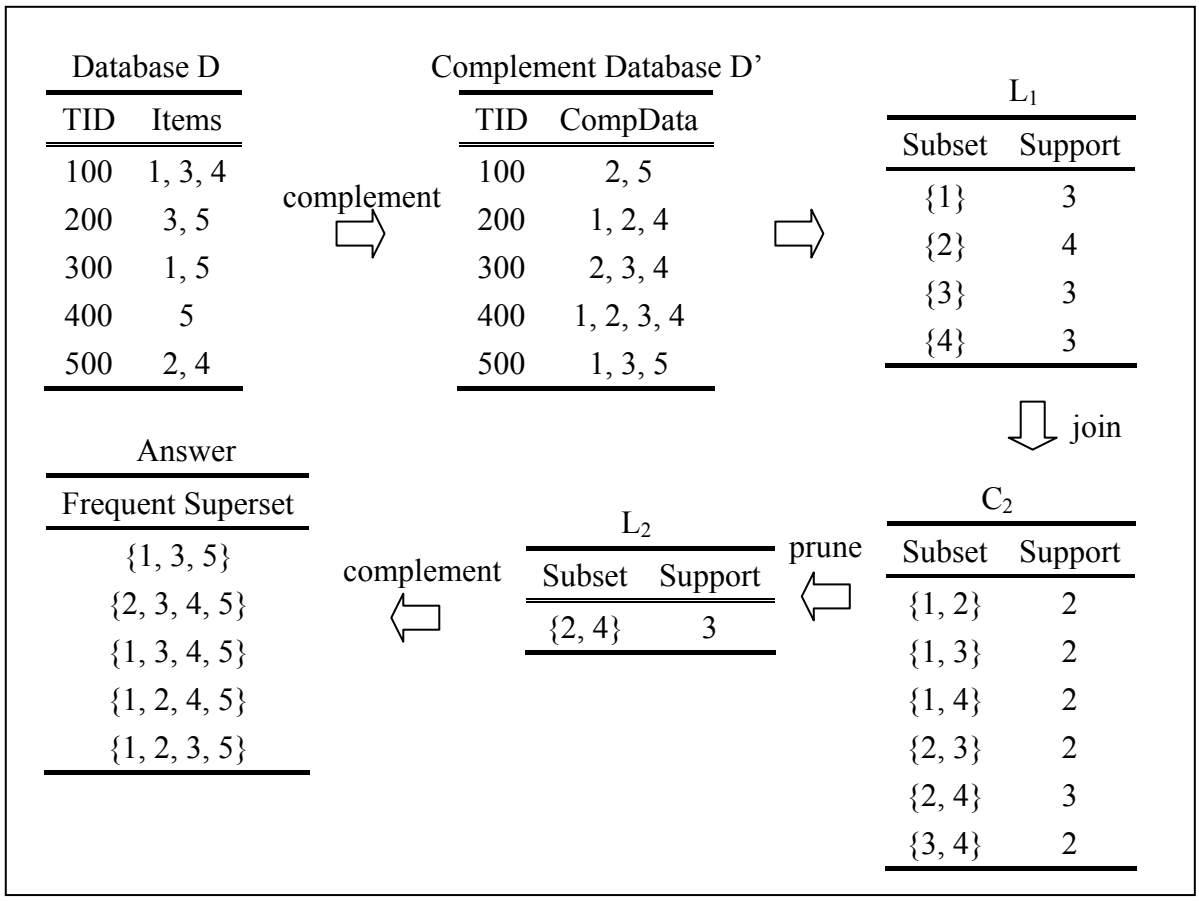

Figure 3.12: An example for DCT with Apriori.

Since DCT algorithm takes the complement of the full database first, it increases the times of database scan by once and furthermore it needs an extra action of writing to save the complement database into auxiliary memory.

In our experiments, we choose Apriori, Eclat and FP-Growth as instances of black boxes to compare the performance with the proposed algorithms, Apriori-C and Eclat-C. FP-Growth is the most efficient algorithm of mining frequent itemset nowadays.

3.6 Summary

In this chapter, we illustrate the Baseline method and three proposed algorithms, Apriori-C, Eclat-C, and Data Complement Technique (DCT). The Baseline method takes the level-wise advantage of Apriori, and reduces the times of database scan. Apriori-C also takes the level-wise advantage of Apriori. Furthermore, it inherits the Apriori property to reduce the number of candidate generation in each level greatly.

Eclat-C is a depth-first search and transaction-ID list intersecting algorithm. The depth-first search strategy makes the maximum complement-frequent superset be discovered earlier, and needn't generate the entire candidates in each level of the Apriori-C method. When the maximum length of frequent superset is short, the Apriori-C has to go more passes to explore the longest pattern. The transaction-ID list intersecting strategy makes it possible to scan database once, and calculates the support values of each candidates by intersecting the transaction-ID lists.

DCT takes the original algorithm of frequent itemset mining as the black box to discover the frequent supersets. We do not have to develop any new mining system, but utilize the well-developed frequent itemset mining system, such as Apriori, FP-growth, Eclat, Tree-Projection, H-Mine, DHP, and so on. The disadvantage of DCT is that it needs an extra I/O for producing the complement database.