Chapter 5

Frequent Itemsets in Dynamic Databases

Most of the databases are dynamic and are updated frequently i.e. new records are added, old records are deleted and existing records are modified very frequently. So, the itemsets which are frequent (or large) may not be frequent when the database is updated and the itemsets which are not frequent may become frequent when the database is updated. So, some algorithms are required to update the set of frequent itemsets when the database is updated. Moreover, new database may contain some new interesting rules which were not present in the old database. One obvious technique is re-running association rule mining algorithms in the updated database to find the frequent itemsets in the updated database. However, this is not optimal solution because of the following reasons.

1. It will require to run the algorithms on adding, deleting or updating a small number of records.

2. It will take too much time because it will scan the same database every time.

3. It will generate most of the itemsets repeatedly.

Some of the popular algorithms to find frequent itemsets in dynamic databases are FUP [CHN+96b], FUP2 [CLK97], DELI [LCK98] and MAAP [ES02]. Some
more algorithms can be found in [TBA+97, FAA+97]. The main requirements of a dynamic association rule mining algorithm are:

1. It should be able to use the already discovered frequent itemsets to discover new frequent itemsets.

2. It should not have to scan the old records/transactions.

3. It should scan the new records/transactions as few number of times as possible.

The most popular and important algorithm, which follows the above criteria to find frequent itemsets in dynamic database is the Borders algorithm [FAL+99, Puj01]. This algorithm has used the concept of border and promoted border set to update the frequent itemsets. However, the algorithm suffers from scalability problem and cannot be used in distributed environment.

As mentioned above, Borders algorithm suffers from scalability problem and cannot be used in distributed environment directly. To address these problems, this chapter presents one modified version of Borders algorithm, which takes less time than that of Borders algorithm. This chapter also presents Distributed Borders algorithm, which is meant for distributed dynamic databases.

5.1 Borders Algorithm

As mentioned above, it is not efficient to rerun frequent itemsets finding algorithms reported so far to find frequent itemsets in dynamic/incremental databases. So, it is desirable to have incremental algorithms, which can generate frequent itemsets in incremental manner by processing only the new transactions/records. Borders is one such algorithm, which satisfies this requirement by using the concept of border set and promoted border set.
Initially the concept of border set was given by Manila and Toivonen [Man97]. An itemset \( X \) is called a border set if \( X \) is not frequent, but all its proper subsets are frequent. Collection of border sets forms the border line between the frequent sets and non-frequent sets. An itemset that was a border set before the database was updated and has become frequent after the database has been updated is called a promoted border set.

Borders algorithm uses the same concept of border and promoted border, and maintains support counts for all the frequent sets as well as for all the border sets. There are three variations of Borders algorithm: addition of transactions/records, addition and deletion of transactions/records and changing of minimum support threshold. The main characteristics of the algorithm are as follows.

1. Entire database is scanned only when new candidate sets are generated.
2. There are only few candidates for which support is counted even if entire database is scanned.

The algorithm is based on the observation that a set is required to be considered as a candidate set only if it has a subset that is a promoted border. The following lemma proves this observation. In addition to Lemma 5.1, the proof of correctness of Borders algorithm also can be found in [FAL+99].

Lemma 5.1

If $X$ is an itemset which is frequent in $T_{whole}$ and not frequent in $T_{old}$, then there exists a subset $Y \subseteq X$ such that $Y$ is promoted border.

Proof: Let $Y$ be a minimal cardinality subset of $X$ which is frequent in $T_{whole}$ but not in $T_{old}$. So, all proper subsets of $Y$ are frequent in $T_{whole}$. However, by minimality of $Y$, none of these subsets is a new frequent set in $T_{whole}$. So, $Y$ is a promoted border. □

Given $L_{old}$ and $B_{old}$, the task of the Borders algorithm is to find $L_{whole}$ and $B_{whole}$. The algorithm (addition) is presented in Algorithm 5.1 on the next page. The algorithm assumes that $L_{old}$ and $B_{old}$ are known with their respective supports. $L_{old}$ and $B_{old}$ can be found by any association rule mining algorithms like Apriori, etc. The algorithm starts by making one pass over the new database $T_{new}$ and counts the supports of the itemsets of $L_{old} \cup B_{old}$ for $T_{new}$. During this pass the algorithm calculates $PB$ and $L_{whole}$. If $PB$ is null then $L_{whole}$ contains all the frequent sets of $T_{whole}$. However, if there is at least one promoted border (i.e. $PB$ is not null), then the algorithm generates new candidate sets (steps 8-10) and scans over the entire database to count the support of them (steps 11-13). Then it finds the large itemsets $L_{i+1}$ based on the support count and updates $L_{whole}$ and $B_{whole}$. This process continues as long as new candidates can be generated. So, the algorithm scans the whole database, if there is some promoted border set. Otherwise, it does not require to scan the whole database.
CHAPTER 5. FREQUENT ITEMSETS IN DYNAMIC DATABASES

Input: $T_{\text{new}}, T_{\text{old}}, \text{minsup}, L_{\text{old}}$ and $B_{\text{old}}$
Output: $L_{\text{whole}}$ and $B_{\text{whole}}$

1. Scan $T_{\text{new}}$ and increment the support count of $X \in (L_{\text{old}} \cup B_{\text{old}})$;
2. $PB := \{X | X \in B_{\text{old}} \text{ and } \text{Sup}(X)_{T_{\text{whole}}} \geq \text{minsup}\}$;
3. $L_{\text{whole}} := PB \cup \{X | X \in L_{\text{old}} \text{ and } \text{Sup}(X)_{T_{\text{whole}}} \geq \text{minsup}\}$;
4. $B_{\text{whole}} = \{X \forall x \in X, X - \{x\} \in L_{\text{whole}}\}$;
5. $m = \max\{i | PB(i) \neq \phi\}$

Steps for Candidate-generation:

6. $L_0 = \phi$, $i = 1$;
7. While ($L_i \neq \phi$ or $i \leq m$) do
8. \[
C_{i+1} = \{X = S_1 \cup S_2 | (i) \ |X| = i + 1, \\
(ii) \exists x \in S_1, S_1 - \{x\} \in PB(i) \cup L_i, \\
(iii) \forall x \in S_2, S_2 - \{x\} \in L_{\text{whole}}(i) \cup L_i\}
\]
9. Scan $T_{\text{whole}}$ and obtain $\text{Sup}(X)_{T_{\text{whole}}}$ for all $X \in C_{i+1}$;
10. $L_{i+1} = \{X | X \in C_{i+1} \text{ and } \text{Sup}(X)_{T_{\text{whole}}} \geq \text{minsup}\}$;
11. $L_{\text{whole}} = L_{\text{whole}} \cup L_{i+1}$;
12. $B_{\text{whole}} = B_{\text{whole}} \cup (C_{i+1} - L_{i+1})$;
13. $i = i + 1$;
14. Enddo

Algorithm 5.1: Borders (Addition)
5.1.1 An Example

Let us take $T_{old}$ and $T_{new}$ as given in Table 5.1 and Table 5.2 respectively and the minimum support be 40%.

$$
\begin{array}{cccc}
A1 & A2 & A3 & A4 & A5 \\
1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 \\
\end{array}
$$

Table 5.1: $T_{old}$ - I

$$
\begin{array}{cccc}
A1 & A2 & A3 & A4 & A5 \\
0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 \\
\end{array}
$$

Table 5.2: $T_{new}$ - I

Then $L_{old}$ is $\{(A1,A2,A3,A5),(A1A3),(A1A5),(A3A5),(A1A3A5)\}$

$B_{old}$ is $\{(A4),(A1A2),(A2A3),(A2A5)\}$

Next, $T_{new}$ is added with $T_{old}$. Now, $T_{new}$ is scanned, and $PB$, $L_{whole}$ and $B_{whole}$ are calculated as follows.

$PB = \{(A4)\}$

$L_{whole} = \{(A4),(A1),(A5),(A1A5)\}$

$B_{whole} = \{(A1A4),(A4A5),(A1A5)\}$

Here, only one promoted border $\{(A4)\}$ has been found. So, new candidate itemsets are to be generated. New Candidates are generated in level-wise fashion.

Candidate 2-itemsets are $C_2 = \{(A1A4), (A4A5)\}$

Now, $T_{whole}$ is to be scanned to update $L_{whole}$ and $B_{whole}$. 
5.1.2 Deletion

In some situations, records/transactions may have to be deleted from the database. So, it will affect the already discovered frequent itemsets. Deletion will have following effects.

1. Some frequent sets may not be frequent any more.

2. Some new frequent sets may emerge because absolute frequency count will decrease.

Borders algorithm has been modified to handle the deletion of records. The algorithm is given in Algorithm 5.2 on the following page.

5.1.3 Changing of Threshold

In some situations, users may have to change the threshold of minimum support. If the new minimum support is greater than the previous minimum support, then new frequent sets and border sets can be found easily by excluding the frequent sets with minimum support less than the new minimum support. However, if the new minimum support is less than the previous minimum support, then new frequent sets may emerge. Borders handles this as a case of deletion.

5.1.4 Discussion

The Borders algorithm is robust enough to find the frequent itemsets in a dynamic database. However, from the experimental study it has been observed that

- with the increase in the volume of $T_{old}$ and $T_{new}$, the cost of scanning of $T_{whole}$ in the every iteration becomes too expensive.

- it suffers from scalability problem.
Input: \( T_{new}, T_{old}, T_{del}, \text{minsup}, L_{old} \) and \( B_{old} \)
Output: \( L_{whole} \) and \( B_{whole} \)

1. Scan \( T_{new} \) and update the support count of \( X \in L_{old} \cup B_{old} \);

2. Scan \( T_{del} \) and update the support count of \( X \in L_{old} \cup B_{old} \);

3. \( PB := \{X|X \in B_{old} \text{ and } \text{Sup}(X)_{T_{whole}} \geq \text{minsup}\} \);

4. \( L_{whole} := PB \cup \{X|X \in L_{old} \text{ and } \text{Sup}(X)_{T_{whole}} \geq \text{minsup}\} \);

5. \( B_{whole} = \{X|\forall x \in X, X - \{x\} \in L_{whole}\} \);

6. \( m = \max\{i|PB(i) \neq \phi\} \);

**Steps for Candidate-generation:**

7. \( L_{0} = \phi, i=1 \);

8. While \((L_{i} \neq \phi \text{ or } i \leq m)\) do

9. \( C_{i+1} = \{X = S_{1} \cup S_{2} | (i) \ |X| = i + 1, \)

10. \( (ii) \exists x \in S_{1}, S_{1} - \{x\} \in PB(i) \cup L_{i}, \)

11. \( (iii) \forall x \in S_{2}, S_{2} - \{x\} \in L_{whole}(i) \cup L_{i} \);\)

12. Scan \( T_{whole} \) and obtain \( \text{Sup}(X)_{T_{whole}} \) for all \( X \in C_{i+1} \);

13. \( L_{i+1} = \{X|X \in C_{i+1} \text{ and } \text{Sup}(X)_{T_{whole}} \geq \text{minsup}\} \);

14. \( L_{whole} = L_{whole} \cup L_{i+1} \);

15. \( B_{whole} = B_{whole} \cup (C_{i+1} - L_{i+1}) \);

16. \( i = i+1 \);

17. Enddo

**Algorithm 5.2: Borders (Addition and Deletion)**

Some possible solutions to these problems are

- Some other data structure like FP-tree can be used to find the frequent
itemsets faster.

- Probability can be used to predict whether itemsets will be frequent on adding new transactions. The itemsets whose probabilities are very less, can be removed from border sets and itemsets with higher probability can be retained.

- New candidate itemsets $C_{i+1}$ can be generated from $C_i$ instead of from $L_i$ because $|C_i|$ is generally very small. After the candidates are generated, $T_{whole}$ can be scanned to find the large and border sets. So, maximum of one scan of entire database will be required. However, if $C_i$ is large, then this method will not give optimum result.

- New candidate itemsets $C_{i+1}$ can be generated from $C_i$ instead of from $L_i$ as long as $|\cup C_i|$ is less than some threshold value. When $|\cup C_i|$ becomes greater than the threshold value, $T_{whole}$ can be scanned to find frequent and border sets. Then, the next iteration begins.

- More than one border sets can be used.

- Distributed approach also can be used to improve the scalability performance of the algorithm. In case of distributed approach, data will be processed in different sites/nodes resulting in improvement in execution time.

This chapter proposes two enhanced versions of the present Borders (addition) algorithm: Modified Borders and Distributed Borders. Modified Borders has used two border sets to reduce scanning of entire database. On the other hand, Distributed Borders is the extension of Borders in distributed environment.

5.2 Modified Borders Algorithm

As it is mentioned above, Borders has to scan entire database when there are some promoted borders. However, scanning entire database is very expensive, particularly, when the database is very large. Scanning entire database can be avoided if new candidates are not generated frequently. New candidates are
generated, if there is even one promoted borders because borders are not included to generate candidate in the old database. So, if borders are included to generate candidates in the old database, then there will be no new candidates. However, it will be infeasible to include all the borders to generate candidates in the old database. This concept led us to include some of the borders, which are likely to become promoted border, to generate candidate sets in the old database so that if those borders become promoted, no new candidates will be generated. New candidates will be generated only when some borders, which were not included to generate candidates in the old database, become promoted.

Based on the above discussion, Borders has been modified by including two border sets. The first border set is $B'_{old}$ and the second border set is $B''_{old}$. $B'_{old}$ is calculated as $\{X | \forall x \in X, X - \{x\} \in L_{old} \cup B'_{old}, \text{Sup}(X)_{T_{old}} \geq \beta' \text{ and } \text{Sup}(X)_{T_{old}} < \text{minsup}\}$. $B''_{old}$ is calculated as $\{X | \forall x \in X, X - \{x\} \in L_{old} \cup B'_{old}, \text{Sup}(X)_{T_{old}} < \beta'\}$. $B'_{old}$ and $L_{old}$ take part in candidate generation, whereas the elements of $B''_{old}$ are not used in candidate generation. Another requirement in the algorithm is that all the subsets of an itemset $X \in B'_{old} \cup B''_{old}$ must be $\in L_{old} \cup B'_{old}$. Obviously, $B'_{old}$ contains the itemsets with higher probability of becoming promoted when new transactions are added. New candidate sets will be generated only when any of the elements of the $B''_{old}$ becomes promoted. If new candidate itemsets are generated, one scan over the whole database is required to find supports of the new candidate itemsets.

5.2.1 The Algorithm

The algorithm works as follows. $L_{old}$, $B'_{old}$ and $B''_{old}$ are assumed to be known with their respective support counts. The algorithm starts by making one pass over the new database $T_{new}$ and updates supports of the elements of $L_{old} \cup B'_{old} \cup B''_{old}$. During the pass, the algorithm generates four categories of itemsets - $PB'$, $PB''$, $B'''$ and $B''''$. If $PB''$ is null, then no new candidate set is required to be generated. If $PB''$ contains at least one itemset, then new candidate sets are required to be generated. If new candidate sets are generated, the algorithm makes one pass over the entire database to count the support of the new candidate sets. At the end, the algorithm generates $L_{whole}$, $B'_{whole}$ and $B''_{whole}$, which are
CHAPTER 5. FREQUENT ITEMSETS IN DYNAMIC DATABASES

Input: $T_{\text{new}}$, $T_{\text{old}}$, $\text{minsup}$, $\beta'$, $L_{\text{old}}$ and $B_{\text{old}}'$. $B_{\text{old}}''$. Output: $L_{\text{whole}}$ and $B_{\text{whole}}'$, $B_{\text{whole}}''$.

1. Scan $T_{\text{new}}$ and increment the support count of $X \in L_{\text{old}} \cup B_{\text{old}}' \cup B_{\text{old}}''$;

2. $PB' = \{ X | X \in B_{\text{old}}' \text{ and } \text{Sup}(X)_{T_{\text{whole}}} \geq \text{minsup} \}; \quad PB'' = \{ X | X \in B_{\text{old}}'' \text{ and } \text{Sup}(X)_{T_{\text{whole}}} \geq \text{minsup} \}$;

3. $L_{\text{whole}} = PB' \cup PB'' \cup \{ X | X \in L_{\text{old}} \text{ and } \text{Sup}(X)_{T_{\text{whole}}} \geq \text{minsup} \}$;

4. $B'' = \{ X | X \in B_{\text{old}}'' \forall z \in X, X - \{ z \} \in L_{\text{whole}}, \text{Sup}(X)_{T_{\text{whole}}} \geq \beta' \text{ and } \text{Sup}(X)_{T_{\text{whole}}} < \text{minsup} \}$;

5. $B''' = \{ X | X \in B_{\text{old}}'' \cup L_{\text{old}}, \forall z \in X, X - \{ z \} \in L_{\text{whole}}, \text{Sup}(X)_{T_{\text{whole}}} \geq \beta', \text{Sup}(X)_{T_{\text{whole}}} < \text{minsup} \}$;

6. $B_{\text{whole}}' = B'' \cup B'''$; $B_{\text{whole}}'' = \{ X | \forall z \in X, X - \{ z \} \in L_{\text{whole}}, \text{Sup}(X)_{T_{\text{whole}}} < \beta' \}$;

7. If $PB'' \neq \phi$ then

8. $m = \max \{ i | PB''(i) \neq \phi \}$;

Steps for Candidate-generation:

9. $L_0 = \phi$, $B_0 = \phi$, $k=2$.

10. While ($L_{k-1} \neq \phi$ or $B_{k-1} \neq \phi$ or $k \leq m + 1$) do

11. $C_k = \phi$;

12. $L = PB''(k-1) \cup L_{k-1} \cup B'''(k-1) \cup B_{k-1}$: $M = L_{k-1} \cup L_{\text{whole}}(k-1) \cup B_{\text{whole}}'(k-1)$;

13. For all itemsets in $l_1 \in L$ do begin

14. For all itemsets in $l_2 \in M$ do begin

15. If $l_1[i] = l_2[i]$ ($1 \leq i \leq k - 2$) and $l_1[k-1] < l_2[k-1]$ then

16. $C = \{ l_1[1], l_1[2], \ldots, l_1[k-2], l_1[k-1], l_2[k-1] \}$;

17. $C_k = C_k \cup C$;

18. End for

19. End for

20. Prune $C_k$: All the subsets of $C_k$ of size $k-1$ must be present in $M$;

21. Scan $T_{\text{whole}}$ and obtain support $\text{Sup}(X)_{T_{\text{whole}}}$ for all $X \in C_k$;

22. $L_k = \{ X | X \in C_k \text{ and } \text{Sup}(X)_{T_{\text{whole}}} \geq \text{minsup} \}$;

23. $L_{\text{whole}} = L_{\text{whole}} \cup L_k$;

24. $B_k = \{ X | X \in (C_k - L_k), \forall z \in X, X - \{ z \} \in L_{\text{whole}}, \text{Sup}(X)_{T_{\text{whole}}} \geq \beta' \text{ and } \text{Sup}(X)_{T_{\text{whole}}} < \text{minsup} \}$;

25. $B_{\text{whole}}' = B_{\text{whole}}' \cup B_k$;

26. $B_{\text{whole}}'' = B_{\text{whole}}'' \cup \{ X | X \in (C_k - L_k), \forall z \in X, X - \{ z \} \in L_{\text{whole}}, \text{Sup}(X)_{T_{\text{whole}}} < \beta' \}$;

27. $k = k + 1$;

28. Enddo

29. Endif

Algorithm 5.3: Modified.Borders
counterparts of the $L_{old}$, $B'_{old}$ and $B''_{old}$ respectively, for the whole database $T_{whole}$. The algorithm is presented in the Algorithm 5.3 on the previous page.

5.2.2 An Example

Let us consider $T_{old}$ as given in Table 5.3 and assume $\minsup = 40\% \& \beta' = 30\%;$

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
<th>$A_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.3: $T_{old}$ - II

Now, $L_{old}$, $B'_{old}$, $B''_{old}$ are obtained as given in Table 5.4.

![Table 5.4: Results - I](image)

Now, suppose $T_{new}$ (Table 5.5 on the following page) added to the $T_{old}$ (Table 5.3).


When $T_{new}$ is scanned, $P_{B'}$, $P_{B''}$, $B'''$, $L_{whole}$, $B'_{whole}$ and $B''_{whole}$ are obtained as given in Table 5.6.

<table>
<thead>
<tr>
<th>$PB'$</th>
<th>$P_{B''}$</th>
<th>$B'''$</th>
<th>$B''''$</th>
<th>$L_{whole}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>{A3}</td>
<td>$\phi$</td>
<td>{$(A_1, A_2, A_4), (A_3), (A_5), (A_1 A_2)$}</td>
</tr>
</tbody>
</table>

Table 5.6: Results - II

Since $P_{B''}$ is not null, new candidate itemsets will be generated. The new candidate itemsets after pruning will be $\{(A_1 A_5, A_2 A_5, A_3 A_5, A_4 A_5), (A_1 A_2 A_3)\}$. Then, entire database is scanned to find the support count of the new candidates.

5.2.3 Experimental Results

$\text{Modified}\_\text{Borders}$ was compared with $\text{Borders}$ algorithm using two synthetic databases and one real database. Both the algorithms were implemented on a Intel PIV based WS (with 256 MB RAM). $L_{old}$, $B'_{old}$, $B''_{old}$ had been computed separately.

Test Data: Two synthetic databases, which were generated using the technique given in [AMS+96], and the Connect-4 dataset, which was downloaded from UCI machine learning repository (www.ics.uci.edu), were used for the experiments. All the synthetic datasets contain 100K records and dimensionality of
each record is 500. Other parameters for synthetic databases are shown in the Table 5.7. For all the experiments, initial sizes of the synthetic databases and Connect-4 were taken as 80K records and 47557 records respectively.

| Data Set     | |T|   | |ML|   |
|--------------|-----|-----|-----|
| T20I4100K    | 20  | 4   | 100K |
| T20I6100K    | 20  | 6   | 100K |

Table 5.7: Parameters for Synthetic Databases - VI

Each of the experiments were executed several times. Tables 5.8 through 5.10 show average number of full database scan required in Borders and Modified_Borders as the value of $\beta'$ increased from 3.5% to 4.5% and size of incremental database is increased from 5K records to 20K records. The value of \textit{mnsup} was taken as 5% for all the experiments. Average execution times over all increments are given in Figures 5.2 on the following page and 5.3 on page 114.

<table>
<thead>
<tr>
<th>Increment</th>
<th>Borders</th>
<th>Modified_Borders ($\beta'=3.5%$)</th>
<th>Modified_Borders ($\beta'=4%$)</th>
<th>Modified_Borders ($\beta'=4.5%$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5K</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10K</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>15K</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>20K</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 5.8: Comparison on Whole Database Scan for Database T20I4100K
CHAPTER 5. FREQUENT ITEMSETS IN DYNAMIC DATABASES

<table>
<thead>
<tr>
<th>Increment</th>
<th>Borders $\beta' = 3.5%$</th>
<th>Modified_Borders $\beta' = 3.5%$</th>
<th>Borders $\beta' = 4%$</th>
<th>Modified_Borders $\beta' = 4%$</th>
<th>Borders $\beta' = 4.5%$</th>
<th>Modified_Borders $\beta' = 4.5%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5K</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10K</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>15K</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20K</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5.9: Comparison on Whole Database Scan for Database T2016100K

<table>
<thead>
<tr>
<th>Increment</th>
<th>Borders $\beta' = 3.5%$</th>
<th>Modified_Borders $\beta' = 3.5%$</th>
<th>Borders $\beta' = 4%$</th>
<th>Modified_Borders $\beta' = 4%$</th>
<th>Borders $\beta' = 4.5%$</th>
<th>Modified_Borders $\beta' = 4.5%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5K</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10K</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15K</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20K</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.10: Comparison on Whole Database Scan for Database Connect4

![Figure 5.2: Average Execution Time for Borders & Modified_Borders (For Various Values of $\beta'$)](image)

Figure 5.2: Average Execution Time for Borders & Modified_Borders (For Various Values of $\beta'$) - I
Figure 5.3: Average Execution Time for Borders & Modified_Borders (For Various Values of $\beta'$) - II
Observations

Followings are some observations made from the experimental results.

- *Tables 5.8 on page 112, 5.9 on page 113 & 5.10 on page 113 clearly show that Borders requires whole scan of the database several number of times, whereas Modified_Borders requires whole scan of database a few number of times. Thus, Modified_Borders saves the execution time.*

- The value of $\beta'$ has a great effect on the performance of Modified_Borders algorithm. As the value of $\beta'$ increases, number of whole scan also increases. In the experiments, it was found that when $\beta'=4.5\%$, number of full scan of the database is almost same in both the algorithms.

- As far as execution time is concerned, Modified_Borders takes much less time than that of Borders when $\beta'$ is small ( *Figures 5.2 on page 113 & 5.3 on the preceding page*). As the value of $\beta'$ increases, Modified_Borders tend to take little more time. This is because number of full scan tends to increase with the increase in value of $\beta'$.

Selection of $\beta'$

Value of $\beta'$ plays an important role in the algorithm. When $\beta'$ tends to minsup, the algorithm tends to become Borders algorithm. With the decrease in value of $\beta'$, performance of Modified_Borders becomes better than Borders algorithm in terms of execution time. However, with the decrease in $\beta'$ value the memory requirement increases due to the additional candidate sets. So, value of $\beta'$ cannot be decreased too much. This cost of additional memory requirement is quite negligible in comparison to the requirement of full database scanning, particularly when database is very large. So, there should be some trade off in choosing the value of $\beta'$. If there is not enough memory and the database is dense, the value of $\beta'$ can be set to a higher value. For sparse databases, $\beta'$ can be set to a lower value.

Here is a very simple technique to choose value of $\beta'$. The algorithm is run with small random sample of the database, incremental database, and an initial lower
value of \( \beta' \). If the result is not satisfactory, \( \beta' \) is increased by a little amount and the algorithm is rerun. The process is continued several times until number of candidates is manageable and gives desired results. When optimum results are obtained, the corresponding value of \( \beta' \) can be used for the entire database. It has been found that the value of \( \beta' \) in range of 80% - 90% of \( \text{minsup} \) gives satisfactory results. As for example, if \( \text{minsup} \) is 5% then \( \beta' \) will be in the range of 4% - 4.5%.

5.3 Distributed_Borders

Basically, there are three approaches for distributed algorithms - fully distributed, central machine based and self-organized. In case of fully distributed algorithm, processing takes place in every node and every node can act as a merger machine. Nodes also can exchange data among themselves. In case of central machine based algorithm, most of the processing takes place in a central machine (server). The disadvantage is that data replication is required in this environment. Self-organized distributed algorithms are more intelligent. The algorithm assigns tasks to different nodes according to requirement. These systems are more robust and fault-tolerant.

This section presents a fully distributed version of the Borders algorithm. The Borders algorithm discussed above is sequential in nature and is meant for centralized database. However, nowadays, most of the databases are distributed in nature. So, a distributed version of Borders algorithm called Distributed_Borders has been proposed. It can also be used in a centralized database by partitioning the database and placing the partitions in different nodes of a distributed systems. This process reduces the number of candidate sets to a great extent resulting in high flexibility, scalability and low cost performance ratio [CHN+96a]. However, distributed algorithms posses some problems such as locally large or border sets may not be globally large or border sets. Again, message passing being a costly affair, processing should be confined in the local sites as much as possible.

Let us consider a transactional database, where each record is a transaction in a supermarket made by the customers. Each transaction is of the form
<TID,1.1,0..0,1>. Here, \( TID \) is the transaction id, which is unique for each transaction. 1 and 0 represents the corresponding item has been bought and not bought respectively in the transaction. It is also assumed that the database is horizontally partitioned and allocated in \( n_s \) sites \( S_i (i=1,2,3...n_s) \) in a fully distributed system; incremental database also is partitioned and added to database of each site. Now, the task is to maintain the global frequent itemsets and global border sets in this distributed environment when the database is updated.

5.3.1 Distributed Algorithm For Maintaining Frequent Itemsets in Dynamic Database

Here, \textit{Borders} algorithm has been examined in a distributed environment. Let \( T_{old} \) be the old transaction database distributed in \( n_s \) sites. \( T_{old}^i \) is the old transaction database at the site \( i (i=1,2,3...n_s) \). \( T_{new} \) and \( T_{new}^i \) are the new transactions to be added to the whole database and to the transactions at the site \( i \) respectively. For a given minimum support threshold \( \text{minsup} \), an itemset \( X \) is globally large in the old database (updated database) if \( \text{Sup}(X)_{T_{old}} \geq \text{minsup} \) (\( \text{Sup}(X)_{T_{whole}} \geq \text{minsup} \)). Similarly, an itemset \( X \) is locally large in the old database (updated database) at some site \( i \), if \( \text{Sup}(X)_{T_{old}^i} \geq \text{minsup} \) (\( \text{Sup}(X)_{T_{whole}^i} \geq \text{minsup} \)).

Like the \textit{Borders} algorithm, this algorithm also uses the concept of border set and promoted border set. The only difference is that all the concepts have been used in the context of the distributed environment. An itemset \( X \) is a \textit{global border}, if \( X \) is not globally large, but all its subsets are globally large. An itemset \( X \) becomes \textit{globally promoted border} on adding the new transactions, if \( X \) is a globally border in the old database and globally large in the updated database. \( L_{old} \) is the global large itemsets in the old database and \( B_{old} \) is the global border sets in the old database. Given \( L_{old} \) and \( B_{old} \), the problem is to find the updated large itemsets \( L_{whole} \) and border sets \( B_{whole} \) for the updated database \( T_{whole} \). The main purpose of the \textit{Distributed Borders} algorithm is to reduce the number of candidate sets and in turn reduce the number of messages to be passed across the network and execution time. To reduce the number of messages, polling technique as discussed in \textit{DMA} [CNF+96] was used.

Some interesting observations, which are listed below, can be made relating to large, border and promoted border sets in distributed environment. Some of
Figure 5.4: Distributed Borders Architecture
these observations were discussed in [CNF+96].

Observation 5.1 Every global large itemset $X$ must be large in at least one site $S_i$.

Observation 5.2 If an itemset $X$ is locally large at some site $S_i$, then all its subsets are also locally large in the site $S_i$.

Observation 5.3 If an itemset $X$ is globally large at some site $S_i$, then all its subsets are also globally large at the site $S_i$.

Observation 5.4 If an itemset $X$ is globally large or promoted border, then $X$ must be large in at least one site $i$.

Proof: This is obvious, because if an itemset $X$ is small in all the sites, it cannot be large in the whole database.

Observation 5.5 If an itemset $X$ is a global promoted border set, then $X$ must be large in $T_{new}$ for some site $S_i$.

Proof: Let an itemset $X$ is a promoted border set in the updated database $T_{whole}$. Then $Sup(X)_{T_{old}} < \text{minsup}$ and $Sup(X)_{T_{whole}} \geq \text{minsup}$. Since $T_{whole} = T_{old} \cup T_{new}$. $Sup(X)_{T_{new}} \geq \text{minsup}$. So, $Sup(X)_{T_{new}} \geq \text{minsup}$ for at least one site $S_i$.

Observation 5.6 If $X$ is global border set, then there exist $Y \subset X$ so that $Y$ is local border in some site $S_i$.

Proof: If $X$ is a global border set, then $X$ must be small/infrequent in at least one site $S_i$. Therefore there exist at least one subset of $X$, which is a local border set in the site.

Observation 5.7 If a new candidate set $c$ in $T_{whole}$ has to be large or border in $T_{whole}$, then either $c$ or one of its immediate subsets must be locally large in one $T_{new}$. 

CHAPTER 5. FREQUENT ITEMSETS IN DYNAMIC DATABASES

Proof: If \( c \) is a new candidate set there can be two possible cases:

1. \( c \) may be large in the updated database \( T_{\text{whole}} \): In this case \( c \) must be large in \( T_{\text{new}} \), i.e. \( c \) must be large in \( T_{\text{new}} \) for some \( i \).

2. \( c \) may be a border set in the updated database \( T_{\text{whole}} \): In this case \( c \) will be small and all of its subsets will be large in the updated database \( T_{\text{whole}} \). So, there exists at least one \( c' \subset c \), which was small in the old database \( T_{\text{old}} \). Otherwise, \( c \) would have been generated in the old database \( T_{\text{old}} \). This \( c' \) will be large in the updated database \( T_{\text{whole}} \). So, \( c' \) must be large in the \( T_{\text{new}} \), i.e. \( c' \) must be large in \( T_{\text{new}} \) for some \( i \).

Observation 5.8 If a candidate set \( X \) in the updated database is either large or border set, all of its immediate subsets must be either \( \in F \cup PB \) or large in at least one site \( i \).

Proof. There can be two possible cases:

1. \( X \) is large in the updated database: If \( X \) is large then all the subsets of \( X \) must also be large in the updated database. Let \( Y \subset X \). Then \( Y \) is either a candidate set or \( Y \in F \cup PB \). If \( Y \) is a candidate and \( Y \) is large, then \( Y \) must be large in at least one site because if \( Y \) cannot be large if it is small in all the sites.

2. \( X \) is border in the updated database: In this case, all the subsets of \( X \) must be large in the updated database. So, by first option, all the subsets are either \( \in F \cup PB \) or large in at least one site.

5.3.2 Local Pruning

Using the above observations many unnecessary candidates can be pruned locally. If an itemset \( X \) is locally small in all the sites, then \( X \) can not be large globally. That is why, itemsets are first checked if they are locally large or not. Their global supports are found only when they are locally large in at least one site \( S_i \). Similarly some promoted border sets also can be pruned away locally using
CHAPTER 5. FREQUENT ITEMSETS IN DYNAMIC DATABASES

Input: $L_{\text{old}}, B_{\text{old}}, T_{\text{new}}, T_{\text{old}}$ and $\text{minsup}$. 
Output: Updated $L_{\text{whole}}$ and $B_{\text{whole}}$

Repeat the following steps at each site $i$ distributively

1. Scan $T_{\text{new}}$ and count the support of all the itemsets $X \in \{L_{\text{old}} \cup B_{\text{old}}\}$ and find
   (a) $PB' = \{X | X \in B_{\text{old}} \text{ and } \text{Sup}(X)_{T_{\text{new}}} \geq \text{minsup}\}$;
   (b) $L_{\text{whole}}' = PB' \cup \{X | X \in L_{\text{old}} \text{ and } \text{Sup}(X)_{T_{\text{whole}}} \geq \text{minsup}\}$; (by observation 5.4 on page 119)

2. Broadcast $X \in L_{\text{whole}}'$ to other sites along with their supports;

3. Prune $L_{\text{whole}}'$ and $PB'$:
   $L_{\text{whole}}' = \{X | X \in L_{\text{whole}}' \text{ and } \text{Sup}(X)_{T_{\text{whole}}} \geq \text{minsup}\}$;
   $PB' = \{X | X \in PB' \text{ and } \text{Sup}(X)_{T_{\text{whole}}} \geq \text{minsup}\}$;

4. Compute $PB = \bigcup PB'$ and $L_{\text{whole}} = \bigcup L_{\text{whole}}'$;

5. $B_{\text{whole}} = \{X | \forall z \in X, X - \{x\} \in L_{\text{whole}}\}$;
   Generate candidates:

6. $m = \max\{i | PB(i) \neq \phi\}$;

7. $i = 1$;

8. While ($L_i \neq \phi$ or $i \leq m$) do

9.    $C_{i+1} = \{X | S_1 \cup S_2 | (i) |X| = i + 1$,

10.   (ii) $\exists x \in X, X - \{x\} \in PB(i) \cup L_i$,

11.   (iii) $\forall x \in X, X - \{x\} \in L_{\text{whole}}(i) \cup L_i$;

12.   Scan $T_{\text{new}}$ and compute $\text{Sup}(X)_{T_{\text{new}}}$ for all $X \in C_{i+1}$;

13.   Remove any candidate set $X \in C_{i+1}$, which is or at least one of its immediate subsets is not large in $T_{\text{new}}$; (by observation 5.7 on page 119)

14.   Scan $T_{\text{old}}$ and find the support $\text{Sup}(X)_{T_{\text{old}}}$ for all $X \in C_{i+1}$; ($T_{\text{new}}$ has already been scanned).

15.   $C_{i+1} = \{X | C_{i+1} | \forall x \in X, Y = X - \{x\}, Y \in L_{\text{whole}} \text{ or } \text{Sup}(Y)_{T_{\text{whole}}} \geq \text{minsup}\}$ (by observation 5.8 on the previous page)

16.   Collect all $\text{Sup}(X)_{T_{\text{whole}}}$ and find $\text{Sup}(X)_{T_{\text{whole}}}$ for all $X \in C_{i+1}$;

17.   $L_{i+1} = \{X | X \in C_{i+1} \text{ and } \text{Sup}(X)_{T_{\text{whole}}} \geq \text{minsup}\}$;

18.   $L_{\text{whole}} = L_{\text{whole}} \cup L_{i+1}$;

19.   $B_{\text{whole}} = B_{\text{whole}} \cup (C_{i+1} - L_{i+1})$;

20.   $i = i + 1$;

21.   Enddo

22. Return $L_{\text{whole}}$ and $B_{\text{whole}}$;

Algorithm 5.4: Distributed Borders
Observation 5.4 on page 119. So, if a border set $X$ is not large in any site, then it is not tested for global promoted border. Observation 5.7 on page 119 is very significant in the pruning away of the candidate sets locally. After the candidate sets are generated, support of the candidate sets are counted in the incremental part $T_{\text{new}}$. If any candidate set or at least one of its immediate subsets is not large in at least one $T_{\text{new}}$, it can be pruned away because it can be neither large set nor border set by Observation 5.7 on page 119. Observation 5.8 on page 120 is also helpful in pruning away unnecessary candidate sets.

5.3.3 Explanation of the Algorithm

It is assumed that $L_{\text{old}}$ and $B_{\text{old}}$ are available with the local support to all the sites. The algorithm starts with scanning the incremental portion $T_{\text{new}}$ and finds local support for all $X \in L_{\text{old}} \cup B_{\text{old}}$. This is because frequent and border sets, which are locally large in at least one site, can only be large globally. Then comes to the second step, which finds the global support for $L'_{\text{whole}}$. This can be done by simply broadcasting the local support of $X \in L'_{\text{whole}}$. If all the items are broadcast to all the sites, then for each item $X$, $O(n_s^2)$ messages will be required, where $n_s$ is the number of sites. Here, polling techniques as described in [CHN+96a] can be used to reduce the number of messages to $O(n_s)$ for each itemset. Third step prunes away the $X \in L'_{\text{whole}}$, which are not globally large. The fourth step just broadcasts the $L'_{\text{whole}}$ to other sites and receives the same from other sites to compute $L_{\text{whole}}$ and $PB$. It can be noted that, all the sites will be having the same set of $L_{\text{whole}}$ and $PB$. Steps 6-11 are responsible for generating the candidate sets. The candidate sets are generated using methods like Apriori. Some kind of pruning techniques are required to prune away some unnecessary candidate sets. Observation 5.4 on page 119 helps prune away some candidate sets. Steps 13-15 are basically pruning steps. It scans the $T_{\text{new}}$ and finds the $\text{Sup}(X)T_{\text{new}}$ for all the candidate sets. According to Observation 5.4 on page 119, a candidate $X$ can be neither large nor border if neither $X$ is large nor at least one of its immediate subsets is large in any $T_{\text{new}}$. So, the candidates, which do not conform to Observation 5.4 on page 119, can easily be removed from the candidate sets. At last, step 16 finds the global support for all the candidate sets. Polling technique as given in [CHN+96a] can be used here also.
5.3.4 Experimental Results

The algorithm was simulated on a share-nothing environment. A 10/100 Mb LAN was used to connect six PIV machines running Windows NT. Each machine had 20GB disk space and 256MB memory. The datasets used in the experiments were T2014200K and T2016200K, which were generated using the technique given in [AMS+94]. Each dataset contained 200K tuples (transactions). Each dataset was partitioned and corresponding partitions were loaded in the machines before the experiments started.

| Data Set    | |LM| |D|
|-------------|----|----|
| T20142000K  | 20 | 4  | 200K |
| T2016200K   | 20 | 6  | 200K |

Table 5.11: Parameters for Synthetic Databases - VII

Three experiments were carried out. In the first experiment, three machines (sites) were used. The purpose of the experiment was to find the execution time for different minimum supports. Each machine initially contained 63K transactions and 3K transactions were added to each machine as incremental database. The results are given Figure 5.5 on the next page.

The second experiment was the scale up experiment. The testbed of the second experiment was same as that of first experiment. Here also, three machines (sites) were used. The purpose of the second experiment was to evaluate the scalability performance of the algorithm. Three machines initially contained 30%, 30% and 25% transactions respectively. Size of incremental database was 5% for each of the machines and minimum support was 1%. The results are given in Figure 5.6 on page 125.

The third experiment was the speedup experiment. For n sites, speedup factor is defined as \( S(n) = T(1)/T(n) \) and efficiency is defined as \( S(n)/n \), where \( T(n) \) is the execution time with n sites. Here, number of machines (sites) were increased from 1 to 6. Sizes of initial database and incremental database were taken as 80% and 20% respectively. Initial and incremental database were divided
equally among the machines and minimum support was taken as 1%. When 1 machine(site) was used, it was the sequential run time of the Borders algorithm. The results are given in Figure 5.7 on the next page.

Figure 5.5: Execution Time of DistributedBorders for different Minimum Supports (minsup)
CHAPTER 5. FREQUENT ITEMSETS IN DYNAMIC DATABASES

Figure 5.6: Execution Time of Distributed_Borders for different Database Sizes

Figure 5.7: Execution Time of Distributed_Borders for different Number of Sites
Observations: The results of the first experiment was obvious and straightforward. In some cases, execution time performance did not improve with the increase of minimum support. This was because whole scan of the database might be required in some sites. It was evident from the second experiment that execution time increased with the increase of the size of initial database and incremental database. However, it increased linearly. Third experiment measured speedup and efficiency of the algorithm. Average efficiency of 63% and 67% for T2016200K and T2014200K respectively were found, which showed that the algorithm achieved sublinear speedup. This speedup is acceptable for any distributed algorithm. However, like other distributed algorithms, performance of this algorithm also depends on the factors such as database types, distribution of data, skewness of data, network speed and other network related problems.

5.4 Discussion

The chapter has presented two enhanced versions of the Borders algorithm: ModifiedBorders and DistributedBorders. ModifiedBorders has tried to reduce the execution time by avoiding full scan of the database in most of the cases. On the other hand, DistributedBorders is the modification of the Borders algorithm to make it suitable for distributed dynamic databases.

Based on the algorithms reported so far, it is quite evident that finding frequent itemsets is a crucial phase in association rule mining. With the increase in dimensionality of databases, the cost of frequent itemset finding also increases. Therefore, the frequent itemset finding task should be limited to those features appropriate or relevant to the task, which increases the efficiency of the algorithms to a great extent. Next chapter attempts to highlight some of the popular feature selection methods and presents a novel method for relevant feature selection.