A comprehensive method for discovering the maximal frequent set

Nivedita Pandey, Prof. Love Verma

Department of Computer Science & Engineering, Raipur Institute of Technology, Raipur, Chhattisgarh

Abstract: The association rule mining can be divided into two steps. The first step is to find out all frequent itemsets, whose occurrences are greater than or equal to the user-specified threshold. The second step is to generate reliable association rules based on all frequent itemsets found in the first step. Identifying all frequent itemsets in a large database dominates the overall performance in the association rule mining. In this paper, we propose an efficient method, INCREMENTAL PINCER, for discovering the maximal frequent itemsets. The INCREMENTAL PINCER method combines the advantages of both the DHP and the Pincer-Search algorithms. The combination leads to two advantages. First, the INCREMENTAL PINCER method, in general, can reduce the number of database scans. Second, the INCREMENTAL PINCER can filter the infrequent candidate itemsets and can use the filtered itemsets to find the maximal frequent itemsets. These two advantages can reduce the overall computing time of finding the maximal frequent itemsets. In addition, the INCREMENTAL PINCER method also provides an efficient mechanism to construct the maximal frequent candidate itemsets to reduce the search space.

Keyterms: association rules, data mining, frequent itemsets, the INCREMENTAL PINCER method

I. Introduction

The process of mining association rules can be decomposed into two steps [13]. The first step is to find out all frequent itemsets, whose occurrences are greater than or equal to the user-specified threshold. The second step is to generate reliable association rules based on all frequent itemsets found in the first step. The cost of the first step is much more expensive than the second step. Therefore, much research focused on developing efficient algorithms for finding frequent itemsets. A well-known Apriori algorithm proposed by R. Agrawal and R. Srikant [13] was the first efficient method to find the frequent itemsets. The main contribution of the Apriori algorithm is it utilizes the downward closure property, i.e., any superset of an infrequent itemset must be an infrequent itemset, to efficiently generate candidate itemsets for the next database scan. By canning a database k times, the Apriori algorithm can find all frequent itemsets of a database, where k is the length of the longest frequent itemset in the database. Many methods based on the Apriori algorithm have been proposed in the literature. In general, they can be classified into three categories: reduce the number of candidate itemsets, reduce the number of database scans, and the combination of top-down and bottom-up search. Reduce the number of candidate itemsets: Methods in this category try to generate a small number of candidate itemsets efficiently in order to reduce the computational cost. The hash-based algorithm DHP proposed by Park et al. [6] is an example. The main contribution of the DHP algorithm is it uses a hash table to filter the huge infrequent candidate itemsets before the next database scan. However, the DHP algorithm needs to perform database scans as many times as the length of the longest frequent itemset in a database. Reduce the number of database scans: Scanning a database iteratively is time consuming. Thus, methods in this category try to reduce database scans aim at reducing disk I/O costs. The Partition algorithm proposed by Savasere et al. [1] generates all frequent itemsets with two database scans. The Partition algorithm divides the database into several blocks such that each block in the database can be fitted into the main memory and can be processed by the Apriori algorithm. However, the Partition algorithm examines much more candidate itemsets than the Apriori algorithm. Brin et al. [17] proposed the DIC algorithm that also divides the database into several blocks like the Partition algorithm. Unlike the Apriori algorithm, once some frequent itemsets are obtained, the DIC algorithm can generate the candidate itemsets in different blocks and then add them to count for the rest blocks. However, the DIC algorithm is very sensitive to the datadistribution of a database. The combination of bottom-up and top-down search: Methods in this category are also based on the downward closure. They obtain the frequent itemsets from the bottom-up direction like the Apriori algorithm. In the mean time, they use the infrequent itemsets found in the bottom-up direction to split the maximal frequent candidate itemsets in the top-down direction in each round. The advantage is that once the maximal frequent itemsets are obtained, all subsets of the maximal frequent itemsets are also identified. Therefore, all subsets of the maximal frequent itemsets do not need to examine from the bottom-up direction. Without the top-down pruning, they need to scan...
A comprehensive method for discovering the maximal frequent set

database as many times as the length of the longest frequent itemset. However, the improvement is not clear when the length of the longest frequent itemset is relatively short. The Pincer-Search algorithm proposed by D. Lin et al. [2] and the MaxMiner algorithm proposed by R.J. Bayardo [6] are two examples. In these two methods, the generation of the maximal frequent candidate itemsets is not efficient. They may spend a lot of time on finding the maximal frequent itemsets. In this paper, we propose an efficient method, INCREMENTAL PINCER, to generate the maximal frequent itemsets in the category of the combination of bottom-up and top-down search. The proposed method combines the advantages of both the DHP and the Pincer-Search algorithms. Unlike the DHP algorithm, the

II. Incremental Pincer

method is very efficient in reducing the number of database scans when the length of the longest frequent itemset is relatively long. Unlike the Pincer-Search algorithm, the INCREMENTAL PINCER method can filter the infrequent itemsets with the hash technique from the bottom-up direction and then can use the filtered itemsets to find the maximal frequent itemsets in the top-down direction. In addition, the INCREMENTAL PINCER method also provides an efficient mechanism to construct the maximal frequent candidate itemsets.

Table 1: Database D.

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A, C, D</td>
</tr>
<tr>
<td>2</td>
<td>B, C, E, F</td>
</tr>
<tr>
<td>3</td>
<td>A, B, C, E, F</td>
</tr>
<tr>
<td>4</td>
<td>B, E</td>
</tr>
<tr>
<td>5</td>
<td>A, C, F</td>
</tr>
</tbody>
</table>

Table 2: All frequent itemsets (the minimum support = 40%).

<table>
<thead>
<tr>
<th>Support</th>
<th>Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>AF, BC, BF, CE, EF, ACF, BCE, BCF, CEF, BCEF</td>
</tr>
<tr>
<td>3</td>
<td>A, B, E, F, AC, BE, CF</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
</tr>
</tbody>
</table>

III. Related Algorithms

Many methods have been proposed to determine all frequent itemsets in the association rule mining [2, 3, 4, 6, 8, 9, 14, 15, 17]. Since our method combines the advantages of the DHP and Pincer-Search algorithms, in this section, we briefly describe the Apriori, the DHP, and the Pincer-Search algorithms.

2.1 The Apriori Algorithm

The Apriori algorithm is given as follows.

Algorithm Apriori()
1. Scan D to obtain L1, the set of frequent 1-itemsets;
2. for (k = 2; Lk-1 Ø; k++) do
3. Ck = apriori-gen(Lk-1); // Generate new candidates from Lk-1
4. for all transactions t D do
5. Ct = subset(Ck, t); // Candidates contained in t
6. for all c Ct do
7. c. count++;
8. Lk = {c Ck | c. count minimum support};
9. All frequent itemsets = kLk; end_of_Apriori

In the first round, the Apriori algorithm scans the database to determine L1 (line 1). In the kth round, where k 2, the process of the Apriori algorithm can be divided into the following three steps. Step 1. Line 3 constructs Ck from Lk-1, th Step 3. Line 9 determines the Lk, whose support is greater than or equal to the user-specified minimum support, from Ck. The algorithm terminates when no more candidate itemsets can be
A comprehensive method for discovering the maximal frequent set

delivered for next round. The algorithm needs to do multiple database scans as many times as the length of the longest frequent itemset. Therefore, its performance decreases which was determined in the (k-1) round. dramatically when the length of the longest

Step 2. Lines 4-7 scan the database to count the support of each k-itemset in Ck.

2.2 The DHP Algorithm
The DHP algorithm is given as follows. //Step 1 of the DHP algorithm

Function build_hash_table()
1. Initialize all hash buckets in the hash table H2 to zero;
2. for all transactions t ∈ D do
3. Insert and count the supports of all 1-itemsets in a hash tree;
4. for all 2-item subsets x of t do
5. H2[h2(x)]++;
6. L1 = {c | c. count minimum support, c is in the hash tree};
end_of_build_hash_table //Step 2 of the DHP algorithm

Function gen_candidate(L1, H2, C2)
1. C2 = L1 L1 = {X Y | X, Y ∈ L1}
2. for all 2-itemsets c ∈ C2 do
3. if H2[h2(c)] < the minimum support then C2 = C2 \ {c}
end_of_gen_candidate

In the Apriori algorithm, it actually counts the support of every itemset in Ck by scanning the database in each round. The main contribution of the DHP algorithm is that it filters the infrequent itemsets in Ck by using the hash technique and then counts the support of frequent itemset is relatively long, every unfiltered itemset in Ck. Since the number of the itemsets in Ck is decreased substantially for next database scan, the overall performance is improved. The process of the DHP algorithm can be divided into two steps. In step 1, function build_hash_table() identifies L1 and builds a hash table H2 in the first database scan. The hash table H2 is built by determining the hash value of each 2-item subset of each transaction by a hash function h2, and then add it to the corresponding hash bucket. In step 2, function gen_candidate() generates C2 and checks each 2-itemset in C2 according to the value of the corresponding hash bucket in H2. If the value of the corresponding hash bucket is smaller than the minimum support, this 2-itemset is an infrequent itemset and its support does not need to be counted in next database scan. An example of the construction of the hash table and the generation of C2 is shown in Figure 1. Like the Apriori algorithm, the DHP algorithm needs to scan a database as many times as the length of the longest frequent itemset. It is inefficient if the length of the longest frequent itemset is long.

Figure 1: An example of the construction of the hash table and the generation of the C2.
3.3 The Pincer-Search Algorithm

The Pincer-Search algorithm is given as follows. Algorithm Pincer-Search (){

1. C1 = {all distinct 1-itemsets in D};
2. n = the number of 1-itemsets in C1;
3. MFCS = {n-itemset}; //the set of maximal frequent candidate itemsets
4. MFS = ; //the set of maximal frequent itemsets
5. k = 1; //pass
6. while Ck do
7. Scan the database and count the supports for MFCS and Ck;
8. MFS = MFS {frequent itemsets in MFCS};
9. Lk = {frequent itemsets in Ck} {subsets of MFS};
10. Sk = {infrequent itemsets in Ck};
11. if Sk then call MFCS_gen();
12. Ck+1 = Lk Lk;
13. if any frequent itemset in Lk was removed then call recover() to recover Ck+1;
14. for all itemsets c Ck+1 do
15. if c any element in MFCS then remove c from Ck+1;
16. k = k+1;
17. return MFS;
}

end_of_Pincer-Search function MFCS_gen(){
1. for all itemsets s Sk do
2. for all itemsets m MFCS do
3. if s m then
4. MFCS = MFCS [m];
5. for all items e itemset s
6. if m [e] is not a subset of any itemset in MFCS then
7. MFCS = MFCS [m [e]];
8. return MFCS;
}

end_of_MFCS_gen function recover(){
1. for all itemset l Lk do
2. for all itemsets m MFS do
3. if the first k-1 items in l are also in m then
4. for i from j+1 to |m| do
5. Ck +1 = Ck +1{l.item1, l.item2, …, l.itemk, m.itemi};
}

end_of_recover

contains all infrequent itemsets will be used to split the maximal frequent candidate itemsets in MFCS in the top-down direction (function MFCS_gen()). The algorithm will be terminated when there are no itemsets in MFCS.

The Pincer-Search algorithm also uses the downward closure. The downward closure consists of two properties. The first property is that all supersets of the infrequent itemsets must also be infrequent. This property is used in many typical bottom-up algorithms of the association rule mining, such as the Apriori algorithm. The second property is that all subsets of a frequent itemsets must also be frequent. This property can be used in a top-down algorithm of the association rule mining. The Pincer-Search algorithm is very efficient when the length of the longest frequent itemset of a database is long. However, its disadvantage is that the initialization of the maximal frequent candidate set is not efficient. It may spend a lot of time on finding the set of maximal frequent itemsets. Given the database shown in Table 1 as an example, we have C1 = {A, B, C, D, E, F} and the set of maximal frequent candidate itemsets MFCS = {ABCDEF}. After the first round, the infrequent 1-itemset is D, and MFCS becomes {ABCEF}. Assume that AB and AE are infrequent 2-itemsets in the second round. Consider the 2-item subset AB in ABCF, {ABCEF} will be split into {BCEF, ACEF}. Consider the 2-item subset AE in ACEF, the {ACEF} will be split into {CEF, ACF}. Thus,
Algorithm INCREMENTAL_PINCER()
1. In the first round, scan the database D to count the support of all 1 itemsets and build a hash table H2;
2. C2 is filtered by the H2;
3. call construct_maximal_frequent_candidate_itemsets(C2, H2);
4. In the second round, divide the database into several blocks;
5. for all blocks b D do
6. Count the supports of itemsets in C2 and MFCS;
7. call process_collision(C2, H2) to process the collisions of the hash bucket;
8. Move the maximal frequent itemsets from MFCS to the hash tree;
9. Apply the Pincer/Search algorithm to the rest of rounds; end_of_algorithm

Function construct_maximal_frequent_candidate_itemsets( C2,  H2)
1. Cmax = \{x = x_1x_2x_3…x_n | x_1x_2, x_1x_3, x_1x_n \in C2, where n > 2\};
2. m = 3; MFCS = \{\};
3. for all x = x_1x_2x_3…x_n \in Cmax do
4. Push x into the stack initially;
5. while the stack is not empty do
6. Pop up an element x from the stack;
7. while m < n do
8. k = h_2(x_ix_m), for i = 2, 3,…, m 1;
9. if (H_2(k) < minimum support) then
10. Split x_1x_2x_3…x_n into two (n-1)-itemsets, x_1x_2x_3…x_ix_m-1x_m+1…x_n and x_1x_2x_3…x_1x_i+1…x_m+1…x_n;
11. if is_maximal_candidate_itemset(x_1x_2x_3…x_1x_i+1…x_mx_m+1…x_n) = true then push x_1x_2x_3…x_1x_i+1…x_m+1…x_n into the stack;
12. else discard x_1x_2x_3…x_1x_i+1…x_mx_m+1…x_n;
13. if is_maximal_candidate_itemset(x_1x_2x_3…x_1x_i…x_m-1x_m+1…x_n) = true
14. then continue processing x_1x_2x_3…x_1x_i…x_m-1x_m+1…x_n;
15. m = m+1;
16. else x_1x_2x_3…x_1x_i…x_m-1x_m+1…x_n is discarded;
17. break;
18. if (m = n and the length of x > 2) then MFCS= MFCS + {x};
19. return MFCS; end_of_construct_maximal_frequent_candidate_itemsets
Function is_maximal_candidate_itemset(itemsetx){
1. for all itemset s in the stack do
2. if all items in x are also in s then return false;
3. else return true;end_of_is_maximal_candidate_itemset
Function process_collision(C2, H2)
1. for all blocks b D do
2. for all H_2(k) minimum support do
3. for all ci C2 that hashed into H_2(k), where \( i = 1, 2,…, n \) don
4. then use the infrequent 2-itemset cj into split itemsets in MFCS;
5. Remove the infrequent 2- itemset cj from C2;end_of_process_collision

IV. The Incremental Pincer Method
Our method, INCREMENTAL PINCER, combines the advantages of both the DHP and Pincer-Search algorithms. In the INCREMENTAL PINCER method, it uses the hash technique of the DHP algorithm to filter the infrequent itemsets in the bottom-up direction. Then it uses a top-down technique that is similar to the Pincer-Search algorithm to find the maximal frequent itemsets. The main difference of the top-down techniques between the INCREMENTAL PINCER method and the Pincer-Search algorithm is that the INCREMENTAL PINCER method provides a more efficient mechanism to initialize the set of maximal frequent candidate itemsets than that of the Pincer-Search algorithm. By combining the advantages of the DHP and Pincer-Search algorithms, the number of database scan and the search space of items can be reduced. The algorithm of the INCREMENTAL PINCER method is given as follows
4. \textbf{if} \ ((H_2(k) \text{ support}(c_j) > \text{ support}(c_j)) \text{ minimum support, } j = 1, 2, \ldots, n) \text{ then } i = 1

In algorithm INCREMENTAL PINCER(), lines 1-2 use the hash technique to filter the infrequent itemsets in \( C_2 \) in the bottom-up direction. Line 3 constructs the set of maximal frequent candidate itemsets MFCS. Line 6 counts the supports of itemsets in MFCS and \( C_2 \). Line 7 splits the maximal frequent candidate itemsets if some conditions are satisfied. Line 8 moves the maximal frequent itemsets from MFCS to the hash tree. Line 9 performs the Pincer-Search algorithm to get the maximal frequent itemsets.

We first explain how function \text{construct_maximal_frequent_candidate_itemsets()} works. Line 1 constructs \( C_{\text{max}} \) with all 2-itemsets that have the same first item in \( C_2 \). Lines 3-19 generate the set of maximal frequent candidate itemsets, MFCS. The generation process is as follows. Assume that an itemset \( x \) in \( C_{\text{max}} \) is denoted as \( x_1x_2x_3\ldots x_n \). Consider the first \( m \) items in \( x_1x_2x_3\ldots x_n \), for \( m = 3, \ldots, n \), and examine the 2-item subset \( x_{im} \) of \( x \), for \( i = 2, 3, \ldots, m \). If the number of 2-itemsets in the corresponding hash bucket of \( x_{im} \) is smaller than the minimum support, i.e., \( x_{im} \) is not in \( C_2 \), split \( x_1x_2x_3\ldots x_{im-1}x_{im+1}\ldots x_n \) into \( x_1x_2x_3\ldots x_{im-1}x_{im+1}\ldots x_n \) and \( x_1x_2x_3\ldots x_{im-1}x_{im+1}\ldots x_n \). Itemsets \( x_1x_2x_3\ldots x_{im-1}x_{im+1}\ldots x_n \) and \( x_{im} \) are also in any element in the stack. The generation process continues.

An itemset is popped up from the stack and the generation process continues. Case 1. All items in \( x_1x_2x_3\ldots x_{im-1}x_{im+1}\ldots x_n \) are also in any element in the stack. Itemset \( x_1x_2x_3\ldots x_{im-1}x_{im+1}\ldots x_n \) is discarded. The generation process continues to examine the first \( m-1 \) items in \( x_{im+1} \) of \( x_1x_2x_3\ldots x_{im-1}x_{im+1}\ldots x_n \). Case 3. Only items in \( x_1x_2x_3\ldots x_{im-1}x_{im+1}\ldots x_n \) are also in any element in the stack. Itemset \( x_1x_2x_3\ldots x_{im-1}x_{im+1}\ldots x_n \) is discarded. The generation process continues to examine the first \( m-1 \) items in \( x_{im+1} \) of \( x_1x_2x_3\ldots x_{im-1}x_{im+1}\ldots x_n \). Case 4. Otherwise, itemset \( x_1x_2x_3\ldots x_{im-1}x_{im+1}\ldots x_n \) is pushed into the stack and the generation process continues to examine the first \( m-1 \) items in \( x_{im+1} \) of \( x_1x_2x_3\ldots x_{im-1}x_{im+1}\ldots x_n \). The generation process is continuing until \( m = n \). Then we get a maximal frequent candidate itemset. Once one maximal frequent candidate itemset is generated, one of the itemsets in the stack is popped up and the generation process is applied until the stack is empty.

An example of the generation process is shown in Figure 2. Let \( C_2 = \{ AB, AC, AD, AE, AF, BC, BF, CD, CE, CF \} \). \( C_{\text{max}} = \{ ABCDEF \} \).

\[ C_1 = \{ AB, AC, AD, AE, AF, BC, BF, CD, CE, CF \} \]
\[ C_{\text{max}} = \{ ABCDEF \} \]
\[ [ABCD] \]
\[ DEF, BC \text{ is in } C_1 \]
\[ [ABCD] \]
\[ EF, BD \text{ is not in } C_1 \]
\[ ABCF \text{ maximal frequent candidate itemsets} \]

Figure 2: An example of the split process

www.iosrjournals.org 36 | Page
In INCREMENTAL PINCER method, the collision of the hash buckets cannot be avoided by using the hash technique. The collision may result in an infrequent itemset be used to construct the maximal frequent candidate itemsets. For example, assume that C2 = \{AB, AC, AD, AE, AF, BC, BF, CD, CE, CF\} is given. One of the maximal frequent candidate itemsets of C2 is ABCF. Assume that AC, a frequent itemset, and AF, an infrequent itemset, are hashed into bucket2. Since AC and AF are in the same bucket, AF cannot be filtered and will be used to construct the maximal frequent candidate itemsets. Function process_collision() provides a solution of this problem. In the following, we explain how it works. First, it divides the database into several blocks. In the second round, the supports of elements in C2 and MFCS are counted. The number of 2-itemsets hashed into bucket k in H2 is denoted as H2(k). Assume that there are n 2-itemsets, c1, c2,..., cn, in C2, are hashed into bucket k. An infrequent itemset cj can be identified by the following equation:

\[ H2(k) \text{ support}(c_i) \leq \text{support}(c_j) \leq \text{minimum support}, \quad j = 1, 2, ..., n \]

where the supports of ci and cj among the k blocks are denoted as support(ci) and support(cj), respectively. In each block scanning, all infrequent itemsets in C2 are identified and are deleted from C2. The identified infrequent itemsets are used to split itemsets in MFCS as well. We now give an example to explain Equation (1). Assume that H2(k) is 100 and the minimum support is 50. After scanning several blocks in the database, support(AC) is 70 and support(AF) is 10. By applying Equation (1), 100 \( (70+10)+10 = 30 \) < the minimum support. Thus, we can identify AF is an infrequent itemset and AF can be discarded. The purpose of dividing a database into several blocks is that some infrequent itemsets in C2 may be determined earlier when some blocks are scanned. The maximal frequent candidate itemsets that contain these infrequent itemsets cannot be counted further. Therefore, the division may lead us to identify those maximal frequent candidate itemsets that contain infrequent itemsets earlier and reduce the time of finding the maximal frequent itemsets.

V. Experimental Results

To evaluate the performance of the proposed method, we have implemented the INCREMENTAL PINCER method in VB.NET language along with the DHP and the Pincer-Search algorithms on a Pentium III 800 MHz PC with 512 MB of main memory. The program designed by IBM Almaden Research Center is used to generate synthetic databases [5]. This program has been widely used by many researchers [1, 2, 6, 7, 8, 9, 12, 14, 17]. By setting up parameters of the program, we can generate desired databases as benchmarks to evaluate the performance of our method. Table 3 shows the meanings of all parameters used in the program. In our experiments, we set \( N = 1000 \) and \( L = 2000 \). Table 4 shows the values of other parameters, T, D, and I. The number of the hash buckets is 500,000. We designed two tests. In the first test, we compare the relative performance and the number of database scans for the three algorithms on five databases shown in Table 4. The results of the first test are shown in Figure 3 and Figure 4.

<table>
<thead>
<tr>
<th>Database</th>
<th>D</th>
<th>File Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T104D100K-500K</td>
<td>10,000,000</td>
<td>3,755-18,827</td>
</tr>
<tr>
<td>T154D100K-500K</td>
<td>15,000,000</td>
<td>5,656-28,253</td>
</tr>
<tr>
<td>T516D100K-500K</td>
<td>15,000,000</td>
<td>5,654-28,267</td>
</tr>
<tr>
<td>T2014D100K-500K</td>
<td>20,000,000</td>
<td>7,535-37,650</td>
</tr>
<tr>
<td>T2016D100K-500K</td>
<td>20,000,000</td>
<td>7,535-37,678</td>
</tr>
</tbody>
</table>

Figure 3 shows the execution time of these three algorithms for test databases with various minimum supports. In Figure 3, our method is a little slower than the DHP algorithm on T104D100K when the minimum support is 1%. In this case, the execution time of the DHP algorithm and the INCREMENTAL PINCER method are 4 and 6 seconds, respectively. The reason is that the length of the longest itemset is two for T104D100K when the minimum support is 1%, i.e., only two database scans are
A comprehensive method for discovering the maximal frequent set

required for T10I4D100K. The INCREMENTAL PINCER method and the DHP algorithm all required two database scans. However, the INCREMENTAL PINCER method needs to spend some time on constructing the maximal frequent candidate itemsets based on C2. Therefore, it takes more time than the DHP algorithm. For other test cases, the INCREMENTAL PINCER method outperforms the DHP and the Pincer-Search algorithms. The summary reasons are given as follows. 1. In contrast with the DHP algorithm, the INCREMENTAL PINCER method finds the frequent itemsets not only in the bottom-up direction but also in the top-down direction. The execution time is improved since the number of database scans is reduced. The number of database scans is shown in Figure 4. The number of database scans required by the DHP algorithm is the length of the longest frequent itemset. In general, the number of database scans of the INCREMENTAL PINCER method is half of that of the DHP algorithm when the minimum support = 0.25% and 0.5%.

2. In contrast with the Pincer-Search algorithm, the INCREMENTAL PINCER method still has better performance than the Pincer-Search algorithm even though the number of database scans required by the INCREMENTAL PINCER method is the same as the Pincer-Search algorithm. There are two reasons. First, the INCREMENTAL PINCER method uses the hash table to filter the huge infrequent 2-itemsets in the C2 instead of actually counting the supports of all 2-itemsets. Second, it constructs the maximal frequent candidate itemsets by using the hash technique instead of the combination of all distinct 1-itemsets in a database. The search space is reduced substantially. In the second test, we evaluate the performance of the INCREMENTAL PINCER method and the DHP algorithm on the test databases with various database sizes. The results of the second test are shown in Figure 5. The performance of the Pincer-Search algorithm is not included since it takes too much time to get the execution results for test databases. In Figure 5, the number of transactions in the test databases is set from 100K to 500K and the minimum support is 0.75%. From Figure 5, we can see that both the execution time of INCREMENTAL PINCER and DHP increases when the number of transactions increases. However, the execution time of the DHP algorithm is near linear to the size of test databases. The INCREMENTAL PINCER method is not so sensitive to the size of a database compared to the DHP algorithm.

![Graphs](image-url)

Figure 5: The run time of the INCREMENTAL PINCER method and the DHP algorithm on various test databases with increasing database size. (Minimum Support = 0.75%)
VI. Conclusions

In this paper, we have proposed an efficient hash-based method, INCREMENTAL PINCER, for discovering the maximal frequent itemsets. The method combines the advantages of the DHP and the Pincer-Search algorithms. The combination leads to two advantages. First, the INCREMENTAL PINCER method, in general, can reduce the number of database scans. Second, the INCREMENTAL PINCER method can filter the infrequent itemsets and can use the filtered itemsets to find the maximal frequent itemsets. In addition, an efficient mechanism to construct the maximal frequent candidate itemsets is provided. The experimental results show that our method has better performance than the DHP and the Pincer-Search algorithms for most of test cases. In particular, our method has significant improvement over the DHP and the Pincer-Search algorithms when the size of a database is large and the length of the longest itemset is relatively long.

Reference