

Mining Frequent Itemsets with Partial Enumeration

Peiyi Tang
Department of Computer Science
University of Arkansas at Little Rock
2801 S. University Ave.
Little Rock, AR 72204

Markus P. Turkia
Department of Computer Science
University of Arkansas at Little Rock
2801 S. University Ave.
Little Rock, AR 72204

ABSTRACT

In this paper, we present an algorithm of mining frequent itemsets using partial enumeration and the FP-growth function with reduced depth of recursion. The experimental results show that our algorithm outperforms the original FP-growth algorithm without partial enumeration for the databases with high density.

Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications—*Data mining*; D.3.3 [Programming Languages]: Language Constructs and Features—*Recursion*

General Terms

Frequent Itemsets, Conditional Databases, Performance

Keywords

Partial Enumeration, k -Prefix Partitioning

1. INTRODUCTION

Mining frequent itemsets from a database is the first step in finding association rules among the items of the database. It is also the most expensive step. Since Han et al. proposed the FP-tree (frequent pattern tree) as a compact representation of a database and the recursive FP-growth algorithm of mining frequent itemsets [1], there has been tremendous interest in adopting and improving [2, 3] or parallelizing it [4]. The basic idea of FP-growth algorithm is to grow the pattern fragment through recursive calls to avoid candidate enumeration. The depth of the recursive calls grows only if the current pattern fragment is frequent. The performance study showed that the FP-growth algorithm is at least a magnitude faster than Apriori-based algorithms [1].

Given the set of frequent items $I = i_1 \cdots i_n$ in the increasing order of support, the FP-growth algorithm basically tries to find all the frequent itemsets containing i_1 ,

then those containing i_2 but not i_1 , those containing i_3 but not i_1i_2 and so on. To find the frequent itemsets containing i_1 , it constructs the conditional database on i_1 and calls the same algorithm recursively. The frequent pattern¹ grows as the recursive calls get deeper. Thus, the set of all frequent itemsets can be found without enumerating the candidates. But, it comes with the cost of deep recursive calls when the frequent patterns are long.

In our recent effort to parallelize the FP-growth algorithm [4], we used k -prefix partitioning [5] to partition the search space and have more parallel tasks for better load balancing and high speedup. In our experiments, we found incidentally that our parallel FP-growth algorithm running on a single processor is faster than the original FP-growth algorithm [1] for some databases. This prompted us to study why our algorithm is faster and to do further experiments.

It turned out that when we partition the search space for more parallel tasks, we actually enumerate the prefix of each possible frequent pattern and then call the FP-growth algorithm to find the rest of it. The extent of this enumeration is controlled by a parameter $1 \leq k < n$. Since $k < n$, it is called *partial enumeration*. When $k = 1$, our algorithm degenerates to the original FP-growth algorithm of [1]. When $k > 1$, we actually turn the construction of part of the frequent pattern from recursion to enumeration. We did extensive experiments to evaluate our algorithm with partial enumeration and found that mining with partial enumeration ($k > 1$) consistently outperforms the mining without enumeration ($k = 1$)[1] for the databases with high density.

The rest of the paper is organized as follows. In Section 2, we present our mining algorithm with partial enumeration and show that when the enumeration depth $k = 1$, our algorithm degenerates to the FP-growth algorithm of [1]. In section 3, we report and analyze the results of our experiments on five datasets from [6]. Section 4 concludes the paper.

2. PARTIAL ENUMERATION

It is necessary to provide preliminary concepts and notations first. A database D is a bag² of transactions. A transaction is a set of items which is a subset of the *total set of items* $I = \{i_1, \dots, i_n\}$, the set of all items used in the database. An *itemset* is also a subset of the total set of items I . The *support* of itemset x in database D , denoted

¹The terms of itemset and pattern are used interchangeably in this paper.

²A bag is a set which allows multiple occurrences of an element.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM SE'06 March 10-12, 2006, Melbourne, Florida, USA
Copyright 2006 ACM 1-59593-315-8/06/0004 ...\$5.00.

as $Sup_D(x)$, is the number of transactions in D containing x . That is,

$$Sup_D(x) = |\{t \mid t \in D \wedge x \subseteq t\}| \quad (1)$$

Given a threshold $1 \leq \xi \leq |D|$, an itemset x is *frequent* if $Sup_D(x) \geq \xi$. The frequent itemset mining problem is to find all the itemsets x with $Sup_D(x) \geq \xi$ for a given threshold ξ . Let the set of frequent itemsets of D with respect to ξ be denoted as $F_D(\xi)$, or F_D if ξ is understood, and we have:

$$F_D(\xi) = \{x \mid x \in 2^I \wedge Sup_D(x) \geq \xi\} \quad (2)$$

where 2^I is the power set of I , the set of all subsets of I including the empty itemset and I itself. The support of the empty itemset \emptyset is $|D|$, i.e. $Sup_D(\emptyset) = |D|$, because every transaction in D contains \emptyset . Thus, the empty itemset \emptyset is a trivial frequent itemset due to $|D| \geq \xi$.

A subset of any frequent itemset is also frequent, because $Sup_D(y) \geq Sup_D(x)$ if $y \subseteq x$, according to (1). An item i is frequent, if the support of its singleton set is greater or equal to the threshold, i.e. $Sup_D(\{i\}) \geq \xi$. Obviously, non-empty frequent itemsets can only be made of *frequent items*. Without loss of generality, we can assume that I is the set of frequent items of database D [7]. We also assume that the items of I are sorted according to the increasing order of support, i.e. $Sup_D(\{i_1\}) \leq \dots \leq Sup_D(\{i_n\})$.

In this paper, we use strings to represent itemsets. Therefore, the total set of items is written as $I = i_1 i_2 \dots i_n$. The search space is the power set of I denoted as 2^I . It is a complete Boolean lattice, $\langle 2^I, \cup, \cap \rangle$. Any subset of I is written as a string of items in the increasing order of their supports. The contiguous substring of I is written as $I[i..j]$ with $1 \leq i < j \leq n$. In particular, the k -prefix of I is denoted as $I_k = I[1..k]$. The empty subset of I is denoted as ϵ .

The idea of partial enumeration comes from k -prefix partitioning of the search space [5, 4]. Given an integer $k < n$, we use k -prefix of I , I_k , to partition 2^I as follows: for each subset x of k -prefix I_k (including the empty subset ϵ), i.e. $x \in 2^{I_k}$, we define partition P_x as:

$$P_x = \{x \cup y \mid y \in 2^{I[(k+1)..n]}\} \quad (3)$$

We have

$$P_{x_1} \cap P_{x_2} = \emptyset, \forall x_1, x_2 \in 2^{I[1..k]} \wedge (x_1 \neq x_2)$$

because if $x_1 \neq x_2$, P_{x_1} and P_{x_2} cannot have a common element. It is obvious that

$$\bigcup_{x \in 2^{I_k}} P_x = 2^I$$

Therefore, the task of mining all the frequent itemsets in 2^I from D can be reduced to mine the frequent itemsets in each individual partition P_x for all $x \in 2^{I_k}$. The x can be enumerated from I_k . Since x would be part of a frequent itemset to be found in P_x , we call the enumeration of x *partial enumeration*. Parameter k determines length of I_k and it is called *enumeration depth*.

To mine the frequent itemsets in a particular partition P_x , we do not need the whole database D , but only the conditional database on x [4] defined as follows:

DEFINITION 1. Given a database D and a subset of prefix I_k , $x \in 2^{I_k}$, the conditional database of D on x , denoted as

D_x , is

$$D_x = \{t - I_k \mid x \subseteq t \wedge t \in D\}$$

Here, the set subtraction $t - I_k$ is $t \cap \overline{I_k} = t \cap I[(k+1)..n]$.

It is important to note that conditional database D_x may have empty transactions ϵ . This happens when the original database D has transactions which contain x and contain the items in I_k only. It is important to include these empty transactions in D_x , because they will affect the decision on if x is a frequent itemset of D as we will see later.

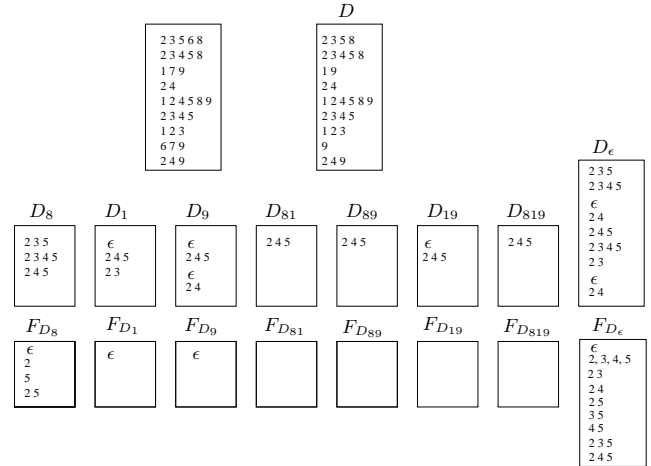


Figure 1: Example Database

Figure 1 shows an example of database and its conditional databases [4]. The original database at the top-left shows that it has 9 transactions using 9 items named $1, \dots, 9$. Suppose that the threshold is $\xi = 3$ and the frequent items in the non-decreasing order of support are: $8(3)$, $1(3)$, $9(4)$, $5(4)$, $3(4)$, $4(5)$, $2(7)$. The numbers in the parentheses above are the supports of the items. The database containing only these frequent items is shown as D . Therefore, the total set of (frequent) items is $I = 8195342$. Suppose that $k = 3$. So $I_k = 819$ and we have eight conditional databases: $D_8, D_1, D_9, D_{81}, D_{89}, D_{19}, D_{819}$ and D_ϵ , as shown in Figure 1. Note the empty transactions ϵ in D_1, D_9, D_{19} and D_ϵ . For example, D has three transactions containing 1: 19, 124589 and 123. Subtracting 819 from each of them gives the three transactions in D_1 : ϵ , 2345 and 23, respectively.

According to Definition 1, we can prove [8] that the support of any itemset $y \subseteq I - I_k$ in D_x is the same as the support of itemset $x \cup y$ in D :

$$Sup_{D_x}(y) = Sup_D(x \cup y) \quad (4)$$

Therefore, for an itemset $y \subseteq I - I_k$, itemset $x \cup y$ is frequent in D if and only if itemset y is frequent in D_x .

Based on the discussion above, the mining of frequent itemsets for D can be reduced to mining the frequent itemsets for each conditional database D_x for all $x \in 2^{I_k}$, as summarized in the following theorem [4].

THEOREM 1. Given a database D with the total set of items I , a threshold ξ , and the k -prefix I_k of I , the set of frequent itemsets of D , can be partitioned to the sets of frequent itemsets mined from each individual conditional database D_x

for all $x \in 2^{I^k}$ as follows:

$$F_D = \bigcup_{x \in 2^{I^k}} F'_{D_x}$$

and

$$F'_{D_{x_1}} \cap F'_{D_{x_2}} = \emptyset$$

for $x_1 \neq x_2$, where F'_{D_x} is the set of frequent itemsets mined from D_x as $F'_{D_x} = \{x \cup y \mid y \in F_{D_x}\}$.

Recall that F_D and F_{D_x} are the sets of frequent itemsets of D and D_x , respectively. When applying Theorem 1 to mine frequent itemsets of D , it is important to consider if the empty itemset ϵ is frequent or not in each conditional database D_x , because x is a frequent itemset of D if and only if ϵ is frequent in D_x . The support of the empty itemset ϵ is the total number of transactions in the database, because every transaction contains it. Therefore, empty itemset ϵ is frequent in D_x if and only if there are at least ξ transactions including empty transactions ϵ in D_x . In the conditional databases shown in Figure 1, empty itemset ϵ is frequent only for D_8, D_1, D_9 and D_ϵ , because each of them has at least $\xi = 3$ transactions. The set of frequent itemsets of each conditional database in the example is also shown in Figure 1. Note that $F_{D_{81}}, F_{D_{89}}, F_{D_{19}}$ and $F_{D_{819}}$ are empty, but F_{D_1} and F_{D_9} are not empty, as they contain ϵ . As a result, $F'_{D_{81}}, F'_{D_{89}}, F'_{D_{19}}$ and $F'_{D_{819}}$ are empty, but $F'_{D_1} = \{1\}$ and $F'_{D_9} = \{9\}$. We also have $F'_{D_8} = \{8, 82, 85, 825\}$ because $F_{D_8} = \{\epsilon, 2, 5, 25\}$.

Note that $F'_{D_\epsilon} = F_{D_\epsilon}$ according to $F'_{D_x} = \{x \cup y \mid y \in F_{D_x}\}$ in Theorem 1. Therefore, the equation for the set of frequent items of D can be written as:

$$F_D = \left(\bigcup_{x \in 2^{I^k \wedge x \neq \epsilon}} F'_{D_x} \right) \cup F_{D_\epsilon} \quad (5)$$

Note that the total set of items of D_ϵ is $I[(k+1)..n]$. To find the frequent itemsets of D_ϵ , we can use the same mechanism to partition the search space $2^{I[(k+1)..n]}$ for D_ϵ with its k -prefix $I[(k+1)..2k]$. The conditional database of D_ϵ on $x \in 2^{I[(k+1)..2k]}$, D_{ϵ_x} , should be

$$D_{\epsilon_x} = \{t - I[(k+1)..2k] \mid x \subseteq t \wedge t \in D_\epsilon\}$$

according to Definition 1. Let us use I_k^j to denote the j -th k -prefix:

$$I_k^j = I[((j-1)k+1)..jk] \quad (6)$$

So, our previous k -prefix I_k is denoted as I_k^1 now. Now, equation (5) can be rewritten as

$$F_D = \left(\bigcup_{x \in 2^{I_k^1 \wedge x \neq \epsilon}} F'_{D_x} \right) \cup \left(\bigcup_{x \in 2^{I_k^2 \wedge x \neq \epsilon}} F'_{D_{\epsilon_x}} \right) \cup F_{D_{\epsilon_\epsilon}} \quad (7)$$

This process of partitioning the mining of the last ϵ conditional database continues until no items are left to partition.

To avoid the notation of deep subscripts, we need to use new notation for ϵ conditional databases. Let us use D_1 for D_ϵ , D_2 for D_{ϵ_ϵ} and so on. So we have

$$D_2 = \{t - I[(k+1)..2k] \mid \epsilon \subseteq t \wedge t \in D_1\}$$

Because $\epsilon \subseteq t$ is always true, we can drop it and have

$$D_2 = \{t - I[(k+1)..2k] \mid t \in D_1\}$$

Note that we had $D_1 = \{t - I[1..k] \mid t \in D\}$. Therefore, we have

$$D_2 = \{t - I[1..2k] \mid t \in D\}$$

In general, we have

$$D_j = \{t - I[1..jk] \mid t \in D\} \quad (8)$$

The original database D is, therefore, denoted as D_0 .

Similarly, we use $D_{1,x}$ to denote D_x , $D_{2,x}$, D_{ϵ_x} , and so on for non-empty x . Thus, we have

$$D_{2,x} = \{t - I[(k+1)..2k] \mid x \subseteq t \wedge t \in D_1\}$$

according to Definition 1. In general, we have

$$D_{j,x} = \{t - I[((j-1)k+1)..jk] \mid x \subseteq t \wedge t \in D_{j-1}\} \quad (9)$$

for non-empty x . The set of frequent itemsets of $D_{j,x}$ is denoted as $F_{D_{j,x}}$.

According to (6), the total set of items I is divided to $\lfloor \frac{n}{k} \rfloor$ k -prefixes, I_k^j ($j = 1, \dots, \lfloor \frac{n}{k} \rfloor$), plus a possible $(n \bmod k)$ -prefix $I[(\lfloor \frac{n}{k} \rfloor k + 1)..n]$ if k does not divide n . Each of them is used to partition the search space and database according to Theorem 1.

After the repeated applications of Theorem 1 ($\lceil \frac{n}{k} \rceil$ times), we have the following equation for the set of frequent items of D :

$$F_D = \left(\bigcup_{j=1}^{\lfloor \frac{n}{k} \rfloor} \bigcup_{x \in 2^{I_k^j \wedge x \neq \epsilon}} F'_{D_{j,x}} \right) \cup \left(\bigcup_{x \in 2^{I[(\lfloor \frac{n}{k} \rfloor k + 1)..n] \wedge x \neq \epsilon}} F'_{D_{\lfloor \frac{n}{k} \rfloor + 1, x}} \right) \cup F_{D_E} \quad (10)$$

Here, $F'_{D_{j,x}}$ is the set of frequent itemsets of D obtained from $F_{D_{j,x}}$ by

$$F'_{D_{j,x}} = \{x \cup y \mid y \in F_{D_{j,x}}\}$$

D_E is the last ϵ conditional database:

$$D_E = \{t - I[1..n] \mid t \in D\}$$

which contains $|D|$ empty transactions ϵ . F_{D_E} can have only one frequent itemset ϵ if $|D| > \xi$ which always is true. Each of the non-empty x in (10) is an enumeration with length less or equal to k . The total number of enumerations is

$$E_n^k = \lfloor \frac{n}{k} \rfloor (2^k - 1) + (2^{n \bmod k} - 1) \quad (11)$$

For each enumeration x , we need to mine the frequent items from the corresponding conditional database. We use FP-growth function of [1] to mine conditional databases. The code of FP-growth algorithm is repeated in function **FP-growth**(D, α) shown in Figure 2. Our mining algorithm **PE**(D, I, n, k) based on (10) is also shown in Figure 2. The enumeration of x of (10) is done by the **for** loop with index i changing from 1 to the total number of enumerations, E_n^k . Each i is then converted to indexes j and l which are used to construct the partial enumeration x and the corresponding conditional database $D_{j,x}$. The **if** statement distinguishes the k -prefix enumeration when $j \leq \lfloor \frac{n}{k} \rfloor$ and the possible last $(n \bmod k)$ -prefix enumeration for $j = \lfloor \frac{n}{k} \rfloor$, if k does not divide n . x is extracted from I according to j and l by using operator $\overset{\circ}{\wedge}$ between a string (ordered set) $S = s_1 \dots s_q$ and a binary string $B = b_1 \dots b_q$ defined as follows. $T = S \overset{\circ}{\wedge} B$

```

function FP-growth( $D, \alpha$ )
begin
  if ( $D$  contains a single path  $P$ ) then
    foreach combination  $\beta$  of  $P$  do
      generate and collect itemset  $\alpha \cup \beta$ ;
    endfor
    return the set of itemsets collected;
  else
     $F \leftarrow \emptyset$ ;
    foreach item  $i$  in  $I$  of  $D$  do
      let  $\beta = \alpha \cup \{i\}$ ;
      Construct  $\beta$ 's conditional database  $D'$ ;
      if ( $D'$  is not empty) then
         $F \leftarrow F \cup \text{FP-growth}(D', \beta)$ ;
      endif
    endfor
    return  $F$ ;
  endif
end

PE( $D, I, n, k$ )
begin
   $F_D \leftarrow \{\epsilon\}$ ;
   $total \leftarrow \lfloor \frac{n}{k} \rfloor (2^k - 1) + (2^{n \bmod k} - 1)$ ;
  for ( $i \leftarrow 1; i \leq total; i \leftarrow i + 1$ ) do
     $j \leftarrow \lceil \frac{i}{2^k - 1} \rceil$ ;
     $l \leftarrow ((i - 1) \bmod (2^k - 1)) + 1$ ;
    if ( $j \leq \lfloor \frac{n}{k} \rfloor$ ) then
      Let  $B$  be the  $k$ -bit binary vector of  $l$ ;
       $x \leftarrow I[((j - 1)k + 1)..jk] \hat{\wedge} B$ ;
    else
      Let  $B$  be the  $(n \bmod k)$ -bit binary vector of  $l$ ;
       $x \leftarrow I[((j - 1)k + 1)..n] \hat{\wedge} B$ ;
    endif
    Construct  $D_{j,x}$  using using (9) and (8);
     $F_{D_{j,x}} \leftarrow \text{FP-growth}(D_{j,x}, \epsilon)$ ;
     $F'_{D_{j,x}} \leftarrow \{x \cup y \mid y \in F_{j,x}\}$ ;
     $F_D \leftarrow F_D \cup F'_{D_{j,x}}$ ;
  endfor
end

```

Figure 2: FP-Growth with Partial Enumeration

Databases	N	$ D $	L	L/N	L^2/N
chess	75	3,196	37	0.493	18.24
connect	129	67,557	43	0.333	14.19
mushroom	119	8,124	23	0.193	4.431
pumsb	2,113	49,046	74	0.035	2.59
kosarak	41,270	990,002	8.1	0.000196	0.00158

Table 1: Database Characteristics

satisfies

$$\begin{cases} s_i \in T & \text{if } b_i = 1 \\ s_i \notin T & \text{if } b_i = 0 \end{cases}$$

for each $1 \leq i \leq q$. Then, $\text{FP-growth}(D_{j,x}, \epsilon)$ is called to obtain the set of frequent itemsets $F_{D_{j,x}}$ from $D_{j,x}$ followed by constructing $F'_{D_{j,x}}$ from $F_{D_{j,x}}$. $F'_{D_{j,x}}$ is then added to F_D , the set of frequent itemsets of D . Note that F_D contains ϵ initially. This is because F_{D_E} in (10) always has it as its only element.

The conditional database $D_{j,x}$ can be constructed from the FP-tree of D . The detailed algorithm can be found in [7].

Let us look at the two extremes of the k value. When $k = n$, there are $E_n^n = 2^n - 1$ enumerations. This is the full enumeration of the entire search space 2^I . Construction of the conditional database D_x for each enumeration $x \in 2^I$ is essentially a direct checking of D against x . Of course, we are not going to take this extreme.

When $k = 1$, there are $E_n^1 = n$ enumerations, each of which is a singleton itemset: $\{i_1\}, \dots, \{i_n\}$. What our PE algorithm does is to construct n conditional databases $D_{j,\{i_j\}}$, ($1 \leq j \leq n$), and call $\text{FP-growth}(D_{j,\{i_j\}}, \{i_j\})$ for each of them to find $F_{D_{j,\{i_j\}}}$ and $F'_{D_{j,\{i_j\}}}$. By examining the algorithm of FP-growth in Figure 2, this is exactly what would be done by calling $\text{FP-growth}(D, \epsilon)$ for the original database D . Therefore, when $k = 1$ our algorithm is actually the FP-growth mining of [1] with the least enumeration (which is rightly named by [1] as the mining without candidate generation). However, when the length of frequent itemsets is large, FP-growth mining comes with the cost of deep recursive calls, as it needs a recursive call to grow the frequent pattern by one item.

When $1 < k < n$, what our algorithm does is to use enumeration x as part of frequent itemset and let the FP-growth mining to find the rest of it. In other words, our algorithm only *partially* enumerates the candidates and the degree of enumeration is controlled by k . Larger the value of k , the more enumeration and the less recursion. As we discovered in the experimental evaluation, a small k larger than 1 can reduce the mining time for the databases with high density.

3. EXPERIMENTAL EVALUATION

We have implemented our partial enumeration algorithm in Figure 2. The code for FP-growth function is taken from [9], which is an implementation of the FP-growth algorithm of [1].

In order to find out the impact of partial enumeration, we took five datasets from [6] for experimental evaluation: chess, connect, mushroom, pumsb and kosarak. The characteristics of the databases are shown in Table 1.

The column N and $|D|$ are the number of items and the number of transactions of the database, respectively. L is the average transaction length

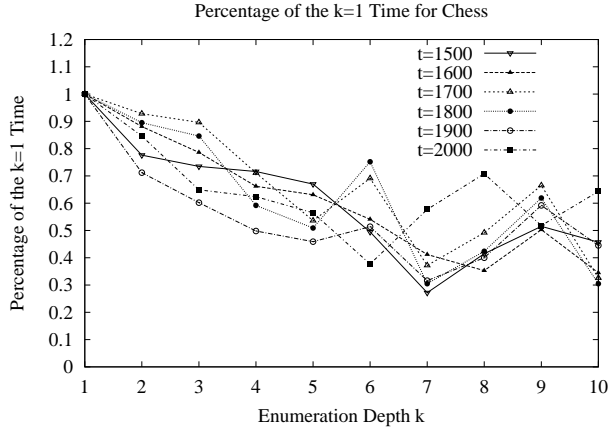
$$L = \left(\sum_{t \in D} |t| \right) / |D|$$

of the database. L/N is the canonical (relative) transaction length or *density* of the database:

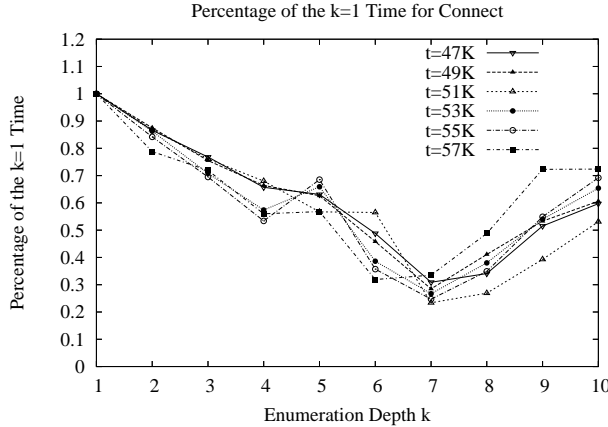
$$L/N = \left(\sum_{t \in D} |t| \right) / (|D| \cdot N)$$

Database	Execution Times in seconds						
chess	ξ	1.5K	1.6K	1.7K	1.8K	1.9K	2.0K
	T_1	4871.902	1789.748	519.277	164.916	52.464	16.177
connect	ξ	47K	49K	51K	53K	55K	57K
	T_1	22925.714	7796.067	2327.291	557.707	102.380	17.104
mushroom	ξ	0.4K	0.45K	0.5K	0.55K	0.6K	0.65K
	T_1	2771.124	409.662	330.114	363.211	97.920	68.951
pumsb	ξ	35K	36K	37K	38K	39K	40K
	T_1	6137.523	1899.209	516.022	124.163	33.804	16.320
kosarak	ξ	1.5K	2K	2.5K	3K	3.5K	4K
	T_1	284.540	108.179	56.196	36.268	26.459	19.399

Table 2: Execution Times for $k = 1$



(a) Percentages of chess



(b) Percentages of connect

Figure 3: Performance of PE

because if the database is represented by a binary bit matrix, $|D| \cdot N$ is the size of the matrix and $\sum_{t \in D} |t|$ is the number of 1-bits in the matrix. We have $L/N \leq 1$. We divided the five datasets into two groups with high and low density L/N . The group with high L/N includes chess, connect and mushroom. The other group includes pumsb and kosarak.

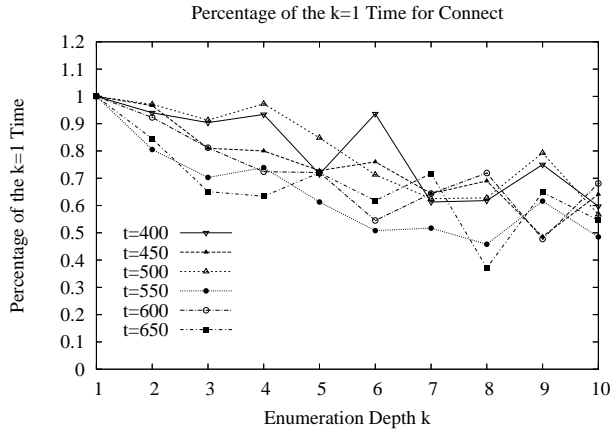
For each of the datasets in these two groups, we run our mining program with partial enumeration ($k > 1$) and without enumeration ($k = 1$) for different thresholds. All our experiments are running on a Dell desktop of 497 MHz Pentium III CPU with 512KB cache and 256MB RAM. Table 2 shows the execution times for $k = 1$, denoted as T_1 , with different thresholds for each dataset.

Figures 3 and 4 show the percentage of the execution time with partial enumeration ($k > 1$) compared with the time without partial enumeration ($k = 1$) for each dataset. It is calculated by T_k/T_1 , where T_k is the execution time with enumeration depth k . The results show that chess, connect and mushroom show consistent improvement as enumeration depth k increases. Figure 3(b) shows that enumeration depth $k = 7$ reduces the original execution time of connect without partial enumeration ($k = 1$) to its 23.4%–33.5%. Figure 3(a) shows the similar improvement for chess: 27.2%–57.7% for $k = 7$. Figure 4(a) shows that the best performance for mushroom occurs when $k = 8$: 36.8%–69.0%. Note the high density L/N (above 19.3%) for each of them shown in Table 1.

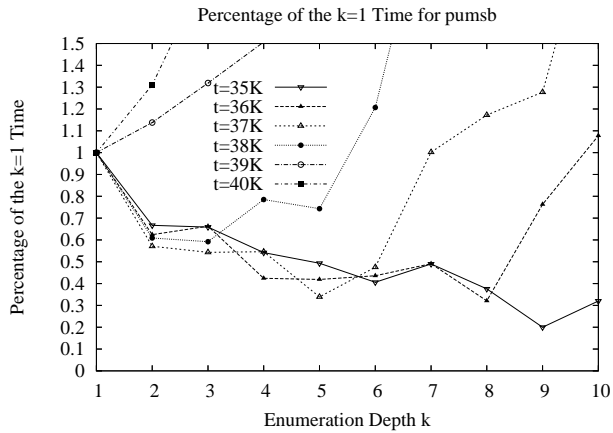
Figure 4(c) shows that partial enumeration does not help reduce the time for kosarak. The average transaction length L of kosarak is only 8.1. It has $n = 41,270$ items. Its density L/N is very low: 0.0196%.

The performance of pumsb is a mixed story as shown in Figure 4(b). The smallest threshold $t=35K$ shows consistent improvement for all $2 \leq k \leq 10$. For thresholds 36K, 37K and 38K, the performances improve until certain points of k , then start to deteriorate. For 39K and 40K, there is no improvement for all $2 \leq k \leq 10$. The L of pumsb is 74 which is large, but its L/N is small (3.5%), much smaller than those of chess, connect and mushroom (49.3%, 33.3% and 19.3%). But, it is much larger than that of kosarak (0.0196%).

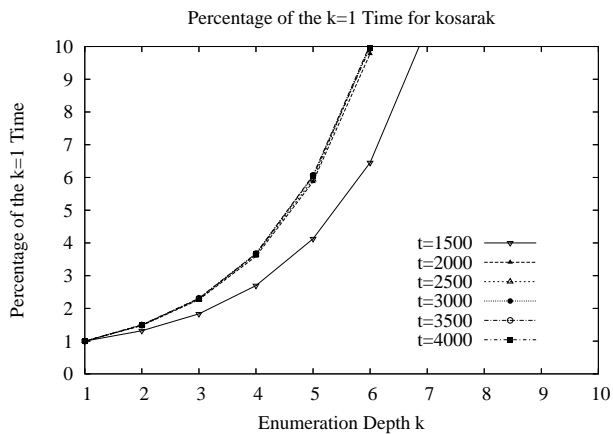
It seems that density L/N can indicate what kinds of databases will benefit from partial enumeration. The databases with high density L/N tend to have long frequent patterns. These long frequent patterns cause deep recursive calls (one item per call) in the FP-growth recursive algorithm. The partial enumeration with small enumeration depth k helps avoid deep recursive calls, thus improving the



(a) Percentages of mushroom



(b) Percentages of pumsb



(c) Percentages of kosarak

Figure 4: Performance of PE (continued)

performance.

In table 1, we list another column of L^2/N as a combined index for the suitability for partial enumeration. As a future work, we plan to do more experimental evaluations and try to find a more accurate model to predict what kind of databases are suitable for partial enumeration and what is the best enumeration depth for such a database.

4. CONCLUSION

We have presented an algorithm of mining frequent items using partial enumeration. By enumerating partial fragments before calling the recursive FP-growth function, we can partially turn recursion to enumeration and reduce the cost of deep recursive calls. We presented the result of experimental evaluation which shows that partial enumeration can reduce the time of the FP-growth mining [1] significantly for the databases with high density.

5. REFERENCES

- [1] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD International on Management of Data*, pages 1–12. ACM Press, 2000.
- [2] G. Grahne and J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset*, 2003.
- [3] Yin-Ling Cheung and Ada Wai-Chee Fu. Mining frequent itemsets without support threshold: With and without item constraints. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1052–1069, 2004.
- [4] Peiyi Tang and Markus P. Turkia. Parallelizing frequent itemset mining with FP-trees. To appear in *Proceedings of the the 21st 2006 International Conference on Computers and Their Applications (CATA'06)*, March 2006.
- [5] Peiyi Tang, Li Ning, and Ningning Wu. Domain and data partitioning for parallel mining of frequent closed itemsets. In *Proceedings of the 43rd Annual Association for Computing Machinery Southeast Conference (ACMSE'05)*, volume 1, pages 250–255, Atlanta, USA, March 2005.
- [6] Bart Goethals and Mohammed J. Zaki. FIMI'03: Workshop on frequent itemset mining implementations. Technical report, <http://fimi.cs.helsinki.fi/>, 2003.
- [7] Peiyi Tang and Markus P. Turkia. Mining frequent itemsets with partial enumeration. Technical Report titus.compsc.ualr.edu/~ptang/papers/partial.pdf, Department of Computer Science, University of Arkansas at Little Rock, 2005.
- [8] Peiyi Tang and Markus P. Turkia. Parallelizing frequent itemset mining with FP-trees. Technical Report titus.compsc.ualr.edu/~ptang/papers/par-fi.pdf, Department of Computer Science, University of Arkansas at Little Rock, 2005.
- [9] Yin ling Cheung. FP-tree/FP-growth: Mining large itemsets using FP-tree algorithm. Technical Report www.cse.cuhk.edu.hk/~kdd/freq/Nmost/ftp.zip, Chinese University of Hong Kong, 2002.