COBBLER: Combining Column and Row Enumeration for Closed Pattern Discovery

Feng Pan Gao Cong Xu Xin Anthony K. H. Tung *

National University of Singapore
email: {panfeng,conggao,xuxin, atung}@comp.nus.edu.sg
*Contact Author

Abstract

The problem of mining frequent closed patterns has receive considerable attention recently as it promises to have much less redundancy compared to discovering all frequent patterns. Existing algorithms can presently be separated into two groups, **feature (column)** [1] **enumeration** *and* row enumeration. *Feature enumeration algorithms like CHARM and CLOSET+ are efficient for datasets with small number of features and large number of rows since the number of feature combinations to be enumerated will be small. Row enumeration algorithms like CARPENTER on the other hand are more suitable for datasets (eg. bioinformatics data) with large number of features and small number of rows. Both groups of algorithms, however, will encounter problem for datasets that have large number of rows and features.*

In this paper, we describe a new algorithm called COBBLER which can efficiently mine such datasets . COBBLER is designed to dynamically switch between feature enumeration and row enumeration depending on the data characteristic in the process of mining. As such, each portion of the dataset can be processed using the most suitable method making the mining more efficient. Several experiments on real-life and synthetic datasets show that COBBLER is order of magnitude better than previous closed pattern mining algorithm like CHARM, CLOSET+ and CARPENTER.

1 Introduction

The problem of mining frequent closed patterns has received considerable attention recently as it promises to have much less redundancy compared to discovering all frequent patterns [8]. Existing algorithms can presently be separated into two groups, *feature (column) enumeration* and *row enumeration*. In feature enumeration algorithms like CHARM [9] and CLOSET+ [7], combinations of features are tested systematically to look for frequent closed patterns. Such an approach is suitable for datasets with small number of features and large number of rows since the number of feature combinations to be tested will be small.

However, for bioinformatics data with large number of features and small number of rows, the performance of these algorithms deteoriate due to the large number of feature combinations. To go around this problem, the algorithm CARPENTER [3] is developed to perform row enumeration on bioinformatics datasets instead. CARPENTER is a row enumeration algorithm which looks for frequent closed patterns by testing various combinations of rows. Since the bioinformatics datasets have small number of rows and large number of features, the number of row combinations will be much smaller than the number of feature combinations. As such, row enumeration algorithms like CARPENTER will be more efficient than feature enumeration algorithms on these kinds of datasets.

From the above, it is natural to make two observations.

First, we can conclude that different datasets will have different characteristics and thus require a different enumeration method in order to make closed pattern mining efficient. Furthermore, since these algorithms typically focus on processing different subset of the data during the mining, the characteristics of the data subset being handled will change from one subset to another. For example, a dataset that have much more rows than features may be partitioned into sub-datasets with more features than rows. Therefore a single feature enumeration method or a single row enumeration method may become inefficient in some phases of the enumeration even if they are the better choice at the start of the algorithm. As such, it makes sense to try to switch the enumeration method dynamically as different subsets of the data are being processed.

Second, both classes of algorithms will have problem handling datasets with large number of features and large number of rows. This can be seen if we understand the basic philosophy of these algorithms. In both classes of algorithms, the aim is to reduce the amount of data being considered by searching in the smaller enumeration space. For example, when performing feature enumeration, the number of rows being considered will decrease as the number of features in a feature set grow. It is thus possible to partition the large number of rows into smaller subset for efficient mining. However, for datasets with large number of rows and large number of features, adopting only one single enumeration method will make it difficult to reduce the data being considered in another dimension.

Motivated by these observations, we derived a new algorithm called COBBLER [2] in this paper. COBBLER is designed to automatically switch between feature enumeration and row enumeration during the mining process based on the characteristics of the data subset being considered. As experiments will show later, such an approach will produce good results when handling different kinds of datasets. Ex-

[1] Although column is a more suitable term here, we will use the term feature in this paper to avoid potential confusion during the technical discussion

[2] COBBLER stands for Combining Row and Column Enumeration. The letter 'b' is counted twice here.

periments show that COBBLER outperforms other closed pattern mining algorithms like CHARM [9], CLOSET+[7] and CARPENTER [3].

In the next section, we will introduce some preliminaries and give our problem definition. The COBBLER algorithm will be explained in Section 3. To show the advantage of COBBLER's dynamic enumeration, experiments will be conducted on both real-life and synthetic datasets in Section 4. Section 5 introduces some of the related work for this paper. We will conclude our discussion in Section 6.

2. Preliminary

We will give a problem description and define some notations for further discussion.

We denote our dataset as D. Let the set of binary features/columns be F=$\{f_1, f_2, .., f_m\}$ and let the set of rows be R=$\{r_1, .., r_n\}$. We abuse our notation slightly by saying that a row r_i contain a feature f_j if f_j have a value of 1 in r_i. Thus we can also say that $r_i \subseteq F$.

For example, in Figure 1(a), the dataset has 5 features represented by alphabet set $\{a, b, c, d, e\}$ and there are 5 rows, $\{r_1, \ldots, r_5\}$, in the dataset, The first row r_1 contains feature set $\{a, c, d\}$ i.e. these binary features have a value of "1" for r_1. To simplify notation, we will use the row number to represent a set of rows hereafter. For example, "23" will be used to denote row set $\{r_2, r_3\}$. And a feature set like $\{a, b\}$ will also be represented as ab.

Here, we give two concepts called **feature support set** and **row support set**.

Definition 2.1 Feature Support Set, $\mathcal{R}(F')$
Given a set of features $F' \subseteq F$, we use $\mathcal{R}(F') \subseteq R$ to denote the maximal set of rows that contain F'. □

Definition 2.2 Row Support Set, $\mathcal{F}(R')$
Given a set of rows $R' \subseteq R$, we use $\mathcal{F}(R') \subseteq F$ to denote the largest set of features that are common amount the rows in R'. □

Example 1 $\mathcal{R}(F')$ and $\mathcal{F}(R')$
Let's use the table in Figure 1(a). Let F' be the feature set $\{a, c\}$, then $\mathcal{R}(F') = \{r_1, r_5\}$ since both r_1 and r_5 contain F' and no other rows in table contain F'. Also let R' be the set of rows $\{r_1, r_2\}$, then $\mathcal{F}(R') = \{a, d\}$ since both feature a and feature d occur in r_1 and r_2 and no other features occur in both r_1 and r_2. □

Definition 2.3 Support, $|\mathcal{R}(F')|$
*Given a set of features F', the number of rows in the dataset that contain F' is called the **support** of F'. Using earlier definition, we can denote the support of F' as $|\mathcal{R}(F')|$.* □

Definition 2.4 Closed Patterns
A set of features $F' \subseteq F$ is called a closed pattern if there exists no F" such that $F' \subset F$" and $|\mathcal{R}(F")| = |\mathcal{R}(F')|$. □

Definition 2.5 Frequent Closed Patterns
*A set of features $F' \subseteq F$ is called a **frequent closed pattern** if (1) $|\mathcal{R}(F')|$, the support of F', is higher than a minimum support threshold. (2) F' is a closed pattern.* □

i	$\mathcal{F}(r_i)$
1	a,c,d
2	a,b,d,e
3	b,e
4	b,c,d,e
5	a,b,c,e

(a) Original Example Table,T

f_j	$\mathcal{R}(f_j)$
a	1,2,5
b	2,3,4,5
c	1,4,5
d	1,2,4
e	2,3,4,5

(b) Transposed Table, TT

Figure 1. Running Example

Let us illustrate these notions with another example below.

Example 2 *Given that minsup=1, the feature set $\{b, e\}$ will be a frequent closed pattern in the table of Figure 1(a) since the feature set occurs four times in the table. $\{b, d\}$ on the other hand is not a frequent closed pattern although it occurs two times in the table which is more than the minsup threshold. This is because that it has a superset $\{b, d, e\}$ and $|\mathcal{R}(\{b, d, e\})| = |\mathcal{R}(\{b, e\})|$.* □

We will now define our problem as following:
Problem Definition: Given a dataset D which contains records that are subset of a feature set F, our problem is to discover all frequent closed patterns with respect to a user support threshold *minsup*.

3. The COBBLER Algorithm

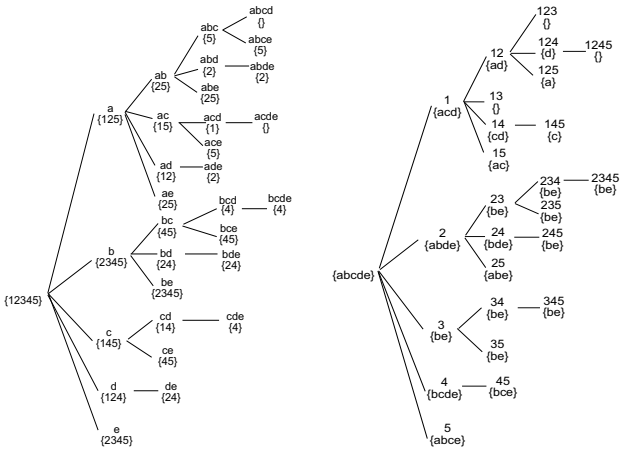
To illustrate our algorithm, we will use the tables in Figure 1 as a running example. Table 1(a) is the original table T and Table 1(b) is the transposed version of Table 1(a), TT. In TT, the row ids are the features in T while the features are the row ids in T. A row number i exists in the row f_j of TT if and only if the feature f_j occurs in row i in T.

For example, since feature "c" occurs in r_1, r_4 and r_5 in the original table, row ids "1", "4" and "5" occur in row "c" in the transposed table. To avoid confusion, we will hereafter use **tuples** to refer to the rows in the transposed table and use **rows** to refer to the rows in the original table.

3.1 Static Enumeration Tree

Algorithms for discovering closed patterns can be represented as a search in an enumeration tree. An enumeration tree can either be a *feature enumeration tree* or a *row enumeration tree*. Figure 2(a) shows a feature enumeration tree in which each possible combination of features is represented as an unique node in the tree. Node "ab" in the tree for example represents the feature combination $\{a, b\}$ while the bracket below (i.e. $\{25\}$) indicates that row r_2 and r_5 contain $\{a, b\}$. Algorithms like CHARM and CLOSET+ find closed pattern by performing depth-first search (DFS) in the feature enumeration tree (starting from the root). By imposing an order $\mathcal{ORD}_f$ on the feature, each possible combination of features will be systematically visited following a lexicographical order. In Figure 2(a), the order of enumeration will be $\{a, ab, abc, \ldots, de, e\}$ (in absence of any optimization and pruning strategies).

The concept of a *row enumeration tree* is similar to a feature enumeration tree except that in a row enumeration tree,



(a) feature enumeration tree (b) row enumeration tree

Figure 2. Traditional row and feature enumeration tree.

i	$\mathcal{F}(r_i)$
2	d,e
3	e
4	c,d,e
5	c,e

(a) $\{b\}$-conditional table, $T|_b$

f_j	$\mathcal{R}(f_j)$
a	5
d	4

(b) $\{1,2\}$-conditional transposed table, $TT|_{\{1,2\}}$

Figure 3. Conditional Table

each possible combination of rows(instead of features), R', is represented as a node in the tree. Figure 2(b) shows a row enumeration tree. Node "12" in the figure represents row combination $\{1, 2\}$ while the bracket "{ad}" below denotes the fact the "{ad}" is found in both r_1 and r_2 (i.e. $\mathcal{F}(\{1, 2\}) = \{a, d\}$. Again, by imposing a order $\mathcal{ORD}_r$ on the rows, row enumeration algorithm like CARPENTER will be able to visit each possible combination of rows in a DFS manner on the enumeration tree. The order of node visited in Figure 2(b) will be $\{1, 12, 123, \ldots, 45, 5\}$ when no pruning strategies are adopted.

Regardless of row or feature enumeration, searches in the enumeration tree are simulated by successive generation of **conditional (original) table** and **conditional transpose table** defined as the followings.

Definition 3.1 Conditional Table, $T|_X$
Let X be a subset of features. Given the original table T, an X-conditional original table denoted as $T|_X$ is a subset of rows from T such that:

1. *Each row is a superset of X in T*
2. *Let f_i be the feature with lowest order in X according to $\mathcal{ORD}_f$. Feature f_i and all f_j that having higher order than f_i according to $\mathcal{ORD}_f$ are removed from each row in $T|_X$*

□

Example 3 *Let the original table in Figure 1(a) be T. When the node "b" in the enumeration tree of Figure 2(a) is visited, an X-conditional table, $T|_b$ (note: X=$\{b\}$) will be created and is as shown in Figure 3(a). From $T|_b$, we can infer that there are 4 rows which contain "b".* □

Definition 3.2 Conditional Transposed Table, $TT|_X$
Let X be a subset of rows (in the original table). Given the transposed table TT, an X-conditional transposed table denoted as $TT|_X$ is a subset of tuples from TT such that:

1. *Each tuple is a superset of X in TT*
2. *Let r_i be the row with lowest order in X according to $\mathcal{ORD}_r$. Row r_i and all r_j that having higher order than r_i according to $\mathcal{ORD}_r$ are removed from each tuple in $TT|_X$*

□

Example 4 *Let the transposed table in Figure 1(b) be TT. When the node "12" in the row enumeration tree of Figure 2(b) is visited, an X-conditional transposed table, $TT|_{12}$ (note: X=$\{1,2\}$) will be created and is as shown in Figure 3(b). The inference we make from $TT|_{12}$ is slightly different from that we make from the earlier example. Here we can infer that $\{a,d\}$ occurs in two rows of the dataset (i.e. r_1 and r_2).* □

In both Example 3 and 4, it is easy to see that the number of rows (tuples) in the conditional (transposed) table will be reduced as the search move down the enumeration tree. This enhanced the efficiency of mining since the number of rows (tuples) being processed at deeper level of the tree will also be reduced. Furthermore, the conditional (transposed) table of a node can be easily obtained from that of its parent. Searching the enumeration tree is thus a successive generation of conditional tables where the conditional table at each node is obtained by scanning the conditional table of its parent node.

3.2 Dynamic Enumeration Tree

As we can see, the basic characteristic of a row enumeration tree or a feature enumeration tree is that the tree is static. The current solution is to make a selection between these approaches based on the characteristic of T at the start of the algorithm. For datasets with many rows and few features, algorithms like CHARM [9] and CLOSET+ [7] that search in the feature enumeration tree will be more efficient since the number of possible feature combinations will be small. However, when the number of features is much larger than the number of rows, a row enumeration algorithm like CARPENTER [3] was shown to be much more efficient.

There are two motivations for adopting a more dynamic approach.

- First, the characteristics of the conditional tables could be different from the orignal table. Since the number of rows (or tuples) can be reduced as we move down the enumeration tree, it is possible that a table T which has more rows than features initially, could have the characteristic reversed for it's conditional tables $T|_x$ (i.e. more features than rows). As such, it makes sense to adopt a different enumeration approach as the data characteristic changes.

- Second, for datasets with large number of rows and also large number of features, a combination of row and feature enumeration could help to reduce both the number of rows and features being considered in the conditional tables thus enhancing the efficiency of mining.

Next, we will illustrate with a simple example on what we mean by dynamic switching of enumeration method:

Example 5 *Consider the table T in Figure 1(a). Let us assume that the order for features, $\mathcal{ORD}_f$ is $\{a, b, c, d, e\}$ and the order for rows, $\mathcal{ORD}_r$ is $\{1, 2, 3, 4, 5\}$. Suppose, we first perform a feature enumeration generating the $\{b\}$-conditional table (shown earlier in Figure 3(a)) followed by the $\{b,c\}$-conditional table in Figure 4(a). To switch to row enumeration, $T|_{bc}$ will first be transposed to create $TT(T|_{bc})$ [3] in Figure 4(b). Since only row 4 and 5 are in the tuples of $TT(T|_{bc})$, we next perform row enumeration on row 4, which give $TT(T|_{bc})|_4$ in Figure 4(c). From $TT(T|_{bc})|_4$, we see that feature "d" and "e" are both in row 4. Thus, we can conclude that only 1 row (i.e. row 4) contains the feature set $\{b,c\}$ + $\{d,e\}$ = $\{b,c,d,e\}$ ($\{b,c\}$ is obtained from feature enumeration while $\{d,e\}$ is obtained from row enumeration).* □

i	$\mathcal{F}(r_i)$
4	d,e
5	e

(a) $T|_{b,c}$

f_j	$\mathcal{R}(f_j)$
d	4
e	4,5

(b) TT($T|_{\{b,c\}}$)

f_j	$\mathcal{R}(f_j)$
d	
e	5

(c) TT($T|_{\{b,c\}}$)$|_4$

Figure 4. Conditional Table

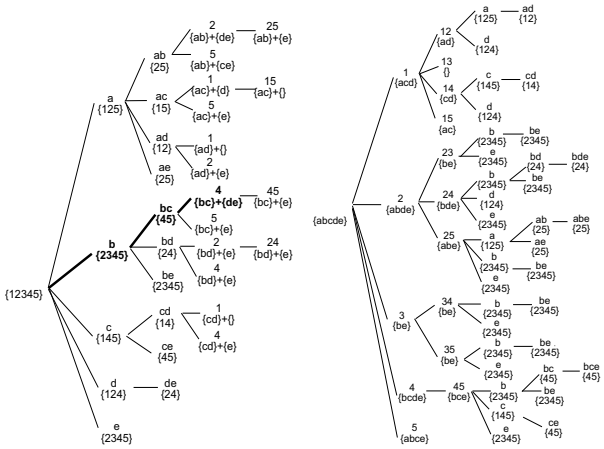
Figure 5(a) and 5(b) show examples of possible dynamic enumeration tree that could be generated from table T in our running example. In Figure 5(a), we highlight the path linking nodes "b", "bc" and "4" as they correspond to the nodes we visited in Example 5. Switching from row enumeration to feature enumeration is also possible as shown in Figure 5(b).

Like previous algorithms, COBBLER will also perform a depth first search on the enumeration tree. To ensure a systematic search, enumeration is done based on $\mathcal{ORD}_r$ for row enumeration and on $\mathcal{ORD}_f$ for feature enumeration.

To formalize the actual enumeration switching procedure, let us first divide all the nodes in our dynamic enumeration tree into two classes, *row enumerated node* and *feature enumerated node*. As the name implies, *row enumerated node* is a node which represents a subset of rows R' being enumerated while a *feature enumerated node* is a node which represents a subset of features F' being enumerated. For example, in Figure 5(a), the node "bc" is a feature enumerated node while its children node "4" is a row enumerated node.

Definition 3.3 Feature to Row Enumeration Switch *Let N be a feature enumerated node representing the feature subset F' and let $\mathcal{R}(F')$ be the rows containing F' in T. In additional, let f_l be the lowest ranking feature in F' based on $\mathcal{ORD}_f$. A switch from feature to row enumeration will follow these steps:*

[3] TT stand for transposed



(a) Switching from feature-wise to row-wise enumeration.

(b) Switching from row-wise to feature-wise enumeration.

Figure 5. Dynamic enumeration trees.

1. *Create transposed table $TT(T|_{F'})$ such that*
   - *we have a tuple for each feature $f \in F$, f having lower rank than f_l*
   - *given a tuple in $TT(T|_{F'})$ representing a feature f, the tuple contains all row r such that $r \in \mathcal{R}(F')$ and $r \in \mathcal{R}(\{f\})$*
2. *Perform row enumeration on $TT(T|_{F'})$ following the order $\mathcal{ORD}_r$.*

□

Example 6 *For example, in Figure 5(a), while node "ab" enumerates feature set, its descendant will switch to enumerate row set. The sub-tree of node "ab" will create a transposed table with one tuple for each feature c, d and e since c,d,e are of lower rank than b in $\mathcal{ORD}|_f$. Since $\mathcal{R}(\{a, b\})$=$\{2, 5\}$, the tuples in the enumeration table will only contain some subsets of $\{2, 5\}$. We thus have the enumerating order $\{2, 25, 5\}$ on the transposed table* □

To define the procedure for switching from row to feature enumeration, we first introduce the concept of **Direct Feature Enumerated Ancestor**

Definition 3.4 Direct Feature Enumerated Ancestor, $DFA(N)$
Given a row enumerated node N, its nearest ancestor which enumerates feature subsets is called its direct feature enumerated ancestor, $DFA(N)$. *In addition, we will use $F'_{DFA(N)}$ to denote the feature set represented by $DFA(N)$ The root node of the enumerating tree can be considered to enumerate both row set and feature set.* □

For example,in Figure 5(b), $DFA(\text{"}bd\text{"}) = \text{"}24\text{"}$.

Definition 3.5 Row to Feature Enumeration Switch
Let N be a row enumerated node representing the row subset R' and let $\mathcal{F}(R')$ be the maximal set of features that is found in every row of R' in T. In addition, let $F'_{DFA(N)}$ be the feature set that is represented by $DFA(N)$ and let f_l be the lowest ranking feature in $F'_{DFA(N)}$ based on $\mathcal{ORD}_f$. A switch from row to feature enumeration will follow these steps:

1. *Create table T' such that for each row r in $\mathcal{R}(F'_{DFA(N)})$, a correspond row r' is in T' with*
   - $r' \subseteq r$
   - $F'_{DFA(N)} \subseteq r'$
   - $r' \subseteq \mathcal{F}(R')$
2. *Remove all features which have lower rank than f_l from all the r'*
3. *Perform feature enumeration on T' following the order $\mathcal{ORD}_f$.*

□

In essence, a row to feature enumeration create a conditional table T' such all features combinations that is a superset of $F'_{DFA(N)}$ but a subset of $\mathcal{F}(R')$ can be tested systematically based on feature enumeration.

Example 7 *For example, in Figure 5(b), while node "24" enumerates row set, its descendant will switch to enumerate feature set. T' will thus be generated for finding all frequent closed patterns that is a subset of $\{b, d, e\}$ (i.e. $\mathcal{F}(\{2, 4\})$ but a superset of $\{\}$ (since that is the DFA of node "24"). Since $\mathcal{R}(\{\})$ contain rows r_1, r_2, r_3, r_4 and r_5, we will create 5 corresponds rows r'_1,...,r'_5 such that $r'_i \subseteq r_i$, $\{\} \subseteq r'_i$ and $r'_i \subseteq \{b, d, e\}$ for $1 \leq i \leq 5$. Based on $\mathcal{ORD}_f$, the enumeration order will be $\{b, bd, bde, d, de, e\}$.* □

Having specified the operation for switching enumeration method, we will next prove that no closed frequent patterns are missed by our algorithm. Our main argument here is that switching the enumeration method at a node N will not effect the set of closed patterns that are tested at the descendants of N. We will first prove that this is true for switching from feature to row enumeration.

Lemma 3.1 *Given a feature enumerated node N, let T_{row} be the enumeration subtree rooted at N after switching from feature to row enumeration. Let $T_{feature}$ be the imaginary subtree rooted at node N if there is no switch in enumeration method. Let $C(T_{row})$ be the set of frequent closed patterns found in enumeration tree T_{row} and $C_(T_{feature})$ be the set of frequent closed patterns that are found in enumeration tree $T_{feature}$. We claim that $C(T_{feature}) = C(T_{row})$.*

Proof:

We first prove that $C(T_{feature}) \subseteq C(T_{row})$ and then that $C(T_{row}) \subseteq C(T_{feature})$.

Suppose node N represents the feature set F'. Assuming that in $T_{feature}$, a depth first search will produce a frequent closed pattern F_C. In this case $F_C = F' + F_T$ with F_T being the additional feature set that are added onto F' when searching in subtree $T_{feature}$. It can be deduced that $\mathcal{R}(F_C) \subseteq \mathcal{R}(F')$ because $F' \subseteq F_C$. Since F_C is a frequent closed pattern, F_T being its subset will also be a frequent pattern in $\mathcal{R}(F')$. Let $R' \subseteq \mathcal{R}(F')$ be the unique maximal set of rows that contain F_T. It is easy to see that R' will also be enumerated in T_{row} since all combinations of rows in $\mathcal{R}(F')$ are enumerated in T_{row}. We can now see that both F' (since $R' \subseteq \mathcal{R}(F')$) and F_T are in R' which means that F_C will be enumerated in T_{row}. Since all closed pattern enumerated in $T_{feature}$ will be enumerated in T_{row}. Therefore, $C(T_{feature}) \subseteq C(T_{row})$.

On the other hand, assuming that F_C is a frequent closed pattern that is found under T_{row}. Let R_T be the row combination enumerated in subtree T_{row} that give F_C (i.e $F_C = \mathcal{F}(R_T)$). Since T_{row} essentially enumerate all row combinations from $\mathcal{R}(F')$, we know $R_T \subseteq \mathcal{R}(F')$ and thus F' is in every row of R_T. By definition of $\mathcal{F}(R_T)$, we know $F' \subseteq F_C$ which means that all rows containing F_C are in $\mathcal{R}(F')$. Since $T_{feature}$ will enumerate all combination of features which are in $\mathcal{R}(F')$, we know F_C will be enumerated in $T_{feature}$. Since all closed pattern enumerated in T_{row} will be enumerated in $T_{feature}$. Therefore, $C(T_{row}) \subseteq C(T_{feature})$.

We can now conclude that $C(T_{feature}) = C(T_{row})$ since $C(T_{feature}) \subseteq C(T_{row})$ and $C(T_{row}) \subseteq C(T_{feature})$. □

We next look at the proceduce for switching from row to feature enumeration. Our argument will go along the same line as Lemma 3.1.

Lemma 3.2 *Given a row enumerated node N, let $T_{feature}$ be the enumeration subtree rooted at N after switching from row to feature enumeration. Let T_{row} be the imaginary subtree rooted at node N if there is no switch in enumeration method. Let $C(T_{row})$ be the set of frequent closed patterns found in enumeration tree T_{row} and $C_(T_{feature})$ be the set of frequent closed patterns that are found in $T_{feature}$. We claim that $C(T_{feature}) = C(T_{row})$.*

□

We omitted the proof for Lemma 3.1 due to lack of space. The gist of the proof is however similar to the proof for Lemma 3.2

With Lemma 3.1 and Lemma 3.2, we are sure that the set of frequent closed patterns found by our dynamic enumeration tree is equal to the set found by a pure row enumeration or feature enumeration tree. Therefore, by a depth first search of the dynamic enumeration tree, we can be sure that all the frequent closed patterns in the database can be found. It is obvious that a complete traversal of the dynamic enumeration tree is not efficient and pruning methods must be introduced to prune off unnecessary searches. Before we explain these methods, we will first introduce the framework of out algorithm in the next section.

3.3. Algorithm

Our formal algorithm is shown in Figure 6 and the details about the subroutines are in Figure 7.

We use both the original table T and the transposed table TT in our algorithm with infrequent features removed. Our algorithm involves recursive computation of conditional tables and conditional transposed tables for performing a depth-first traversal of the dynamic enumeration tree. Each conditional table represents a feature enumerated node while each conditional transposed table represents a row enumerated node. For example, the $\{a, b\}$-conditional table represents the node "a b" in Figure 5(a) while the $\{2, 5\}$-conditional transposed table represents the node "2 5" in Figure 5(b). After setting FCP, the set of frequent closed

Algorithm
Input: Original table T, transposed table TT, features set F, row set R and support level $minsup$
Output: Complete set of frequent closed patterns, FCP
Method:

1. *Initialization.* $FCP = \emptyset$;
2. *Check switching conditions.* SwitchingCondition();
3. *If mine frequent closed patterns in row enumeration first.* RowMine($TT|_\emptyset$,R,FCP);
4. *If mine frequent closed patterns in feature enumeration first.* FeatureMine($T|_\emptyset$,F,FCP);

Figure 6. The Main Algorithm

patterns, to be empty, our algorithm will check a switching condition [4] to decide whether to perform row enumeration or feature enumeration. Depending on the switch condition, either subroutine $RowMine$ or $FeatureMine$ will be called.

The subroutine $RowMine$ takes in three parameters $TT'|_X$, R' and FCP. $TT'|_X$ is an X-conditional transposed table while R' contains the set of rows that will be considered for row enumeration according to $\mathcal{ORD}_r$. FCP contains the frequent closed patterns which have been found so far. Step 1 to 3 in the subroutine performs the counting and pruning. We will delay all discussion on pruning to Section 3.5. Step 4 in the subroutine will output the frequent closed pattern. The switching condition will be checked in Step 5 to decide whether a row enumeration or a feature enumeration will be executed next. Based on this condition, the subroutine will either continue to Step 6 for row enumeration or to Step 7 for feature enumeration. Note that the $RowMine$ subroutine has essentially no difference from the row enumeration algorithm, CARPENTER in [3] except for Step 7 where we switch to feature enumeration. Since CARPENTER is proven to be correct and Lemma 3.2 has shown that the switch to feature enumeration does not affect our result, we know that the $RowMine$ subroutine will output the correct set of frequent closed patterns.

The subroutine $FeatureMine$ takes in three parameters $T'|_X$, F' and FCP. $T'|_X$ is an X-conditional original table. F' contains the set of features that will be considered for feature enumeration according to $\mathcal{ORD}_f$. FCP contains the frequent closed patters which have been found so far. Step 1 to 3 performs counting and pruning and their explanation will also be done in later section. Step 4 will output the frequent closed pattern while Step 5 will check the switching condition to decide on the enumeration method. Based on the switching condition, the subroutine will either continue to Step 6 for feature enumeration or to Step 7 for row enumeration. We again note that the $FeatureMine$ subroutine has essentially no difference from other feature enumeration algorithm like CHARM [9] and CLOSET+ [7] except for Step 7 where we switch to row enumeration. Since these algorithms are proven to be correct and Lemma 3.1 has shown that switch to row enumeration does not affect our result. We know that the $FeatureMine$ subroutine will output the correct set of frequent closed pattern.

We can observe that the recursive computation will stop when in $RowMine$, the R' becomes empty or in

[4] We will delay the discussion for this switching condition to the next section.

Subroutine: RowMine($TT'|_X$,R',FCP).
Parameters:

- $TT'|_X$: A X-conditional transposed table;
- R': A subset of rows which have not been considered in the enumeration;
- FCP: The set of frequent closed patterns that have been found;

Method:

1. Scan $TT'|_X$ and count the frequency of occurrences for each row, $r_i \in R'$. $Y = \emptyset$.
2. **Pruning 1:** Let $U \subset R'$ be the set of rows in R' which occur in at least one tuple of $TT'|_X$. If $|U| + |X| \leq minsup$, then return; else $R' = U$;
3. **Pruning 2:** Let Y be the set of rows which are found in every tuple of the X-conditional transposed table. *Let* $R' = R' - Y$ *and remove all rows of* Y *from* $TT'|_X$;
4. If $|X| + |Y| \geq minsup$ and $\mathcal{F}(X) \notin FCP$, add $\mathcal{F}(X)$ into FCP;
5. Check the switching condition, SwitchingCondition();
6. If go on for row enumeration, for each $r_i \in R'$,
   $R' = R' - \{r_i\}$
   RowMine($TT'|_X|_{r_i}, R', FCP$);
7. If switch to feature enumeration, for each $f_i \in \mathcal{F}(X)$,
   $F' = \mathcal{F}(X) - \{f_i\}$
   FeatureMine($T|_{f_i}, F', FCP$);

Subroutine: FeatureMine($T'|_X$,F',FCP).
Parameters:

- $T'|_X$: A X-conditional original table;
- F': A subset of features which have not been considered in the enumeration;
- FCP: The set of frequent closed patterns that have been found;

Method:

1. Scan $T'|_X$ and count the frequency of occurrences for each feature, $f_i \in F'$. $Y = \emptyset$.
2. **Pruning 1:** Let $U \subset F'$ be the set of features in F' which occur in at least $minsup$ rows of $T'|_X$. $F' = U$;
3. **Pruning 2:** Let Y be the set of features which are found in every row of the X-conditional original table. *Let* $F' = F' - Y$ *and remove all features of* Y *from* $T'|_X$;
4. If $X + Y \notin FCP$ and $\mathcal{R}(X) \geq minsup$, add $X + Y$ into FCP;
5. Check the switching condition, SwitchingCondition();
6. If go on the feature enumeration, for each $f_i \in F'$,
   $F' = F' - \{f_i\}$
   FeatureMine($T'|_X|_{f_i}, F', FCP$);
7. If switch to row enumeration, transpose X conditional table $T'|_X$ to a transposed table TT_{new}, for each $r_i \in \mathcal{R}(X + Y)$,
   $R' = \mathcal{R}(X + Y) - \{r_i\}$
   RowMine($TT'_{new}|_{r_i}, R', FCP$);

Figure 7. The Subroutines

$FeatureMine$, the F' becomes empty.

3.4 Switching Condition

Switching condition are used to decide whether to switch from row enumeration to feature enumeration or vice verse. To determine that, our main idea is to estimate the enumeration cost for the subtree at a node and select the smaller one between a feature enumeration subtree and a row enumeration subtree.

The enumeration cost of a tree can be estimated from two components, the size of the tree and the computation cost at each node of the tree. The size of a tree is judged based on the estimated number of nodes it contains while the computation cost at a node is measured using the estimated number of rows (or features) that will be processed at the node.

For example, if a feature enumeration tree T_{enum} contains m nodes $\{N_1, N_2, \ldots, N_m\}$ and node N_i will process R_i rows, the enumeration cost of T_{enum} is $\sum_{i=1}^{m} R_i$. To simplify explanation, we will focus on estimating the enumeration cost of a feature enumeration tree. The estimation of the enumeration cost of a row enumeration tree will be similar.

Assume that a feature enumeration tree, T_{enum}, rooted at node N_{root} which representing F and $\mathcal{R}(F)$ and contains m sub-nodes $\{N_1, N_2, \ldots, N_m\}$. Let N_{root} correspond to conditional table $T|_F$. We give some definitions below.

- $F = \{f_1, f_2, \ldots, f_n\}$.
- $\mathcal{S}(f_j, T|_F)$, the frequency of feature f_j in $T|_F$.
- $r = |\mathcal{R}(F)|$, the number of rows conditional table $T|_F$ contains.
- $\mathcal{H}(N_i)$, the estimated maximum height of the subtree rooted at node N_i.

Given one of the node N_i representing feature set F_i', we will first use a simple probability deduction to calculate $\mathcal{H}(N_i)$. Suppose the node on level $\mathcal{H}(N_i)$ is represented as $N_{\mathcal{H}(N_i)}$, we then calculate $\overline{R}(N_{\mathcal{H}(N_i)})$, the estimated number of relevant rows being processed at the node $N_{\mathcal{H}(N_i)}$.

Assume that the set of features which have not been considered is $\{f_j | 1 \leq j \leq q\} = \{f_1, f_2, \ldots, f_q\}$ and f_j are sorted by descending order of $\mathcal{S}(f_j, T|_{F_i'})$. Let h be a value such that

$$r \cdot \prod_{j=1}^{(h+1)} \mathcal{S}(f_j, T|_{F_i'}) < minsup \leq r \cdot \prod_{j=1}^{h} \mathcal{S}(f_j, T|_{F_i'})$$

Then we calculate $\mathcal{H}(N_i)$ and $\overline{R}(N_{\mathcal{H}(N_i)})$ as

$$\mathcal{H}(N_i) = h$$

$$\overline{R}(N_{\mathcal{H}(N_i)}) = r \cdot \prod_{j=1}^{h} \mathcal{S}(f_j, T|_{F_i'}).$$

Intuitively, $\mathcal{H}(N_i)$ corresponds to the expected maximum number of levels enumeration will take place before support pruning take place.

Thus the estimated enumeration cost on node $N_{\mathcal{H}(N_i)}$ is

$$r \cdot RowProcessTime \cdot \prod_{j=1}^{h} \mathcal{S}(f_j, T|_{F_i'})$$

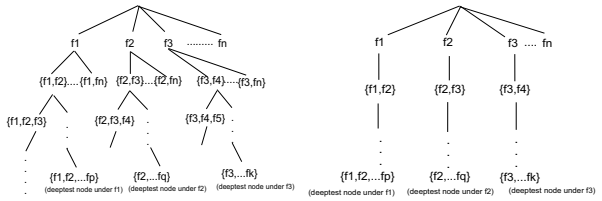
where $RowProcessTime$ is the average processing time of rows.

On the path from node N_i to node $N_{\mathcal{H}(N_i)}$, the k^{th} node will represent feature set $\{f_1, f_2, \ldots, f_k\}$ and its estimated enumeration cost is

$$r \cdot RowProcessTime \cdot \prod_{j=1}^{k} \mathcal{S}(f_j, T|_{F_i'})$$

Let $\mathcal{L}(N_i)$ be the estimated enumeration cost of enumerating through the entire path from node N_i to node $N_{\mathcal{H}(N_i)}$,

$$\mathcal{L}(N_i) = \sum_{k=1}^{h} (r \cdot RowProcessTime \cdot \prod_{j=1}^{k} \mathcal{S}(f_j, T|_{F_i'})).$$



(a) The entire feature enumeration tree, T_{enum}. (b) Simplified feature enumeration tree, T'_{enum}.

Figure 8. Entire and simplified enumeration tree

Figure 8(a) shows the entire representation of feature enumeration tree T_{enum}. Figure 8(b) is a simplified enumeration tree T'_{enum} of T_{enum} in which only the longest pathes in each sub-tree rooted at node N_{f_i} are retained. The estimated enumeration cost of T'_{enum} is $\sum_{i=1}^{n} \mathcal{L}(N_{f_i})$. We use the estimated enumeration cost of T'_{enum} as an criterion for the estimated enumeration coat of T_{enum}. Therefore, the estimated enumeration cost of the feature enumeration tree is

$$\sum_{i=1}^{n} \mathcal{L}(N_{f_i}).$$

The estimated enumeration cost of a row enumeration tree is computed in the similar way. Having computed these two estimated values, we will select the searching method that has a smaller estimated enumeration cost in the next level of enumeration.

3.5. Prune Method

Both subroutines $RowMine$ and $FeatureMine$ applies pruning strategies. We will only give a brief discussion here since they are developed in previous work and not the emphasis of our work here.

The correctness of pruning strategy 1 and 2 used in subroutine $RowMine$ has been proven in [3]. Here we will only prove the correctness of the pruning strategy applied in subroutine $FeatureMine$.

In step 3 of subroutine $FeatureMinePattern$, all the features which occur in every row of X-conditional original table $T'|_X$ will be removed from $T'|_X$ and will be considered to be already enumerated. We will prove its correctness by the following Lemma.

Lemma 3.3 *Let $T'|_X$ be an X conditional original table and Y be the set of features which occur in every row of $T'|_X$. Given any subset $F' \subseteq F$, we have $\mathcal{R}(X + F') = \mathcal{R}(X + Y + F')$.*

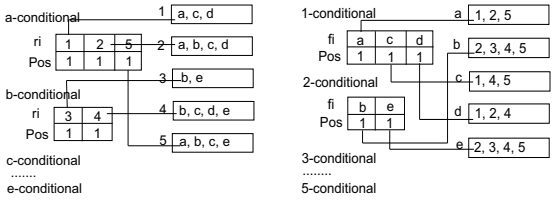
Proof: *By definition, $\mathcal{R}(X + F')$ contains a set of rows, all of which contain feature set $X + F'$. Since the features in Y occur in every row of $T'|_X$, this means that these features also occur in every row of $T'|_{(X+F')}$ (Note: $T'|_{(X+F')} \subset T'|_X$). Thus, the set of rows in $T'|_{(X+F')}$ is exactly the set of rows in $T'|_{(X+F'+Y)}$. From this, we can conclude that $\mathcal{R}(X + F') = \mathcal{R}(X + Y + F')$.* □

Example 8 *As an example to illustrate Lemma 3.3, let us consider the b-conditional table in Figure 3(a). Since feature "e" occurs in every row of $T|_b$, we can conclude that $\mathcal{R}(b)$=$\mathcal{R}(be)$ =2345. Thus, we need not create $T|_{be}$ in our search and feature "e" need not be considered for further enumeration down that branch of the enumeration tree.* □

Lemma 3.3 proves that all the frequent closed patterns found in the X-conditional table $T'|_X$ will contain feature set Y, since for each feature set $X + F'$ found in $T'|_X$, we can get its superset $X + Y + F'$ and $\mathcal{R}(X + Y + F') = \mathcal{R}(X + F')$. Thus it is correct to remove Y from all the rows of $T'|_X$ and consider Y to be enumerated.

3.6. Implementation

To show the feasibility of implementation, we will show some details about the implementation of COBBLER.



(a) feature enumeration Conditional Pointer List at Node "a"

(b) row enumeration Conditional Pointer List at Node "1"

Figure 9. Conditional Pointer List

The data structure for enumeration we used in COBBLER is similar to that we used in CARPENTER. Dataset are organized in a table and memory pointers pointing to various positions in the table are organized in a **conditional pointer list** [3]. Since we enumerate both row and feature in COBBLER, we maintain two sets of conditional pointer list for original table T and transposed table TT respectively. The conditional pointer list for row enumeration is the same as the conditional pointer list used in CARPENTER while the conditional pointer list for feature enumeration is create simply by replacing the feature ids with row ids and pointing them to the original table T. Figure 9 gives an example for feature enumeration conditional pointer list and row enumeration conditional pointer list. Most of the operations we take to maintain the conditional pointer lists are similar to CARPENTER. Interested readers are referred to [3] for details.

4. Performance

In this section we will compare the performance of COBBLER against other algorithms. All our experiments were performed on a PC with Pentium IV 2.4Ghz CPU, 1 G RAM and a 30GB hard-disk. Algorithms were coded in Standard C.

Algorithms: We compare COBBLER against two other closed pattern discovery algorithms, CHARM [9] and CLOSET+ [7]. CHARM and CLOSET+ are both feature enumeration algorithms. We also compared the performance of CARPENTER [3] and COBBLER, but since COBBLER's performance is always better than CARPENTER, we do not present the result for CARPENTER here. To make a fair comparison, CHARM and CLOSET+ are also run in the main memory after one disk scan is done to load the datasets.

Datasets: We choose 1 real-life datasets and 1 synthetic dataset to analyze the performance of COBBLER. The characteristics of the 2 datasets are shown in the table below.

Dataset	# items	# rows	row length
thrombin	139351	1316	29745
synthetic data	100000	15000	1700

As we can see, the 2 datasets we used have different characteristics. The **thrombin** dataset[5] consists of compounds tested for their ability to bind to a target site on thrombin, a key receptor in blood clotting. Each compound is described by a single feature vector comprised of a class value (A for active, I for inactive) and 139,351 binary features, which describe three-dimensional properties of the molecule. The **synthetic dataset** is generated by IBM data generator. It is a dense dataset and contains long frequent patterns even with relatively high support value.

Parameters: Three parameters are varied in our experiment, *minimum support* ($minsup$), *row ratio* (r) and *length ratio* (l). The parameter *minimum support*, $minsup$, is a minimum threshold of support which has been explained earlier. The parameters r and l are used to varying the size of the synthetic dataset we used for scalability test. The parameter *row ratio*, r, has a value above 0. It is used to generate new datasets with different number of rows using IBM data generator. All dataset with different row ratio of r was generated using a same set of parameters except that each time, the *number of rows* is changed to $15000 * r$. The parameter *length ratio*, l, has a value between 0 and 1. It is used to generate new datasets with different average row size from the original synthetic dataset listed in the table above. A dataset with a length ratio of l retains on average $l * 100\%$ of the columns in the original dataset. Columns to be retained are randomly selected for each row. The default value of r is 1 and the default value of l is 0.95. Because the real-life data is very different from the synthetic dataset, we will only use r and l for the synthetic dataset.

[5] http://www.biostat.wisc.edu/ page/Thrombin.testset.zip

4.1. Varying Minimum Support

In this set of experiments, we set l and r to their default value, 0.95 and 1, and vary the minimum support. Because of the different characteristics of the 2 datasets, we vary the minimum support in different ranges. The thrombin dataset is relatively sparse and its minimum support varies in a range which has low minimum support value. The synthetic dataset is relatively dense and the number of frequent items is quite sensitive to the minimum support, so its minimum support varies in a smaller range which has relatively high minimum support value.

Figure 10 and 11 show how COBBLER compares against CHARM and CLOSET+ as $minsup$ is varied. We can observe that on the real-life dataset, CLOSET+ performs worst for most of the time while CHARM performs best when $minsup$ is relatively high and when the $minsup$ is decreased to be low, COBBLER performs the best. This is because when the $minsup$ is high, the structure of the dataset after removing all the infrequent items is relatively simple. Because the characteristic of the data subset seldom changes during the enumeration, COBBLER will only use one of the enumeration method and become either a pure feature enumeration algorithm or a pure row enumeration algorithm. The advantage of COBBLER's dynamic enumeration cannot been seen and therefore COBBLER is outperformed by CHARM which is a highly optimized feature enumeration algorithm.

With the decrease of $minsup$, the structure of the dataset after removing infrequent items will become more complex. COBBLER begins to switch between feature enumeration method and row enumeration method according to the varying characteristic of the data subset. Therefore COBBLER outperforms CHARM in low $minsup$ on the real-life datasets.

On the synthetic dataset, COBBLER performs the best for most of the time since the synthetic dataset is dense and complex enough. CHARM performs worst on this dataset, even at very high $minsup$. This is due to the fact that the synthetic dataset is a very dense one which results in a very large feature enumeration space for CHARM.

4.2. Vary Length Ratio

In this set of experiments, we varying the size of the synthetic dataset by changing the length ratio, l. We set $minsup$ to 0.15%, r to 1 and vary l from 0.8 to 1. If l is set to values smaller than 0.8, the generated dataset will be too sparse for any interesting result. Figure 12 shows the performance comparison of COBBLER, CHARM and CLOSET+ on the synthetic dataset when we vary l. For CHARM and CLOSET+, it takes too much time to run on dataset with $l = 1$, so the result is not included in Figure 12. As we can see from the graph, COBBLER outperforms CHARM and CLOSET+ in most cases. CHARM is always the worst among these 3 algorithms and both COBBLER and CLOSET+ are order of magnitude better than it. CLOSET+ has a steep increase in run time as length ratio is increased. Its performance is as good as COBBLER when l is low but is soon outperformed by COBBLER when l is increased to some higher values.

COBBLER performance is not significantlly better than CLOSET+ with low l values because a low value of l will destroy many of the frequent patterns in the dataset, making the dataset sparse. This will cause COBBLER to perform pure feature enumeration method and lose the advantage of performing dynamic enumeration. With the increase of l, the dataset will become more complex and COBBLER will show its advantage over CLOSET+ and also CHARM.

4.3. Varying Row Ratio

In this set of experiments, we varying the size of the synthetic dataset by varying row ratio, r. We set $minsup$ to 0.15%, l to its default value of 0.95 and varying r from 0.6 to 2. Figure 13 shows the performance comparison of COBBLER, CHARM and CLOSET+ on the synthetic dataset when we vary r. As we can see, with the increase of the number of rows, the datasets become more complex and COBBLER 's dynamic enumeration strategy shows its advantage over the other two algorithms. In all the cases, COBBLER outperforms CHARM and CLOSET+ by an order of magnitude and also has a smoothest increase in run time.

As can be seen, in all the experiments we conducted, COBBLER outperforms CLOSET+ in most cases and outperforms CHARM when the dataset becomes complicated for increased l and r or decreased $minsup$. This result also demonstrates that COBBLER is efficient in datasets with different characteristics as it uses combined row and feature enumeration and can switch between these two enumeration methods according to the characteristics of a dataset while in the searching process.

5. Related Work

Frequent pattern mining [1, 2, 6, 10] as a vital topic has received a significant amount of attention during the past decade. The number of frequent patterns in a large data set can be very large and many of these frequent patterns may be redundant. To reduce the frequent patterns to a compact size, mining frequent closed patterns has been proposed. The followings are some new advances for mining closed frequent patterns.

CLOSET [5] and CLOSET+ [7] are two algorithms which discover closed patterns by depth-first, feature enumeration. CLOSET uses a frequent pattern tree (FP-structure) for a compressed representation of the dataset. CLOSET+ is an updated version of CLOSET. In CLOSET+, a hybrid tree-projection method is implemented and it builds conditional projected table in two different ways according to the density of the dataset. As shown in our experiment, both CLOSET and CLOSET+ are unable to handle long datasets due to their pure feature enumeration strategy.

CHARM [9] is a feature enumeration algorithm for mining frequent closed pattern. Like CLOSET+, CHARM performs depth-first, feature enumeration. But instead of using FP-tree structure, CHARM use a vertical format to store the dataset in which a list of row ids is stored for each feature. These row id lists are then merged during the feature enumeration to generate new row id lists that represent corresponding feature sets in the enumeration tree. In addition, a

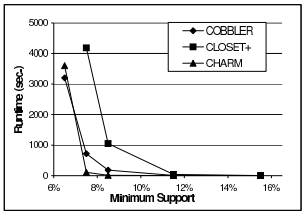


Figure 10. Varying $minsup$ (thrombin)

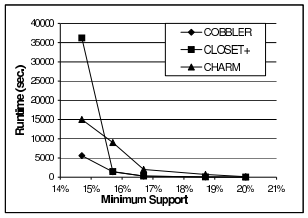


Figure 11. Varying $minsup$ (synthetic data)

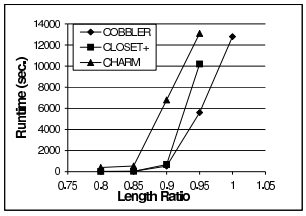


Figure 12. Varying l (*synthetic data*)

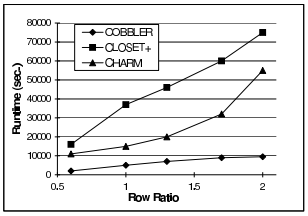


Figure 13. Varying r (*synthetic data*)

technique called *diffset* is used to reduce the size of the row id lists and the computational complexity for merging them.

Another algorithm for mining frequent closed pattern is CARPENTER [3]. CARPENTER is a pure row enumeration algorithm. CARPENTER discovers frequent closed patterns by performing depth-first, row enumeration combined with efficient search pruning techniques. CARPENTER is especially designed to mine frequent closed patterns in datasets containing large number of columns and small number of rows.

6. Conclusion

In this paper, we proposed an algorithm called COBBLER which can dynamically switch between row and feature enumeration for frequent closed pattern discovery. COBBLER can automatically select an enumeration method according to the characteristics of the datasets before and during the enumeration. This dynamic strategy helps COBBLER to deal with different kind of dataset including large, dense datasets that have varying characteristics on different data subsets. Experiments show that our approach yields good payoff as COBBLER outperforms existing frequent closed pattern discovery algorithms like CLOSET+, CHARM and CARPENTER on several kinds of datasets. In the future, we will look at how COBBLER can be extended to handle datasets that can't be fitted into the main memory.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)*, pages 487–499, Sept. 1994.

[2] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *Proc. AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94)*.

[3] F. Pan, G. Cong, and A. K. H. Tung. Carpenter: Finding closed patterns in long biological datasets. In *Proc. Of ACM-SIGKDD Int'l Conference on Knowledge Discovery and Data Mining*, 2003.

[4] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang. H-mine: Hyper-structure mining of frequent patterns in large databases. In *Proc. IEEE 2001 Int. Conf. Data Mining (ICDM'01)*, Novermber.

[5] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proc. 2000 ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD'00)*.

[6] P. Shenoy, J. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah. Turbo-charging vertical mining of large databases. In *Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00)*, pages 22–23, Dallas, TX, May 2000.

[7] J. Wang, J. Han, and J. Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *Proc. 2003 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'03)*, Washington, D.C., Aug 2003.

[8] M. Zaki. Generating non-redundant association rules. In *Proc. 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00)*, 2000.

[9] M. Zaki and C. Hsiao. Charm: An efficient algorithm for closed association rule mining. In *Proc. of SDM 2002*, 2002.

[10] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 283–286, Newport Beach, CA, Aug. 1997.