Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 04-025

SUMMARY: Efficiently Summarizing Transactions for Clustering

Jianyong Wang and George Karypis

June 22, 2004

SUMMARY: Efficiently Summarizing Transactions for Clustering *

Jianyong Wang and George Karypis
Department of Computer Science, Digital Technology Center, & Army HPC Research Center
University of Minnesota, Minneapolis, MN 55455
{jianyong, karypis}@cs.umn.edu

Abstract

Frequent itemset mining was initially proposed and has been studied extensively in the context of association rule mining. In recent years, several studies have also extended its application to the transaction (or document) classification and clustering. However, most of the frequent-itemset based clustering algorithms need to first mine a large intermediate set of frequent itemsets in order to identify a subset of the most promising ones that can be used for clustering. In this paper, we study how to directly find a subset of high quality frequent itemsets that can be used as a concise summary of the transaction database and to cluster the categorical data. By exploring some properties of the subset of itemsets that we are interested in, we proposed several search space pruning methods and designed an efficient algorithm called SUMMARY. Our empirical results have shown that SUMMARY runs very fast even when the minimum support is extremely low and scales very well with respect to the database size, and surprisingly, as a pure frequent itemset mining algorithm it is very effective in clustering the categorical data and summarizing the dense transaction databases.

1 Introduction

Frequent itemset mining was initially proposed and has been studied extensively in the context of association rule mining [2, 3, 24, 29, 15, 9, 18, 34]. In recent years, some studies have also demonstrated the usefulness of frequent itemset mining in serving as a condensed representation of the input data in order for answering various types of queries [22, 8], and the transactional data (or document) classification [5, 20, 19, 4] and clustering [32, 7, 11, 33].

Most frequent-itemset based clustering algorithms need to first mine a large intermediate set of frequent itemsets (in many cases, it is the complete set of frequent itemsets), on which some further post-processing can be performed in order to generate the final result set which can be used for clustering purposes. In this paper we consider directly mining a final subset of frequent itemsets which can be used as a concise summary of the original database and to cluster the categorical data. To serve these purposes, we require the final set of frequent itemsets have the following properties: (1) it maximally covers the original database given a minimum support; (2) each final frequent itemset can be used as a description for a group of transactions, and the transactions with the same description can be grouped into a cluster with approximately maximal intra-cluster similarity. To achieve this goal, our solution to this problem formulation is that for each transaction we find one of the longest frequent itemsets that it contains and use this longest frequent itemset as the corresponding transaction's description. The set of so mined frequent itemsets is called a *summary set*.

One significant advantage of directly mining the final subset of frequent itemsets is that it provides some chances to design more efficient algorithm. We proved that each itemset in the *summary set* must be closed, thus some search space pruning methods proposed for frequent closed itemset mining can be borrowed to accelerate the *summary set* mining. In addition, based on some properties of the *summary set*, we proposed several novel pruning methods which greatly improve the algorithm efficiency. By incorporating these pruning methods with a traditional frequent itemset min-

*This work was supported in part by NSF CCR-9972519, EIA-9986042, ACI-9982274, ACI-0133464, and ACI-0312828; the Digital Technology Center at the University of Minnesota; and by the Army High Performance Computing Research Center (AHPCRC) under the auspices of the Department of the Army, Army Research Laboratory (ARL) under Cooperative Agreement number DAAD19-01-2-0014. The content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. Access to research and computing facilities was provided by the Digital Technology Center and the Minnesota Super-computing Institute.

ing framework, we designed an efficient *summary set* mining algorithm, SUMMARY. Our thorough empirical tests show that SUMMARY runs very fast even when the minimum support is extremely low and scales very well w.r.t. the database size, and its result set is very effective in clustering the categorical data and summarizing the dense transaction databases.

The rest of this paper is organized as follows. Section 2 and Section 3 introduce the problem definition and some related work. Section 4 describes the algorithm in detail. Section 5 presents the empirical results. Section 6 presents an application of the algorithm in clustering the categorical data, and the paper ends with some discussions and conclusion in Section 7.

2 Problem Definition

A *transaction database* TDB is a set of transactions, where each transaction, denoted as a tuple $\langle tid, X\rangle$, contains a set of items (i.e., X) and is associated with a unique transaction identifier *tid*. Let $I=\{i_1, i_2, \ldots, i_n\}$ be the complete set of distinct items appearing in TDB. An *itemset* Y is a non-empty subset of I and is called an l*-itemset* if it contains l items. An itemset $\{x_1, \ldots, x_l\}$ is also denoted by $x_1 \cdots x_l$. A transaction $\langle tid, X\rangle$ is said to *contain* itemset Y if $Y \subseteq X$. The number of transactions in TDB containing itemset Y is called the (absolute) *support* of itemset Y, denoted by $sup(Y)$. In addition, we use $|TDB|$ and $|Y|$ to denote the number of transactions in database TDB, and the number of items in itemset Y, respectively.

Given a minimum support threshold, *min_sup*, an itemset Y is *frequent* if $sup(Y) \geq min\_sup$. Among the longest frequent itemsets supported by transaction T_i, we choose any one of them and denote it by SI_{T_i}. SI_{T_i} is called the *summary itemset* of T_i[1]. The set of the *summary itemsets* w.r.t. the transactions in TDB (i.e., $\cup_{i=1}^{|TDB|}\{SI_{T_i}\}$) is called a *summary set* w.r.t. database TDB. Note that the *summary set* of a database may not be unique, this is because a transaction may support more than one *summary itemset*.

Given a transaction database TDB and a minimum support threshold *min_sup*, the problem of this study is to find any one of the *summary sets* w.r.t. TDB.

Example 2.1 The first two columns in Table 1 show the transaction database TDB in our running example. Let *min_sup*=2, we sort the list of frequent items in support ascending order[2] and get the sorted item list which is called *f_list*. In this example *f_list* = <a:3, b:4, c:4, d:5, e:5, f:5>. The list of frequent items in each transaction are sorted according to *f_list* and shown in the third column of Table 1. It is easy to figure out that $\{ace$:2, acf:2, bcd:2, bd:4, bde:2, df:2, ef:3$\}$ is one *summary set* w.r.t. TDB. □

Tid	Set of items	Ordered frequent item list
01	a, c, e, g	a, c, e
02	b, d, e	b, d, e
03	d, f, i	d, f
04	e, f, h	e, f
05	a, b, c, d, e, f	a, b, c, d, e, f
06	b, c, d	b, c, d
07	a, c, f	a, c, f
08	e, f	e, f
09	b, d	b, d

Table 1. A transaction database TDB

3 Related Research

Since the introduction of the association rule mining [2, 3], numerous frequent itemset mining algorithms have been proposed. In essence, SUMMARY is a projection-based frequent itemset mining algorithm [18, 1] and adopts the natural matrix structure instead of the FP-tree to represent the (conditional) database [26, 12]. It grows a current prefix itemset by physically building and scanning its projected matrix. In [15] an algorithm was proposed to mine all most specific sentences, however, both the problem and the algorithm in this study are different from those in [15].

In Section 4 we prove that each *summary itemset* must be closed, thus some pruning methods previously proposed in the closed (or maximal) itemset mining algorithms [6, 25, 27, 10, 34, 30, 23, 21] can be used to enhance the efficiency of SUMMARY. Like several itemset mining algorithms with length-decreasing support constraint [28, 31], SUMMARY adopts some pruning methods to prune the unpromising transactions and prefixes. However, because the problem formulations are different, the pruning methods in SUMMARY are different from the previous studies.

One important application of the SUMMARY algorithm is to concisely summarize the transactions and cluster the categorical data. There are many algorithms designed for clustering categorical data, typical examples include ROCK [14] and CACTUS [13]. Recently several frequent-itemset based clustering algorithms have also been proposed to cluster categorical or

[1]Transaction T_i may contain no frequent itemset, in this case SI_{T_i} is empty and T_i can be treated as an outlier.

[2]Our experience tells that there is no clear winer between the support descending order and ascending order.

numerical data [7, 11, 33]. These methods first mine an intermediate set of frequent itemsets, and some post-processing are needed in order to get the clustering solution. SUMMARY mines the final subset of frequent itemsets which can be directly used to group the transactions to form clusters and enables us to design more effective pruning methods to enhance the performance.
Contributions. The contributions of this paper can be summarized as follows.

1. We proposed a new problem formulation of mining the *summary set* of frequent itemsets with the application of summarizing transactions and clustering categorical data.

2. By exploring the properties of the *summary set*, we have proposed several pruning methods to effectively reduce the search space and enhance the efficiency of the SUMMARY algorithm.

3. Thorough performance study has been performed and shown that SUMMARY has high efficiency and good scalability, and can be used to cluster categorical data with high accuracy.

4 SUMMARY: An Efficient Algorithm to Summarize the Transactions

In this section we first briefly introduce a traditional framework for enumerating the set of frequent itemsets, which forms the basis of the SUMMARY algorithm. Then we discuss how to design some pruning methods to speed up the mining of the *summary set*. Finally we present the integrated SUMMARY algorithm, and discuss how to revise SUMMARY to mine all the *summary itemsets* for each transaction.

4.1 Frequent Itemset Enumeration

Like some other projection-based frequent itemset mining algorithms, SUMMARY employs the *divide-and-conquer* and *depth-first search* strategies [18, 30], which are applied according to the *f_list* order. In Example 2.1, SUMMARY first mines all the frequent itemsets containing item a, then mines all frequent itemsets containing b but no a, ..., and finally mines frequent itemsets containing only f. In mining itemsets containing a, SUMMARY treats a as the current prefix, and builds its conditional database, denoted by $TDB|_a$=$\{\langle 01, ec\rangle$, $\langle 05, efc\rangle$, $\langle 07, fc\rangle\}$ (where the local infrequent items b, d, and g have been pruned and the frequent items in each projected transaction are sorted in support ascending order). By recursively applying the *divide-and-conquer* and *depth-first search* methods to $TDB|_a$, SUMMARY can find the set of frequent itemsets containing a. Note instead of using the FP-tree structure, SUMMARY adopts the natural matrix structure to store the physically projected database [12]. This is because the matrix structure allows us to easily maintain the *tid*s in order to determine which set of transactions the prefix itemset covers. In addition, in the above enumeration process, SUMMARY always maintains the current longest frequent itemset for each transaction T_i that was discovered first so far. In the following we call it the *current longest covering frequent itemset* w.r.t. T_i (denoted by LCF_{T_i}).

4.2 Search Space Pruning

The above frequent itemset enumeration method can be simply revised to mine the *summary set*: Upon getting a frequent itemset, we check if it is longer than the current longest covering frequent itemset w.r.t. any transaction that this itemset covers. If so, this newly mined itemset becomes the current longest covering frequent itemset for the corresponding transactions. Notice that this naïve method is no more efficient than the traditional all frequent itemset mining algorithm. However, the above algorithm for finding the *summary set* can be improved in two ways. First, as we will prove later in this section, any *summary itemset* must be closed and thus, the pruning methods proposed for closed itemset mining can be used. Second, since during the mining process we maintain the length of the current longest covering itemset for each transaction, we can employ additional *branch-and-bound* techniques to further prune the overall search space.

Definition 4.1 *(Closed itemset)* An itemset X is a **closed itemset** if there exists no proper superset $X' \supset X$ such that $sup(X') = sup(X)$. □

Lemma 4.1 *(Closure of a summary itemset) Any summary itemset w.r.t. a transaction T_i, SI_{T_i}, must be a closed itemset.*

Proof. We will prove it by contradiction. Assume SI_{T_i} is not closed, which means there must exist an itemset Y, such that $SI_{T_i} \subset Y$ and $sup(SI_{T_i}) = sup(Y)$. Thus, Y is also supported by transaction T_i and is frequent. However, $|Y| > |SI_{T_i}|$ contradicts with the fact that SI_{T_i} is the summary itemset of transaction T_i. □

Lemma 4.1 suggests that any pruning method proposed for closed itemset mining can be used to enhance the performance of the *summary set* mining. In SUMMARY, only one such technique, *item merging* [30], is adopted that works as follows. For a prefix itemset P, the complete set of its local frequent items that have the same support as P are merged with P to form a new prefix, and these items are removed from the list of the local frequent items of the new prefix. It is easy to see that such a scheme does not affect the correctness of the algorithm [30].

Example 4.1 Assume the current prefix is a:3, whose local frequent item list is $<e{:}2,\ f{:}2,\ c{:}3>$, among which c:3 can be merged with a:3 to form a new prefix ac:3 with local frequent item list $<e{:}2,\ f{:}2>$. □

Besides the above pruning method, we developed two new pruning methods called *conditional transaction* and *conditional database* pruning that given the set of the currently maintained longest covering frequent itemsets w.r.t. TDB, they remove some conditional transactions and databases that are guaranteed not to contribute to and generate any *summary itemsets*.

Specifically, let P be the prefix itemset that is currently under consideration, $sup(P)$ its support, and $TDB|_P = \{\langle T_{P_1}, X_{P_1}\rangle,\ \langle T_{P_2}, X_{P_2}\rangle,\ \cdots,\ \langle T_{P_{sup(P)}}, X_{P_{sup(P)}}\rangle\}$ its conditional database. Note that some (or all) of the transactions X_{P_i} $(1 \le i \le sup(P))$ can be empty.

Definition 4.2 *(Invalid conditional transaction)* A conditional transaction T_{P_i} in $TDB|_P$ (where $1{\le}i{\le}sup(P)$), is an **invalid** conditional transaction if it falls into one of the following two cases:

1. $|X_{P_i}| \le (|LCF_{T_{P_i}}| - |P|)$;
2. $|X_{P_i}| > (|LCF_{T_{P_i}}| - |P|)$, but $|\{\ \forall j \in [1..sup(P)],\ T_{P_j} \mid |X_{P_j}| > (|LCF_{T_{P_i}}| - |P|)\ \}| < min\_sup$.

Otherwise, T_{P_i} *is called a* **valid** *conditional transaction.* □

The first condition states that a conditional transaction is invalid if its size is no greater than the difference between its current longest covering frequent itemset and the length of the prefix itemset, whereas the second condition states that the number of conditional transactions which can be used to derive itemsets longer than $LCF_{T_{P_i}}$ by extending prefix P is smaller than the minimum support.

Lemma 4.2 *(Unpromising summary itemset generation) If* T_{P_i} *is an invalid conditional transaction, there will be no frequent itemset derived by extending prefix* P *that* T_{P_i} *supports and is longer than* $LCF_{T_{P_i}}$.

Proof. Follows directly from Definition 4.2. (i) If a transaction T_{P_i} is invalid because of the first condition, it will not contain sufficient items in its conditional transaction to identify a longer covering itemset. (ii) If a transaction T_{P_i} is invalid because of the second condition, the conditional database will not contain a sufficiently large number of long conditional transactions to obtain an itemset that is longer than $LCF_{T_{P_i}}$ and frequent. □

Note it is possible for an invalid conditional transaction to be used to mine summary itemsets for other valid conditional transactions w.r.t. prefix P; thus, we cannot simply prune any invalid conditional transaction. Instead, we can safely prune some invalid conditional transactions according to the following Lemma.

Lemma 4.3 *(Conditional transaction pruning) An invalid conditional transaction,* T_{P_i}*, can be safely pruned, if it satisfies:*

$$|X_{P_i}| \le \min_{\forall j,\ T_{P_j}\ is\ valid} (|LCF_{T_{P_j}}| - |P|) \qquad (1)$$

Proof. Consider an invalid conditional transaction T_{P_i} that satisfies Equation 1. Then in order for a frequent itemset supported by the conditional transaction T_{P_i} and prefix P to replace the current longest covering frequent itemset of a valid conditional transaction T_{P_j}, T_{P_i} needs to contain more than $|X_{P_i}|$ items in its conditional transaction. As a result, T_{P_i} can never contribute to the support of such an itemset and can be safely pruned from the conditional database. □

Lemma 4.3 can be used to prune from the conditional database some unpromising transactions satisfying Equation 1 even when there exist some valid conditional transactions. However, in many cases, there may exist no valid conditional transactions, in this case the whole conditional database can be safely pruned.

Lemma 4.4 *(Conditional database pruning) Given the current prefix itemset* P *and its projected conditional database* $TDB|_P$*, if each of its conditional transactions,* T_{P_i}*, is invalid,* $TDB|_P$ *can be safely pruned.*

Proof. According to Lemma 4.2, for any invalid conditional transaction, T_{P_i}, we cannot generate any frequent itemsets longer than $LCF_{T_{P_i}}$ by growing prefix P. This means that if each conditional transaction is invalid, we can no longer change the current status of the set of the currently maintained longest covering frequent itemsets w.r.t. prefix P, $\cup_{i=1}^{sup(P)}\{LCF_{T_{P_i}}\}$, by extending P; thus, $TDB|_P$ can be safely pruned. □

Example 4.2 Assume the prefix is c:4 (i.e., P=c). From Table 1 we get that $TDB|_c$=$\{\langle 01,\ e\rangle,\ \langle 05,\ def\rangle,\ \langle 06,\ d\rangle,\ \langle 07,\ f\rangle\}$, and LCF_{01}=ace:2, LCF_{05}=ace:2, LCF_{06}=bcd:2, and LCF_{07}=acf:2. Conditional transactions $\langle 01,\ e\rangle$, $\langle 06,\ d\rangle$, and $\langle 07,\ f\rangle$ fall into case 1 of Definition 4.2, while $\langle 05, def \rangle$ falls into case 2 of Definition 4.2, thus all the conditional transactions in $TDB|_c$ are invalid. According to Lemma 4.4, conditional database $TDB|_c$ can be pruned. $\square$

```
ALGORITHM 1: SUMMARY(TDB, min_sup)

INPUT: (1) TDB: a transaction database, and (2) min_sup: a minimum support threshold.
OUTPUT: (1) SI: the summary set.
BEGIN
01. for all t_i ∈ TDB
02.     SI_{t_i} ← ∅;
03. call summary(∅, TDB);
END
```

```
SUBROUTINE 1 : summary(pi, cdb)

INPUT: (1) pi: a prefix itemset, and (2) cdb: the conditional database w.r.t. prefix pi.
BEGIN
04. I ← find_frequent_items(cdb,min_sup);
05. S ← item_merging(I);  pi ← pi ∪ S;  I ← I - S;
06. if(pi ≠ ∅)
07.     for all t_i ∈ cdb
08.         if(|SI_{t_i}| < |pi|)
09.             SI_{t_i} ← pi;
10. if(I ≠ ∅)
11.     if(conditional_database_pruning(I,pi,cdb))
12.         return;
13.     cdb ← conditional_transaction_pruning(I,pi,cdb);
14.     for all i∈I do
15.         pi' ← pi ∪ {i};
16.         cdb' ← build_cond_database(pi', cdb);
17.         call summary(pi', cdb');
END
```

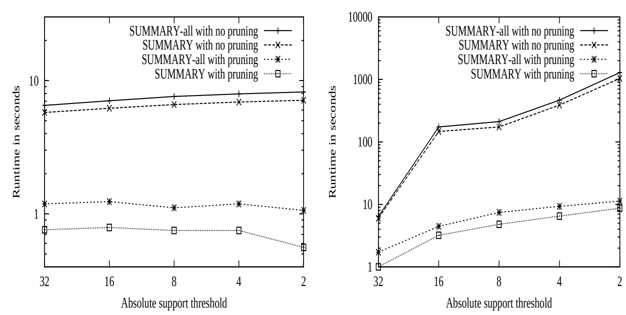
4.3 The Algorithm

By pushing deeply the search space pruning methods of Section 4.2 into the frequent itemset mining framework described in Section 4.1, we can mine the *summary set* as described in the SUMMARY algorithm shown in ALGORITHM 1. It first initializes the summary itemset to empty for each transaction (lines 01-02) and calls the SUBROUTINE 1 (i.e., *summary(∅, TDB)*) to mine the *summary set* (line 03). SUBROUTINE *summary(pi, cdb)* applies the search space pruning methods such as the *item_merging* (line 05), *conditional_database_pruning* (lines 11-12), and *conditional_transaction_pruning* (line 13), updates the *summary set* information for conditional database *cdb* w.r.t. prefix itemset *pi* (lines 06-09), and grows the current prefix, builds the new conditional database, and recursively calls itself under the projection-based frequent itemset mining framework (lines 14-17).

Discussion. A transaction may be covered by multiple summary itemsets. In this paper we mainly focus on the SUMMARY algorithm, which for each transaction, only inserts into the *summary set* the summary itemset that was discovered first. However, it is rather straightforward to revise SUMMARY to find all the summary itemsets supported by each transaction. Specifically, if we change the '$\leq$' to '$<$' in case 1 of Definition 4.2, all the '$>$' to '$\geq$' in case 2 of Definition 4.2, the '$\leq$' to '$<$' in Equation 1 of Lemma 4.3, and the '$<$' to '$\leq$' in line 08 of ALGORITHM 1, the revised SUMMARY algorithm will find all the summary itemsets. We denote the so-derived algorithm by SUMMARY-all.

5 Experimental Results

We have implemented both the SUMMARY and SUMMARY-all algorithms, and performed a thorough experimental study to evaluate the effectiveness of the pruning methods, their algorithmic efficiency, and their overall scalability. All the experiments except the efficiency test were performed on a 2.4GHz Intel PC with 1GB memory and Windows XP installed. In our experiments, we used some databases which were popularly used in evaluating various frequent itemset mining algorithms [34, 30, 16], such as *connect*, *chess*, *pumsb**, *mushroom*, and *gazelle*, and some categorical databases obtained from the UCI Machine Learning repository, such as *SPECT*, *Letter Recognition*, and so on.

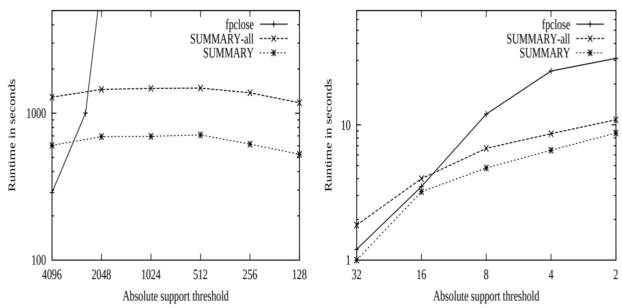


a) Database (*mushroom*) b) Database (*gazelle*)

Figure 1. Effectiveness of pruning methods

Effectiveness of the Pruning Methods. We first evaluated the effectiveness of the pruning methods by comparing SUMMARY and SUMMARY-all themselves with or without the *conditional database* and *transaction* pruning methods. Figure 1 shows that the algorithms with pruning can be over an order of magnitude faster than the corresponding algorithms without pruning for both *mushroom* and *gazelle* databases. This illustrates that the pruning methods newly proposed in this paper are very effective in reducing search space.

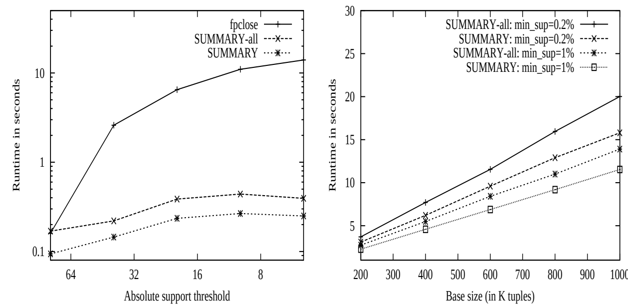
Efficiency. To mine the *summary set*, a naïve method is to first mine the complete set of frequent closed itemsets, from which the *summary set* can be further identified. Our comparison with fpclose [17], one of the most recently developed efficient closed itemset mining algorithms [16], shows that such a solution is not practical when the minimum support is low. As we will discuss in Section 6, such low minimum support values are beneficial for clustering applications. The efficiency comparison was performed on a 1.8GHz Linux machine with 1GB memory by varying the absolute support threshold and turning off the output of fpclose. The experiments for all the databases we used show consistent results. Due to limited space, we only report the results for dense database *connect*, sparse database *gazelle*, and categorical database *SPECT*.



a) Database (*connect*) b) Database (*gazelle*)

Figure 2. Efficiency test for *connect* and *gazelle*

Figure 2 shows the runtimes for databases *connect* and *gazelle*. It shows that both SUMMARY and SUMMARY-all scale very well w.r.t. the support threshold, and for *connect* database, they even run faster at low support value of 128 than at high support value of 512. This is because these two algorithms usually mine longer itemsets at lower support, which makes the pruning methods more effective in removing some short transactions and conditional databases. Because the FP-tree structure adopted by fpclose is very effective in condensing dense databases, at high support fpclose is much faster than SUMMARY and SUMMARY-all for dense databases like *connect*, but once we continue to lower the support, it can be orders of magnitude slower. While for sparse databases like *gazelle*, fpclose can be several times slower. The *SPECT* database is very small and only contains 267 instances (i.e., patient image sets) and 23 attributes per instance. Figure 3a shows that even for such a small database, both SUMMARY and SUMMARY-all can be over an order of magnitude faster than fpclose. In addition, the above results also demonstrate that SUMMARY always runs a little faster than SUMMARY-all, this is because SUMMARY-all mines more *summary itemsets* than SUMMARY. For example, at absolute minimum support threshold 32, on average SUMMARY-all finds 11.1 *summary itemsets* for each transaction of the dense database *mushroom*, and finds 1.3 *summary itemsets* for each transaction of the sparse database *gazelle*.



a) Efficiency(*SPECT*) b) Scalability(*T10I4Dx*)

Figure 3. Efficiency and scalability test

Scalability. We also tested the algorithm scalability using the IBM synthetic database series *T10I4Dx* by setting the average transaction length at 10 and changing the number of transactions from 200K to 1000K. We ran both SUMMARY and SUMMARY-all at two different minimum relative supports 0.2% and 1%. Figure 3b shows that these two algorithms scale very well against the database size.

6 Application - *Summary Set* based Clustering

One important application of the SUMMARY algorithm is to cluster the categorical data by treating each summary itemset as a cluster description and grouping the transactions with the same cluster description into a cluster. In SUMMARY, we adopt a prefix tree structure to facilitate this task, which has been used extensively in performing different data mining tasks [18, 30]. For each transaction, T_i, if its summary itemset SI_{T_i} is not empty, we sort the items in SI_{T_i} in lexicographic order and insert it into the prefix tree. The tree node corresponding to the last item of the sorted summary itemset represents a cluster, to which the transaction T_i belongs.

Example 6.1 The summary itemsets for the transactions in our running example are SI_{01}=*ace*, SI_{02}=*bde*, SI_{03}=*df*, SI_{04}=*ef*, SI_{05}=*ace*, SI_{06}=*bcd*, SI_{07}=*acf*, SI_{08}=*ef*, and SI_{09}=*bd*. If we insert these summary itemsets into the prefix tree in sequence, we can get seven clusters with cluster descriptions *ace*, *bde*, *df*,

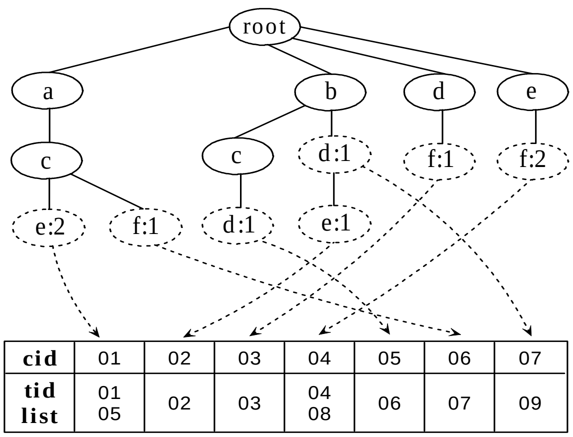


Figure 4. Clustering based on *summary set*

ef, bcd, acf, and bd, as shown in Figure 4. From Figure 4 we see that transactions 01 and 05 are grouped into cluster 01, transactions 04 and 08 are grouped into cluster 04, while each of the other transactions forms a separate cluster of their own. Note that a non-leaf node *summary itemset* in the prefix tree represents a non-maximal frequent itemset in the sense that one of its proper supersets must be frequent. For example, *summary itemset bd* is non-maximal, because *summary itemset bde* is a proper superset of *bd*. In this case, we have an alternative clustering option: merge the non-leaf node clusters with their corresponding leaf node clusters to form larger clusters. In Figure 4, we can merge cluster 07 with cluster 02 to form a cluster. □

Clustering Evaluation. We have used many categorical databases to evaluate the clustering quality of the SUMMARY algorithm, including *mushroom*, *SPECT*, *Letter Recognition*, and *Congressional Voting*, which all contain class labels and are available at *http://www.ics.uci.edu/~mlearn/*. We did not use the class labels in mining the *summary set* and clustering, instead, we only used them to evaluate the clustering accuracy, which is defined by the number of correctly clustered instances (i.e., the instances with dominant class labels in the computed clusters) as a percentage of the database size [3]. SUMMARY runs very fast and can achieve very good clustering accuracy for these databases, especially when the minimum support is low. Due to limited space, we only show results for *mushroom* and *Congressional Voting* databases, which have been widely used in the previous studies [14, 32, 33].

[3]Note we also used the average cluster purity and entropy to evaluate the clustering quality, and our results show that the clustering of SUMMARY has high purity and low entropy, which is consistent with the clustering accuracy measure. Due to limited space, we will not report them here.

The *mushroom* database contains some physical characteristics of various mushrooms. It has 8124 instances and two classes: poisonous or edible. Table 2 shows the clustering results for this database, including the *minimum support* used in the tests, the *number of clusters* found by SUMMARY, the *number of misclustered instances*, *clustering accuracy*, *compression ratio* and *runtime* (in seconds) for both the *summary set* discovery and clustering. The compression ratio is defined as the total number of items in the database divided by the total number of items in the *summary set*. We can see that SUMMARY has a clustering accuracy higher than 97% and a runtime less than 0.85 seconds for a wide range of support thresholds. At support 25, it can even achieve a 100% accuracy. The MineClus algorithm is one of the most recently developed clustering algorithm for this type of databases [33]. Its reported clustering solution for this database finds 20 clusters with an accuracy 96.41% and in the meantime declares 0.59% of the instances as outliers, which means it misclusters about 290 instances and treats about another 48 instances as outliers. Compared to this algorithm, SUMMARY is very competitive in considering both of its high efficiency and clustering accuracy. In addition, the high compression ratios demonstrate that the *summary set* can be used as a concise summary of the original database (Note in each case of Table 2, the *summary set* covers each instance of the original database, which means there is no outlier in our solution).

sup.	# clu.	# miscl.	accur.	com. rat.	time
1400	30	32	99.6	660	0.38s
1200	35	32	99.6	549	0.42s
1000	37	32	99.6	509	0.44s
800	63	208	97.4	268	0.48s
400	128	8	99.9	120	0.66s
200	140	6	99.93	97	0.77s
100	197	32	99.6	62	0.81s
50	298	1	99.99	37	0.79s
25	438	0	100	23	0.75s

Table 2. Clustering *mushroom* **database**

The *Congressional Voting* database contains the 1984 United States Congressional Voting Records and has two class labels: Republican and Democrat. In our experiments, we removed four outlier instances whose most attribute values are missing and used the left 431 instances. Table 3 shows the clustering solution of SUMMARY at a minimum support 245, at which

cid	# Rep.	# Demo.	cid	# Rep.	# Demo.
1	2	244	4	1	3
2	155	16	5	2	1
3	5	0	6	1	1

Table 3. Clustering *Congressional Voting* **database**

point the clusters produced by SUMMARY covers the whole database (while a minimum support higher than 245 will make SUMMARY miss some instances), and SUMMARY only uses 0.001 seconds to find the six clusters with an accuracy higher than 95% and a compression ratio higher than 1164. Even we simply merge the four small clusters with the two large clusters in order to get exact two clusters, the accuracy is still higher than 93% in the worst case (e.g., clusters 3 and 5 are merged into cluster 1, and clusters 4 and 6 are merged into cluster 2), and is much better than the reported accuracy, 86.67%, of the MineClus algorithm [33].

7 Discussions and Conclusion

In this paper we proposed to mine the *summary set* that can maximally cover the input database. Each *summary itemset* can be treated as a distinct cluster description and the transactions with the same description can be grouped together to form a cluster. Because the *summary itemset* of a cluster is one of the longest frequent itemsets that is common among the corresponding transactions of the same cluster, it can approximately maximize the intra-cluster similarity, while different clusters are dissimilar with each other because they support distinct *summary itemsets*. In addition, we require each *summary itemset* be frequent in order to make sure it is statistically significant.

Directly mining the *summary set* also enabled us to design an efficient algorithm, SUMMARY. By exploring some properties of the *summary set*, we developed two novel pruning methods, which significantly reduce the search space. Our performance study showed that SUMMARY runs very fast even when the minimum support is extremely low and the *summary set* is very effective in clustering categorical data. In addition, we also evaluated SUMMARY-all, a variant of SUMMARY, which mines all the summary itemsets for each transaction. In future, we plan to explore how to choose the one among the summary itemsets supported by a transaction which can reduce the number of clusters while achieving a high clustering accuracy.

References

[1] R. Agarwal, C. Aggarwal, V. Prasad. A Tree Projection Algorithm for Generation of Frequent Item Sets, J. Para. Distr. Comp. 61(3), 2001.

[2] R. Agrawal, T. Imielinski, A. Swami. *Mining Association Rules between Sets of Items in Large Databases*, SIGMOD'93.

[3] R. Agrawal, R. Srikant. *Fast Algorithms for Mining Association Rules*, VLDB'94.

[4] M. Antonie, O. Zaiane. *Text Document Categorization by Term Association*, ICDM'02.

[5] R.J. Bayardo. *Brute-force Mining of High-confidence Classification rules*, KDD'97.

[6] R.J. Bayardo. *Efficiently Mining Long Patterns from Databases*, SIGMOD'98.

[7] F. Beil, M. Ester, X. Xu. *Frequent Term-Based Text Clustering*, KDD'02.

[8] J. Boulicaut, A. Bykowski, C. Rigotti. *Free-Sets: A Condensed Representation of Boolean Data for the Approximation of Frequency Queries*, J. of Data Mining and Knowl. Discov. 7(1), 2003.

[9] S. Brin, R. Motwani, J.D. Ullman, S. Tsur. *Dynamic Itemset Counting and Implication Rules for Market Basket Data*, SIGMOD'97.

[10] D. Burdick, M. Calimlim, J. Gehrke. *MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases*, ICDE'01.

[11] B. Fung, K. Wang, M. Ester. *Hierachical Document Clustering Using Frequent Itemsets*, SDM'03.

[12] K. Gade, J. Wang, G. Karypis. *Efficient Closed Pattern Mining in the Presence of Tough Block Constraints*, to appear in KDD'04.

[13] V. Ganti, J. Gehrke, R. Ramakrishnan. *CACTUS: Clustering Categorical Data Using Summaries*, KDD'99.

[14] S. Guha, R. Rastogi, K. Shim. *ROCK: A Robust Clustering Algorithm for Categorical Attributes*, ICDE'99.

[15] D. Gunopulos, H. Mannila, S.Saluja. *Discovering All Most Specific Sentences by Randomized Algorithms*, ICDT'97.

[16] B. Goethals, M. Zaki. *An Introduction to FIMI'03 Workshop on Frequent Itemset Mining Implementations*, ICDM-FIMI'03.

[17] G. Grahne, J. Zhu. *Efficiently Using Prefix-trees in Mining Frequent Itemsets*, ICDM-FIMI'03.

[18] J. Han, J. Pei, Y. Yin. *Mining Frequent Patterns without Candidate Generation*, SIGMOD'00.

[19] W. Li, J. Han, J. Pei. *CMAR: Accurate and Efficient Classification based on multiple class-association rules*, ICDM'01.

[20] B. Liu, W. Hsu, Y. Ma. *Integrating Classification and Association Rule Mining*, KDD'98.

[21] G. Liu, H. Lu, W. Lou, J. X. Yu. *On Computing, Storing and Querying Frequent Patterns*, KDD'03.

[22] H. Mannila, H. Toivonen. *Multiple Uses of Frequent Sets and Condensed Representations*, KDD'96.

[23] F. Pan, G. Cong, A.K.H. Tung, J. Yang, M. Zaki. *CARPENTER: Finding Closed Patterns in Long Biological Datasets*, KDD'03.

[24] J. Park, M. Chen, P. S. Yu. *An Effective Hash Based Algorithm for Mining Association Rules*, SIGMOD'95.

[25] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal. *Discovering Frequent Closed Itemsets for Association Rules*, ICDT'99.

[26] J. Pei, et al. *H-Mine: Hyper-structure Mining of Frequent Patterns in Large Databases*, ICDM'01.

[27] J. Pei, J. Han, R. Mao. *CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets*, DMKD'00.

[28] M. Seno, G. Karypis. *LPMiner: An Algorithm for Finding Frequent Itemsets Using Length-Decreasing Support Constraint*, ICDM'01.

[29] H. Toivonen. *Sampling Large Databases for Association Rules*, VLDB'96.

[30] J. Wang, J. Han, J. Pei. *CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets*, KDD'03.

[31] J. Wang, G. Karypis. *BAMBOO: Accelerating Closed Itemset Mining by Deeply Pushing the Length-Decreasing Support Constraint*, SDM'04.

[32] K. Wang, C. Xu, B. Liu. *Clustering Transactions using Large Items*, CIKM'99.

[33] M. Yiu, N. Mamoulis. *Frequent-Pattern based Iterative Projected Clustering*, ICDM'03.

[34] M. Zaki, C. Hsiao. *CHARM: An Efficient Algorithm for Closed Itemset Mining*, SDM'02.