Knowl Inf Syst (2005) 00: 1–19
DOI 10.1007/s10115-005-0216-7

Knowledge and Information Systems

RESEARCH ARTICLE

Jianyong Wang · George Karypis

On efficiently summarizing categorical databases

Received: 1 June 2004 / Revised: 15 January 2005 / Accepted 30 January 2005 /
Published online: 10 May 2005

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 transaction or document 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 key 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 show 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.

Keywords Data mining · Frequent itemset · Categorical database · Clustering

1 Introduction

Frequent itemset mining was initially proposed and has been studied extensively in the context of association rule mining [2, 3, 9, 15, 18, 24, 29, 35]. In recent years,

J. Wang
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

G. Karypis (✉)
Department of Computer Science, Digital Technology Center and Army HPC Research Center, University of Minnesota, Minneapolis, MN 55455, USA
E-mail: karypis@cs.umn.edu

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 [8, 22], and the transactional data (or document) classification [4, 5, 19, 20] and clustering [7, 11, 32, 33, 34].

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. This set of mined frequent itemsets is called a *summary set*.

One significant advantage of directly mining the final subset of frequent itemsets is that it provides the possibility of designing a 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 mining 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 with respect to the database size. Moreover, 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. Sections 2 and 3 introduce the problem definition and some related work, respectively. Section 4 describes the algorithm in detail. Section 5 presents the empirical results. Section 6 shows an application of the algorithm in clustering 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 $\langle \text{tid}, X \rangle$, 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 $x_1 \ldots x_l$. A transaction $\langle \text{tid}, X \rangle$ 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 $|\text{TDB}|$ and $|Y|$ to denote the number of transactions in database TDB, and the number of items in itemset Y, respectively.

Table 1 A transaction database 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

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* with respect to (w.r.t.) the transactions in TDB (i.e., $\cup_{i=1}^{|\mathrm{TDB}|}\{\mathrm{SI}_{T_i}\}$) 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 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 and get the sorted item list which is called f_list. In this example f_list = $\langle a{:}3, b{:}4, c{:}4, d{:}5, e{:}5, f{:}5\rangle$. 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.

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 [1, 18] and adopts the natural matrix structure instead of the FP-tree to represent the (conditional) database [12, 26]. 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, 10, 21, 23, 25, 27, 30, 35] 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

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

the unpromising transactions and prefixes. However, as 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 numerical data [7, 11, 34]. 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* and present the integrated SUMMARY algorithm. Finally we discuss the local item ordering schemes and how to revise SUMMARY to mine K *summary itemsets* for each transaction.

4.1 Frequent itemset enumeration

Like most of the 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 1, SUMMARY first mines all the frequent itemsets containing item a, then mines all frequent itemsets containing b but no $a, \ldots$, 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 $\text{TDB}|_a = \{\langle 01, ec\rangle, \langle 05, efc\rangle, \langle 07, fc\rangle\}$ (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 $\text{TDB}|_a$, 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 tids 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, by maintaining the length of the current longest covering itemset for each transaction during the mining process, we can employ additional *branch-and-bound* techniques to further prune the overall search space.

Definition 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 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 $\text{SI}_{T_i} \subset Y$ and $\sup(\text{SI}_{T_i}) = \sup(Y)$. Thus, Y is also supported by transaction T_i and is frequent. However, $|Y| > |\text{SI}_{T_i}|$ contradicts with the fact that SI_{T_i} is the summary itemset of transaction T_i. □

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

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 $\text{TDB}|_P = \{\langle T_{P_1}, X_{P_1}\rangle, \langle T_{P_2}, X_{P_2}\rangle, \ldots, \langle T_{P_{\sup(P)}}, X_{P_{\sup(P)}}\rangle\}$ its conditional database. Note that some (or all) of the transactions X_{P_i} $(1 \leq i \leq \sup(P))$ can be empty.

Definition 2 (*Invalid conditional transaction*) A conditional transaction T_{P_i} in $\text{TDB}|_P$ (where $1 \leq i \leq \sup(P)$), is an **invalid** conditional transaction if it falls into one of the following two cases:

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

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 $\text{LCF}_{T_{P_i}}$ by extending prefix P is smaller than the minimum support.

Lemma 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* $\text{LCF}_{T_{P_i}}$.

Proof Follows directly from Definition 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 $\text{LCF}_{T_{P_i}}$ and frequent. □

Note that 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 3 (*Conditional transaction pruning*) *An invalid conditional transaction, T_{P_i}, can be safely pruned, if it satisfies:*

$$|X_{P_i}| \leq \min_{\forall j,\ T_{P_j} \text{ is valid}} \left(\left|\text{LCF}_{T_{P_j}}\right| - |P|\right). \tag{1}$$

Proof Consider an invalid conditional transaction T_{P_i} that satisfies Eq. (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 3 can be used to prune from the conditional database some unpromising transactions satisfying Eq. (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 (*Conditional database pruning*) *Given the current prefix itemset P and its projected conditional database* $\text{TDB}|_P$*, if each of its conditional transactions, T_{P_i}, is invalid,* $\text{TDB}|_P$ *can be safely pruned.*

Proof According to Lemma 2, for any invalid conditional transaction, T_{P_i}, we cannot generate any frequent itemsets longer than $\text{LCF}_{T_{P_i}}$ 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, $\cup_{i=1}^{\text{sup}(P)}\{\text{LCF}_{T_{P_i}}\}$, by extending P; thus, $\text{TDB}|_P$ can be safely pruned. □

Example 3 Assume the prefix is c:4 (i.e., P=c). From Table 1 we get that $\text{TDB}|_c$={⟨01, e⟩, ⟨05, def⟩, ⟨06, d⟩, ⟨07, f⟩}, and LCF_{01}=ace:2, LCF_{05}=ace:2, LCF_{06}=bcd:2, and LCF_{07}=acf:2. Conditional transactions ⟨01, e⟩, ⟨06, d⟩, and ⟨07, f⟩ fall into case 1 of Definition 2, while ⟨05, def⟩ falls into case 2 of Definition 2; thus, all the conditional transactions in $\text{TDB}|_c$ are invalid. According to Lemma 4, conditional database $\text{TDB}|_c$ can be pruned.

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*.

```
01. for all t_i ∈ TDB
02.   SI_{t_i} ← ∅;
03. call summary(∅, TDB);
```

SUBROUTINE 1: **SUMMARY**(*pi*, *cdb*)

INPUT: (1) *pi*: *a prefix itemset*, and (2) *cdb*: *the conditional database w.r.t. prefix pi*.

```
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');
```

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*($\emptyset$, TDB)) to mine the *summary set* (line 03). Subroutine *summary(pi, cdb)* finds the set of local frequent items by scanning conditional database *cdb* once (line 04) and 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).

4.4 Discussions and extensions

4.4.1 Ordering of local items

In some FP-tree based frequent itemset mining algorithms [18, 30], the support descending ordering scheme is popularly used to sort the local items w.r.t. a prefix, while as stated in Section 4.1, SUMMARY adopts the support ascending ordering scheme. We make this decision due to the following considerations. First, the support descending ordering generally helps in generating more compact FP-tree structures; thus, leads to more efficient memory usage and support counting. However, SUMMARY does not use the FP-tree structure to represent the conditional database; thus, it cannot get the benefit from the support descending ordering as the FP-tree based algorithms usually do. Second, a transaction may support multiple summary itemsets. By adopting the support ascending ordering scheme, SUMMARY prefers the summary itemset whose constituent items have relatively lower support. This heuristic is very important for the purpose of clustering due to the fact that some items with very high support (e.g., as high as the number of transactions in the database) are not differentiable in terms of different classes. We will justify this heuristic with our experiments on some real datasets.

4.4.2 Mining K longest itemsets w.r.t. each transaction

As mentioned above, a transaction may be covered by multiple summary itemsets in many cases. The SUMMARY algorithm described in Algorithm 1, only inserts for each transaction into the *summary set* one summary itemset, i.e., the one that was discovered first. We can revise SUMMARY to find K summary itemsets for each transaction which supports no less than K summary itemsets,[2] where K is a user input parameter. We denote the so-derived algorithm by SUMMARY-K.

[2] If a transaction supports less than K summary itemsets, all its summary itemsets will be mined.

The SUMMARY-K algorithm is very similar to the SUMMARY algorithm. To avoid the unnecessary repetition, here we mainly describe their major difference, that is, how to adapt Definition 2 and Lemma 3 to mine K summary itemsets, and will leave other minor revisions to the interested readers.

Invalid conditional transaction for mining K summary itemsets. Similar to the notations used in Definition 2, let P be the prefix itemset that is currently under consideration, $\sup(P)$ its support, and $\text{TDB}|_P = \{\langle T_{P_1}, X_{P_1}\rangle, \langle T_{P_2}, X_{P_2}\rangle, \ldots, \langle T_{P_{\sup(P)}}, X_{P_{\sup(P)}}\rangle\}$ its conditional database. In addition, we use $K_{T_{P_i}}$ to denote the number of the currently longest itemsets maintained by algorithm SUMMARY-K w.r.t. a conditional transaction T_{P_i}.

Definition 3 (*Invalid conditional transaction for mining K summary itemsets*) A conditional transaction T_{P_i} in $\text{TDB}|_P$ (where $1 \leq i \leq \sup(P)$), is an *invalid* conditional transaction if it falls into one of the following two cases:

1. $|X_{P_i}| < (|\text{LCF}_{T_{P_i}}| - |P|)$ or $|X_{P_i}| = (|\text{LCF}_{T_{P_i}}| - |P|)$ but $K_{T_{P_i}} \geq K$;
2. $|X_{P_i}| > (|\text{LCF}_{T_{P_i}}| - |P|)$ or $|X_{P_i}| = (|\text{LCF}_{T_{P_i}}| - |P|)$ and $K_{T_{P_i}} < K$ but $|\{\, \forall j \in [1..\sup(P)], T_{P_j} || X_{P_j}| > (|\text{LCF}_{T_{P_i}}| - |P|)\,\}| < \text{min\_sup}$.

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

Conditional transaction pruning for mining K summary itemsets. Similar to SUMMARY algorithm, an invalid conditional transaction can be used to mine summary itemsets for other valid conditional transactions and thus cannot be simply pruned. As a result, we need to design new conditions in order to safely prune some invalid conditional transactions.

Lemma 5 (*Conditional transaction pruning for mining K summary itemsets*) *An invalid conditional transaction, T_{P_i}, can be safely pruned, if it satisfies:*

$$\left|X_{P_i}\right| < \min_{\forall j,\ T_{P_j} \text{ is valid}} \left(\left|\text{LCF}_{T_{P_j}}\right| - |P|\right). \tag{2}$$

Proof Similar to the proof of Lemma 3. □

5 Experimental results

In this section, we will first present a thorough experimental study to evaluate the effectiveness of the pruning methods, the overall scalability, and the efficiency of the SUMMARY algorithm. Then we will evaluate the SUMMARY-K algorithm by varying the K parameter. All the experiments except the efficiency test were performed on a 2.4 GHz Intel PC with 1 GB memory and Windows XP installed. In our experiments, we used some databases which were popularly used in evaluating various frequent itemset mining algorithms [16, 30, 35], 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.

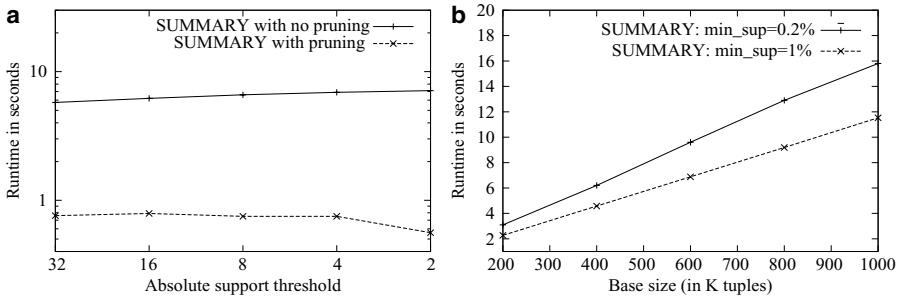

Fig. 1 Effectiveness of the pruning methods and the scalability test. **a** Database (*mushroom*). **b** Scalability (*T10I4Dx*)

5.1 Effectiveness of the pruning methods

We first evaluated the effectiveness of the pruning methods by comparing SUMMARY itself with or without the *conditional database* and *transaction* pruning methods. Figure 1a shows that the SUMMARY algorithm with pruning can be over an order of magnitude faster than the one without pruning for the *mushroom* database. This illustrates that the pruning methods proposed in this paper are very effective in reducing search space.

5.2 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 200 K to 1000 K. We ran SUMMARY at two different minimum relative supports of 0.2% and 1%. Figure 1b shows that SUMMARY scales very well against the database size.

5.3 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 1.8 GHz Linux machine with 1 GB 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 databases *connect* and *gazelle*.

Figure 2 shows the runtime for databases *connect* and *gazelle*. It shows that SUMMARY scales very well w.r.t. the support threshold, and for *connect*

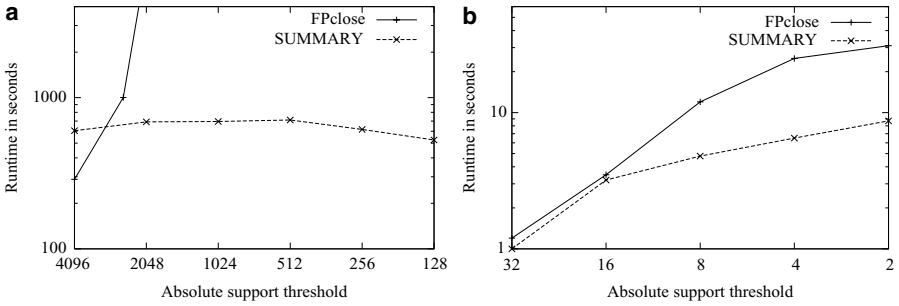

Fig. 2 Efficiency test for *connect* and *gazelle*. **a** Database (*connect*). **b** Database (*gazelle*)

database, it even runs faster at low support value of 128 than at high support value of 512. This is because SUMMARY usually mines longer itemsets at lower support, which makes the pruning methods more effective in removing some short transactions and conditional databases. As the FP-tree structure adopted by FPclose is very effective in condensing dense databases, at high support, FPclose is faster than SUMMARY 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.

5.4 Test of SUMMARY-K algorithm

We also evaluated the efficiency of the SUMMARY-K algorithm by varying the value of the K parameter from 1 to 8 against FPclose (When K equals 1, SUMMARY-K is the same as SUMMARY). Figure 3 shows the comparison result for database *spect*. From Fig. 3a we can see that when K is larger, SUMMARY-K is a little slower, but it is always faster than FPclose. Figure 3b compares the number of patterns mined by SUMMARY-K and FPclose, which shows that FPclose mines orders of magnitude more patterns than SUMMARY-K. The high efficiency of the SUMMARY-K algorithm illustrates the effectiveness of the pruning methods newly proposed in this paper from another angle.

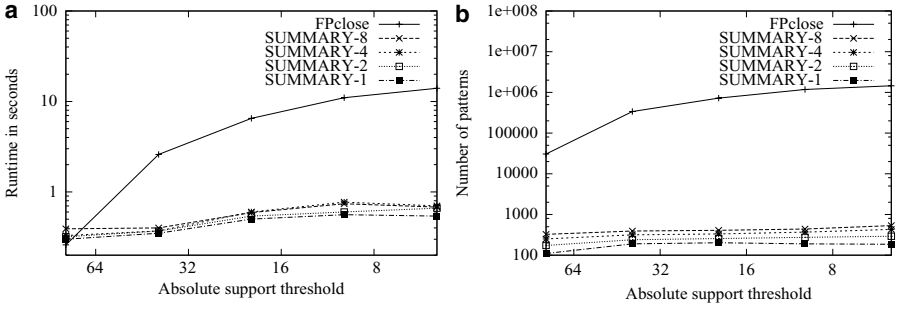

Fig. 3 Test for SUMMARY-K on database *spect*. **a** Runtime. **b** Number of patterns

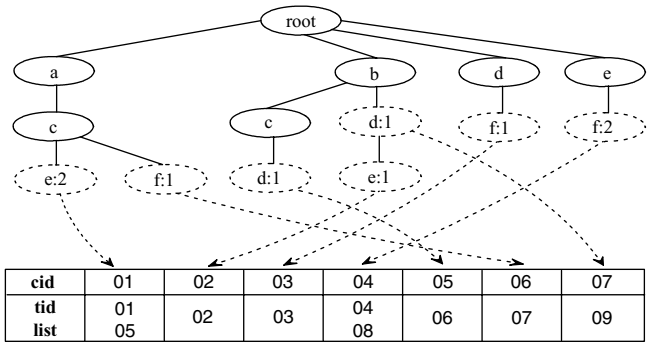

Fig. 4 Clustering based on *summary set*

6 Application—*summary set* based clustering

6.1 Clustering based on summary set

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 4 The summary itemsets for the transactions in our running example are $\mathrm{SI}_{01} = ace$, $\mathrm{SI}_{02} = bde$, $\mathrm{SI}_{03} = df$, $\mathrm{SI}_{04} = ef$, $\mathrm{SI}_{05} = ace$, $\mathrm{SI}_{06} = bcd$, $\mathrm{SI}_{07} = acf$, $\mathrm{SI}_{08} = ef$, and $\mathrm{SI}_{09} = bd$. If we insert these summary itemsets into the prefix tree in sequence, we can get seven clusters with cluster descriptions *ace*, *bde*, *df*, *ef*, *bcd*, *acf*, and *bd*, as shown in Fig. 4. From Fig. 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 Fig. 4, we can merge cluster 07 with cluster 02 to form a cluster.

6.2 Clustering evaluation

We have used several 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/~xmlearn/. 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. 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, 34].

The *mushroom* database contains some physical characteristics of various mushrooms. It has 8124 instances and two classes: poisonous and 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 of 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 [34]. 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).

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 of 245, at which point the clusters produced by SUMMARY covers the entire database (while a minimum support higher than 245

Table 2 Clustering *mushroom* database

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 3 Clustering *congressional voting* database

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

will make SUMMARY miss some instances), and SUMMARY only uses 0.001 s 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 [34].

6.3 Comparison with ROCK

ROCK is one of the most well-known categorical clustering algorithms [14]. Following we will compare SUMMARY with ROCK. For *mushroom* database, ROCK generates 21 clusters as shown in Table 4. In the table we also list the 21 largest clusters generated by SUMMARY at absolute support 1400. From Table 4, we see that although SUMMARY and ROCK are two different clustering algorithms, their clustering solutions are similar in many aspects. For example, each

Table 4 SUMMARY vs. ROCK (*mushroom* database)

ROCK			SUMMARY		
cid	# Edi.	# Pois.	cid	# Edi.	# Pois.
1	1728	0	1	1728	0
2	0	1728	2	0	1728
3	0	1296	3	0	1296
4	768	0	4	704	0
5	704	0	5	0	576
6	0	288	6	256	0
7	288	0	7	0	192
8	0	256	8	0	192
9	192	0	9	0	192
10	0	192	10	0	144
11	192	0	11	0	144
12	96	0	12	96	0
13	96	0	13	96	0
14	48	0	14	0	96
15	48	0	15	0	96
16	0	36	16	72	0
17	0	32	17	72	0
18	16	0	18	72	0
19	0	8	19	72	0
20	0	8	20	0	48
21	32	72	21	72	32

Table 5 SUMMARY vs. ROCK (*congressional voting* database)

ROCK			SUMMARY		
cid	# Rep.	# Demo.	cid	# Rep.	# Demo.
1	5	201	1	2	244
2	144	22	2	155	16

of these two algorithms only generates one impure cluster (i.e., the 21st cluster), and their top 3 largest clusters (i.e., the first, the second, and the third one) have the same size and class distribution.

Table 5 shows the comparison result for *Congressional Voting* database. Similarly, as ROCK generates exact two clusters, here we only list the two largest clusters for SUMMARY at support 245. We see that the two clusters generated by SUMMARY contains more tuples but they are purer than the corresponding ones of ROCK. As a result, SUMMARY has better clustering solution than ROCK for this database.

6.4 Evaluation of item ordering scheme

In Section 4.4.1 we analyzed why SUMMARY adopts the support ascending ordering scheme instead of the support descending ordering for sorting the local items. Our experiments demonstrate that the support ascending ordering scheme usually leads to better clustering solution than the support descending ordering. Table 6 shows the comparison result for *Congressional Voting* database at absolute support of 245. The two different ordering schemes generate the same number of clusters, but it is very clear that the clustering computed with the support ascending ordering scheme has better quality.

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

Table 6 Ascending ordering vs. descending ordering (*congressional voting* database)

SUMMARY					
Support-Ascending			Support-Descending		
cid	# Rep.	#Demo.	cid	# Rep.	# Demo.
1	2	244	1	149	123
2	155	16	2	14	97
3	5	0	3	2	27
4	1	3	4	0	16
5	2	1	5	0	2
6	1	1	6	1	0

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-K, a variant of SUMMARY, which mines K 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.

Acknowledgements 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. Part of this work was done while Jianyong Wang was doing research in University of Minnesota and this paper is a major-value added version of a conference paper that appeared in the 2004 IEEE International Conference on Data Mining (ICDM'04).

References

1. Agarwal, R., Aggarwal, C., Prasad, V.: A tree projection algorithm for generation of frequent item sets. J Parallel Distrib Comput **61**(3), 350–371 (2001)
2. Agrawal, R., Imielinski, T., Swami, A.: Mining associations between sets of items in massive databases. In: Buneman, P., Jajodia, S. (eds.) Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 207–216. Washington DC (1993)
3. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Bocca, J.B., Jarke, M., Zaniolo, C. (eds.) Proceedings of 20th International Conference on Very Large Data Bases, pp. 487–499. Santiago de Chile, Chile (1994)
4. Antonie, M., Zaiane, O.: Text document categorization by term association. In: Proceedings of the 2002 IEEE International Conference on Data Mining, pp. 19–26. Maebashi City, Japan (2002)
5. Bayardo, R.J.: Brute-force mining of high-confidence classification rules. In: Heckerman, D., Mannila, H., Pregibon, D. (eds.) Proceedings of the 3rd International Conference on Knowledge Discovery and Data mining), pp. 123–126. Newport Beach, California, USA (1997)
6. Bayardo, R.J.: Efficiently mining long patterns from databases. In: Haas, L.M., Tiwary, A. (eds.) Proceedings ACM SIGMOD International Conference on Management of Data, pp. 85–93. Seattle, Washington (1998)
7. Beil, F., Ester, M., Xu, X.: Frequent term-based text clustering. In: KDD'02 Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 436–442. Edmonton, Alberta, Canada (2002)

8. Boulicaut, J., Bykowski, A., Rigotti, C.: Free-sets: a condensed representation of Boolean data for the approximation of frequency queries. J Data Mining Knowl Discovery **7**(1), 5–22 (2003)
9. Brin, S., Motwani, R., Ullman, J.D., Tsur, S.: Dynamic itemset counting and implication rules for market basket data. In: Peckham, J. (ed.) Proceedings ACM SIGMOD International Conference on Management of Data, pp. 255–264. Tucson, Arizona, USA (1997)
10. Burdick, D., Calimlim, M., Gehrke, J.: MAFIA: a maximal frequent itemset algorithm for transactional databases. In: Proceedings of the 17th International Conference on Data Engineering, pp. 443–452. Heidelberg, Germany (2001)
11. Fung, B., Wang, K., Ester, M.: Hierachical document clustering using frequent itemsets. In: Barbara, D., Kamath, C. (eds.) Proceedings of the 3rd SIAM International Conference on Data Mining. USA, San Francisco, CA (2003)
12. Gade, K., Wang, J., Karypis, G.: Efficient closed pattern mining in the presence of tough block constraints. In: Kim, W., Kohavi, R., Gehrke, J., DuMouchel, W. (eds.) Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, pp. 138–147. Washington, USA (2004)
13. Ganti, V., Gehrke, J., Ramakrishnan, R.: CACTUS: clustering categorical data using Summaries. In: Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 73–83. San Diego, CA, USA (1999)
14. Guha, S., Rastogi, R., Shim, K.: ROCK: a robut clustering algorithm for categorical attributes. In: Proceedings of the 15th International Conference on Data Engineering, pp. 512–521. Sydney, Austrialia (1999)
15. Gunopulos, D., Mannila, H., Saluja, S.: Discovering all most specific sentences by randomized algorithms. In: Afrati, F.N., Kolaitis, P.G. (eds.) Proceedings of the 6th International Conference on Database Theory, pp. 215–229. Delphi, Greece (1997)
16. Goethals, B., Zaki, M.: Advances in frequent itemset mining implementations: Introduction to FIMI03. In: Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations. Melbourne, Florida, USA (2003)
17. Grahne, G., Zhu, J.: Efficiently using prefix-trees in mining frequent itemsets. In: Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations. Melbourne, Florida, USA (2003)
18. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Chen, W., Naughton, J.F., Bernstein, P.A. (eds.) Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pp. 1–12. Dallas, Texas, USA (2000)
19. Li, W., Han, J., Pei, J.: CMAR: accurate and efficient classification based on multiple class-association rules. In: Cercone, N., Lin, T.Y., Wu, X. (eds.) Proceedings of the 2001 IEEE International Conference on Data Mining, pp. 369–376. San Jose, California, USA (2001)
20. Liu, B., Hsu, W., Ma, Y.: Integrating classification and association rule mining. In: Agrawal, R., Stolorz, P.E., Piatetsky-Shapiro, G. (eds.) Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining, pp. 80–86. New York City, New York, USA (1998)
21. Liu, G., Lu, H., Lou, W., Yu, J.X.: On computing, storing and querying frequent patterns. In: Getoor, L., Senator, T.E., Domingos, P., Faloutsos, C. (eds.) Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 607–612. Washington, District of Columbia, USA (2003)
22. Mannila, H., Toivonen, H.: Multiple uses of frequent sets and condensed representations. In: Simoudis, E., Han, J., Fayyad, U.M. (eds.) Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, pp. 189–194. Portland, Oregon, USA (1996)
23. Pan, F., Cong, G., Tung, A.K.H., Yang, J., Zaki, M.: CARPENTER: finding closed patterns in long biological datasets. In: Getoor, L., Senator, T.E., Domingos, P., Faloutsos, C. (eds.) Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 637–642. Washington, District of Columbia, USA (2003)
24. Park, J., Chen, M., Yu, P.S.: An effective hash based algorithm for mining association rules. In: Carey, M.J., Schneider, D.A. (eds.) Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, pp. 175–186. San Jose, California (1995)
25. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. In: Beeri, C., Buneman, P. (eds.) Proceedings of the 6th International Conference on Database Theory, pp. 398–416. Jerusalem, Israel (1999)

26. Pei, J., Han, J., Lu, H., Nishio, S., Tang, S., Yang, D.: H-mine: hyper-structure mining of frequent patterns in large databases. In: Cercone, N., Lin, T.Y., Wu, X. (eds.) Proceedings of the 2001 IEEE International Conference on Data Mining, pp. 441–448. San Jose, California, USA (2001)
27. Pei, J., Han, J., Mao, R.: CLOSET: an efficient algorithm for mining frequent closed itemsets. In: Gunopulos, D., Rastogi, R. (eds.) Proceedings of 2000 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 21–30. Dallas, Texas, USA (2000)
28. Seno, M., Karypis, G.: LPMiner: an algorithm for finding frequent itemsets using length-decreasing support constraint. In: Cercone, N., Lin, T.Y., Wu, X. (eds.) Proceedings of the 2001 IEEE International Conference on Data Mining, pp 505–512. San Jose, California, USA (2001)
29. Toivonen, H.: Sampling large databases for association rules. In: Vijayaraman, T.M., Buchmann, A.P., Mohan, C., Sarda, N.L. (eds.) Proceedings of 22th International Conference on Very Large Data Bases, pp 134–145. Mumbai, India (1996)
30. Wang, J., Han, J., Pei, J.: CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In: Getoor, L., Senator, T.E., Domingos, P., Faloutsos, C. (eds.) Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 236–245. Washington, District of Columbia, USA (2003)
31. Wang, J., Karypis, G.: BAMBOO: Accelerating closed itemset mining by deeply pushing the length-decreasing support constraint. In: Proceedings of the 4th SIAM International Conference on Data Mining. Lake Buena Vista, Florida, USA (2004)
32. Wang, K., Xu, C., Liu, B.: Clustering transactions using large items. In: Proceedings of the 1999 ACM CIKM International Conference on Information and Knowledge Management, pp 483–490. Kansas City, Missouri, USA (1999)
33. Xiong, H., Steinbach, M., Tan, P., Kumar, V.: HICAP:Hierarchial clustering with pattern preservation. In: Proceedings of the 4th SIAM International Conference on Data Mining. Lake Buena Vista, Florida, USA (2004)
34. Yiu, M., Mamoulis, N.: Frequent pattern based iterative projected clustering. In: Proceedings of the 3rd IEEE International Conference on Data Mining, pp. 689–692. Melbourne, Florida, USA (2003)
35. Zaki, M., Hsiao, C.: CHARM: an Efficient algorithm for closed itemset mining. In: Grossman, R.L., Han, J., Kumar, V., Mannila, H., Motwani, R. (eds.) Proceedings of the 4th SIAM International Conference on Data Mining. Arlington, VA, USA (2002)

Jianyong Wang received the Ph.D. degree in computer science in 1999 from the Institute of Computing Technology, the Chinese Academy of Sciences. Since then, he ever worked as an assistant professor in the Department of Computer Science and Technology, Peking (Beijing) University in the areas of distributed systems and Web search engines, and visited the School of Computing Science at Simon Fraser University, the Department of Computer Science at the University of Illinois at Urbana-Champaign, and the Digital Technology Center and the Department of Computer Science at the University of Minnesota, mainly working in the area of data mining. He is currently an associate professor of the Department of Computer Science and Technology at Tsinghua University, P.R. China.

George Karypis received his Ph.D. degree in computer science at the University of Minnesota and he is currently an associate professor at the Department of Computer Science and Engineering at the University of Minnesota. His research interests spans the areas of parallel algorithm design, data mining, bioinformatics, information retrieval, applications of parallel processing in scientific computing and optimization, sparse matrix computations, parallel preconditioners, and parallel programming languages and libraries. His research has resulted in the development of software libraries for serial and parallel graph partitioning (METIS and ParMETIS), hypergraph partitioning (hMETIS), for parallel Cholesky factorization (PSPASES), for collaborative filtering-based recommendation algorithms (SUGGEST), clustering high dimensional datasets (CLUTO), and finding frequent patterns in diverse datasets (PAFI). He has coauthored over ninety journal and conference papers on these topics and a book title "Introduction to Parallel Computing" (Publ. Addison Wesley, 2003, 2nd edition). In addition, he is serving on the program committees of many conferences and workshops on these topics and is an associate editor of the IEEE Transactions on Parallel and Distributed Systems.