Itemset Support Queries using Frequent Itemsets and Their Condensed Representations

Taneli Mielikäinen[1], Panče Panov[2], and Sašo Džeroski[2]

[1] HIIT BRU, Department of Computer Science, University of Helsinki, Finland
[2] Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia

Abstract. The purpose of this paper is two-fold: First, we give efficient algorithms for answering itemset support queries for collections of itemsets from various representations of the frequency information. As index structures we use itemset tries of transaction databases, frequent itemsets and their condensed representations. Second, we evaluate the usefulness of condensed representations of frequent itemsets to answer itemset support queries using the proposed query algorithms and index structures. We study analytically the worst-case time complexities of querying condensed representations and evaluate experimentally the query efficiency with random itemset queries to several benchmark transaction databases.

1 Introduction

Discovery of frequent itemsets aims to find all itemsets occurring sufficiently many times in a given transaction database [1, 2]. An example of a transaction database is a collection of documents represented as sets of words occurring in them. There, an itemset is a set of words and the frequency of an itemset is the fraction of the documents containing all the words in the itemset. Some other examples of frequent itemsets include sets of products often bought together from a supermarket and sets of web pages often referred together by other web pages. A variety of methods enabling the discovery of large numbers of frequent itemsets have been developed; see [3, 4] for a representative collection of the state of the art methods for frequent itemset mining.

Frequent itemsets can be used to summarize data directly or as an intermediate step to construct association rules [1, 5]. Frequent itemset collections describing the data in detail tend to be very large and hence they are rather problematic as summaries of the data. Due to such problems major efforts have been done to exclude redundant itemsets from the output, i.e., to obtain a condensed representation of frequent itemsets; see [6, 7].

Condensed representations of frequent itemsets are considered also as promising building blocks in inductive databases [8–11]. One important issue in (inductive) databases is query answering. An inductive database should be able to answer efficiently data mining queries, for example itemset support queries. An itemset support query asks how large fraction of transactions in the given transaction database contain a given itemset.

In this paper we study the task of answering itemset support queries and examine how the condensed representations of frequent itemsets could be used for such queries. Efficient answering of itemset support queries can be used in many data mining algorithms such as rule induction [12], decision tree construction [13] and learning Naïve Bayes classifiers [14], see [15]. Also, they have a close connection to statistical queries studied in learning theory: an infrastructure for answering itemset support queries can be considered as an efficient implementation of the statistical query oracle for the concept class of conjunctions of positive literals [16]. Furthermore, assessing the applicability of different condensed representations for itemset support queries would be helpful also in focusing the research and use of different condensed representations.

Related work. There has been some work on approximate frequency estimates for and using frequent itemsets. In [17] frequent itemsets are used to induce probabilistic models describing the joint probability distributions of binary datasets. [5, 18] study the approximations of the frequencies of boolean formulas in transaction data. Techniques for obtaining frequency approximations based on random subsets of transactions are described in [19, 20]. Also the problem of determining exact frequencies of itemsets have been studied. [21] gives algorithms for answering several types queries to a transaction database represented as a tree in a main memory. [22] proposes FP-trees that are trie structures for transaction data developed for frequent itemset mining. A data structure called AD-tree is proposed in [23]. AD-tree represents for all attribute-value combinations occurring in the projections of a relational database their counts in a trie. [24] studies the use of automata to represent itemset collections and their condensed representations. Protocols for private itemset support queries are given in [25]. To the best of our knowledge this is the first systematic study of using condensed representations of frequent itemsets to facilitate the answering of itemset support queries.

Roadmap. Section 2 gives the central concepts in frequent itemset mining. Section 3 defines the itemset support query problem, and describes data structures and algorithms for answering itemset support queries using different condensed representations. The proposed approaches for itemset support query answering are evaluated experimentally in Section 4. Section 5 concludes the paper.

2 Preliminaries

In this section we briefly define the central concepts in frequent itemset mining and their condensed representations. See [2–4, 6, 7] for more details.

Two most important ingredients in frequent itemset mining are transaction databases and itemsets. Transaction databases comprise the data. Itemsets are in dual role as building blocks of transaction databases and representation of the discovered knowledge:

Definition 1 (Transaction databases and itemsets). *A transaction database $\mathcal{D}$ is a set of transactions. A transaction t is a pair $\langle i, X\rangle$ consisting of a transaction identifier $tid(t) = i$ and an itemset $is(t) = X$. A transaction identifier is a natural number occurring in at most one transaction in a transaction database. An itemset X is a finite subset of $\mathcal{I}$, the set of possible items. The set $\mathcal{D}_u$ consists of all different itemsets in $\mathcal{D}$, i.e., $\mathcal{D}_u = \{X : \langle i, X\rangle \in \mathcal{D}\}$.*

Example 1. $\mathcal{D} = \{\langle 1, \{1,2,3,4\}\rangle, \langle 2, \{3,4,5,6\}\rangle, \langle 3, \{1,2,3,4,5,6\}\rangle\}$ is an example of a transaction database.

As examples of itemsets, the itemsets occurring in that database, i.e., the sets in $\mathcal{D}_u$ are $\{1,2,3,4\}$, $\{3,4,5,6\}$, and $\{1,2,3,4,5,6\}$.

Also certain statistics computed from the transaction database to itemsets are central for frequent itemset mining:

Definition 2 (Occurrences, counts, covers, supports and frequencies). *The occurrence set of an itemset X in a transaction database $\mathcal{D}$ is $occ(X, \mathcal{D}) = \{i : \langle i, Y\rangle \in \mathcal{D}, X = Y\}$. The count of X in $\mathcal{D}$ is $count(X, \mathcal{D}) = |occ(X, \mathcal{D})|$. The cover of X in $\mathcal{D}$ is $cover(X, \mathcal{D}) = \{i : \langle i, Y\rangle, X \subseteq Y\}$, the support of X in $\mathcal{D}$ is $supp(X, \mathcal{D}) = |cover(X, \mathcal{D})|$ and the frequency of X in $\mathcal{D}$ is $fr(X, \mathcal{D}) = supp(X, \mathcal{D})/|\mathcal{D}|$.*

Using the concepts defined above we can define frequent itemsets:

Definition 3 (Frequent itemsets). *An itemset X is σ-frequent in a transaction database $\mathcal{D}$ if $fr(X, \mathcal{D}) \geq \sigma$. The collection of all σ-frequent itemsets in $\mathcal{D}$ is $\mathcal{F}(\sigma, \mathcal{D})$.*

Frequent itemset mining has been a central issue in data mining since its introduction almost 15 years ago [1], many efficient algorithms for frequent itemset mining have been proposed [2–4], and similar techniques have been applied to a variety of other types of patterns than itemsets and quality functions than support.

The frequent itemset collections representing the data well are often very large—even larger than the underlying transaction database—and contain much redundant information. To cope with this problem many techniques for removing redundant itemsets from the frequent itemset collection have been proposed, see [6, 7].

For example, in many transaction databases there are several frequent itemsets with exactly the same cover. This can be used to partition the collection of frequent itemsets to equivalence classes. From each equivalence class we can select, e.g., all maximal or minimal itemsets (called closed and free itemsets):

Definition 4 (Closed and free frequent itemsets). *$X \in \mathcal{F}(\sigma, \mathcal{D})$ is closed σ-frequent itemset in $\mathcal{D}$, if $fr(X, \mathcal{D}) > fr(Y, \mathcal{D})$ for all $Y \supsetneq X$. The collection of closed σ-frequent itemsets is $\mathcal{C}(\sigma, \mathcal{D})$.*

$X \in \mathcal{F}(\sigma, \mathcal{D})$ is free σ-frequent itemset in $\mathcal{D}$, if $fr(X, \mathcal{D}) < fr(Y, \mathcal{D})$ for all $Y \subsetneq X$. The collection of free σ-frequent itemsets is $\mathcal{G}(\sigma, \mathcal{D})$. Free itemsets are known also as generators.

Example 2. The frequent itemsets in the transaction database given in Example 1 with minimum frequency threshold 2/3 are the itemsets in $2^{\{1,2,3,4\}} \cup 2^{\{3,4,5,6\}}$, in total 27 itemsets. The closed frequent itemsets comprise in the itemsets $\{1,2,3,4\}$, $\{3,4\}$, and $\{3,4,5,6\}$. The free frequent itemsets are $\emptyset$, $\{1\}$, $\{2\}$, $\{5\}$, and $\{6\}$.

Although there is a many-to-one mapping between free and closed frequent itemsets in $\mathcal{D}$, i.e., that there are always at least as many free frequent itemsets as there are closed frequent itemsets, the collections of free frequent itemsets have the advantage of being downward closed. (An itemset collection $\mathcal{S}$ is downward closed if $X \in \mathcal{S}, Y \subseteq X \Rightarrow Y \in \mathcal{S}$.) This enables the re-use of virtually all known frequent itemset mining algorithms with minor modifications to discover only free frequent itemsets.

The deduction rules for closed and free itemsets to purge the frequent itemset collection are quite simple. Hence, they do not purge the collection very much. Smaller condensed representations can be obtained using more powerful deduction rules, such as Bonferroni inequalities [6, 7]:

Definition 5 (Non-derivable frequent itemsets). *$X \in \mathcal{F}(\sigma, \mathcal{D})$ is non-derivable σ-frequent itemset in $\mathcal{D}$, if $\underline{supp}(X, \mathcal{D}) < \overline{supp}(X, \mathcal{D})$ where*

$$\underline{supp}(X, \mathcal{D}) = \max_{Y \subsetneq X, |X \setminus Y| \text{ odd}} \sum_{Y \subseteq Z \subsetneq X} (-1)^{|X \setminus Z|+1} supp(Z, \mathcal{D})$$

$$\overline{supp}(X, \mathcal{D}) = \min_{Y \subsetneq X, |X \setminus Y| \text{ even}} \sum_{Y \subseteq Z \subsetneq X} (-1)^{|X \setminus Z|+1} supp(Z, \mathcal{D})$$

are the greatest lower bound $\underline{supp}(X, \mathcal{D})$ and smallest upper bound $\overline{supp}(X, \mathcal{D})$ for the support of an itemset X given the supports of all subsets of X. The collection of non-derivable σ-frequent itemsets is $\mathcal{N}(\sigma, \mathcal{D})$.

3 Itemset support queries

The itemset support query problem asks for the supports of a collection of itemsets w.r.t. some support function $supp|_{\mathcal{S}}$. The problem can be formulated as follows:

Problem 1 (Itemset support query). Given a representation of supports of itemsets in a collection $\mathcal{S}$, and a collection $\mathcal{Q}$ of itemsets, find the supports for all itemsets in $\mathcal{Q} \cap \mathcal{S}$.

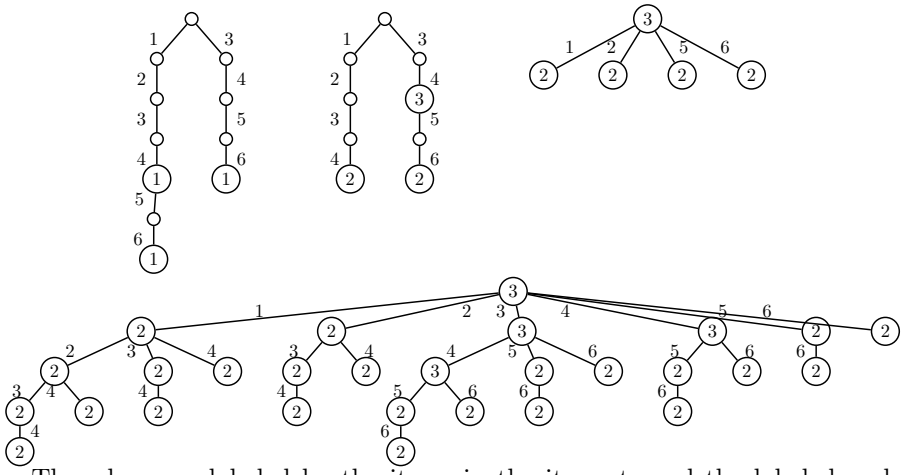
In this paper, we focus on the case where the collection $\mathcal{S}$ is equal to $\mathcal{F}(\sigma, \mathcal{D})$ in some transaction database $\mathcal{D}$ and $\mathcal{Q}$ being a single itemset, as the main goal of the current paper is study how well condensed representations of itemset collections are suited to answer itemset support queries.

The representations we consider are $supp|_{\mathcal{F}(\sigma,\mathcal{D})}$, $supp|_{\mathcal{C}(\sigma,\mathcal{D})}$, $supp|_{\mathcal{G}(\sigma,\mathcal{D})}$ $supp|_{\mathcal{N}(\sigma,\mathcal{D})}$ and $count|_{\mathcal{D}_u}$. That is, we use index structures containing supports for all, closed, free and non-derivable frequent itemsets, and the occurrence

counts of different itemsets in the database $\mathcal{D}$, respectively, to answer the itemset support queries. We represent itemset collections $\mathcal{S}$ that we use in the itemset support queries as tries, similarly to [26].

Definition 6 (Itemset tries). *An itemset trie for an itemset collection $\mathcal{S} \subseteq 2^{\mathcal{I}}$ is a rooted labeled tree $T(\mathcal{S}) = T = \langle V, E, l : E \to \mathcal{I} \rangle$, such that for each itemset $X \in \mathcal{S}, |X| = k$, there is an unique node $v_X \in V$ such that the labels $label(\langle v_{i-1}, v_i \rangle)$ of the path $\langle root(T), v_1 \rangle, \ldots, \langle v_{k-1}, v_X \rangle$ form an increasing sequence corresponding to the itemset $X = \{label(\langle root(T), v_1 \rangle), \ldots, label(\langle v_{k-1}, v_X \rangle)\}$. The itemset corresponding to a node v is denoted by X_v.*

Example 3. The tries $T(\mathcal{D}_u)$, $T(\mathcal{C}(\sigma, \mathcal{D}))$, $T(\mathcal{G}(\sigma, \mathcal{D}))$ and $T(\mathcal{F}(\sigma, \mathcal{D}))$ of Examples 1 and 2 are as follows:



The edges are labeled by the items in the itemsets and the labeled nodes correspond to the itemsets in the collection: the itemset associated to a label node is determined by the edge labels of the path from root to that node, and the node label corresponds to the value of the node.

Itemset tries provide simple representations of itemset collections $\mathcal{S}$ for deciding very efficiently whether a given itemset is in the collection or not. The tries are adapted straightforwardly to answer also many other kinds of queries by adding to each node v in $T(\mathcal{S})$ a value $val(v)$. The exact use of the values of the nodes depend on what kind of function is represented by the trie. For example, $val(v) = count(X_v, \mathcal{D}), X_v \in \mathcal{D}_u$, and $val(v) = supp(X_v, \mathcal{D}), X_v \in \mathcal{S}$ for all $supp|_{\mathcal{S}}, \mathcal{S} \in \{\mathcal{F}(\sigma, \mathcal{D}), \mathcal{C}(\sigma, \mathcal{D}), \mathcal{G}(\sigma, \mathcal{D}), \mathcal{N}(\sigma, \mathcal{D})\}$.

The algorithm for querying the value of a given itemset Q from an itemset trie $T(\mathcal{S})$ is given in Algorithm 1. The algorithm retrieves the value of an itemset Q (represented as an ascending sequence $Q_1, \ldots, Q_{|Q|}$ of items) in time $\mathcal{O}(|Q|)$. We assume that $val(v) = 0$ for all $X_v \notin \mathcal{S}$. Hence, the the value returned for an itemset not in $\mathcal{S}$ is always 0.

Algorithm 1 provides the worst-case optimal solution for querying the values associated to itemsets from an itemset trie representing the collection explicitly.

Algorithm 1 The algorithm for obtaining the value of an itemset Q from $T(\mathcal{S})$ if $Q \in \mathcal{S}$.

```
1: function IVQ-Eq(T(S), Q)
2:     u ← root(T(S))
3:     for all j = 1, ..., |Q| do
4:         if child(u, Q_j) exists then
5:             u ← child(u, Q_j)
6:         else
7:             return 0
8:     return val(u)
```

As the condensed representations of frequent itemsets are proper subcollections of frequent itemsets, Algorithm 1 can be used to query only the supports of the itemsets that appear explicitly in the condensed representations. For example, Algorithm 1 returns non-zero supports only for closed frequent itemsets from $T(\mathcal{C}(\sigma, \mathcal{D}))$ and only for free frequent itemsets from $T(\mathcal{G}(\sigma, \mathcal{D}))$.

The supports of frequent itemsets can be retrieved from the supports of closed and free frequent itemsets, and the counts in $\mathcal{D}_u$ using the formula of form

$$val(X) = \circ \{val(Y) : Y \in \mathcal{S}, Y \supseteq X\} \tag{1}$$

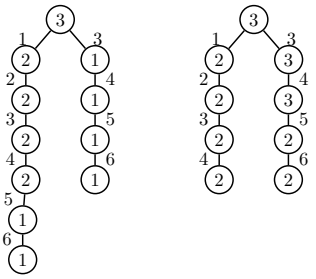
where the operation $\circ$ forms a commutative monoid with some value set M. ($M = \mathbb{N}$ for supports and counts.) Namely, the support of $X \in \mathcal{F}(\sigma, \mathcal{D})$ can be obtained from those representations as follows:

$$\begin{aligned} supp(X, \mathcal{D}) &= \sum_{Y \in \mathcal{D}_u, Y \supseteq X} count(Y, \mathcal{D}) \\ &= \max_{Y \in \mathcal{C}(\sigma, \mathcal{D}), Y \supseteq X} supp(Y, \mathcal{D}) \\ &= \min_{Y \in \mathcal{G}(\sigma, \mathcal{D}), \mathcal{I} \setminus Y \supseteq \mathcal{I} \setminus X} supp(Y, \mathcal{D}) \end{aligned}$$

Hence, the operations $\circ$ are in these cases $+$, max and $\min^{>0}$, respectively. (These formulas to deduce the supports follow immediately from the definitions of $\mathcal{D}_u$, $\mathcal{C}(\sigma, \mathcal{D})$, and $\mathcal{G}(\sigma, \mathcal{D})$.)

To answer the itemset value queries using Equation 1, the itemset tries $T(\mathcal{S})$ are preprocessed for these queries in such a way that the values $val(v)$ of the nodes v in $T(\mathcal{S})$ are replaced by $\circ \{val(u) : u \in subtrie(v, T(\mathcal{S}))\}$. Such preprocessing can be done in time $\mathcal{O}(|V|)$ for an itemset trie $T = (V, E)$.

Example 4. After the preprocessing, the tries $T(\mathcal{D}_u)$ and $T(\mathcal{C}(\sigma, \mathcal{D}))$ of Example 3 look the following:



Preprocessing replaces the values $val(v)$ in $T(\mathcal{D}_u)$ by $\sum_{u \in subtrie(v,T(\mathcal{S}))} val(u)$ and the values $val(v)$ in $T(\mathcal{C}(\sigma, \mathcal{D}))$ by $\max\{val(u) : u \in subtrie(v, T(\mathcal{S}))\}$.

The evaluation of Equation 1 for (preprocessed) itemset tries $T(\mathcal{S})$ is described as Algorithm 2. (Note that the pseudocode omits many lower-level details of the algorithm for the sake of readability. For example, recursive function calls are not be used in practice as they cause some slowdown.)

Algorithm 2 The algorithm for obtaining the value itemset Q from $T(\mathcal{S})$ by Equation 1.

```
1: function IVQ-Supset(T(S), Q)
2:     return IVQ-Supset-Node(T(S), Q, root(T(S)), 1)

1: function IVQ-Supset-Node(T(S), Q, v, j)
2:     val ← 0
3:     for all u ∈ children(v), label(v, u) ≤ Q_j and subtrie(u, T(S)) can contain the
       itemset {Q_j, ..., Q_|Q|} do
4:         if label(v, u) < Q_j then
5:             val ← val ∘ IVQ-Supset-Node(T(S), Q, u, j)
6:         else if j < |Q| then
7:             val ← val ∘ IVQ-Supset-Node(T(S), Q, u, j + 1)
8:         else
9:             val ← val ∘ val(u)
10:    return val
```

The worst-case time complexity of Algorithm 2 is not as good as of Algorithm 1. Namely, in the worst case the whole itemset trie $T(\mathcal{S})$ have to be traversed. This is the case, for example, when $\mathcal{S} = \{\{i, n/2, \ldots, n-1\} : i = 1, \ldots, n/2-1\}$ and $Q = \{n\}$. In this example the ordering of the items in the trie makes a great difference in terms of time and space requirements. Even more severe example is the following: $\mathcal{S}\{X \cup \{2n+1\} : X \subseteq \{1, \ldots, 2n\}, |X| = n\}$ and $Q = \{2n+1\}$.

The performance of the query evaluation can be improved by efficient and effective detection of subtries that cannot have paths containing the query itemset. A few examples of such criteria are $depth(subtrie(u, T(\mathcal{S}))) < |Q| - i$ (i.e., in the subtrie of u there should be a path of length at least $|Q| - i$) and

$\max label(subtrie(u, T(\mathcal{S}))) < Q_{|Q|}$ (i.e., it should be possible that the subtrie contains the item $Q_{|Q|}$)).

Algorithm 2 is not very appropriate for free frequent itemsets because then the complements of the itemsets must be used. This often leads to much larger itemsets. An alternative of using the complements of the free frequent itemsets is to use a simple modification of Equation 1:

$$val(X) = \circ\{val(Y) : Y \in \mathcal{S}, Y \subseteq X\} \tag{2}$$

The support of $X \in \mathcal{F}(\sigma, \mathcal{D})$ can be obtained from the representations $T(\mathcal{D}_u)$, $T(\mathcal{C}(\sigma, \mathcal{D}))$ and $T(\mathcal{G}(\sigma, \mathcal{D}))$ using Equation 2 by

$$\begin{aligned} supp(X, \mathcal{D}) &= \sum_{Y \in \mathcal{D}_u, \mathcal{I} \setminus Y \subseteq \mathcal{I} \setminus X} count(Y, \mathcal{D}) \\ &= \max_{Y \in \mathcal{C}(\sigma, \mathcal{D}), \mathcal{I} \setminus Y \subseteq \mathcal{I} \setminus X} supp(Y, \mathcal{D}) \\ &= \min_{Y \in \mathcal{G}(\sigma, \mathcal{D}), Y \subseteq X} supp(Y, \mathcal{D}) \end{aligned}$$

The method implementing Equation 2 is given as Algorithm 3. For this algorithm the itemset tries $T(\mathcal{S})$ should represent the itemset collection and their values as they are, i.e., without any preprocessing.

Algorithm 3 The algorithm for obtaining the value itemset Q from $T(\mathcal{S})$ by Equation2.

```
1: function IVQ-Subset(T(S), Q)
2:     return IVQ-Subset-Node(T(S), Q, root(T(S)), 1)

1: function IVQ-Subset-Node(T(S), Q, v, j)
2:     val ← 0
3:     for all u ∈ children(v), label(v, u) ∈ Q do
4:         if label(v, u) < Q_|Q| then
5:             val ← val ∘ IVQ-Subset-Node(T(S), Q, u, min{i : Q_i > label(v, u)})
6:         else
7:             val ← val ∘ val(u)
8:     return val
```

In this case, the time complexity can also be quite high: in the worst case the whole trie $T(\mathcal{S})$ has to be traversed even when no itemset in $\mathcal{S}$ is contained in any itemset in $\mathcal{Q}$. As an example, consider the collection $\mathcal{S} = \{X \cup \{n\} : X \subseteq \{1, \ldots, n-2\}\}$ and the query $Q = \{1, \ldots, n-1\}$.

In general case not much additional pruning of unpromising parts of the search trie can be done. In some special cases, however, that is possible. For example, in the case free frequent itemsets, maintaining the smallest support in the subtrie can be used to skip those subtries that cannot have smaller supports than the value obtained that far in the traversal.

Answering itemset support queries using the representation $T(\mathcal{N}(\sigma, \mathcal{D}))$ is slightly more complicated, but can be implemented as follows: (1) query $\mathcal{Q}$ is transformed into downward closed collection $\mathcal{Q}'$, (2) Algorithm 1 is applied to $T(\mathcal{Q}')$, (3) the still undetermined supports of the itemsets in $T(\mathcal{Q}')$ are deduced using Bonferroni-inequalities [27], and (4) Algorithm 1 is applied to retrieve supports of itemsets in $\mathcal{Q}$ from $T(\mathcal{Q}')$.

Unfortunately the proposed algorithm for itemset support queries from the representation based on non-derivable itemsets is not very practical, because the fastest known algorithm to obtain the support of an itemset of size k using Bonferroni-inequalities has the worst-case time complexity $\mathcal{O}(2^k)$ [27]. Hence, we omit non-derivable itemsets from our experiments.

4 Experiments

We examine the performance of itemset query algorithms using random queries to several benchmark databases in frequent itemset mining from the FIMI transaction database repository (http://fimi.cs.helsinki.fi). To see the potential differences in the performance of different representations, we selected 8 transaction databases where the number of all frequent itemsets is higher than the number of closed or free frequent itemsets. For each database $\mathcal{D}$ we used a minimum support threshold $\sigma\,|\mathcal{D}|$ that produces a large number of frequent itemsets, because the goal of using itemset collections as index structures for itemset support queries is to be able to answer as large fraction of the queries as possible. This differs somewhat from the classical use of frequent itemsets and association rules as a small number of nuggets of knowledge. The basic statistics of the selected databases are shown in Table 1.

Table 1. The transaction databases used in the experiments. The columns are the transaction database name, the minimum support threshold, the number frequent items, the number of transactions, the number of different transactions after removing infrequent items, the number of frequent itemsets, the number of free frequent itemsets, and the number of closed frequent itemsets.

$\mathcal{D}$	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{D}\rvert$	$\sigma\,\lvert\mathcal{D}\rvert$	$\lvert\mathcal{I}^{\geq\sigma}\rvert$	$\lvert\mathcal{D}_u^{\geq\sigma}\rvert$	$\lvert\mathcal{F}(\sigma,\mathcal{D})\rvert$	$\lvert\mathcal{G}(\sigma,\mathcal{D})\rvert$	$\lvert\mathcal{C}(\sigma,\mathcal{D})\rvert$
CONNECT	129	67 557	50 000	30	214	1 928 335	26 417	26 417
KOSARAK	41 270	990 002	900	1 384	387 603	1 598 294	1 143 594	1 124 819
MUSHROOM	119	8 124	500	67	7 032	1 442 503	14 925	9 864
PUMSB	2 113	49 046	35 000	34	3305	1897479	519 725	194 538
PUMSB*	2 088	49 046	13 000	63	25 404	1 293 828	57 172	32 115
RETAIL	16 470	88 162	5	10 988	83 119	1 506 775	532 342	504 142
WEBVIEW-1	497	59 601	35	369	18 184	1 177 607	118 696	76 260
WEBVIEW-2	3 340	77 511	15	2 643	48 117	1 599 210	397 283	343 818

We selected randomly 1 000 000 itemsets from $\mathcal{F}(\sigma, \mathcal{D})$ with uniform distribution over the collection to assess average-case efficiency of different represen-

tations of the frequent itemsets to answer itemset support queries. As a baseline representation we used the representation of $\mathcal{D}_u^{\geq\sigma}$ as list of itemsets together with their occurrence counts. The sizes of the trie representations and the average query answering performances for 1 000 000 random itemset support queries from the collection $\mathcal{F}(\sigma, \mathcal{D})$ for the databases of Table 1 are shown in Table 2.

Table 2. The space requirements of different trie representations and their query answering time speedups w.r.t. a list representation of $\mathcal{D}_u^{\geq\sigma}$ for 1 000 000 random itemset support queries selected from frequent itemsets.

	the sizes of the tries in KBs				the speed-ups in the query times			
$\mathcal{D}$	$\mathcal{D}_u^{\geq\sigma}$	$\mathcal{F}(\sigma,\mathcal{D})$	$\mathcal{G}(\sigma,\mathcal{D})$	$\mathcal{C}(\sigma,\mathcal{D})$	$\mathcal{D}_u^{\geq\sigma}$	$\mathcal{F}(\sigma,\mathcal{D})$	$\mathcal{G}(\sigma,\mathcal{D})$	$\mathcal{C}(\sigma,\mathcal{D})$
CONNECT	36	75 326	1 032	1 032	14.4	7.0	1.1	1.4
KOSARAK	117 735	62 433	44 672	44 286	0.8	8 705.7	170.8	65.7
MUSHROOM	776	56 348	583	420	29.5	244.7	71.6	126.2
PUMSB	752	74 120	20 302	10 416	43.9	117.8	6.2	5.6
PUMSB*	6 695	50 540	2 233	1 395	41.1	1 505.3	248.1	302.9
RETAIL	26 024	58 858	20 795	20 240	1.5	1 759.3	35.0	6.4
WEBVIEW-1	2 166	46 000	4 637	3 308	5.2	320.5	38.0	11.4
WEBVIEW-2	8 692	62 469	15 519	13 950	22.5	820.8	24.8	11.7

The trie representations show often considerable speedup compared to the baseline approach. The experimental results show also that answering itemset support queries to $T(\mathcal{D}_u^{\geq\sigma})$, $T(\mathcal{C}(\sigma,\mathcal{D}))$ and $T(\mathcal{G}(\sigma,\mathcal{D}))$ are often much slower than answering them using $T(\mathcal{F}(\sigma,\mathcal{D}))$. Especially queries to $T(\mathcal{D}_u^{\geq\sigma})$ for KOSARAK were considerable slower than to the other representations, even slower than the baseline representation. One possible explanation is that the branching factor of the trie $T(\mathcal{D}_u^{\geq\sigma})$ is higher than in $T(\mathcal{F}(\sigma,\mathcal{D}))$, $T(\mathcal{C}(\sigma,\mathcal{D}))$ and $T(\mathcal{G}(\sigma,\mathcal{D}))$. Furthermore, the pruning conditions in Algorithm 2 are more expensive than in the baseline approach. If the pruning fails, then the pruning attempts result just slowdown.

In general, condensed representations of frequent itemsets seem to offer a reasonable alternative for itemset support queries using all frequent itemsets only when very few frequent itemsets are contained in the condensed representation. (Note that in the case of $T(\mathcal{G}(\sigma,\mathcal{D}))$ and $T(\mathcal{C}(\sigma,\mathcal{D}))$ the query optimization of first checking whether the itemset is in the collection could improve the performance considerably.) The preliminary experimental results are promising, but more comprehensive evaluation of itemset support queries is needed.

5 Conclusions

In this paper we studied the use of frequent itemsets, the condensed representations of the frequent itemsets, and the concise representations of transaction databases for answering itemset support queries. Answering such queries is an

important task in inductive databases for performing interactive data mining queries and as building blocks of data mining algorithms. We proposed efficient trie structures for representing itemset collections and the answering itemset support queries. We evaluated experimentally the applicability of the major condensed representations of frequent itemsets to answer itemset support queries. Trie representations seem to offer a reasonable approach for facilitating itemset support querying.

As a future work we plan to work on efficient indices for querying approximate representations of frequent itemsets. Some representations, e.g., [19, 28], fit readily to itemset tries, but it is not clear what would be the best representation for answering multiple itemset support queries. Another potential line of research is of hierarchical descriptions of data to facilitate the efficient query answering. For example, using the projections of the transaction database to the maximal frequent itemsets can result significant space saving compared other condensed representations [29]. Querying projections with only a very small number of different transactions is fast. Most likely there is no particular represenation that would be always the best for all itemset support queries, but different indices suit for different kinds of transaction databases. This suggests to develop techniques for finding most efficient representation for a particular database and distribution queries. Furthermore, combining different representations for a single transaction databases have some promise in both helping to comprehend the underlying itemset collections and speeding up the itemset support queries.

References

1. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In Buneman, P., Jajodia, S., eds.: SIGMOD Conference. (1993) 207–216
2. Goethals, B.: Frequent set mining. In Maimon, O., Rokach, L., eds.: The Data Mining and Knowledge Discovery Handbook. Springer (2005) 377–397
3. Goethals, B., Zaki, M.J., eds.: FIMI '03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 19 December 2003, Melbourne, Florida, USA. Volume 90 of CEUR Workshop Proceedings. (2003)
4. Bayardo Jr., R.J., Goethals, B., Zaki, M.J., eds.: FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004. Volume 126 of CEUR Workshop Proceedings. (2004)
5. Mannila, H., Toivonen, H.: Multiple uses of frequent sets and condensed representations (extended abstract). In: KDD. (1996) 189–194
6. Calders, T., Rigotti, C., Boulicaut, J.F.: A survey on condensed representations for frequent sets. [30] 64–80
7. Mielikäinen, T.: Transaction databases, frequent itemsets, and their condensed representations. [31] 139–164
8. Boulicaut, J.F.: Inductive databases and multiple uses of frequent itemsets: The cInQ approach. In Boulicaut, J.F., Raedt, L.D., Mannila, H., eds.: Database Support for Data Mining Applications. Volume 3848 of LNCS. (2004) 1–23
9. Imielinski, T., Mannila, H.: A database perspective on knowledge discovery. Communications of the ACM **39** (1996) 58–64

10. Mannila, H.: Inductive databases and condensed representations for data mining. In: ILPS. (1997) 21–30
11. Siebes, A.: Data mining in inductive databases. [31] 1–23
12. Clark, P., Niblett, T.: The CN2 induction algorithm. Machine Learning **3** (1989) 261–283
13. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1992)
14. Maron, M.E.: Automatic indexing: An experimental inquiry. J. ACM **8** (1961) 404–417
15. Panov, P., Džeroski, S., Blockeel, H., Loškovska, S.: Predictive data mining using itemset frequencies. In: Proceedings of the 8th International Multiconference Information Society. (2005) 224–227
16. Kearns, M.J.: Efficient noise-tolerant learning from statistical queries. J. ACM **45** (1998) 983–1006
17. Pavlov, D., Mannila, H., Smyth, P.: Beyond independence: Probabilistic models for query approximation on binary transaction data. IEEE Transactions on Knowledge and Data Engineering **15** (2003) 1409–1421
18. Seppänen, J.K., Mannila, H.: Boolean formulas and frequent sets. [30] 348–361
19. Mielikäinen, T.: Separating structure from interestingness. In Dai, H., Srikant, R., Zhang, C., eds.: PAKDD. Volume 3056 of LNCS. (2004) 476–485
20. Toivonen, H.: Sampling large databases for association rules. In Vijayaraman, T.M., Buchmann, A.P., Mohan, C., Sarda, N.L., eds.: VLDB. (1996) 134–145
21. Kubat, M., Hafez, A., Raghavan, V.V., Lekkala, J.R., Chen, W.K.: Itemset trees for targeted association querying. IEEE Transactions on Knowledge and Data Engineering **15** (2003) 1522–1534
22. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Min. Knowl. Discov. **8** (2004) 53–87
23. Moore, A.W., Lee, M.S.: Cached sufficient statistics for efficient machine learning with large datasets. JAIR **8** (1998) 67–91
24. Mielikäinen, T.: Implicit enumeration of patterns. [32] 150–172
25. Laur, S., Lipmaa, H., Mielikäinen, T.: Private itemset support counting. In Qing, S., Mao, W., Lopez, J., Wang, G., eds.: ICICS. Volume 3783 of LNCS. (2005) 97–111
26. Mielikäinen, T.: An automata approach to pattern collections. [32] 130–149
27. Calders, T., Goethals, B.: Quick inclusion-exclusion. [31] 86–103
28. Geerts, F., Goethals, B., Mielikäinen, T.: What you store is what you get. [33] 60–69
29. Mielikäinen, T.: Finding all occurring patterns of interest. [33] 97–106
30. Boulicaut, J.F., Raedt, L.D., Mannila, H., eds.: Constraint-Based Mining and Inductive Databases, European Workshop on Inductive Databases and Constraint Based Mining, Hinterzarten, Germany, March 11-13, 2004, Revised Selected Papers. Volume 3848 of LNCS. (2005)
31. Bonchi, F., Boulicaut, J.F., eds.: Knowledge Discovery in Inductive Databases, 4th International Workshop, KDID 2005, Porto, Portugal, October 3, 2005, Revised Selected and Invited Papers. Volume 3933 of LNCS. (2006)
32. Goethals, B., Siebes, A., eds.: KDID 2004, Knowledge Discovery in Inductive Databases, Proceedings of the Third International Workshop on Knowledge Discovery in Inductive Databases, Pisa, Italy, September 20, 2004, Revised Selected and Invited Papers. Volume 3377 of LNCS. (2005)
33. Boulicaut, J.F., Dzeroski, S., eds.: Proceedings of the Second International Workshop on Inductive Databases, 22 September, Cavtat-Dubrovnik, Croatia. (2003)