Computing Frequent k-Itemsets Directly in Sparse Datasets

Maurizio Atzori[1,2], Paolo Mancarella[1], and Franco Turini[1]

[1] Dipartimento di Informatica, University of Pisa, Italy
{atzori,paolo,turini}@di.unipi.it
[2] ISTI-CNR, Area della Ricerca di Pisa, Italy

Abstract. In this paper we show that the well known problem of computing frequent k-itemsets (i.e. itemsets of cardinality k) in a given dataset can be reduced to the problem of finding iceberg queries from a stream of queries suitably constructed from the original dataset. Hence, algorithms for computing frequent k-itemsets can be obtained by adapting algorithms for computing iceberg queries. In the paper we show that, for sparse datasets, this can be done directly, i.e. without generating frequent x-itemsets, for each $x < k$, as done in the most common algorithms based on a level-wise approach. We exploit a recent algorithm for finding iceberg queries and define an algorithm which requires only three sequential passes over the dataset to compute the frequent k-itemsets (even for $k > 3$). An important feature of the algorithm is that the amount of main memory required can be determined in advance, and it is shown to be very low for sparse datasets. Experiments show that for very large datasets with millions of small transactions our proposal outperforms the state-of-the-art algorithms.

1 Introduction

The field of Data Mining concerns the extraction of useful information from raw data. This is usually done by generalizing data to induce models from datasets. Among the models considered important for decision making, association rules [1, 2] play an important role, in that they allow us to highlight relevant trends in the data and also to gain some improvements when dealing with other models, such as classification (see e.g. [3]) and clustering [4]. Roughly speaking, mining association rules from a given set of transactions (e.g. a given set of supermarket receipts) amounts at finding rules of the form $i_1, \ldots, i_n \Rightarrow i_{n+1}, \ldots, i_k$, where each i_j is an item (e.g. a supermarket good). The intended meaning of such a rule is that it is likely that a transaction containing the items in the premise contains also the items in the conclusion.

It is well known that the most expensive task in mining association rules is the extraction of frequent itemsets, i.e. sets of items which occur together in at least a given percentage of the whole set of transactions. Once frequent itemsets are produced from the dataset, the generation of the association rules is rather straightforward. Since the datasets we are interested in are typically huge, it is

important to devise algorithms which try to minimize both the time and the space required for the analysis. Most of the known proposals that solve this kind of problem for huge datasets are based on level-wise algorithms that compute frequent itemsets of increasing cardinality, up to a given one, thus requiring several passes through the dataset. This is done in order to maintain the main memory space usage acceptable. Non level-wise, depth-first algorithms also exist [5], but they usually require that the whole dataset (even in a compressed form) fits in main memory.

In this paper we will focus on the problem of finding itemsets of a given size directly, i.e. without generating smaller itemsets as done in level-wise approaches. This is particularly useful in the context of *Inductive Databases* since in order to answer some queries, it would be necessary to know the exact support or the number of the frequent k-itemsets. Mining all the frequent itemsets would reduce memory and time efficiency, and maximal itemset mining could be not sufficient to answer the query (in fact maximal itemsets do not allow us to compute the exact support of smaller itemsets). As shown later in the experiment section, in very large datasets with thousands of items and millions of small transactions our proposal is able to compute frequent k-itemsets while the current state-of-the-art algorithms fail due to huge main memory requirements.

First we show that the problem of finding k-frequent itemsets addressed in this paper can be transformed into the problem of finding frequent symbols over a (huge) stream of symbols over a given alphabet, often referred to as the iceberg queries problem or the hot list analysis. Then, we exploit a recent algorithm for the iceberg queries problem which allows us to solve the original frequent itemset problem by only two sequential passes over the dataset plus a preprocessing step aimed at computing some statistics on the dataset (three passes in total). We will see that, for sparse datasets (i.e. datasets with few items per transactions w.r.t. the total number of possible items) the amount of main memory required by the proposed algorithm is very low and indipendent from both the number of items and the size of the dataset, as shown by some experiments we have conducted on a prototype implementation of the algorithm. Notice that, when looking for association rules of the form $i_1 \dots i_n \Rightarrow i_{n+1} \dots i_k$, with $k > n$, we need to determine frequent itemsets of cardinality n and k. Using standard level-wise algorithms such as *Apriori* [6], k passes through the dataset have to be performed. Using our approach, we can run two instances of the proposed algorithm in parallel, thus requiring three passes through the dataset overall. The main contribution of this paper is the developement of an algorithm that, for sparse datasets, requires a limited amount of memory while keeping a (small) constant number of passes over the input.

The paper is organized as follows. In Section 2 we set up the notation used in the rest of the paper, formally defining both the frequent itemsets and the iceberg queries frameworks and we briefly recall some of the existing works related to these problems. In Section 3 we present a new approach to the problem of finding frequent itemsets of size k, based on the reduction to the iceberg queries problem. In Section 4 we present an algorithm that computes the exact set of

k-itemsets reading sequentially the dataset only twice. Section 5 is devoted to present some experiments we have conducted in order to show the effectiveness of our algorithm in terms of the amount of main memory needed. Finally, Section 6 contains some discussions on future works and directions we are planning to follow.

2 Preliminaries

In this section we set up the basic notations and terminology that we are going to adopt in the rest of the paper, and we formally define the so called *frequent itemset mining* problem and the *iceberg queries* problem.

2.1 Frequent Itemset Mining Problem

Let us first set up some notational conventions used in the sequel.

Set of items A set of items is a finite set, denoted by $\mathcal{I}$. Elements of $\mathcal{I}$ will be denoted by $i, i', i_1, \ldots$ and are referred to as *items*.

Itemset An itemset is a subset of $\mathcal{I}$. Itemsets will be denoted by $I, I', \ldots$.

k-itemset An itemset I is called a k-itemset if $|I| = k$, where $|I|$ denotes the cardinality of I.

Transaction A transaction T is an itemset, denoted by $T, T', \ldots$.

Dataset A *dataset* $\mathcal{D}$ is a multiset (a bag) of transactions. Given $\mathcal{D}$, the *maximal transaction length* of $\mathcal{D}$ is

$$m_{\mathcal{D}} = \max\{|T| \mid T \in \mathcal{D}\}.$$

Definition 1 (FIM Problem). *Let $\mathcal{D}$ be a dataset and I be an itemset. I is called a* frequent *itemset with respect to $\mathcal{D}$ and a support σ, with $0 < \sigma \leq 1$ if:*

$$|\{T \in \mathcal{D} \mid I \subseteq I(T)\}| \geq \sigma|\mathcal{D}|.$$

Let $\mathcal{F}_k(\sigma, \mathcal{D})$ be the set of all k-itemset that are frequent w.r.t. σ and $\mathcal{D}$. Then, the FIM problem is defined as the task of determining $\mathcal{F}_k(\sigma, \mathcal{D})$ for each k such that $0 < k \leq m_{\mathcal{D}}$.

In the sequel we will often write simply $\mathcal{F}_k$ instead of $\mathcal{F}_k(\sigma, \mathcal{D})$, whenever the parameters σ and $\mathcal{D}$ are either clear from the context or irrelevant.

2.2 Iceberg Queries

Our approach is based on the reduction of the problem of frequent itemsets computation to the problem of finding iceberg queries. Let us define the so called *Iceberg Queries* problem (also known as *Hot List Analysis*).

Alphabet By $\mathcal{Q}$ we denote a finite alphabet. Elements of $\mathcal{Q}$ are denoted by $q, q', q_1, \ldots$ and are referred to as *queries*.

Stream A *stream of queries* is a sequence $s = \langle q_1, \ldots, q_n \rangle$, such that $q_i \in \mathcal{Q}$, for each $1 \leq i \leq n$; the length n of the stream is referred to as $|s|$.

Frequence Given a stream s and a query q, $f_s(q)$ denotes the number of occurrences of q in s.

Definition 2 (IQ Problem). *Let $\mathcal{Q}$ be a set of queries, s be a stream of queries and ϑ be a real number such that $0 < \vartheta \leq 1$. The IQ problem is defined as the task of determining the subset $Q(\vartheta, s)$ defined as follows:*

$$Q(\vartheta, s) = \{q \in \mathcal{Q} \mid f_s(q) > \vartheta |s|\}.$$

In the sequel, if a query q belongs to $Q(\vartheta, s)$ we will say that q is an *iceberg query* with respect to $\mathcal{Q}$, s and ϑ.

Before going on, it is worth pointing out that, in concrete applications of both FIM and IQ problems, the input ($\mathcal{D}$ and s, respectively) is usually huge and it can only be red sequentially (e.g. according to transaction identifiers in the first case and to the sequence order in the second case). Moreover, in FIM problems we usually have $|\mathcal{D}| \gg |\mathcal{I}|$ and in IQ problems we usually have $|s| \gg \mathcal{Q} \gg 1/\vartheta$.

2.3 Related Work

Almost all the algorithms for finding frequent itemsets are based on the level-wise generation of candidates of the *Apriori* algorithm [6]. The level-wise approach is performed in order to maintain the search space small enough to fit into the main memory. This strategy necessarily leads to several passes through the dataset.

Some other papers present different approaches in order to keep the number of passes through the dataset constant. In [7] the authors show a partitioning technique that needs two passes through the database. First, the dataset is partitioned into several parts which are small enough to fit into the memory. Every partition is elaborated using a level-wise algorithm and then the results of each partition are merged. This leads to a superset of the solution. A second scan then removes the false positive elements of the superset. Unfortunately, if the dataset is very large then the resulting partitions can be too small with respect to the dataset, leading to a huge superset, and this can reduce the effectiveness of the algorithm.

Another important approach to reduce the number of passes through the dataset is the one proposed by Toivonen in [8] and then refined in [9, 10], based on the evaluation of a small random sample of the dataset. A set of patterns that are probably frequent in the whole dataset are generated, and then their exact frequencies are verified in the rest of the dataset. If a failure occurs in the generation of candidates (i.e., there are false negatives), a mechanism is provided which, in a second pass, computes the remaining frequent patterns. By decreasing the support threshold the probability of failure can be decreased, but for low probabilities this drastically increments the number of candidates. Furthermore, if we are dealing with very large datasets, it is possible that the (small) sample is not very representative of the whole dataset, and this means

that there is a high probability of failure. In this case the candidates to be verified in the second pass can be too many to be fitted into the main memory (i.e. more than two passes are needed).

Main memory usage of depth-first (i.e., non-levelwise) frequent pattern mining algorithms is discussed in [11]: two state-of-the-art algorithms, FP-Growth [5] and Eclat [12], are tested and shown to be very memory consuming even for medium-size datasets. A simple improvement of Eclat, named Medic, is proposed but it is empirically shown to reduce the amount of memory needed of $\approx 50\%$ in the best case: the memory requirements still depend on the size of the dataset, and this fact leaves the algorithm impractical when datasets are very large. Another confirmation of the scalability limitations of current state-of-the-art algorithms for frequent itemset mining came from the First IEEE ICDM Workshop on Frequent Itemset Mining Implementations, FIMI 2003 [13], where several well-known algorithms were implemented and independently tested. The results show that "*none of the algorithms is able to gracefully scale-up to very large datasets, with millions of transactions*".

Our technique is based on a novel approach to the Iceberg Queries problem, proposed in [14]. The authors present a (surprisingly) simple algorithm, able to find all queries with frequency greater than or equal to a given threshold ϑ, from a given stream of queries, by using $O(1/\vartheta)$ main memory cells and performing two passes through the stream. Notice that, in the worst-case, the output size is exactly $1/\vartheta$. Furthermore, the algorithm proposed in [14], that we will call *KPS*, does $O(1)$ operations per query (under the reasonable assumption that hash tables make $O(1)$ operations for insertion, search and deletion).

3 Transforming FIM problems into IQ problems

In this section we show how a FIM problem can be transformed into an IQ problem. Roughly speaking, the idea is to associate to each k-itemset a query and to construct a suitable stream $s_{\mathcal{D}}$ of queries starting from the given dataset $\mathcal{D}$, in such a way that the problem of determining $\mathcal{F}_k(\sigma, \mathcal{D})$ is transformed into the problem of determining $Q(\vartheta, s_{\mathcal{D}})$, where ϑ is a function of $\sigma, m_{\mathcal{D}}$ and k. Once we have defined such transformation, we can adopt any algorithm for the IQ problem in order to solve the original FIM problem. In particular, we can adopt algorithms which keep the number of passes through the dataset as small as possible. In the next section, we will show such an algorithm which is based on the one proposed by [14] for the IQ problem.

In the sequel, given a finite set S and a natural number k, we denote by S^k the set of all the subsets $P \subseteq S$ such that $|P| = k$. Moreover, given two sequences $s = \langle x_1, \ldots, x_n \rangle$ and $s' = \langle y_1, \ldots, y_m \rangle$, we denote by $s :: s'$ the sequence $\langle x_1, \ldots, x_n, y_1, \ldots, y_m \rangle$.

We first define the transformation which, given a FIM problem, constructs a corresponding IQ problem.

Definition 3 (FIM to IQ). *Let $\mathcal{I}$ be an itemset, $\mathcal{D}$ be a dataset, and k be a natural number such that $k \leq m_{\mathcal{D}}$. Then:*

(i) *The alphabet $Q^{\mathcal{I}}$ is defined as the set $\mathcal{I}^k$ (each set in $\mathcal{I}^k$ is a symbol in the alphabet).*

(ii) *For each $T \in \mathcal{D}$, a stream associated with T is a sequence $s_T = \langle I_1, \ldots, I_{n_T} \rangle$ such that*
- *$\{I_1, \ldots, I_{n_T}\} = T^k$*
- *each $I_j \in T^k$ occurs in s_T exactly once.*

(iii) *If $\mathcal{D} = \{T_1, \ldots, T_n\}$, then a stream $s_{\mathcal{D}}$ associated with $\mathcal{D}$ is a sequence*

$$s = s_{T_1} :: s_{T_2} :: \ldots :: s_{T_n}.$$

Notice that, in the above definition, we do not define *the* stream associated with a transaction, but rather *a* stream associated with it. Similarly we talk about *a* stream associated with the dataset $\mathcal{D}$. Indeed, given a transaction T_i there are many ways of constructing a stream s_{T_i} corresponding to it, and consequently, there may be many ways of constructing $s_{\mathcal{D}}$. Actually, the choice of $s_{\mathcal{D}}$ is irrelevant as far as the correctness of the transformation is concerned.

In the next theorem we show that a FIM problem $\mathcal{F}_k(\sigma, \mathcal{D})$ can be mapped into an IQ problem $Q(\vartheta, s_{\mathcal{D}})$ where $s_{\mathcal{D}}$ is any stream constructed as in the previous definition and ϑ is a suitable function of σ, k and $m_{\mathcal{D}}$.

Theorem 1. *Let $\mathcal{I}$ be an itemset, $\mathcal{D}$ be a dataset, k be a natural number such that $k \leq m_{\mathcal{D}}$, and σ be a real number such that $0 < \sigma \leq 1$. Let $Q^{\mathcal{I}}$ and $s_{\mathcal{D}}$ be the alphabet and a stream of queries as in Definition 3. Let also $\bar{e} = \binom{m_{\mathcal{D}}}{k}$.*
If an itemset I is a frequent k-itemset with respect to σ and $\mathcal{D}$, then I is an iceberg query with respect to $Q^{\mathcal{I}}$, $s_{\mathcal{D}}$, and $\vartheta = \frac{\sigma}{\bar{e}}$. Formally:

$$I \in \mathcal{F}_k(\sigma, \mathcal{D}) \Longrightarrow I \in Q(\vartheta, s_{\mathcal{D}}).$$

Proof. Let $|\mathcal{D}| = N$ and $\mathcal{D} = \{T_1, \ldots, T_N\}$. We observe:

(1) $|s_{\mathcal{D}}| = \sum_{i=1}^{N} \binom{|T_i|}{k} \leq \sum_{i=1}^{N} \binom{m_{\mathcal{D}}}{k} = \bar{e}N$

(2) By construction of $s_{\mathcal{D}}$, $|\{T \in \mathcal{D} | I \subseteq I(T)\}| = f_{s_{\mathcal{D}}}(I)$.

$$\begin{array}{ll} & I \in \mathcal{F}_k(\sigma, \mathcal{D}) \\ \Longrightarrow & \{\text{By definition of } \mathcal{F}_k(\sigma, \mathcal{D})\} \\ & |\{T \in \mathcal{D} | I \subseteq I(T)\}| \geq \sigma N \\ \Longleftrightarrow & \{\text{By observation (2)}\} \\ & f_{s_{\mathcal{D}}}(I) \geq \sigma N \\ \Longleftrightarrow & \{\text{algebra}\} \\ & f_{s_{\mathcal{D}}}(I) \geq \frac{\sigma}{\bar{e}} N\bar{e} \\ \Longrightarrow & \{\text{By observation (1) and definition of } \vartheta\} \\ & f_{s_{\mathcal{D}}}(I) \geq \vartheta |s_{\mathcal{D}}| \\ \Longrightarrow & \{\text{by definition of } Q(\vartheta, s_{\mathcal{D}})\} \\ & I \in Q(\vartheta, s_{\mathcal{D}}). \end{array}$$

□

The previous theorem suggests a new technique for finding frequent k-itemsets, which can be sketched as follows:

(1) Construct a stream $s_{\mathcal{D}}$ corresponding to $\mathcal{D}$;
(2) Find the set of Iceberg Queries $Q(\vartheta, s_{\mathcal{D}})$.

It is worth noting that, in our proposal, step (1) and (2) are not performed sequentially, but rather they are interleaved.

Example 1. Let us give a simple example of the above transformation. Let $\mathcal{I} = \{a, b, c, d, e, f\}$ be the underlying set of items and let $\mathcal{D}$ be the following set of transactions:

$$\mathcal{D} = \{\{a, b, d\}, \{a, c, e\}, \{a, d, f\}, \{b, c\}, \{b, d, e\}, \{c, d, f\}\}$$

Assume that we are interested in 2-itemsets which occur in at least $\sigma = 1/3$ of the transactions. It is easy to see that, among all possible 2-itemsets ($\binom{|\mathcal{I}|}{k} = \binom{6}{2} = 15$), only $\{a, d\}$, $\{b, d\}$ and $\{d, f\}$ appear at least twice.

It is worth noting that anti-monotonicity[3] here is not useful at all. In fact, every item (1-itemset) in this example is frequent: it means that 15 candidated 2-itemsets have to be tested by a levelwise algorithm (e.g. Apriori) in a second pass. Even constraint-based techniques (e.g. [15]) cannot be used here since there are no transactions containing less than 2 items (they would be removed since they do not generate any 2-itemset, possibly reducing the search space) and thus the constraint on the size of the itemsets cannot be exploited.

Let us consider now our *FIM-to-IQ* transformation. For each transaction T_i in $\mathcal{D}$, we build a stream s_i associated with it:

$s_1 = \langle\{a, b\}, \{a, d\}, \{b, d\}\rangle$
$s_2 = \langle\{a, c\}, \{a, e\}, \{c, e\}\rangle$
$s_2 = \langle\{a, d\}, \{a, f\}, \{d, f\}\rangle$
$s_4 = \langle\{b, c\}\rangle$
$s_5 = \langle\{b, d\}, \{b, e\}, \{d, e\}\rangle$
$s_2 = \langle\{c, d\}, \{c, f\}, \{d, f\}\rangle$

The stream associated with $\mathcal{D}$ is then $s_{\mathcal{D}} = s_1 :: s_2 :: s_3 :: s_4 :: s_5 :: s_6$. Since $m_{\mathcal{D}} = 3$, in this case the threshold for the IQ problem is set to

$$\vartheta = \frac{\sigma}{\binom{m_{\mathcal{D}}}{k}} = \frac{\frac{1}{3}}{\binom{3}{2}} = \frac{1}{9}$$

Notice that in the stream $s_{\mathcal{D}}$, the "queries" $\{a, d\}$, $\{b, d\}$ and $\{d, f\}$ are the only ones occuring with a frequency of $1/8 \geq \vartheta$. In this case the KPS algorithm only requires $1/\vartheta = 9$ counters instead of the 15 needed by a levelwise algorithm for frequent itemset mining, and only two passes over the dataset (but we need to know $m_{\mathcal{D}}$ in advance). If $|I| > 10000$ and most of the items are frequent, then the number of candidate 2-itemsets can be too large to be fitted in main memory. In such a case an Apriori-like approach will lead to "out of memory errors", while our transformation approach is still effective (see Section 5).

[3] The anti-motonicity property of the itemset frequency asserts that if I is frequent, then all the subsets of I have to be frequent.

4 The Proposed Algorithm

In this section we propose an instance of the technique based on the result of the previous section, which exploits an algorithm for determining Iceberg Queries recently proposed in [14]. Given a stream s and a real number ϑ (called the *threshold*), the algorithm in [14], that we will refer to as the *KPS*-algorithm, requires one pass through the input stream in order to find a superset of the required $Q(\vartheta, s)$. A trivial second pass can be done to find exactly $Q(\vartheta, s)$, keeping the same performance characteristics. In particular, the authors show that their algorithm requires only $O(1/\vartheta)$ memory cells.

In order to exploit the *KPS*-algorithm, given a dataset $\mathcal{D}$, for each transaction $T \in \mathcal{D}$ we feed the *KPS*-algorithm by each k-itemset contained in $I(T)$, where k is the given size of itemsets we are interested in. The parameters for the *KPS*-algorithm are set as suggested in Theorem 1, i.e. the threshold is set to $\frac{\sigma}{\bar{e}}$ (recall that $\bar{e} = \binom{m_{\mathcal{D}}}{k}$, where $m_{\mathcal{D}}$ is the maximal length of a transaction in $\mathcal{D}$). In this way we obtain a two-passes algorithm, with the drawback of having to know the value of $m_{\mathcal{D}}$ in advance. If $m_{\mathcal{D}}$ is not known, a "preprocessing pass" is required. Thus, we obtain a three-passes algorithm that saves more memory (w.r.t the two-passes version) since the length of the stream $s_{\mathcal{D}}$ can be computed exactly in the preprocessing pass, instead of being roughly over-approximated in a pessimistic way by $\binom{m_{\mathcal{D}}}{k}|\mathcal{D}|$.

In the specification of our algorithm (see Algorithm 1), we will use the following notations:

- *KPS*`.init` sets up the data structures for the *KPS*-algorithm;
- *KPS*`.threshold` refers to the parameter ϑ of the *KPS*-algorithm;
- *KPS*`.send(I)` sends the itemset `I` to the *KPS*-algorithm as an element of the stream of queries;
- *KPS*`.output` denotes the set computed by the *KPS*-algorithm.

Notice that Pass 1 (lines 10–12) of Algorithm 1 basically corresponds to run *KPS* for computing a superset of the desired set of queries (itemsets in our case). The second, trivial pass of our algorithm (lines 15–22) filters out from this superset those itemsets which do not occur at least $\sigma \cdot |\mathcal{D}|$ times in the given dataset. In Pass 2, `count` is a data structure used to actually count the number of exact occurrences of an itemset in the given dataset. Notice that the same data structure used to represent the output of the *KPS*-algorithm (typically an hash table) can be used to implement the `count` data structure as well. This is indeed what we have done in our prototype implementation of the algorithm.

The correctness of the algorithm is guaranteed by the correctness of the *KPS*-algorithm and by Theorem 1. As far as space complexity is concerned, the following theorems can be easily proven.

Theorem 2. *Pass 1 of the proposed algorithm computes a superset of* $\mathcal{F}_k(\sigma, \mathcal{D})$ *in* $O(\bar{e}/\sigma)$ *space with one sequential pass through the dataset.*

Algorithm 1 Stream Mining Algorithm

Input: $\mathcal{D}, \sigma, k$
Output: $\mathcal{F}_k(\sigma, \mathcal{D})$, $count(I)\ \forall I \in \mathcal{F}_k(\sigma, \mathcal{D})$

```
1: //Pre-processing pass: some statistics on D are computed
2: streamLength ← 0; dbLength ← 0;
3: for all T ∈ D do
4:    dbLength ← dbLength + 1;
5:    streamLength ← streamLength + (|T| choose k);
6: //Initialize the data structures for KPS
7: KPS.threshold ← σ dbLength/streamLength;
8: KPS.init;
9: //Pass 1: a superset of F_k(σ, D) is computed
10: for all T ∈ D do
11:    for all I ⊆ T s.t. |I| = k do
12:       KPS.send(I);
13: //assert: KPS.output is a superset of F_k(σ, D)
14: //Pass 2: the actual counts of frequent k-itemsets are obtained
15: for all T ∈ D do
16:    for all I ⊆ T s.t. |I| = k do
17:       if I ∈ KPS.output then
18:          count(I) ← count(I) + 1;
19: //infrequent k-itemsets are pruned
20: for all I ∈ KPS.output do
21:    if count(I) ≥ σ · dbLength then
22:       F_k(σ, D) ← F_k(σ, D) ∪ I;
```

Trivially, Pass 2 of the algorithm computes the actual $\mathcal{F}_k(\sigma, \mathcal{D})$ reading once more the dataset, but without requiring further data structures. Hence we have the following corollary.

Corollary 1. *The proposed algorithm computes $\mathcal{F}_k(\sigma, \mathcal{D})$ in $O(\bar{e}/\sigma)$ space with two sequential passes through the dataset.*

5 Experiments

Even if our algorithm performs only two or three passes through the dataset (depending on whether we know $m_{\mathcal{D}}$ in advance or not), both time and space complexity linearly depend from $\bar{e}$. The first thing to keep in mind is that, for the class of applications of the algorithm we have in mind, accessing the dataset is much more expensive than performing main memory operations, and if $\bar{e}$ is not too big, we can perform main memory operations while reading the dataset. For example, let us imagine a scenario in which the huge dataset is stored in different servers, and communications are encrypted (e.g., in the context of distributed data mining). It is easy to understand that, in this context, reducing the number of passes through the dataset is preferable, even at the cost of adding some more local main memory computations.

Anyway, we know that $\bar{e} = \binom{m_{\mathcal{D}}}{k}$ is polynomial in $m_{\mathcal{D}}$ (it belongs to $O(n^k)$). This can lead to huge memory requirements, which is obviously not acceptable. The graph in Fig. 1 shows the actual amount of memory needed by our algorithm, when varying both $m_{\mathcal{D}}$ (the maximal transaction length of the dataset) and k (the size of the frequent itemsets we are looking for), and keeping σ fixed to 1/1000 (be aware of the logarithmic scale used for the y axis).

For example, the graph shows that if $m_{\mathcal{D}} = 16$ and $k = 4$ ($\sigma = 1/1000$) then we need less than 128 Mb. Notice that the space requirements are also linearly dependent on $1/\sigma$. Hence, in the previous example, if $\sigma = 1/100$ then we need only ≈ 13 Mb in order to compute the exact set of all frequent 6-itemsets. These results are obtained by using an hashtable for *KPS* which requires 40 bytes per cell and a load factor of 0.75 (it is the only data structure used in Algorithm 1).

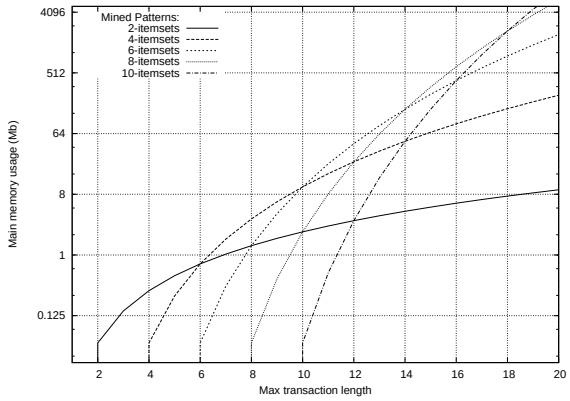

Fig. 1. Memory usage is a function of m_d and k (in this graph σ is fixed to 1/1000)

We believe that these results are quite satisfactory. Notice that they *do not depend on the size of the dataset or the number of possible items* (except for a necessary logarithmic factor due to the number of bits required for the counters) but only on σ, $m_{\mathcal{D}}$ and k.

Furthermore, although the proposed algorithm scales polynomially in the size of the maximal transaction, if the dataset is sufficiently sparse it is possible to divide the transactions into two groups: a small group with the large transactions, and a large group where the proposed algorithm can be applied, containing short transactions. If the former group is small enough, the parameters of the iceberg algorithm would only need to be modified slightly, and the filtering pass would use the original parameter to select the actual frequent itemsets.

We also carried on a set of experiments in order to compare the scalability of our algorithm with the state-of-the-art algorithms FPGrowth [5], Eclat [12], Relim [16] and the well-known Apriori [6]. We generated an artificial dataset Retail-like, by replicating the Retail dataset (from the FIMI Repository [13])

and by inserting 2 different items in each transaction, with the following characteristics:

Number of Transactions	12,497,500
Number of Items	16,470
Size of Largest Transaction	78

Then, we truncated the dataset in order to limit the number of transactions at 1, 2, 3, 4 and 5 millions. On a 512Mb-Ram Pentium machine, Relim was able to find 2-frequent itemsets up to 3 millions of transaction before crashing, while Apriori, FPGrowth and Eclat went up to 4 millions. Only our three-passes algorithm was able to compute all the 2-frequent itemsets in the 5 millions dataset. With respect to time performance, our algorithm, in the current unoptimized Java implementation, is slower than the other algorithms, but it comes out to be very scalable, by requiring 21 minutes for the 1 million dataset, and 41 minutes for the 2 millions dataset.

6 Conclusions and Future Work

We presented an algorithm to compute the exact set of frequent k-itemsets (and their counts) on a given dataset, with reasonable space requirements under the hypothesis that the dataset is sparse. The algorithm is obtained by transforming the set of transactions into streams of k-itemsets and by using an algorithm for iceberg queries in order to compute the output. As far as space complexity is concerned, the algorithm requires an acceptable amount of main memory even for huge datasets, as shown in the paper. We have actually implemented a first, non-optimized version of the algorithm, i.e. without making use of specific (e.g., compressed) data structures. Some experiments on concrete (sparse) datasets are encouraging also as far as time complexity is concerned, showing the effectiveness of the algorithm.

We plan to implement an optimized version of our algorithm that is able to compute all the frequent itemsets with only two passes (having the maximum transaction length in input), running some instances of the current version of the algorithm for different values of k, but saving memory because of the redundancies among different instances (e.g., if the 3-itemset $\{(a, b, c)\}$ is frequent we do not need to store that also the 2-itemsets $\{(a, b), (a, c), (b, c)\}$ are frequent).

Moreover, we plan to study hybrid algorithms which exploit our approach for the first few steps of a level-wise algorithm à la *Apriori*. For example, we can determine in advance suitable bounds for k which ensure that the amount of memory required by our algorithm keeps reasonable in computing x-itemsets, for each level $x \leq k$. Then, we can continue the generation of frequent h-itemsets, for $h > k$, by using a standard level-wise approach. Notice that, for each $x < k$ we can even adopt a simpler version of our algorithm, which performs only the first pass through the dataset (i.e., computing supersets of frequent x-itemsets).

Finally, from the theoretical point of view, we believe that the FIM to IQ transformation presented in the paper can also be useful to get theoretical results

on the trade-off between memory requirements and number of passes over the dataset in the FIM problem. Further work is still needed in this respect.

Acknowledgments. The authors wish to thank the three anonymous reviewers of *KDID05* for their useful comments and suggestions.

References

1. Hipp, J., Güntzer, U., Nakhaeizadeh, G.: Algorithms for association rule mining – a general survey and comparison. SIGKDD Explorations **2** (2000) 58–64
2. Goethals, B.: Survey on frequent pattern mining (2003)
3. Li, W., Han, J., Pei, J.: CMAR: Accurate and efficient classification based on multiple class-association rules. In: ICDM. (2001) 369–376
4. Han, E.H., Karypis, G., Kumar, V., Mobasher, B.: Clustering based on association rule hypergraphs. In: Research Issues on Data Mining and Knowledge Discovery. (1997)
5. 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, May 16-18, 2000, Dallas, Texas, USA, ACM (2000) 1–12
6. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In Bocca, J.B., Jarke, M., Zaniolo, C., eds.: Proc. 20th Int. Conf. Very Large Data Bases, VLDB, Morgan Kaufmann (1994) 487–499
7. Savasere, A., Omiecinski, E., Navathe, S.B.: An efficient algorithm for mining association rules in large databases. In: The VLDB Journal. (1995) 432–444
8. Toivonen, H.: Sampling large databases for association rules. In Vijayaraman, T.M., Buchmann, A.P., Mohan, C., Sarda, N.L., eds.: In Proc. 1996 Int. Conf. Very Large Data Bases, Morgan Kaufman (1996) 134–145
9. Chen, B., Haas, P., Scheuermann, P.: A new two-phase sampling based algorithm for discovering association rules. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM Press (2002) 462–468
10. B. Chen, P.J. Haas, P.S.: Fast: A new sampling-based algorithm for discovering association rules. In: 18th International Conference on Data Engineering. (2002)
11. Goethals, B.: Memory issues in frequent itemset mining. In: Proceedings of the 2004 ACM Symposium on Applied Computing (SAC'04), ACM (2004)
12. Zaki, M.J.: Scalable algorithms for association mining. In: IEEE Transactions on Knowledge and Data Engineering, ACM Press (2000) 372–390
13. 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. In Goethals, B., Zaki, M.J., eds.: FIMI. Volume 90 of CEUR Workshop Proceedings., CEUR-WS.org (2003)
14. Karp, R.M., Papadimitriou, C.H., Shenker, S.: A simple algorithm for finding frequent elements in streams and bags. In: Proceedings of the ACM PODS 2003. Volume 28., ACM (2003)
15. Bonchi, F., Giannotti, F., Mazzanti, A., Pedreschi, D.: Examiner: Optimized level-wise frequent pattern mining with monotone constraint. In: ICDM. (2003) 11–18
16. Borgelt, C.: Keeping things simple: Finding frequent item sets by recursive elimination. In: Workshop Open Software for Data Mining (OSDM05). (2005)