Mining frequent closed itemsets out of core

Claudio Lucchese* Salvatore Orlando† Raffaele Perego‡

Abstract

Extracting frequent itemsets is an important task in many data mining applications. When data are very large, it becomes mandatory to perform the mining task by using an external memory algorithm, but only a few of these algorithms have been proposed so far.

Since also the result set of all the frequent itemsets is likely to be undesirably large, condensed representations, such as closed itemsets, have recently gained a lot of attention. In this paper we discuss the limitations of the partitioning techniques adopted by external memory algorithms for extracting all the frequent itemsets, when applied to closed itemsets mining. The main issue is that the closedness of an itemset cannot be evaluated only using the local knowledge available in a single partition of the input dataset. A further step is thus needed to correctly merge the partial results. We introduce the first algorithm for mining closed itemsets out of core. The algorithm exploits a *divide-et-impera* approach, where the input dataset is split into smaller partitions, such that not only they can be loaded, but also they can be mined entirely into the main memory. Moreover, we devised a simple technique based on a new theoretical result that allows us to reduce the problem of merging partial solutions to an external memory sorting problem.

1 Introduction

Frequent Itemsets Mining (FIM) is a demanding task common to several important data mining applications that looks for interesting patterns within databases (e.g., association rules, correlations, sequences, episodes, classifiers, clusters). The problem can be stated as follows. Let $\mathcal{I} = \{a_1, ..., a_M\}$ be a finite set of items, and $\mathcal{D}$ a dataset containing N transactions, where each transaction $t \in \mathcal{D}$ is a list of *distinct* items $t = \{i_1, ..., i_T\}$, $i_j \in \mathcal{I}$. We call k-itemset a sequence of k *distinct* items $I = \{i_1, ..., i_k\} \mid i_j \in \mathcal{I}$. Given a k-itemset I, let $supp(I)$ be its *support*, defined as the number of transactions in $\mathcal{D}$ that include I. Mining all the frequent itemsets from $\mathcal{D}$ requires to discover all the itemsets having a support higher than (or equal to) a given threshold min_supp. This requires to browse the huge search space given by the power set of $\mathcal{I}$.

The FIM problem has been extensively studied in the last years. Several variations to the original Apriori algorithm [1], as well as completely different approaches, have been proposed [12, 6, 14, 21, 2, 17, 7, 11, 3]. Unfortunately, the collection of frequent itemsets extracted from a dataset is often very large. This makes the task of the analyst hard, since s/he has to extract useful knowledge from a huge amount of frequent patterns.

Closed itemsets are a solution to this problem. They are a condensed, i.e. both concise and lossless, representation of a collection of frequent itemsets. They are concise since a collection of closed itemsets is orders of magnitude smaller than the corresponding collection of frequents. This allows to use very low minimum support thresholds, which would make the extraction of all the frequent itemsets intractable. Moreover, they are lossless, because it is possible to derive the identity and the support of every frequent itemset in the collection from them. Since when we mine closed itemsets, we implicitly discard redundancies, extracting association rules directly from them has been proven to be more meaningful for analysts [18, 20]. Hence, many efficient Frequent Closed Itemsets Mining (FCIM) algorithms have been recently proposed [10, 15, 19, 4, 13, 22, 20].

Several efficient mining algorithms that solve the FIM problem work in-core. Unfortunately, real world datasets may be huge, so that these algorithms cannot store all the data in main memory. To address this issue, a few FIM out-of-core algorithms have been designed [16, 5]. They exploit a *divide-et-impera* approach, by subdividing the original dataset into partitions that can be separately loaded and mined in the main memory. Such out-of-core techniques can be profitably utilized also in case of severe space constraints, e.g. because users have limited capabilities in resource utilization. Consider, for example, a multi-user server, in which single user programs are disallowed to allocate all the main memory available, to avoid swapping all the others.

The problem of mining frequent closed itemsets out-of-core is even tougher. The property of being closed is, in fact, a global property of an itemset in the context of the whole collection of frequent itemsets of the dataset. Itemset closedness can not be thus decided on the basis of the knowledge available in a single partition of the input dataset only. This means that the partitioning-based divide-et-impera approach

---

*University Ca' Foscari of Venice, Department of Computer Science. Via Torino 155, 30172 Mestre (VE), Italy. clucches@dsi.unive.it.

†University Ca' Foscari of Venice, Department of Computer Science. Via Torino 155, 30172 Mestre (VE), Italy. orlando@dsi.unive.it.

‡High Performance Computing Laboratory, ISTI-CNR. Via G. Moruzzi 1, 56126 Pisa (PI), Italy. r.perego@isti.cnr.it.

is harder to apply than in the FIM case. By separately mining with a FCIM algorithm the partitions of a dataset we may in fact generate frequent itemsets that are not globally closed in the whole dataset. These itemsets are to some extent *spurious*, since their existence could be inferred from the closed ones. A further step is thus needed in the FCIM case to correctly merge the partial results obtained, by removing redundancies from the final result.

Another important issue is that most of the in-core FCIM algorithms usually keep the entire collection of frequent closed itemsets mined so far in main memory, for checking whether an itemset is globally closed or not. This makes the realization of an out-of-core FCIM algorithm even more challenging, since it has to deal with strict and predictable memory constraints.

Our final goal is thus to design an intelligent partitioning technique that allows to mine small subsets of the original datasets entirely in main memory, and a merging strategy able to derive the whole collection of closed itemsets from the local results obtained from each partition.

Contribution With this paper we contribute the first algorithm for mining closed itemsets in external memory. We base our algorithm on DCI_CLOSED, a previously proposed, in-core FCIM algorithm [10]. Given a dataset and a minimum support threshold, DCI_CLOSED efficiently performs the mining task using a bounded and predictable amount of memory. This allow us to determine precise bounds on the size of partitions, and to be sure that they can be surely stored and processed separately in main memory using DCI_CLOSED as a mining engine.

To merge partial local results in an efficient way, by fulfilling the requirement concerning memory occupation, we devised a simple technique based on a new theoretical result. This allows the problem of discarding spurious itemsets to be reduced to the problem of external memory sorting.

2 Towards an Out-of-Core Closed Itemsets Mining Algorithm

To design a new out-of-core FCIM algorithm, we used the same framework adopted by state-of-the-art FIM out-of-core algorithms. Since we assume that the whole dataset cannot be mined in the main memory available, we exploit a divide-et-impera approach through the following steps:

1. Subdivide the original dataset into smaller datasets that can be separately processed entirely in main memory.

2. Independently mine each partition in main memory by using a FCIM algorithm, with low and predictable memory requirements.

3. Merge in external memory the local results obtained

from each dataset partition by removing redundancies.

It is clear that the overall effectiveness of this three-phase algorithm depends on the partitioning strategy. The challenge is to devise a partitioning which creates as few subproblems as possible from the original dataset, and that, at the same time, allows a fast merging of the local results in order to get the actual solution of the mining task.

In the following, we will investigate the above three phases of our ideal algorithm. First, in Section 3 we introduce closed itemsets and related issues, and we motivate the choice of an algorithm well suited for mining closed itemsets in-core using bounded amounts of memory. Then, Section 4 discusses some out-of-core FIM algorithm, and their partitioning strategies. One of these strategies will be chosen, by showing its advantages in the context of FCIM. In Section 5 we then describe how to merge local results by removing redundancies. Note that it is important to reduce the size of partitions as much as possible, by pruning from them any unnecessary items and transactions. Section 6 thus discusses how to prune partitions and determine their sizes before subdividing the dataset. Finally, experimental results and concluding remarks are discussed in Section 9.

3 Closed Itemsets Mining Algorithms

The concept of closed itemset is based on the two following functions f and g:

$$f(T) = \{i \in \mathcal{I} \mid \forall t \in T, i \in t\}$$
$$g(I) = \{t \in \mathcal{D} \mid \forall i \in I, i \in t\}.$$

where T and I, $T \subseteq \mathcal{D}$ and $I \subseteq \mathcal{I}$, are subsets of all the transactions and items appearing in $\mathcal{D}$, respectively.

Function f returns the set of items included in all the transactions belonging to T, while function g returns the set of transactions (called *tid-list*) supporting a given itemset I.

DEFINITION 1. *An itemset I is said to be* closed *iff*

$$c(I) = f(g(I)) = f \circ g(I) = I$$

where the composite function $c = f \circ g$ is called Galois operator *or* closure operator.

The closure operator defines a set of equivalence classes over the lattice of frequent itemsets: two itemsets belong to the same equivalence class *iff* they have the same closure, i.e. they are supported by the same set of transactions. We can also show that an itemset I is closed if no superset of I with the same support exists. Therefore mining the *maximal* elements of all the equivalence classes corresponds to mining all the closed itemsets.

Fig. 1(b) shows the lattice of frequent itemsets derived from the simple dataset reported in Fig. 1(a), mined with
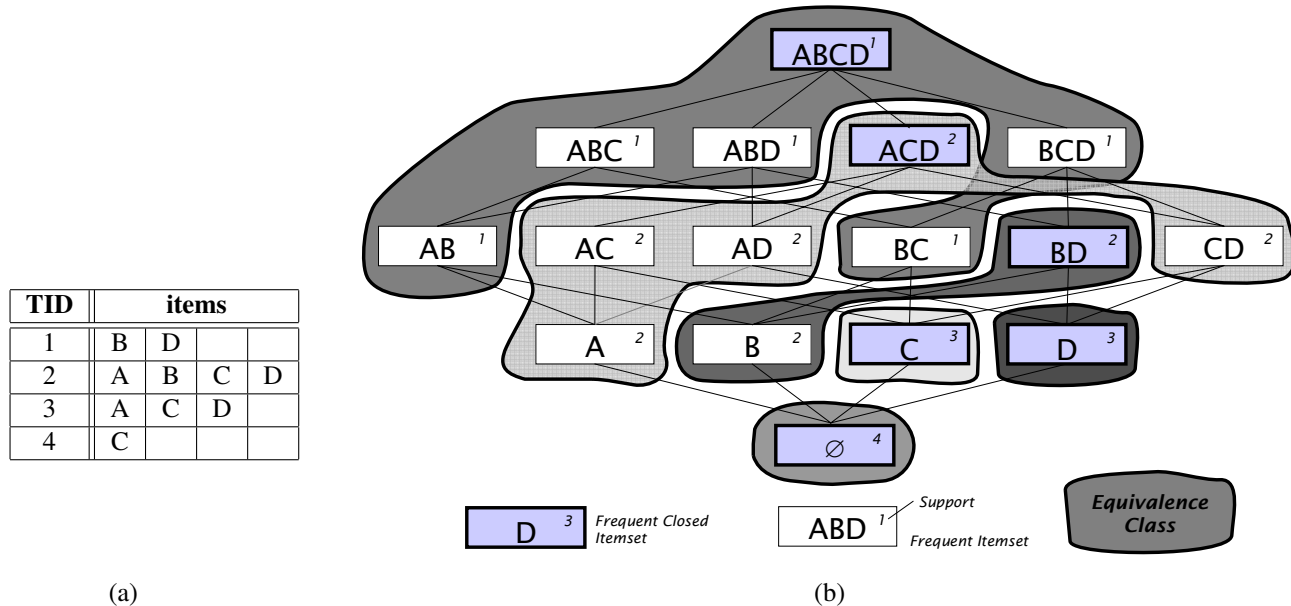
TID	items			
1	B	D		
2	A	B	C	D
3	A	C	D	
4	C			

(a)

(b)

Figure 1: (a) The input transactional dataset $\mathcal{D}$, represented in its horizontal form. (b) Lattice of all the frequent itemsets ($min\_supp = 1$), with closed itemsets and equivalence classes.

$min\_supp = 1$. We can see that the itemsets with the same closure are grouped in the same equivalence class. Each equivalence class contains elements sharing the same supporting transactions, and closed itemsets are their maximal elements. Note that closed itemsets (six) are remarkably less than frequent itemsets (sixteen).

3.1 Visiting the FCIM Search Space and Detecting Duplicates.

The goal of an effective visiting strategy should be to identify exactly a single itemset for each equivalence class. We could in fact mine all the closed itemsets by computing the closure of just this single representative itemset for each equivalence class. Let us call these representative itemsets *closure generators*.

The most efficient FCIM algorithms use a technique that we call *closure climbing*. As soon as a generator is devised, its closure is computed, and new generators are built as supersets of the closed itemset discovered so far. Since closed itemsets are the maximal elements of their own equivalence classes, this strategy always guarantees to jump from an equivalence class to another. Unfortunately, it does not ensure that the new generator belongs to an equivalence class that was not yet visited. Hence, it may happen to visit multiple times the same equivalence class. For example, in Fig. 1 we can see that both $\{A, C\}$ and $\{C, D\}$ are generators of the same closed itemset $\{A, C, D\}$, and they can be obtained as supersets of the closed itemsets $\{C\}$ and $\{D\}$, respectively.

We need thus to introduce some *duplicate* checking technique in order to avoid generating multiple times the same closed itemset. The following *Subsumption Lemma* can be used to identify duplicate generators:

LEMMA 3.1. (SUBSUMPTION LEMMA) *Given two itemsets X and Y, if $X \subset Y$ and $supp(X) = supp(Y)$ (i.e., $|g(X)| = |g(Y)|$), then $c(X) = c(Y)$.*

Proof. If $X \subset Y$, then $g(Y) \subseteq g(X)$. Since $|g(Y)| = |g(X)|$ then $g(Y) = g(X)$. $g(X) = g(Y) \Rightarrow f(g(X)) = f(g(Y)) \Rightarrow c(X) = c(Y)$.

Therefore, given a generator X, if we find an already mined closed itemsets Y that set-includes X, where the supports of Y and X are identical, we can conclude that $c(X) = c(Y) = Y$. In this case we also say that Y *subsumes* X. If this holds, we can safely prune the generator X without computing its closure. Otherwise, we have to compute $c(X)$ in order to obtain a new closed itemset.

Several algorithms, like CHARM, CLOSET, and CLOSET+ [22, 15, 19, 4], base their duplicate avoidance technique on this Lemma. For example, CHARM exploits a hash table to quickly individuate all the already mined closed itemsets Y that subsume a given itemset X.

Unfortunately, this technique may become expensive, both in time and space. In time, because it requires searching the possibly huge set of closed itemsets mined so far for the inclusion of each generator. In space, because in order to efficiently perform set-inclusion checks, all the closed

itemsets have to be kept in the main memory, which means that the size of the output is a lower bound to the space complexity of the algorithm. Unfortunately, when low minimum support threshold are used, it may happen to extract a huge number of closed itemsets, so that maintaining them in main memory for searching purposes may become unfeasible.

3.2 DCI_CLOSED**: our Mining Engine.** In our context we need an FCIM algorithm that meets two important requirements: the amount of memory used must be as *low* as possible and, more importantly, it must be *predictable*. Meeting both these requirements is a prerequisite to the possibility of devising an effective partition strategy able to produce dataset partitions that can be mined respecting a given maximum memory constraint. To the best of our knowledge, the only FCIM algorithm respecting the above requirements is DCI_CLOSED [8, 10].

DCI_CLOSED exploits a divide-et-impera strategy and a bitwise vertical representation of the database. It has been proven to outperforms other state-of-the-art algorithms on most dataset, and furthermore, due to its space efficiency, it completes successfully the mining tasks on large input datasets and with low support thresholds that cause all the other algorithms to fail.

Moreover, since DCI_CLOSED does not need to store the set of closed itemsets mined so far in the main memory, it turns out to have memory requirements much lower than other algorithms. This is because it is based on an innovative strategy to visit the search space, which is derived from an original theoretical framework that formalizes the problem of mining closed itemsets in detail. Differently from other algorithms, DCI_CLOSED exploits duplicate checking just looking at a subset of the original dataset stored in a vertical bitwise format. Thanks to its optimizations, this subset turns out to be pretty small, and experiments have shown that this duplicate checking technique is faster than those directly based on Lemma 3.1.

Lastly, the space complexity of the algorithm depends only on the dataset and can be easily upper bounded. DCI_CLOSED just need the original dataset and the tid-lists of the nodes along the path of the depth first visit along the lattice. This path can be long at most M nodes. Since DCI_CLOSED projects the dataset at the first level of the visit, it requires at most $(3M) \times N$ bits to run over a dataset $\mathcal{D}$ with a minimum support threshold equal to one.

4 Partitioning Strategies

One common feature of FCIM algorithms is that they need to exploit a global knowledge on the dataset at any time of the computation.

This is a tough problem which we have to consider with attention when discussing partitioning strategies suitable for closed itemsets mining algorithms. In fact, the global knowledge required regards the whole dataset, and not just the single partition currently considered. This is because, by definition, to state whether an itemset is closed or not, we need all the transactions supporting it. In the following we analyze the different partitioning strategies of two out-of-core FIM algorithms, *Partition* and *DiskMine*, and discuss their advantages and disadvantages with particular regards to the FCIM problem.

4.1 Partitioning of the Input Dataset. This is the approach adopted by *Partition*, a level-wise apriori-like FIM algorithm that reads the database at most twice to generate all frequent itemsets. *Partition* is based on two main ideas. The first one is to divide the dataset in disjoint partitions that can fit in main memory one at the time, and the second one is that every frequent itemset must be frequent in at least one of these partitions.

Firstly, the dataset is partitioned horizontally, and local frequent itemsets are mined separately from each partition. By summing up the local supports of itemsets we can determine their global support in the original dataset. Unfortunately, some frequent itemsets may happen to be infrequent in some partitions, and thus their precise support is not returned by the algorithm. If this is the case, a second scan is required to calculate the correct global support of these itemsets.

Partition exploits a proper partitioning of the dataset, since it splits the dataset into disjoint subsets of transactions which cover the whole dataset. Each subset can be mined separately, but false positives, i.e. itemsets that are locally frequent in some partition but result to be globally infrequent, may be created. Returning to the FCIM problem, if we adopt a similar partitioning strategy an even worse problem arises with the closed-ness property. In fact, an itemset which is not closed in a partition may be closed when considering the whole dataset. This means that not only we have to discriminate between false and true positives (local and global frequent patterns), but also between false and true negatives, i.e., globally closed itemsets that result not to be closed in some of the partitions of the dataset.

In [9], we have shown that it is however possible to reconstruct the whole set of global closed itemsets even if some closed itemset is not present in any of the sets of local results. Suppose we have two partitions $\mathcal{D}_1$ and $\mathcal{D}_2$, and the two collection, $\mathcal{C}_1$ and $\mathcal{C}_2$, of the closed itemsets mined from them. In this case, the global solution is made by all the closed itemsets mined locally, plus the result of the intersections between any couple of itemsets in the cartesian product $\mathcal{C}_1 \times \mathcal{C}_2$. This result can be easily generalized to the case of P partitions by first merging the two collections $\mathcal{C}_1$ and $\mathcal{C}_2$, then merging this partial result with $\mathcal{C}_3$ and so on.

The cost of the merging step is however very high. As-

suming that a naïve algorithm for merging two sets of partial results takes $|\mathcal{C}^*|^2$ time, where $|\mathcal{C}^*|$ is the average number of local closed itemsets, we will have an overall complexity of about $|\mathcal{C}^*|^P$. The merging phase thus becomes rapidly intractable as the number of partitions increases. The last disatvantage of this approach is that in order to perform the mergin efficiently, each local collection of closed itemsets should be stored in main memory.

4.2 Partitioning the Search Space. *DiskMine* is an FP-GROWTH based FIM algorithm. FP-GROWTH stores the transactions in a trie-like data structure named FP-tree. The initial FP-tree is then recursively projected item by item, thus visiting the whole lattice of frequent itemsets. The idea behind *DiskMine* is that, even if the whole dataset may be large, every projection on single items is likely to be very small. Therefore instances of FP-GROWTH can be run on these projections in main memory. Differently from the horizontal partitioning technique, the set of itemsets mined from each projection produce a proper partitioning of the global collection of frequent itemsets with complete support information. Therefore there is no need for a post processing phase for merging the results or a second scan for calculating correct supports, but it is enough to gather local results.

The projection-based partitioning strategy used by FP-GROWTH may be used within any FIM algorithm. It works as follows. Given a total order $\prec$ among single items $\mathcal{I}$, we put all transactions containing the first item i_1 in the first projection $\mathcal{D}_{i_1}$, then all the transactions containing the second item i_2 in the second projection $\mathcal{D}_{i_2}$, but deleting every occurrence of i_1, and so on. Finally, we independently mine frequent itemsets starting with i_1 from $\mathcal{D}_{i_1}$, then itemsets starting with i_2 from $\mathcal{D}_{i_2}$, and so on. Note that the results sets generated from the various projections are disjoint by construction.

More formally, let $\mathcal{D}_i$ be a projection-based partition of $\mathcal{D}$ over the item $i \in \mathcal{I}$, defined as follows:

$$\mathcal{D}_i = \{t' = t \setminus \{j \in t \mid j \prec i\} \mid t \in \mathcal{D} \wedge i \in t\}.$$

$\mathcal{D}_i$ is thus built only from those transaction t in the original dataset that contain i by removing all the items preceding i according to the total order $\prec$.

DiskMine merges many of such projections together in order to minimize the number of partitions and therefore the number of disk accesses. A possible way is to combine partitions of datasets which have been projected over contiguous items in the total order $\prec$. We thus indicate with $\mathcal{D}_{[x,y)}$ the projected dataset obtained by merging all the projected datasets $\mathcal{D}_i, \forall\, i \in [x,y)$. Formally, we have that:

$$\mathcal{D}_{[x,y)} \equiv \{t' = t \setminus \{j \in t \mid j \prec x\} \mid t \in \mathcal{D} \wedge \exists i \in t \mid x \preceq i \prec y\}.$$

Given the sorted set of single items $\mathcal{I} = \{i_1, \dots, i_M\}$, we can thus create P partitions $\mathcal{D}_{[p_0,p_1)}$, $\mathcal{D}_{[p_1,p_2)}$, ..., $\mathcal{D}_{[p_{P-1},p_P]}$ of the dataset where $i_1 = p_0 \prec p_1 \prec \cdots \prec p_P = i_M$, such that each partition can be mined entirely in main memory. Note that during the mining phase, a FIM algorithm must extract only those (lexicographically ordered) itemsets starting with an item in $[x,y)$.

The above strategy guarantees the possibility of independently mining each projection in order to get the whole set of frequent itemsets. Unfortunately this does not hold when mining closed itemsets. This is because each partition does not enclose knowledge about the global collection of closed itemsets, and therefore it is not possible to locally understand whether an itemset is globally closed or not.

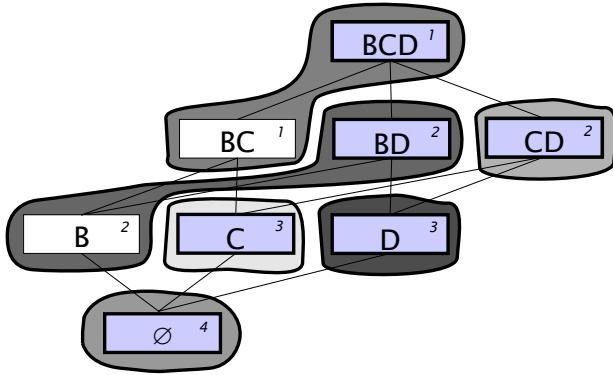| TID	items			
2	A	B	C	D
3	A	C	D	

(a)
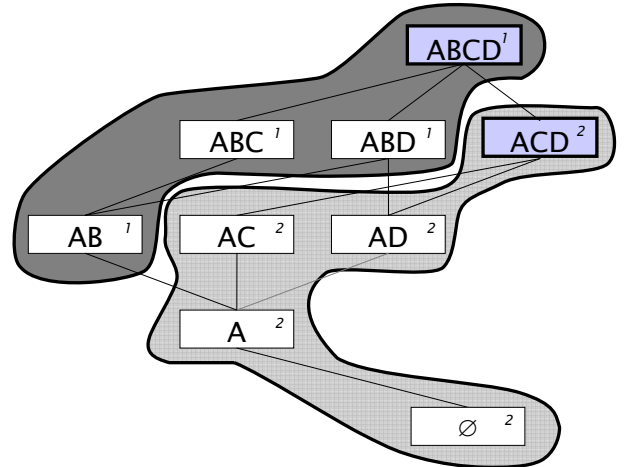
(b)

Figure 2: (a) The projected transactional dataset $\mathcal{D}_{[A,B)}$, represented in its horizontal form. (b) The lattice of all the frequent itemsets in $\mathcal{D}_{[A,B)}$ ($min\_supp = 1$), with closed itemsets and equivalence classes highlighted.

For example, consider Figure 1, which shows a dataset $\mathcal{D}$ and its frequent closed itemsets extracted with $min\_supp = 1$. Note that the items are lexicographically ordered. Consider now that from $\mathcal{D}$, we can build two projected datasets $\mathcal{D}_{[A,B)} \equiv \mathcal{D}_A$ (see Figure 2), and $\mathcal{D}_{[B,D]}$ (see Figure 3), where $\mathcal{D}_{[B,D]}$ is the projected dataset obtained by merging $\mathcal{D}_B$, $\mathcal{D}_C$, and $\mathcal{D}_D$. When we extract frequent closed itemsets from the two projections, the closed itemsets mined from $\mathcal{D}_{[B,D]}$ are incorrect. We can see that itemsets

TID	items		
1	B	D	
2	B	C	D
3	C	D	
4	C		

(a)

(b)

Figure 3: (a) The projected transactional dataset $\mathcal{D}_{[B,E)}$, represented in its horizontal form. (b) The lattice of all the frequent itemsets in $\mathcal{D}_{[B,E)}$ ($min\_supp = 1$), with closed itemsets and equivalence classes evidenced.

$\{B,C,D\}$ and $\{C,D\}$ are locally closed in $\mathcal{D}_{[B,D]}$, but they are not globally closed in $\mathcal{D}$ since they are *subsumed* (see Lemma 3.1) by $\{A,B,C,D\}$ and $\{A,C,D\}$, extracted from $\mathcal{D}_{[A,B)}$.

It is clear that, we can decide locally whether an itemset in $\mathcal{D}_{[B,D]}$ is globally closed or not, becaues there is no knowledge about the occurrences of items preceeding B in the ordering $\prec$. Note that this is different than with horizontal partitioning, where we do not have information about other transaction outside the current projection. Conversely, in this case we just miss information about the items pruned out from the current projection.

It is easy to show that eventually every frequent closed itemset is mined in some of the projections. Given one closed itemset $X \in \mathcal{C}$, where $i' = min_{\prec}(i \in X)$, there must exist one $\mathcal{D}_{[x,y)}$ such that $x \preceq i' \prec y$. Since such projected partition contains by construction all the items of X and all its supporting transactions, X will be returned as a closed itemset from $\mathcal{D}_{[x,y)}$.

If we denote with $\mathcal{C}$ the set of closed itemsets of $\mathcal{D}$, and with $\mathcal{C}_1, \ldots, \mathcal{C}_P$ the closed itemsets extracted from P partitions of the original dataset $\mathcal{D}$, then the following surely holds:

$$\mathcal{C} \subseteq (\mathcal{C}_1 \cup \ldots \cup \mathcal{C}_P) \cup \{\emptyset\}.$$

Note that, since during the mining of each partition $\mathcal{D}_{[x,y)}$, the mining algorithm must extract only those lexicographically (based on $\prec$) ordered itemsets starting with an item in $[x, y)$, the empty set can not be extracted from any projected partition, and therefore must be considered separately.

At first glance, it seems easier to use such projection-based partitioning and to remove some non-closed itemset from the result set, rather than using the horizontal partitioning and constructively derive the solution set. In the next section, we will show that this intuition is true, by reducing the problem of finding such spurious itemsets to the one of external memory sorting.

5 Spurious Itemsets Setection in Search Space Partitioning Approaches

We have seen that some locally closed itemset may be non-closed globally. We referred to these non-closed itemsets as *spurious*. Every spurious itemset X is simply a frequent itemset such that $X \neq c(X)$, and therefore it is an additional representative of the equivalence class of $c(X)$.

In order to detect such redundant itemsets, we could use the usual duplicate detection technique based on the subsumption Lemma 3.1, once that we have mined all the partitions. Given a itemset X which is closed in the partition $\mathcal{D}_{[x,y)}$, we must check whether it is *subsumed* by some other itemset Y mined in some other partitions. Since we left out from $\mathcal{D}_{[x,y)}$ only those items preceeding x according to our ordering $\prec$, we need to look for such Y only among itemsets mined from those projections $\mathcal{D}_{[s,t)}$ where $t \preceq x$.

Unfortunately, also in this case, in order to perform fast searches, it would be necessary to store itemsets mined from those partitions $\mathcal{D}_{[s,t)}$ in memory resident data structures, like the one proposed in [22]. But, even if we analyze those partitions $\mathcal{D}_{[s,t)}$ one at time, we have no guarantee that they would fit in main memory.

In the following, we introduce Lemma 5.1, which suggests a different and innovative technique for detecting spurious itemsets that can be efficiently implemented in an external memory algorithm.

LEMMA 5.1. *Let $\mathcal{C}$ be the collection of closed itemsets in the input dataset $\mathcal{D}$, and let $\mathcal{C}_1, \ldots, \mathcal{C}_P$ be the collections of closed itemsets mined from the P partitions of the original dataset $\mathcal{D}$, respectively $\mathcal{D}_{[p_0,p_1)}, \mathcal{D}_{[p_1,p_2)}, \ldots, \mathcal{D}_{[p_{P-1},p_P]}$, where $\mathcal{I} = \{i_1, \ldots, i_M\}$ are sorted ascendingly according to some order $\prec$ and $i_1 = p_0 \prec p_1 \prec \cdots \prec p_P = i_M$.*

If $X \in \mathcal{C}_i$ and X is not globally closed in $\mathcal{D}$, then there must exist an itemset $Y \in \mathcal{C}_{j \neq i}$ with $Y \supset X$ and $supp(X) = supp(Y)$ such that X is a suffix of Y.

Proof: If $X \in \mathcal{C}_i$ is not closed in $\mathcal{D}$, then there must exist an itemset $Y \in \mathcal{C}$ such that $Y \supset X$ and $supp(X) = supp(Y)$.

422

Since $\mathcal{C} \subseteq \{\mathcal{C}_1, \ldots, \mathcal{C}_P\} \cup \{\emptyset\}$ and since X is closed in C_i, then there exist $C_{j \neq i}$ such that $Y \in C_j$. Let us focus on the items in $\{Y \setminus X\}$. By construction of the various partitions, these items may only preceed the items in X. Thus, since $\forall i \in \{Y \setminus X\}$, $i \prec j, \forall j \in X$, we have that X is a suffix of Y. $\square$

The above Lemma simply says that if X belongs to some local result set $\mathcal{C}_i$ but it is not globally closed in $\mathcal{D}$, then a superset Y of X with the same support must have been mined in some other partition, and X is a suffix of Y.

EXAMPLE 1. *Consider $\mathcal{C}_1$ the closed itemsets extracted from $\mathcal{D}_{[A,B]}$ (see Figure 2), and $\mathcal{C}_2$ the closed itemsets extracted from $\mathcal{D}_{[B,D]}$ (see Figure 3).*

Given a non globally closed itemset $X \in \mathcal{C}_2$, e.g. $X = \{B, C, D\}$, by applying Lemma 5.1, we know that there must exists an itemset $Y \in \mathcal{C}_1$ such that X is subsumed by Y, and X is a suffix of Y. This itemset actually exists, and it is $Y = \{A, B, C, D\}$.

The above lemma suggests a very simple method to identify spurious closed itemsets extracted from distinct partitions. This method is not expensive and can be efficiently implemented by using an external memory algorithm.

First of all, it is worth noting that given any two itemsets $X \in C_i$ and $Y \in C_j$ such that X is subsumed by Y, if we sort X and Y in *descending* order rather than ascending, than X is a *prefix* of Y. Thus, let us consider the list $\mathcal{L}_X$ made with the descendingly sorted items of the itemset X preceded by its support value. We can easily show that if $\mathcal{L}_X$ is a prefix of $\mathcal{L}_Y$, than X is subsumed by Y. This condition in fact ensures that both the subsumption conditions, $Y \supset X$ and $supp(X) = supp(Y)$, actually holds.

In order to detect spurious itemsets, we materialize such lists $\mathcal{L}_X$ from the sets of all the locally mined itemsets, and then we sort all the lists in ascending lexicographic order. This sorting is done in external memory by using a multiway merge-sort algorithm. We read chunks of lists $\mathcal{L}_X$ in a buffer of predefined size. When the buffer is full, we sort it in-core before dumping it to the disk. Finally, a multi-way merge algorithm is applied to get a single sorted set of lists. Detection and removal of spurious itemsets can be done easily during the multi-way merge step: if itemset X is spurious, then the itemset Y that subsumes X can only have an associated $\mathcal{L}_Y$ that comes immediately after $\mathcal{L}_X$.

EXAMPLE 2. *Consider the sets $\mathcal{C}_1$ and $\mathcal{C}_2$ of the closed itemsets extracted from $\mathcal{D}_{[A,B]}$ (see Figure 2), and $\mathcal{D}_{[B,D]}$ (see Figure 3), respectively. From them we obtain the following lists $\mathcal{L}_X$:*

$\mathcal{D}_{[A,B]}$		
supp	*Closed itemset*	*List*
2	ACD	$\mathcal{L}_{ACD} = \textbf{2}, D, C, A$
1	ABCD	$\mathcal{L}_{ABCD} = \textbf{1}, D, C, B, A$

$\mathcal{D}_{[B,D]}$		
supp	*Closed itemset*	*List*
3	C	$\mathcal{L}_C = \textbf{3}, C$
3	D	$\mathcal{L}_D = \textbf{3}, D$
2	BD	$\mathcal{L}_{BD} = \textbf{2}, D, B$
2	CD	$\mathcal{L}_{CD} = \textbf{2}, D, C$
1	BCD	$\mathcal{L}_{BCD} = \textbf{1}, D, C, B$

Once the lists $\mathcal{L}_X$ associated with the various itemsets X are built and stored on disk, we can sort them by using an external memory algorithm. In our example, we eventually obtain:

$\mathcal{L}_{BCD}$	=	$\textbf{1}, D, C, B$	*non closed*
$\mathcal{L}_{ABCD}$	=	$\textbf{1}, D, C, B, A$	
$\mathcal{L}_{BD}$	=	$\textbf{2}, D, B$	
$\mathcal{L}_{CD}$	=	$\textbf{2}, D, C$	*non closed*
$\mathcal{L}_{ACD}$	=	$\textbf{2}, D, C, A$	
$\mathcal{L}_C$	=	$\textbf{3}, C$	
$\mathcal{L}_D$	=	$\textbf{3}, D$	

Since the two lists $\mathcal{L}_{BCD}$ and $\mathcal{L}_{CD}$ result to be prefixes of lists $\mathcal{L}_{ABCD}$ and $\mathcal{L}_{ACD}$, respectively, which occur in the next two positions, the two associated itemsets $\{B, C, D\}$ and $\{C, D\}$ can be safely discarded being spurious itemsets.

As we mentioned before, the closure of the empty set must be considered separately. Since if $c(\emptyset) \neq \emptyset$, then $c(\emptyset)$ would be mined from some partition, we must only consider the case corresponding to $c(\emptyset) = \emptyset$. Note that $c(\emptyset) \neq \emptyset$ only if the most frequent itemset appears in all the transactions of $\mathcal{D}$, i.e. $\forall i \in c(\emptyset)$, $supp(i) = |\mathcal{D}|$. Therefore we must add the empty set to the collection of globally closed itemsets only when no item has support equal to $|\mathcal{D}|$.

6 Creating Partitions

The choice of the projected partitions is actually the first step of our out-of-core algorithm. Given a dataset $\mathcal{D}$, a minimum support threshold min_supp, and a maximum memory size Mem_{size}, we must create the minimum number of partitions such that they can be entirely mined in at most Mem_{size} bytes. Since partitions are built by merging dataset projections of contiguous single items, we can reformulate the problem as follows: given an item $k \in \mathcal{I}$, we have to find the

largest n such that the size of $\mathcal{D}_{[k,k+n)}$ is less than (or equal to) Mem_{size}.

Before starting to subdivide $\mathcal{D}$, we scan the input dataset in order to find the frequent single items $\mathcal{F}_1$, $|\mathcal{F}_1| = M$. Since infrequent items will not contribute to the collection of frequent closed itemsets, we can remove them without affecting the correctness of the algorithm. In this way we obtain a much smaller initial dataset. Moreover, relative frequencies of frequent single items are used to set the global order $\prec$. In fact, it has been shown that by sorting items per ascending frequency order, we better balance the size of the projected partitions and reduce the search space [5].

Hereinafter, we thus assume that the dataset $\mathcal{D}$ to be subdivided does not contain infrequent items, while the frequent items have been re-mapped to $[0, M)$, where 0 $(M-1)$ corresponds to the least (most) frequent item in $\mathcal{F}_1$.

Given a dataset and a minimum support threshold, the amount of memory required for mining frequent closed itemsets depends, of course, on the specific algorithm exploited. DCI_CLOSED uses a limited amount of memory, very close to the size of the bitwise vertical representation of the dataset. As discussed in Section 3.2, DCI_CLOSED memory usage can be accurately estimated. Let $I_{[k,k+n)}$ be the set of frequent single items in the projection $\mathcal{D}_{[k,k+n)}$, then the memory required by DCI_CLOSED is bounded by $(3 \cdot |I_{[k,k+n)}| \times |\mathcal{D}_{[k,k+n)}|)$ bits.

Therefore, given Mem_{size}, we must estimate the cardinality of $\mathcal{D}_{[k,k+n)}$ and $I_{[k,k+n)}$ for every potential partition in order to choose the most suitable partition schema. We can easily compute the size of $\mathcal{D}_{[k,k+n)}$, for every possible value of k and n, during the second scan of $\mathcal{D}$ by using a fixed number of counters. Note that for any given k, we have $M-k$ distinct possible choices for n. We thus need a number of counters equal to $\sum_{k=0}^{M-1}(M-k) = M \cdot (M+1)/2$.

We also need an estimate for the number of frequent single items $I_{[k,k+n)}$ appearing in every possible partition. A broad over-estimation of this number is obviously $M-k$. The estimate can be however more precise if we exploit the knowledge of $\mathcal{F}_2$, i.e. the frequent 2-itemsets in $\mathcal{D}$. During the second dataset scan, we can also compute $\mathcal{F}_2$, by using further $\binom{M}{2}$ counters.

In particular, given a projected dataset $\mathcal{D}_h = \mathcal{D}_{[h,h+1)}$, only h and those items x, $h < x < M$ such that $\{h, x\} \in \mathcal{F}_2$, may belong to frequent closed itemsets mined from $\mathcal{D}_h$. All the other items x corresponding to infrequent pairs can be pruned from $\mathcal{D}_h$. thus:

$$I_{[h,h+1)} = \{h\} \cup \{x \mid h < x < M \ \land \{h, x\} \in \mathcal{F}_2\}.$$

Moreover, if there are no pairs $\{h, x\}$, $h < x < M$, such that $\{h, x\} \in \mathcal{F}_2$, we can avoid mining $\mathcal{D}_h$ at all, since we can not surely extract any frequent closed itemset other than $\{h\}$.

Finally, when we merge n consecutive projected datasets in the partition $\mathcal{D}_{[k,k+n)}$, the number of items that can appear in the transactions of $\mathcal{D}_{[k,k+n)}$ are:

$$\left| \bigcup_{\substack{\forall h \in [k,k+n) \ s.t. \\ I_{[h,h+1)} \neq \{h\}}} I_{[h,h+1)} \right|$$

Note that the above technique requires that M^2 counters be stored in main memory. If M is very large, it may happen that the memory required to implement such technique exceeds Mem_{size}. In this case, to fulfill memory constraints, it is however possible to partition the set of counters, and perform more scans of the dataset.

7 An Algorithm for Mining Frequent Closed Itemsets from Secondary Memory

The pseudo-code of DCI_CLOSED_OOC, our out-of-core FCIM algorithm, is illustrated in Algorithm 1.

The first step relies on two scans of the dataset, during which the horizontal dataset is pruned, and decisions are taken concerning the number and size of partitions.

In the second step the various partitions $\mathcal{D}_{[k,k+n)}$ are mined by using DCI_CLOSED as FCIM mining engine. Note that DCI_CLOSED must extract from each partition $\mathcal{D}_{[k,k+n)}$ only the frequent closed itemsets whose first item belong to $[k, k+n)$. Moreover the list $\mathcal{L}_X$ associated with each closed itemset mined are written to disk for the following step.

Finally, the third step deals with removing non closed itemsets in order to obtain the exact result. It is carried out by means of the external memory sorting algorithm depicted in Section 5.

8 Experimental Evaluation

We conducted a bunch of experiments on a Linux PC equipped with a 2GHz Pentium Xeon processor and 1GB of random-access memory. Three large datasets were used:

- Webdocs. It contains 5,267,657 distinct items in 1,692,082 transactions. The dataset is about 1.4 GB large, and is available from the FIMI repository.

- USCensus1990. It contains 397 distinct items in 2,458,285 transactions. The dataset is about 520 MB large, and is available from the UCI machine learning repository.

- Artificial2GB. The last dataset was synthesised using the IBM generator. It contains 3,000 distinct items in 1,330,293 transactions, and it is about 2 GB large.
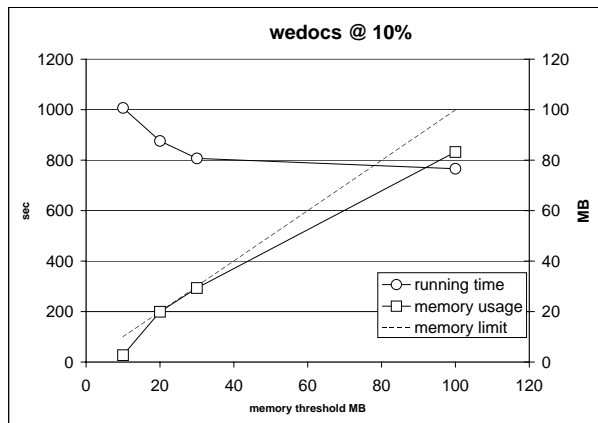
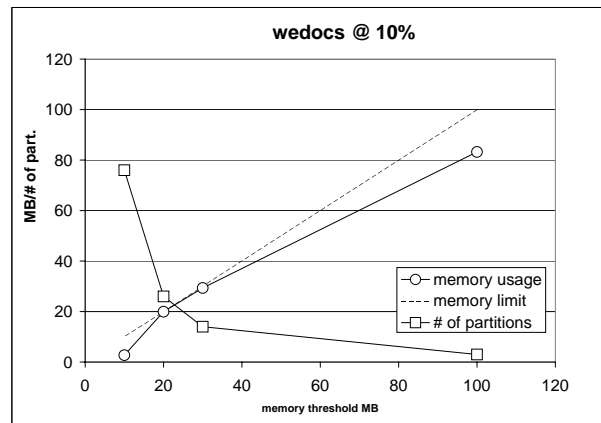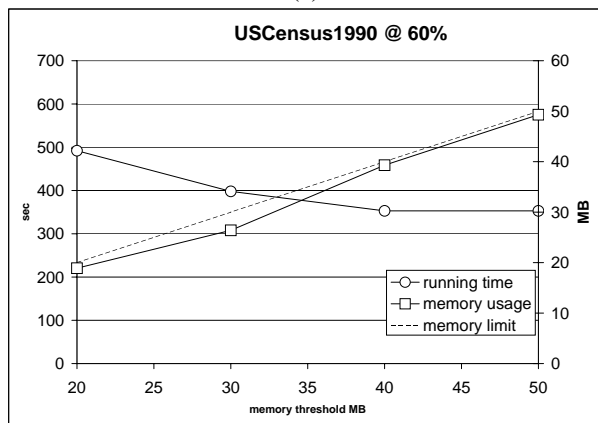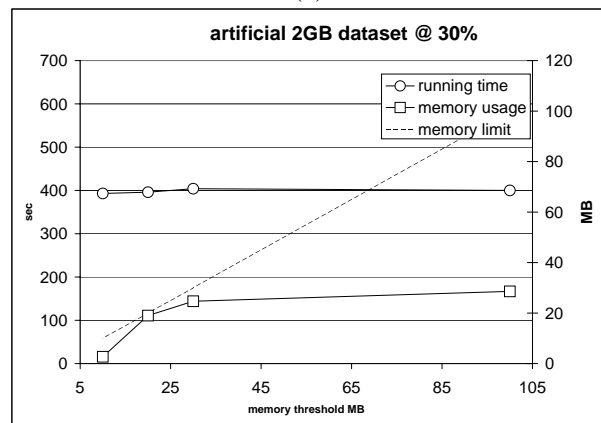In Figure 4(a,b) we compare on the Webdocs dataset the performance of DCI_CLOSED_OOC with FP-CLOSE (a

424
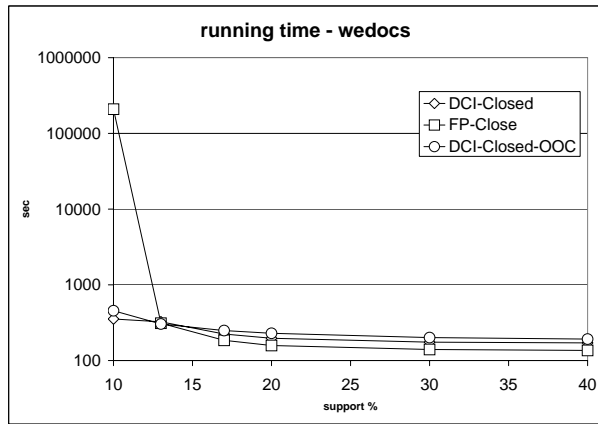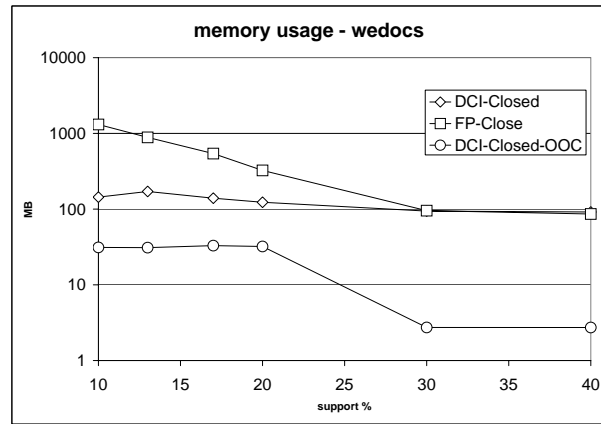
Figure 4: Results of the experiments conducted on two real world datasets and an artificial one.

Algorithm 1 DCI_CLOSED_OOC pseudocode

Step 1: Partitioning. Scan $\mathcal{D}$ twice to make decisions about projected partitions.

1: Scan $\mathcal{D}$ for the *first time* to find out the set of frequent items $\mathcal{F}_1$ and their supports, where $|\mathcal{F}_1| = M$.
2: Scan $\mathcal{D}$ for the *second time*. During the scan: (a) prune transactions on-the-fly by removing infrequent items, and re-map frequent items into the interval $[0, M)$; (b) compute $\mathcal{F}_2$ and collect the information about memory occupancy of all possible partitions.
3: Choose the most suitable partitioning schema by considering the given memory constraint Mem_{size}, and save such information for the following step.

Step 2: Mining. Run DCI_CLOSED to extract frequent closed itemsets from all the partitions.

1: For each partition $\mathcal{D}_{[k,k+n)}$, DCI_CLOSED scan $\mathcal{D}$, create on the fly an in-core (bitwise) vertical representation of $\mathcal{D}_{[k,k+n)}$, and mine from it all the closed itemsets whose first item belong to $[k, k+n)$. All closed itemsets mined are written to disk as lists $\mathcal{L}_X$.

Step 3: Merging. Remove spurious itemsets, and returns the final set of closed itemsets.

1: Run the *external memory* sorting algorithm to lexicographically order all the lists $\mathcal{L}_X$ stored on disk.
2: Remove non closed itemsets by discarding every list $\mathcal{L}_X$ that is a *prefix* of the list that occurs immediately after, and output the final result.

---

fast implementation of CLOSET+ available from the FIMI repository), and our in-core FCIM algorithm DCI_CLOSED. Note that we imposed DCI_CLOSED_OOC to run by using at most 30MB of memory. The aim of this test is to quantify the overhead introduced by our three-steps mining approach: partitioning, separate mining, and merging. From the plot reported in Figure 4(a) we can see that the execution times of the three algorithms are comparable for most of the support thresholds experimented. This means that the overhead introduced does not affect the overall performance remarkably, thus making our out-of-core approach not only viable in the cases where severe memory constraints really exist, but also efficient. Note that in the test conducted with the lowest support threshold, FP-CLOSE resulted very slow due to disk swapping activity. On the other hand, DCI_CLOSED always ran by using remarkably less memory than FP-CLOSE, thus justifying its choice as mining engine.

In Figure 4(c,d), we plotted the execution times, the number of partitions, and the amount of memory actually used by DCI_CLOSED_OOC for mining dataset `Webdocs` as a function of the memory threshold imposed. The performances of the algorithm resulted always to be very stable, since executions times did not increase significantly with the number of partitions. On the other hand, given the partitioning technique adopted, the number of partitions grows more than linearly as expected. More importantly, the plots show that the amount of memory actually used during execution always resulted lower than the memory threshold imposed.

Finally, Figure 4(e,f) reports the results of the tests conducted on datasets `USCensus1990` and `Artificial2GB`. Also in these tests the memory threshold was always respected by DCI_CLOSED_OOC.

9 Conclusion and Future Work

We have presented a novel algorithm able to mine all the frequent closed itemsets from a transactional database using a limited amount of main memory. To our best knowledge, this is the first external memory algorithm for mining closed itemsets.

The two main contributions of this paper are, on the one hand, the optimization of an already known projected-based partitioning technique adapted to our framework, and, on the other hand, an innovative merging technique of the local results extracted from each partition.

We have shown how exploiting such partitioning technique requires a double scan of the dataset to collect enough information to decide how to subdivide it in order to obtain projected partitions that fit the available memory. Such information is also used to prune further the dataset and its partitions.

The main issue we have had to solve regards the possible generation of spurious frequent itemsets, which can be obtained if we simply combine the local results obtained from the separate mining of the partitions. We may in fact generate some additional frequent itemsets besides the truly closed ones. This unpleasant behavior is due to the partial knowledge available in each projected partition. This does not permit us to check, during the local mining of a partition, whether a produced itemset is globally closed or not. We have solved this problem in an elegant way. We have devised a novel out-of-core technique, based on a new theoretical insight, for merging the various local results and removing spurious itemsets. In particular, we have reduced the problem of merging partial solutions to an external memory sorting problem.

The experiments showed that DCI_CLOSED_OOC is able to run by using a very limited amount of main memory. Moreover, its performance is very similar to those of FP-CLOSE and its in-core counterpart. This is mainly due to

the fact that although DCI_Closed_ooc performs many more I/O operations, it subdivides and prunes the dataset effectively, thus producing very compact and cache-friendly in-core data structures for each partition.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of VLDB '94*, pages 487–499, September 1994.

[2] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 255–264. ACM Press, 1997.

[3] Bart Goethals and Mohammed J. Zaki. Advances in Frequent Itemset Mining Implementations: Report on FIMI'03. *SIGKDD Explorations*, 6(1):109–117, 2004.

[4] Gosta Grahne and Jianfei Zhu. Efficiently using prefix-trees in mining frequent itemsets. In *Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, November 2003.

[5] Gösta Grahne and Jianfei Zhu. Mining frequent itemsets from secondary memory. In *Proceedings of the 4th IEEE International Conference on Data Mining (ICDM 2004), 1-4 November 2004, Brighton, UK*, pages 91–98, 2004.

[6] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proc. of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2000.

[7] Junqiang Liu, Yunhe Pan, Ke Wang, and Jiawei Han. Mining frequent item sets by opportunistic projection. In *Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'02)*, pages 229–238. ACM Press, 2002.

[8] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Dci_closed: A fast and memory efficient algorithm to mine frequent closed itemse. In *Proc. of the IEEE ICDM 2004 Workshop on Frequent Itemset Mining Implementations (FIMI'04)*, 2004.

[9] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. On distributed closed itemsets mining: some preliminary results. In *Proc. of the SIAM SDM 2005 Workshop on High Performace Distributed Data Mining*, 2005.

[10] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Fast and memory efficient mining of frequent closed itemsets. *IEEE Transactions on Knowledge and Data Engineering*, 18(1):21–36, 2006.

[11] S. Orlando, P. Palmerini, R. Perego, and F. Silvestri. Adaptive and resource-aware mining of frequent sets. In *Proc. The 2002 IEEE International Conference on Data Mining (ICDM '02)*, pages 338–345, 2002.

[12] J. S. Park, M.-S. Chen, and P. S. Yu. An Effective Hash Based Algorithm for Mining Association Rules. In *Proc. of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 175–186, 1995.

[13] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25–46, 1999.

[14] J. Pei, J. Han, H. Lu, S. Nishio, and D. Tang, S. amd Yang. H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases. In *Proc. of the 2001 IEEE International Conference on Data Mining (ICDM'01)*, San Jose, CA, USA, 2000.

[15] Jian Pei, Jiawei Han, and Runying Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In *Proc. of the SIGMOD International Workshop on Data Mining and Knowledge Discovery*, May 2000.

[16] Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland*, pages 432–444. Morgan Kaufmann, 1995.

[17] Rafik Taouil, Nicolas Pasquier, Yves Bastide, Lotfi Lajhal, and Gerd Stumme. Mining freqent patterns with counting inference. *SIGKDD Explorations*, 2(2):66–75, December 2000.

[18] Rafik Taouil, Nicolas Pasquier, Yves Bastide, and Lotfi Lakhal. Mining bases for association rules using closed sets. In *Proc. of the 16th International Conference on Data Engineering (ICDE'00)*, page 307. IEEE Computer Society, 2000.

[19] Jianyong Wang, Jiawei Han, and Jian Pei. Closet+: searching for the best strategies for mining frequent closed itemsets. In *Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, pages 236–245. ACM Press, 2003.

[20] Mohammed J. Zaki. Mining non-redundant association rules. *Data Mining Knowledge Discovery*, 9(3):223–248, 2004.

[21] Mohammed J. Zaki and Karam Gouda. Fast vertical mining using diffsets. In *Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, pages 326–335. ACM Press, 2003.

[22] Mohammed J. Zaki and Ching-Jui Hsiao. Charm: An efficient algorithm for closed itemsets mining. In *Proc. of the 2nd SIAM International Conference on Data Mining (SDM'02)*, April 2002.