Frequent closed itemset based algorithms: A thorough structural and analytical survey

S. Ben Yahia
Faculté des Sciences de Tunis
Campus Universitaire
1060, Tunis, Tunisie
sadok.benyahia@fst.rnu.tn

T. Hamrouni
Faculté des Sciences de Tunis
Campus Universitaire
1060, Tunis, Tunisie
tarek.hamrouni@fst.rnu.tn

E. Mephu Nguifo
CRIL-CNRS, IUT de Lens
Rue de l'Université SP 16
62307 Lens Cedex, France
mephu@cril.univ-artois.fr

ABSTRACT

As a side effect of the digitalization of unprecedented amount of data, traditional retrieval tools proved to be unable to extract hidden and valuable knowledge. Data Mining, with a clear promise to provide adequate tools and/or techniques to do so, is the discovery of hidden information that can be retrieved from datasets. In this paper, we present a structural and analytical survey of frequent closed itemset (FCI) based algorithms for mining association rules. Indeed, we provide a structural classification, in four categories, and a comparison of these algorithms based on criteria that we introduce. We also present an analytical comparison of FCI-based algorithms using benchmark dense and sparse datasets as well as "worst case" datasets. Aiming to stand beyond classical performance analysis, we intend to provide a focal point on performance analysis based on memory consumption and advantages and/or limitations of optimization strategies, used in the FCI-based algorithms.

1. INTRODUCTION

The survival of the association rule extraction technique is owed to the retrieval of compactly sized with added-value knowledge. In this respect, the last decade witnessed a particular interest in the definition of condensed representations, *e.g.*, closed itemsets [31], free itemsets [6], non-derivable itemsets [7], essential itemsets [9], etc. The definition of these condensed representations mainly relies on the inclusion-exclusion principle and *Bonferroni* inequalities, which were of extensive use in addressing many enumeration problems [14].
The study of the extraction of closed itemsets grasped the interest of the data mining community. Indeed, frequent closed itemset (FCI) based algorithms were introduced to mainly tackle two complementary problems. On the one hand, FCI-based algorithms present an effective mining approach for dense extraction contexts. In such contexts, large equivalence classes are obtained. FCIs, standing on the top of the hierarchy induced by each equivalence class, allow to informatively infer the supports of FIs, standing within the frequent minimal generators (FMGs) and their associated FCIs. Note that the FMGs correspond to the frequent 0-free itemsets [6] and to the frequent key patterns [35].
Unfortunately, typically sparse contexts represent a "nightmare" for FCI-based algorithms, which have to bear useless and costly closure computations. The low performances on sparse contexts are quite expected, since the FCI search space tends to overlap with that of FIs.
On the other hand, FCI-based algorithms, which heavily draw on Formal Concept Analysis (FCA) mathematical settings [40], present a novel alternative with a clear promise to dramatically reduce, without information loss, the size of the association rule set that can be drawn from both synthetic and real-life datasets. The result of such a reduction is a reasonably-sized subset of association rules that can be seen as an irreducible nucleus of association rules, commonly known as "generic basis" of association rules [30].
In this paper, we present a structural and analytical comparative study of FCI-based algorithms. Hence, we classify FCI-based algorithms into four disjoint categories. We then introduce some features (or dimensions) allowing to highlight the major differences between the most prominent FCI-based algorithms for mining association rules (current and future). Performances of these algorithms are assessed and compared on benchmark dense and space datasets as well as "worst case" datasets [18]. Interestingly enough, the proposed analytical comparison goes beyond those respectively proposed by Zheng et *al.* [43; 44], in which only sparse datasets were of interest, and Goethals and Zaki [16], where only performance curves are showed. Indeed, we try not only to show performance curves, but also to explain these performances based on advantages and/or limitations of optimization strategies used in these algorithms. To obtain an in-depth insight, we also assess the memory consumption of the surveyed algorithms in conjunction with the evolution of gathered information, in main memory, during the mining process.
It is important to mention that in this survey, we put the focus on FCI-based algorithms especially designed for datasets having much more transactions than items. Nevertheless, some recent algorithms are dedicated to other kind of datasets like the COFI-CLOSED algorithm [11] for mining extremely large datasets (having millions of transactions) and the CARPENTER [28] and FARMER [10] algorithms for mining genomic datasets (having much more items than transactions).
We think that this survey can be regarded as complementary to some existing ones. For example, for FI-based algorithms, we can cite the survey of Hipp *et al.* [20], that of Aggarwal [1] for maximal FI-based algorithms and that of Kuznetsov and Ob"edkov [22] for concept lattice algorithms. As the FCI representation is concise *w.r.t.* that of FIs, this survey is also closely related to that of Calders *et al.* [8] in

which the authors give a general overview about different concise representations (like closed itemsets, free itemsets, non-derivable itemsets, etc).

The remainder of the paper is organized as follows: section 2 presents basic definitions of the FCA mathematical settings. Section 3 sketches a critical classification and a comparison of FCI-based algorithms thanks to criteria that we introduce. Section 4 reports an analytical comparison of FCI-based algorithms on benchmark and "worst case" datasets. Section 5 concludes this paper and points out future perspectives.

2. BASIC DEFINITIONS

In this section, we present basic definitions that will be of use in the remainder.

Definition 1. (Formal context) A formal context (or an extraction context) is a triplet $\mathcal{K} = (\mathcal{O}, \mathcal{I}, \mathcal{R})$, where $\mathcal{O}$ represents a finite set of objects (or transactions), $\mathcal{I}$ is a finite set of items (or attributes) and $\mathcal{R}$ is a binary (incidence) relation (*i.e.*, $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{I}$). Each couple $(o, i) \in \mathcal{R}$ expresses that the object $o \in \mathcal{O}$ contains the item $i \in \mathcal{I}$.

The closure operator γ induces an equivalence relation on the power set of items portioning it into disjoint subsets called *equivalence classes*. The largest element (*w.r.t.* the number of items) in each equivalence class is called a *closed itemset* and is defined as follows:

Definition 2. (Closed Itemset) An itemset $I \subseteq \mathcal{I}$ is said to be *closed* if and only if $\gamma(I) = I$ [31]. The *support* of I, denoted by Supp(I), is equal to the number of objects in $\mathcal{K}$ that contain I. I is said to be *frequent* if Supp(I) is greater than or equal to a user-specified minimum support threshold, denoted *minsup*. The *frequency* of I in $\mathcal{K}$ is equal to $\frac{Supp(I)}{|\mathcal{O}|}$.

Definition 3. (Iceberg Galois lattice) Let $\mathcal{FCI}_{\mathcal{K}}$ be the set of the FCIs extracted from an extraction context $\mathcal{K}$. When the set $\mathcal{FCI}_{\mathcal{K}}$ is partially ordered with set inclusion, the resulting structure $(\hat{\mathcal{L}}, \subseteq)$ only preserves the *Join* operator [15]. This structure is called a *join semi-lattice* or an *upper semi-lattice* and is, hereafter, referred to as "*Iceberg Galois lattice*" [27; 35].

Definition 4. (Upper cover) The upper cover of an FCI f (denoted Cov$^u(f)$) consists of the FCIs that immediately cover f in the Iceberg Galois lattice. The set Cov$^u(f)$ is given as follows: Cov$^u(f) = \{f_1 \mid f_1 \in \mathcal{FCI}_{\mathcal{K}}$ and $f \subset f_1$ and $\nexists\, f_2 \in \mathcal{FCI}_{\mathcal{K}}$ *s.t.* $f \subset f_2 \subset f_1\}$.

Definition 5. (Minimal Generator) An itemset $g \subseteq \mathcal{I}$ is said to be a *minimal generator* of a closed itemset f, if and only if $\gamma(g) = f$ and $\nexists\, g_1 \subset g$ *s.t.* $\gamma(g_1) = f$ [3]. Thus, the set MG$_f$ of the minimal generators associated to a closed itemset f is: MG$_f = \{g \subseteq \mathcal{I} \mid \gamma(g) = f$ and $\nexists\, g_1 \subset g$ *s.t.* $\gamma(g_1) = f\}$.

Therefore, the problem of mining association rules might be reformulated, under the point of view of the FCI-based algorithms, as follows:

1. Discover both distinct "closure systems", *i.e.*, sets of sets which are closed under the intersection operator, namely the FCI set and the FMG set. Also, the upper cover of each FCI should be available.
2. From the information discovered in the first step, *i.e.*, both closure systems and the upper covers, derive generic bases of association rules (from which all remaining rules can be derived).

In the next section, we present a structural survey of FCI-based algorithms.

3. FCI-BASED ALGORITHMS

In this section, we start by introducing a classification of these algorithms after which we give their main characteristics.

3.1 FCI-based algorithm classification

In general, the criterion used to classify FCI-based algorithms is inherited from the FI mining stuff, *i.e.*, the technique used to traverse the search space. Nevertheless, we add a supplementary category that we call "Hybrid without duplication" and this for different reasons explained hereafter. Hence, FCI-based algorithms can be roughly split into four categories, namely "Test-and-generate", "Divide-and-conquer", "Hybrid" and "Hybrid without duplication".

1. First category: **"Test-and-generate" technique**: The most known algorithms which adopt this technique are Close [31], A-Close [32] and Titanic [35]. These algorithms stress on the optimization of a level-wise process for the discovery of both closure systems previously mentioned. As a starting point, they consider the already known meet-irreducible set, *i.e.*, items of the ground set $\mathcal{I}$. This set is further extended by self-joined compound elements – using the combinatorial phase of Apriori-Gen [2] – during the exploration process. Candidate sets are pruned using the conjunction of statistical metrics (*e.g.*, the support measure) and heuristics essentially based on structural properties of CIs and MGs (*e.g.*, the fact that the MG set is an order ideal (or down-set) of $(2^{\mathcal{I}}, \subseteq)$ [35]).
2. Second category: **"Divide-and-conquer" technique**: The most known algorithm which adopts this technique is Closet [33]. The latter introduced the use of the highly compact data structure FP-tree (Frequent-Pattern tree) [19] within the FCI discovery process. Using a depth-first traversal of the search space, Closet tries to split the extraction context, stored in a global FP-tree, into smaller sub-contexts and to recursively apply the FCI mining process on these sub-contexts. The mining process also heavily relies on the search space pruning. This pruning is also based on statistical metrics in conjunction with introduced heuristics. Some improvements or alternatives of this algorithm were proposed, mainly Closet+ [39] and FP-Close [17], while respecting its driving idea.
3. Third category: **"Hybrid" technique**: Algorithms adopting this technique use properties of both previously mentioned techniques. ChARM [42] algorithm is the most known. Unlike other methods which exploit only the (closed) itemset search space, ChARM simultaneously explores both the CI search space and that of transactions thanks to an introduced data structure called IT-tree (Itemset-Tidset tree) [42]. Each node in the IT-tree contains an FCI candidate and the list

of the transactions to which it belongs. This list is called *tidset* [42]. CHARM explores the search space in a depth-first manner, like algorithms of the "Divide-and-conquer" technique, without splitting the extraction context into smaller sub-contexts. However, it generates each time a single candidate, like algorithms of the "Test-and-generate" technique. Then, it tries to test whether it is an FCI or not, using tidset intersections and subsumption checking. The mining process also heavily draws on the search space pruning. This pruning is based on imposed statistical metrics in conjunction with introduced heuristics.

4. Fourth category: **"Hybrid without duplication" technique**: Two main algorithms were proposed adopting this technique, namely DCI-CLOSED [25; 26] and LCM [36; 37]. Both algorithms can only be considered as improvements of CHARM as they inherit the use of tidsets [1] and the hybrid traverse of the search space adopted in CHARM. However, we choose to classify these two algorithms in a new category. Indeed, they avoid the main drawback of previously proposed algorithms, namely the high cost of subsumption checking allowing to discard generators whose closures were already mined (case of the second and the third category [2]). To this end, DCI-CLOSED and LCM traverse the search space in a depth-first manner. After discovering an FCI f, they generate a new generator g by extending f with a frequent item i, $i \notin f$. Using a total order relation on frequent items, DCI-CLOSED and LCM verify if g infringes this order by performing tests using only the tidset of f and those of the frequent items. If g is not discarded, then g is an order preserving generator of a new FCI. Then, its closure is computed using the previously mentioned tidsets. Hence, both algorithms extract the set of the FCIs in a linear time of its size [25; 36]. In addition, these algorithms do not need to store, in main memory, the set containing the previously mined FCIs since they do not require performing subsumption checking. The differences between DCI-CLOSED and LCM are the strategies for taking closure and the adopted data structures for storing the extraction context in main memory.

3.2 FCI-based algorithm characteristics

Based on the reformulation of the mining problem from the point of view of the FCI-based approach and its promises to losslessly reduce in size the association rule set, we present in what follows some features (or dimensions) allowing: on the one hand, to assess the percentage of achievement of such promised goals and on the other hand, to highlight the major differences between the FCI-based algorithms for mining association rules (current and future).

1. **Exploration technique**: three techniques are used to explore the search space, namely "Test-and-generate", "Divide-and-conquer" and "Hybrid".

2. **Architecture**: the architecture dimension depends on how a given algorithm is designed: a sequential function in a centralized single processor architecture, or more suited to a parallel treatment in a multiprocessor or distributed architecture, *e.g.*, CLOSET and DCI-CLOSED;

3. **Parallelism strategy**: parallel algorithms can be further described as task, data or hybrid parallelism;

4. **Data source type**: this feature indicates the type of the input data: (extended) market basket data (also known as horizontal data), vertical data, relational data, plain text, multimedia data, etc;

5. **Information storage**: different data structures are used to keep track of the information required to an algorithm execution (to store candidates, the extraction context, etc). The most privileged structure seems to be the *trie* structure.

6. **Generator choice**: some algorithms chose FMGs as generators of each FCI. Hence, closure computations are performed after discovering either the whole FMG set or only a subset of it (*e.g.*, FMGs having a common size). Other algorithms adopt a technique based on the fact that each time that a frequent generator is found, its closure is computed. This makes possible to create new frequent generator candidates starting from the already found FCI [25].

7. **Closure computation**: the closure of an itemset X can be computed thanks to the following two different methods. In the first method, the closure of X is the result of the intersection of the transactions to which it belongs. In the second method, its closure is computed in an incremental manner by searching for items that verify the following property [3]: $\psi(X) \subseteq \psi(i) \Rightarrow i \in \gamma(X)$, such that $i \in \mathcal{I}$ and $i \notin X$ [35]. The closure computation can also be performed *on-line* or *off-line*. In the former case, each time that a subset of the whole (minimal) generator set is found, the closure of each element of this subset is calculated. In the latter case, the closure computation of the whole (minimal) generator set is done at the same step [25].

8. **Possibility of redundant closure computation**: Since an FCI can have more than one generator, the same closure can be computed more than once unless some tests are performed. This dimension indicates whether a redundant closure computation is possible. Otherwise, it shows how this redundant computation is avoided.

9. **Output type**: here we focus on the type of the derived association rules: *e.g.*, boolean, fuzzy, spatial, temporal, generalized, qualitative, etc;

10. **Generated output**: this feature indicates the knowledge extracted by a given algorithm. The value "generic ARs" means that only a lossless and non-redundant subset of association rules *is* derived. Whereas, the value "redundant ARs" indicates that all (redundant) association rules *can be* derived from the generated output.

[1]Called *tidlists* in [25; 26] and *denotations* in [36; 37].

[2]In the case of the first category, an FCI can be computed more than once.

[3]$\psi(X)$ indicates the tidset of the itemset X.

Features	First category	Second category	Third category	Fourth category
Architecture	sequential	parallel	sequential	parallel
Parallelism strategy	none	none	none	none
Exploration technique	*Test-and-generate*	*Divide-and-conquer*	*Hybrid*	*Hybrid*
Data source type	horizontal	horizontal	horizontal, vertical	horizontal, vertical
Information storage	*trie*	*trie*	*trie*	bitmap matrix (DCI-Closed), simple arrays (LCM)
Generator choice	*minimal* generators	generators	generators	generators
Closure computation	intersection operation, on-line for both CLOSE and TITANIC and off-line for A-CLOSE	incremental search and on-line	incremental search and on-line	incremental search and on-line
Possibility of redundant closure computation	yes	no, using subsumption checking	no, using subsumption checking	no, using order preserving tests
Output type	boolean	boolean	boolean	boolean
Generated output	the FCIs, their associated FMGs and the redundant ARs	the FCIs and the redundant ARs	the FCIs and the redundant ARs	the FCIs and the redundant ARs
Most known algorithms	CLOSE, A-CLOSE and TITANIC	CLOSET, CLOSET+ and FP-CLOSE	CHARM	DCI-CLOSED and LCM

Table 1: Characteristics of the four categories.

Table 1 summarizes a categorization of the FCI-based algorithms *w.r.t.* the previously described basic dimensions. The last but one entry in Table 1 sheds light on the fact that, unfortunately, none of the reported algorithms is able to fulfill the problem reformulation goals, mentioned in the previous section. The origin of this failure is due to the fact of neglecting the importance – especially from a data mining frenzy towards performances – of maintaining the order covering the relationship between the FCIs. This is quite expected since the aforementioned algorithms avoid bearing a costly precedence-order construction fee. Hence, no more than the generic basis of exact association rules can be straightforwardly derived. To be able to extract approximate generic association rules, they need to be associated with another appropriate algorithm allowing to build the Iceberg Galois lattice, *e.g.*, that proposed by Valtchev *et al.* [38].

By comparing the main FCI-based algorithms, we can also note the following remarks:

- Among the previously mentioned algorithms, only TITANIC, DCI-CLOSED and LCM consider the equivalence class whose the associated FMG is the empty set. Thus, all remaining algorithms need to perform an additional treatment to compute the empty set closure.
- Pruning strategies adopted by TITANIC are an improvement of those of A-CLOSE by avoiding the cost of "unnecessary" traversals of previously extracted FMGs thanks to the estimated support pruning strategy, as will be explained later in the next section. Those of CLOSET and CHARM (resp. DCI-CLOSED and LCM) can also be considered as the same.
- The main weakness of the CLOSE, A-CLOSE and TITANIC algorithms is the redundant computation of the same FCI if this latter has more than one FMG. This problem is avoided by the CLOSET and CHARM algorithms using subsumption checking. Unfortunately, to speed up subsumption checking, both algorithms are obliged to maintain the retained FCIs in main memory. The DCI-CLOSED and LCM algorithms are mainly dedicated to tackle this problem. Thus, they try to extract the set of the FCIs without duplicate generation (hence in a linear time) and without maintaining it in main memory, using astute duplicate detection strategies.

It is important to note that, for all considered FCI-based algorithms, a suitable buffer management scheme is missing. Indeed, this scheme is required to handle this problem whenever necessary since generating a huge number of candidate sets might cause the memory buffer to overflow. To handle this issue, the candidate generation step should be modified in order to consider that a portion of the required (generated) information may be disk resident (saved on disk). In this case, we may need to perform an external sort as done in [34].

4. EXPERIMENTAL COMPARISON

In this section, we give an analytical survey of the FCI-based algorithms considered in the previous section. Experiments were carried out on a Pentium IV with a CPU clock rate of 2.4 GHz and 512 MB of main memory (with 2 GB of swap space). To rate the different behaviors of the considered algorithms, we ran experiments on both benchmark and "worst case" datasets, whose characteristics are detailed in what follows. The CLOSE, A-CLOSE, TITANIC, CHARM (with '-h -e 1 -d -H 1' options), FP-CLOSE, DCI-CLOSED and LCM ver. 2 algorithms were tested on the Linux distribution S.U.S.E 9.0, and their original makefile versions were compiled using gcc 3.3.1. Since CLOSET+ [4] was provided as a Windows executable, it was compared under Windows XP Professional on the same experimental environment. The source codes of the CLOSE, A-CLOSE, TITANIC, FP-CLOSE, CHARM, DCI-CLOSED and LCM algorithms and the CLOSET+ Windows binary executable were kindly provided by their respective authors.

[4]Since in [39], CLOSET+ performances already proved to be definitely better than those of CLOSET, we did not use CLOSET in the tests.

To report our results, we use *runtime*, *i.e.*, the period between input and output, instead of using the CPU time measured in some literature (*e.g.*, [17; 37]). The memory consumption is measured using the GNU glibc tool `memusage`, considering only the maximum heap size since stack use is much smaller than heap size. To make the measurements more reliable, no other application was running on the machine while experiments were running and none of the different executables displays the list of the discovered FCIs on the screen, nor writes it down on a disk-resident file.

4.1 Test dataset description

Algorithms were tested on two types of datasets: benchmark and "worst case". A "worst case" context is introduced thanks to the following definition:

Definition 6. [18] A "worst case" context is a triplet $\mathcal{K} = (\mathcal{O}, \mathcal{I}, \mathcal{R})$ where $\mathcal{I}$ is a finite set of items of size n, $\mathcal{O}$ represents a finite set of objects of size (n+1) and $\mathcal{R}$ is a binary (incidence) relation (*i.e.*, $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{I}$). In such context, each item belongs to n distinct objects. Each object, among the first n ones, contains (n-1) distinct items while the last one is verified by all items.

Thus, in a "worst case" context, each CI is equal to its unique MG. Hence, from a "worst case" context of dimension equal to $n\times(n+1)$, 2^n FCIs can be extracted when the *absolute minsup* value is set to **1**. Even if the worst case is rarely encountered in practice, "worst case" datasets allow to scrutinize the behavior of an algorithm on extremely sparse ones and hence to assess its scalability. Indeed, when n is increased by **1**, the number of FCIs grows up by a factor exactly equal to **2**. Hence, for a given algorithm, these datasets allow to check how it performs with the evolution of the FCI number. To the best of our knowledge, none of the previous algorithm performance surveys has considered such an extreme case. Figure 1 (Top on the left) presents an example of a "worst case" dataset for n = **4**. Figure 1 (Top on the right) presents parameter settings of synthetic datasets. Figure 1 (Down) summarizes the characteristics of benchmark dense and sparse datasets [5] and those of a "worst case" dataset.

4.2 Runtime

For more clarity, we have split the corresponding figures into two pools: the first one depicts reported statistics for the representatives of the first category, namely the Close, A-Close and Titanic algorithms. The second focuses on comparing performances of the representatives of the remaining categories, *i.e.*, the Closet+, FP-Close, ChARM, DCI-Closed and LCM algorithms. This splitting was motivated by the large gap in the inter-pool execution times. In fact, we noticed that the first pool algorithm performances can not compete with those of the second pool and gathering them in the same curve will be misleading. The second pool will also be divided into two sub-groups *w.r.t.* the technique used to traverse the search space, *i.e.*, "Divide-and-conquer" or "Hybrid" technique. Hence, we find the Closet+ and FP-Close algorithms in the first group, while the ChARM, DCI-Closed and LCM algorithms are in the second group. For each dataset, we begin by comparing performances of algorithms belonging to the same group (an "intra-group" comparison). Then, we compare the five algorithms performances (an "inter-group" comparison) by giving the best algorithm(s) by *minsup* interval.

[5]All benchmark datasets are downloadable from: `http://fimi.cs.helsinki.fi/data`.

	i_1	i_2	i_3	i_4
o_1		×	×	×
o_2	×		×	×
o_3	×	×		×
o_4	×	×	×	
o_5	×	×	×	×

|T|	Average transaction size
|I|	Average size of the maximal potentially FIs
|D|	Number of generated transactions

Benchmark datasets	**Type**	**# items**	**# transactions**	**Avg. transaction size**
Mushroom	dense	**119**	**8, 124**	**23**
Chess	dense	**75**	**3, 196**	**37**
Connect	dense	**129**	**67, 557**	**43**
T10I4D100K	sparse	**1, 000**	**100, 000**	**10**
T40I10D100K	sparse	**1, 000**	**100, 000**	**40**
Retail	sparse	**16, 470**	**88, 162**	**10**
"Worst case" datasets	sparse	n	n+1	$\frac{n^2}{n+1}$

Figure 1: (**Top on the left**) A "worst case" dataset for n = **4**. (**Top on the right**) Synthetic dataset settings. (**Down**) Dataset characteristics.

4.2.1 Benchmark dense datasets

First pool: In the first pool, Close presents better performances than A-Close for all datasets. This was expected since Close is known to perform better than A-Close for dense datasets [29].

For the Mushroom dataset, Titanic outperforms Close for *minsup* values ranging between **20%** and **2%**. However, the contrary happens for *minsup* values lower than **2%**, where Close largely outperforms Titanic. Indeed, to compute the FCIs, Titanic tries to extend any FMG with the appropriate items by carrying out costly support computations, especially for low *minsup* values. Indeed, for *minsup* = **0.1%**, **116** items – among **119** – are frequent and the maximum size of an FMG is only equal to **10** items. Note that for this dataset, Close, A-Close and Titanic suffer from redundant computation of the FCIs. Indeed, for a *minsup* value equal to **0.1%**, the number of FMGs, equal to **360, 166**, is almost **2.2** times the number of FCIs, equal to **164, 117** (as shown by Figure 5).

For the Chess dataset, the counting inference [4] adopted by Titanic is more efficient than that based on intersection computations adopted by the Close algorithm. Indeed, Titanic outperforms Close for the different tested *minsup* values. However, for the same reasons as in the case of the Mushroom dataset, this efficiency tends to decrease proportionally with the decrease of *minsup* values.

For the Connect dataset, Titanic largely outperforms Close for all *minsup* values. In fact, the task of Close is considerably complicated by performing intersections on a large number of highly sized transactions. However, due to lack of memory capacity, the execution of Titanic did not come to an end starting from a *minsup* value equal to **50%**. It is important to mention that, even if this dataset is dense, each FCI has a unique FMG (as shown by Figure 5 in which both associated curves are overlapping).

Second pool: For the Mushroom dataset, performances are as follows. For the first group and for all *minsup* values, FP-Close outperforms Closet+ and the gap tends to increase for lower *minsup* values. As Mushroom is a dense dataset, both Closet+ and FP-Close are obliged to recursively build the physically projected FP-trees. The difference between the performances of both algorithms comes from the fact that FP-Close stores the previously mined FCIs in recursively constructed CFI-trees (Closed Frequent Itemset trees), while Closet+ keeps track of all mined FCIs in a global prefix-tree. Consequently, in the case of FP-Close, subsumption checking cost is by far less expensive than that for Closet+, and the difference gets sharper when *minsup* values decrease. In addition, FP-Close stores only a part of each FCI in a CFI-tree [45] which even more reduces the cost of subsumption checking. In the second group and for *minsup* values greater than or equal to **0.3%**, DCI-Closed and LCM performances are very similar, with a slight advantage for the latter. Recall that as mentioned in section 3.1, both algorithms belong to the fourth category and share their main characteristics. For the same *minsup* values, they outperform ChARM, the representative of the third category. This fact can be explained by, on the one hand, the efficiency of duplicate detection strategies performed by DCI-Closed and LCM, and the several optimization techniques used in both algorithms, on the other hand. For *minsup* values lower than **0.3%**, LCM still outperform ChARM. However, surprisingly enough, for a *minsup* value equal to **0.2%**, the execution of DCI-Closed stops for lack of memory capacity after more than one hour and a half. The main reason is that DCI-Closed uses a heuristic allowing to assess whether the dataset is dense or sparse. Using this information – the nature of the dataset – DCI-Closed launches a slightly modified version of the level-wise sweeping kDCI algorithm [6] [24] in the case of a sparse dataset. Hence, strangely in the case of the dense Mushroom dataset and for a *minsup* value equal to **0.2%**, kDCI was executed instead of the procedure normally to be run in the case of dense datasets. This explanation is argued by the subset of FCIs extracted before that the execution stops. Indeed, using DCI-Closed with the option allowing to write the output on a disk-resident file, we found that the extracted FCIs are written in an increasing order of their respective sizes (*i.e.*, by level). Hence, in this particular case, DCI-Closed browses the search space in a breadth-first manner (using kDCI) instead of a depth-first manner, which leads to runtime degradation and to memory saturation. By comparing performances of the five algorithms, FP-Close is the best for *minsup* values greater than or equal to **3%**. DCI-Closed takes the best for *minsup* values greater than or equal to **0.3%**. For *minsup* values less than **0.3%**, LCM is the best.

For the Chess dataset, on the one hand, FP-Close largely outperforms Closet+. For the same reasons as for the Mushroom dataset, the difference between their respective performances sharply increases as much as we lower *minsup* values. For example, FP-Close performs almost **8.5** (resp. **27**) times better than Closet+ for a *minsup* value equal to **40%** (resp. **30%**). On the other hand, ChARM outperforms both DCI-Closed and LCM for *minsup* values greater than or equal to **90%**. For *minsup* values between **90%** and **70%**, LCM is the best and for *minsup* values lower than **70%**, DCI-Closed is the best. By comparing the performances of both groups, ChARM, LCM and DCI-Closed are the best, respectively, for the previously mentioned intervals. It is worth noting that we stopped the execution of ChARM for a *minsup* value equal to **20%** after more than two days and a half. This confirms the prohibitive cost of tidset intersections and subsumption checking, for very low *minsup* values.

For the Connect dataset and for the first group, the scenario of both previous datasets was repeated. Indeed, for all *minsup* values, FP-Close outperforms Closet+. However, both algorithm performances are similar for *minsup* values greater than or equal to **30%** and the advantage is clearly taken by FP-Close for supports lower than **30%** (almost **2.5** times for a *minsup* value equal to **10%**). For the second group, ChARM is largely outperformed by both algorithms of the fourth category, *i.e.*, DCI-Closed and LCM. For example, DCI-Closed performs almost **30** times better than ChARM for *minsup* values equal to **10%**. LCM performs better than DCI-Closed for *minsup* values greater than or equal to **40%**, even though the performance gap is very tight. For *minsup* values lower than **40%**, DCI-Closed takes the lead on LCM and performs almost twice better than the latter for a *minsup* value equal to **10%**. By comparing both groups' performances, FP-Close is the best for a *minsup* value equal to **60%**. For the other tested *minsup* values greater than or equal to **40%**, LCM is the best, whereas DCI-Closed takes the top on the four other algorithms for *minsup* values lower than **40%**.

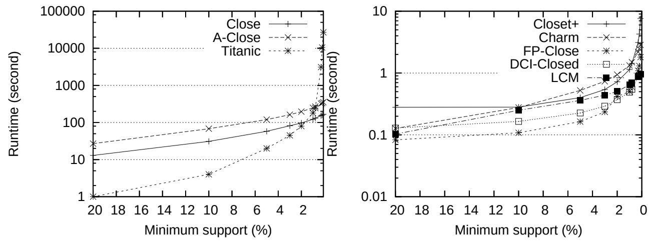

Figure 2: Runtime for the Mushroom dataset.

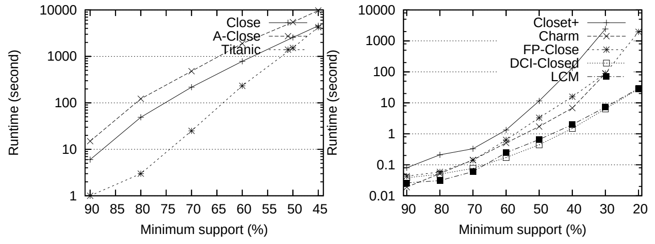

Figure 3: Runtime for the Chess dataset.

4.2.2 Benchmark sparse datasets

First pool: For the T10I4D100K dataset, and for a *minsup* value equal to **0.5%**, Titanic outperforms both Close

[6]kDCI is a breadth-first algorithm that mines FIs. An additional test is thus added to check which FI is also closed.

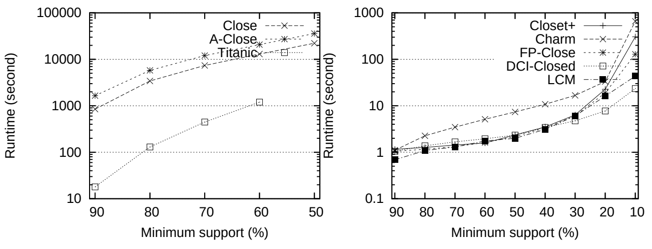

Figure 4: Runtime for the Connect dataset.

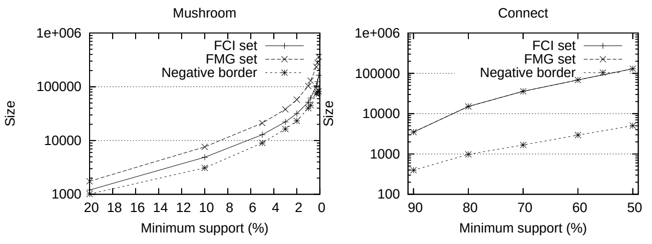

Figure 5: The size of the FCI set, of the FMG set and of the negative border of MGs (denoted Negative border) for, respectively, the Mushroom and Connect datasets.

and A-Close. For the same *minsup* value, A-Close outperforms Close. We can explain this by the fact that, for this *minsup* value, each FMG is equal to its closure (*i.e.*, the variable *Level*, used in A-Close algorithm to check this case, is equal to **0** [7]) and then A-Close does not perform any closure computation. However, A-Close is handicapped by the traversal of the (k-1)-FMG set for each k-candidate g. This traversal allows it to compare the support of g with those of its (k-1)-subsets to check if g is an FMG or not. As outlined earlier, Titanic avoids this overhead by simply comparing the support of g to its estimated support (equal to the minimum of the supports of its (k-1)-subsets). For *minsup* values lower than **0.5%**, Close outperforms both A-Close and Titanic. For a *minsup* value equal to **0.2%**, *Level* is equal to **3**. Thus, A-Close has to compute the closure of the FMGs of size greater than or equal to **2**. For a *minsup* value lower than **0.2%**, A-Close computes the closure of all FMGs since *Level* is equal to **2**. Titanic performances decrease in a significant way for the same reason evoked earlier. Indeed, for *minsup* = **0.02%**, **859** items, among **1, 000**, are frequent and the maximum size of an FMG is only equal to **10** items. In addition, Titanic needs to maintain, in main memory, a highly sized negative border of MGs [21] (as shown by Figure 9).

For the T40I10D100K dataset, and for *minsup* values ranging between **10%** and **1.5%**, Titanic outperforms both Close and A-Close. For this interval of *minsup*, *Level* is equal to **0** and the same scenario as that of the T10I4D100K dataset when *minsup* value is equal to **0.5%** seems to be repeated. For a *minsup* value lower than **1.5%**, *Level* is equal to **4**. However, Close performs better than A-Close most of times. This can be explained by the fact that the size of the longest FMG is equal to **18** and hence A-Close has to compute the closures of FMGs of sizes between **3** and **18**. The decrease of Titanic performances, when lowering *minsup* values, can be explained by what follows: when computing the closure of an FMG and aiming to extend it by the appropriate items, Titanic has to explore a particularly very large search space since the negative border of MGs is maintained (as shown by Figure 9).

For the Retail dataset and for *minsup* values greater than **0.08%**, Close and A-Close performances are similar with a slight advantage for A-Close. For *minsup* values greater than **0.06%**, Close performances considerably degrade. The decrease of its performances can be explained by the enormous influence of the high number of items in the Retail dataset. Indeed, Close is handicapped by a very high number of candidates to which it is obliged to compute respective closures, even though a large number of them is infrequent. The number of candidates also affects the performances of A-Close which, in addition to the closure computation cost, has to traverse the (k-1)-FMG set for each k-candidate. Note that these infrequent candidates belong to the negative border of MGs stored by Titanic what explains why its search space is very large. In addition, Titanic is considerably penalized by the high number of frequent items to consider in closure computations. Indeed, for *minsup* = **0.04%**, **4, 643** items are frequent and the maximum size of an FMG is only equal to **6** items. For support values lower than **0.04%**, Titanic executions stop for lack of memory capacity.

Second pool: For the tested datasets, DCI-Closed uses kDCI algorithm for all *minsup* values. This fact was confirmed by checking DCI-Closed output written on a disk resident file using a specific option.

For the T10I4D100K dataset, and for the first group, the representatives of the second category of FCI-based algorithms, *i.e.*, Closet+ and FP-Close, have very similar performances. For *minsup* values greater than or equal to **0.03%**, FP-Close slightly outperforms Closet+. However, it is the opposite for *minsup* values lower than **0.03%**. As the T10I4D100K dataset is a sparse one, Closet+ uses the top-down pseudo tree-projection method. Hence, it avoids the recursive construction of FP-trees and has only to traverse the global FP-tree. On its side, FP-Close profits from using an array-based technique to avoid traversing previously built FP-trees to construct the respective header tables of the new entries. For the second group, it was expected that DCI-Closed outperforms ChARM and LCM, since it uses the kDCI algorithm. Interestingly enough, experiments showed the opposite. Indeed, for *minsup* values greater than or equal to **0.08%**, ChARM and DCI-Closed have very similar performances with a slight advantage to ChARM and both outperform LCM with a peak of almost **3** times for a *minsup* value equal to **0.5%**. However, the tendency changes for support values lower than **0.08%** since LCM takes the top on ChARM and DCI-Closed with a peak of almost **3** (resp. **8**) times better than ChARM (resp. DCI-Closed) for a *minsup* value equal to **0.02%**. By comparing the five algorithms, ChARM (resp. LCM) is the best for high (resp. low) *minsup* values. It is important to mention that for a *minsup* value equal to **0.02%**, LCM algorithm outputs **107, 825** FCIs. However, this number does not match with that obtained with the other

[7]*Level* is initialized with **0** and takes as value the size k of the smallest FMG candidate having the same support as one of its (k-1)-subsets.

surveyed algorithms (**107, 822** FCIs for Close, A-Close, ChARM, Closet+ and FP-Close, and **107, 823** for Titanic and DCI-Closed since both consider the equivalence class whose the FMG is the empty set). Hence, this fact points out the need for the proof of the algorithm correctness as formerly claimed by Zheng *et al.* [43], especially when many optimizations are used. However, it is worth noting that, in their performance comparison, Zheng *et al.* omitted to use the '-H 1' option allowing ChARM to perform subsumption checking to discard non FCIs. This explains why they found, for a *minsup* value equal to **0.01%**, **303, 610** instead of **283, 397** FCIs. The difference of **20, 213** represents non FCIs, not discarded by ChARM since '-H 1' option was not used.

For the T40I10D100K dataset and for the first group, FP-Close largely outperforms Closet+, for all *minsup* values. The difference reaches a peak of almost **10** times for a *minsup* value equal to **0.5%**. Indeed, subsumption checking highly affects Closet+ performance as it stores all FCIs in the *same* tree (the number of these itemsets reaches **1, 275, 940** for *minsup* = **0.5%**). For the second group, ChARM is the best for *minsup* values greater than or equal to **5%**. DCI-Closed takes the top for *minsup* values between **5%** and **1.5%** and LCM is the best for *minsup* values lower than **1.5%**. In this pool, ChARM and DCI-Closed are the best for the respective intervals previously mentioned, whereas FP-Close slightly outperforms LCM for *minsup* values lower than **1.5%**.

For the Retail dataset and for the first group, the opposite of the scenario of the T40I10D100K dataset happens. Indeed, Closet+ largely outperforms FP-Close since the former performs between **1.5** and **7.5** times better than the latter. This is due to the high number of frequent items extracted for very low *minsup* values. For example, for a *minsup* value equal to **0.01%**, **9, 300** items are frequent. Such number highly increases the cost of the array based technique adopted by FP-Close, whereas Closet+, benefiting from the use of the top-down pseudo tree-projection method, needs only to traverse the global FP-tree. For the second group, LCM largely outperforms ChARM and DCI-Closed and the gap tends to sharply increase proportionally to the decrease of *minsup* values. For example, LCM performs almost **2** (resp. **1.4**) times better than ChARM (resp. DCI-Closed) for a *minsup* value equal to **0.1%**, whereas for a *minsup* value equal to **0.01%**, the difference reaches almost **6** (resp. **10**). In this pool, LCM is the best for all *minsup* values.

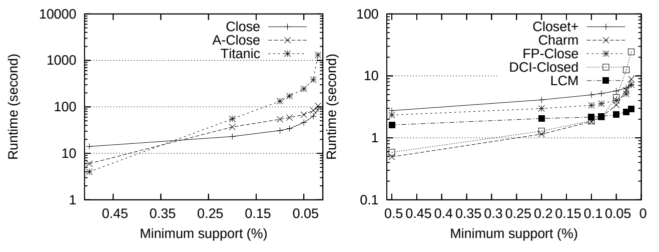

Figure 6: Runtime for the T10I4D100K dataset.

These experiments show that, unfortunately, there is no outstanding algorithm to be qualified as the best for all datasets or at least for a given dataset type, *i.e.*, dense or sparse.

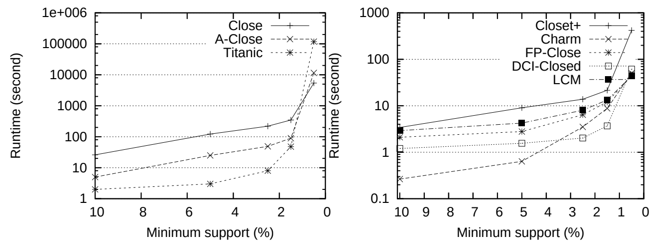

Figure 7: Runtime for the T40I10D100K dataset.

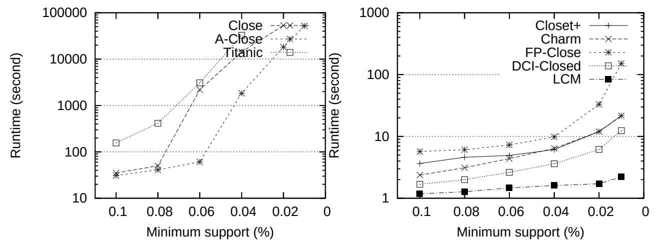

Figure 8: Runtime for the Retail dataset.

Moreover, in general, for a given dataset, there is no best algorithm for all *minsup* values. Indeed, algorithm performances closely rely on *minsup* values. A change in the *minsup* value can lead to different information to be treated by the algorithm and so, an optimization or a heuristic that performs better for a given *minsup* value can slow down performances due to this change. Thus, the use of a multitude of optimizations is not sufficient. Indeed, the most important is to be able to precisely decide when to apply the most appropriate optimization according to the handled input data. On average, LCM performances are not badly affected by the *minsup* value changes.

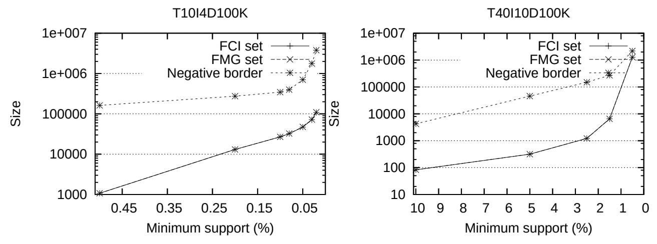

Figure 9: The size of the FCI set, of the FMG set and of the negative border of MGs (denoted Negative border) for, respectively, the T10I4D100K and T40I10D100K datasets.

4.2.3 "Worst case" datasets

First pool: We tested **25** datasets showing the variation of n from **1** to **25**. The *absolute minsup* value was fixed to **1**. Execution times of the three algorithms began to be distinguishable only starting from the value of n equal to **15**. In a "worst case" dataset, each FMG is equal to its closure and hence, A-Close does not perform any closure computation thanks to the use of *Level*. A-Close and Titanic largely outperform Close for the different values of n. The difference between their respective performances began to be clear for a value of n greater that **21**. Note that TITANIC performs a traversal of the (k-1)-FMG set for each k-FMG g. By this sweeping, TITANIC tries to reduce the cost of the computation of g' closure by collecting items belonging to the respective closures of its (k-1)-subsets. In addition, TITANIC performs an incremental search to try to find other items belonging to the closure of g. Hence, the higher is the value of n, the worse is the influence of such treatments on TITANIC performances. For lack of memory space, executions of CLOSE and TITANIC stop for a value of n equal to **24**. It is the same for A-CLOSE when n reaches **25**.

Second pool: We tested **30** datasets showing the variation of n from **1** to **30**. The *absolute minsup* value was fixed to **1**. Execution times of the five algorithms began to be distinguishable only starting from the value of n equal to **15**. DCI-CLOSED and LCM performances are largely better than those of CLOSET+, FP-CLOSE and CHARM, with a clear advantage for DCI-CLOSED for the different values of n. For the tested datasets, DCI-CLOSED executed, against any waiting, the procedure used in the case of dense datasets and not the kDCI algorithm. We expected the latter to be run since "worst case" contexts are the most evident representation of sparse ones. Due to subsumption checking performed in a large set of FCIs, CHARM performances are not interesting even for low values of n. The execution of FP-CLOSE stops for a value of n equal to **22** with "`The blocks are used up`" error message. For $n = 21$, we stopped the execution of CLOSET+ after more than four hours. The respective curves of DCI-CLOSED and LCM have almost the same slope. They grow linearly as much as the value of n increases. This fact confirms that, in the contrary to algorithms belonging to the first three categories, the runtime of the fourth category's algorithms is a linear function of the size of the FCI set. Nevertheless, the fact that DCI-CLOSED presents, in general, very interesting performances must not hide the difficulties encountered by this algorithm to correctly assess the right type (dense or sparse) of a given dataset. Indeed, more than once, DCI-CLOSED uses kDCI instead of the expected procedure to be run for dense dataset (case of the Mushroom dataset) and inversely (case of the "worst case" datasets).

It is important to mention that the treatments performed by all algorithms except A-CLOSE, to compute closures, are useless since each frequent (minimal) generator is equal to its closure.

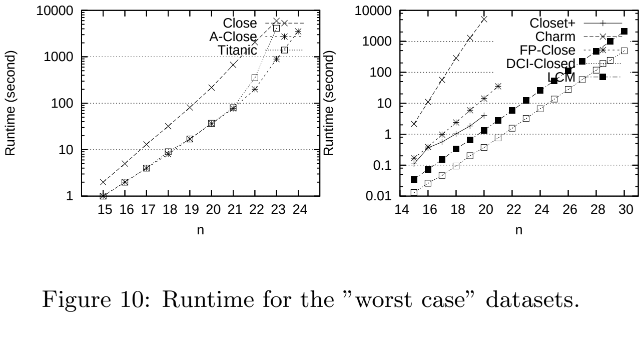

Figure 10: Runtime for the "worst case" datasets.

4.3 Memory consumption

Since memory consumption also demonstrates the quality of an algorithm (and in general, of a category of algorithms), we recorded the main memory consumption peak displayed out for different datasets. The tested algorithms are still divided into two pools.

First pool: Figure 11 plots the peak of the main memory consumption of the CLOSE, A-CLOSE and TITANIC algorithms when running them on the Mushroom and T10I4-D100K datasets. For both datasets, TITANIC uses the maximum amount of main memory. This is a straightforward consequence of the counting inference adopted by TITANIC. Indeed, to correctly apply this mechanism, TITANIC needs to maintain in main memory the set of the FMGs as well as the negative border of MGs. This border is in general of high size (as shown by Figures 5 and 9). By comparing the memory consumption of CLOSE and A-CLOSE, CLOSE uses the lowest amount of main memory for the dense dataset and A-CLOSE for the sparse one. It is worth noting that using an off-line closure computation, A-CLOSE requires saving, in main memory, each mined FMG until obtaining the whole FMG set. Hence, this manner of closure computation is also greedy in memory space. CLOSE, in the contrary of A-CLOSE and TITANIC, does not need to maintain the set of the FMGs in main memory. However, it presents another drawback. Indeed, CLOSE computes the closure of the FMG candidates while counting their respective supports. Hence, CLOSE needs to maintain, in main memory, several CIs whose support values are probably lower than the *minsup* value. The number of such CIs is much higher for sparse datasets.

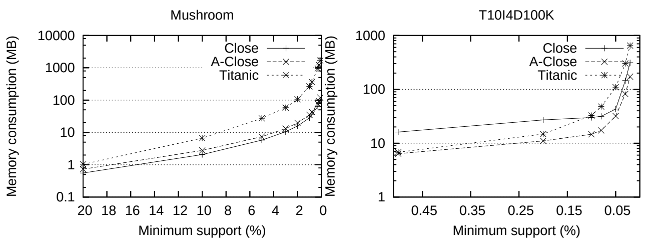

Figure 11: The memory consumption for, respectively, the Mushroom and T10I4D100K datasets.

Second pool: Figure 12 plots the peak of the main memory consumption of the FP-CLOSE, CHARM, DCI-CLOSED and LCM algorithms when running them, respectively, on the Mushroom, Connect, T10I4D100K and T40I10D100K datasets. Obtained results on both dense datasets confirm the claim of the authors of DCI-CLOSED and LCM about the memory consumption of their respective algorithms. Indeed, with an advantage for DCI-CLOSED, the amount of memory used by both algorithms is nearly constant since they do not need to maintain the previously mined FCIs in main memory. Conversely, the amounts of main memory, respectively used by FP-CLOSE and CHARM, rapidly grow up as much as the *minsup* value decreases. Indeed, this increase is due to the huge number of previously extracted FCIs that must be maintained in main memory. For example, considering the Connect dataset when the *minsup* value decreases from **20%** to **10%**, the cardinal of the FCI set grows up from a value equal to **1, 483, 200** to a value equal to **8, 073, 778**. While, the memory consumption of DCI-Closed (resp. LCM) is only multiplied by a factor equal to **1.16** (resp. **1.34**), that of ChARM (resp. FP-Close) is multiplied by a factor equal to **5.91** (resp. **4.91**). It is worth noting that even if FP-Close stores only a part of each FCI in the CFI-trees, the storage of multiple CFI-trees still needs a lot of main memory.

In the case of both sparse datasets T10I4D100K and T40I10-D100K, DCI-Closed uses the kDCI algorithm. Hence, it is expected that DCI-Closed memory requirement will not be constant, in contrary to the case of previously examined dense datasets. Indeed, kDCI needs to maintain, in main memory, the set of the k-FIs and that of the (k-1)-FIs.

Nevertheless, for the T10I4D100K dataset, the memory requirement of DCI-Closed can still be considered as constant, as for dense datasets. Indeed, the maximum multiplicative factor is equal to **1.33**. This can be explained by the fact that the peak of DCI-Closed memory consumption always occurs when both the set of the **3**-FIs and the set of the **2**-FIs are maintained in main memory. The size of these sets increases when *minsup* values decrease, but not exponentially. Hence, DCI-Closed memory requirement appears as if it were constant. Another surprising result, about the T10I4D100K dataset, is that ChARM uses the lowest amount of main memory. As for dense datasets, the amount of memory used by LCM is nearly constant and is slightly lower than that used by DCI-Closed. FP-Close uses the highest amount of main memory since it needs to maintain recursively built FP-trees, known to be large and bushy for sparse datasets, as well as multiple CFI-trees.

For the T40I10D100K dataset, the memory requirement of DCI-Closed is not constant but this algorithm has the best space-efficiency for low *minsup* values. ChARM uses the lowest amount of main memory for high *minsup* values. However, for the T40I10D100K dataset and for low *minsup* values, the number of FCIs considerably increases (equal to **1, 275, 940** for *minsup* = **0.5%**). Hence, since ChARM needs to maintain such number of FCIs in main memory, its memory requirement rapidly grows up when the support threshold decreases. This explains why DCI-Closed, and even if it browses the search space in a level-wise manner, uses less main memory for low *minsup* values. As for the T10I4D100K dataset, FP-Close uses the highest amount of main memory. By comparing the memory requirements of the representative of the third (resp. fourth) category, *i.e.*, ChARM (resp. LCM) for both *minsup* values **1.5%** and **0.5%**, we find that the amount used by ChARM (resp. LCM) grows up by a factor equal to **83.07** (resp. **1.03**). The main reason is that the size of the FCI set, to be maintained in main memory by ChARM, grows up from a value equal to **6, 540**, for *minsup* = **1.5**, to a value equal to **1, 275, 940** for *minsup* = **0.5%**. As the fourth category avoids this storage, its memory requirement is nearly constant and it is far from indicating that the size of the FCI set grows up exponentially.

5. CONCLUSION

In this paper, we presented a structural and analytical comparative study of FCI-based algorithms, towards a theoretical and empirical guidance to choose the most adequate mining algorithm. The main report of this comparative study is to shed light on an "obsessional" algorithmic effort to reduce the computation time of the interesting itemset extraction step.

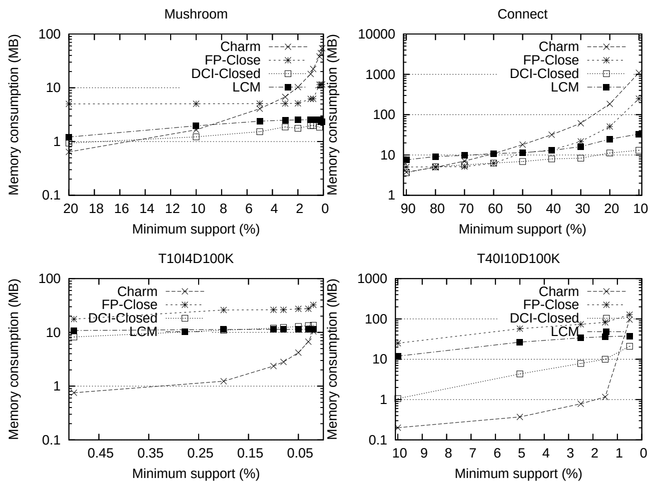

Figure 12: The memory consumption for, respectively, the Mushroom, Connect, T10I4D100K and T40I10D100K datasets.

The obtained success is primarily due to an important programming effort combined with strategies for compacting data structures in the main memory. However, it seems obvious that this frenzied activity loses sight of the essential objective of this step, *i.e.*, to extract a reliable knowledge, of exploitable size for end-users. Thus, almost all these algorithms were focused on enumerating CIs, presenting a frequency of appearance considered to be satisfactory. The dark side of this success is that this enumeration will generate an impressive and not exploitable number of association rules, even for a reasonable context size. Indeed, apart from modest performances on sparse contexts, these algorithms concentrated on extracting the FCIs while neglecting to determine the underlying subset-superset relationship.

We can notice, as highlighted by the second entry in Table 1, the absence of attempts to parallelize FCI-based algorithms. Indeed, most parallel algorithms for mining association rules are based on the sequential Apriori algorithm [8]. We also believe that it will be a very promising approach to combine both paradigms: task parallelism and data parallelism. In this respect, both preliminary works [5; 13] on distributed or parallel computation of the FCIs are worth noting.

Finding a powerful and costless metric for the assessment of an extraction context density is another important compelling issue to be tackled. In fact, FCI-based algorithm performances – behaving badly on sparse datasets on the contrary of FI-based algorithms – motivate setting up a "density depending" approach. This latter seeks to apply the appropriate extraction approach depending on the extraction context density. For example, based on the "Divide-and-conquer" technique, such a novel approach recursively divides the extraction context into sub-contexts. The appropriate frequent (closed) itemset extraction process is applied to each sub-context, depending on its accurate density as-

[8]An excellent survey is given in [41] classifying algorithms by load-balancing strategy, architecture and parallelism.

sessment.
An in-depth study of dataset characteristics is bashfully beginning to grasp the community interest. In fact, these characteristics have an important impact on algorithm performances. A finer structural characterization of handled datasets – beyond the number of transactions/items, the average transaction size or the density – permits a better understanding of the behavior of algorithms. In this respect, two preliminary works are worth mentioning. The first studies the distribution of the negative border and its complement, the positive one [12], while the second work proposes some results about the average number of frequent (closed) itemsets, using probabilistic techniques (*e.g.*, the Bernoulli law) [23].

Other avenues for future work address the challenging issues of mining richly structured and genomic contexts. For example, in genomic contexts, mined knowledge is closely dependent of the discretization method, *i.e.*, how the original context is translated to a binary one. Usually, a gene is considered to be present in a biological situation if its expression level exceeds (or not) a given threshold (*e.g*, its expression level average in the different biological situations). Clearly, such translation is far from being information lossless. Actually, the main drawback is that the discretization is not able to describe the "actual" biological situation. For this reason, it is of paramount importance to handle an extended extraction context, *i.e.*, without binarizing the original context. In this respect, introducing soft computing techniques seems to be a promising issue.
Finally, well adapted visualization models are badly missing. In fact, useful relationships between nonintuitive variables are the jewels that data mining techniques hope to locate. However, the unmanageably large association rule sets, compounded with their low precision, often makes the perusal of knowledge ineffective, their exploitation time-consuming and frustrating for the user. This fact is reinforced since the user does not know beforehand what the data mining process will discover. It is a much bigger leap to take the output of the system and to translate it into a natural representation matching with what the end-user has in mind.

Acknowledgements

We are grateful to the anonymous reviewers for their helpful comments. We also thank C. Dubois, J.J. Givry and C. Zimny for English proofreading. This work is partially supported by the French-Tunisian project CMCU 05G1412.

6. REFERENCES

[1] C. C. Aggarwal. Towards long pattern generation in dense databases. In *ACM-SIGKDD Explorations*, volume 3(1), pages 20–26, July 2001.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proceedings of the 20th International Conference on Very Large Databases, Santiago, Chile*, pages 478–499, June 1994.

[3] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, and L. Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. In *Proceedings of the 1st International Conference on Computational Logic (DOOD 2000), Springer-Verlag, LNAI, volume 1861, London, UK*, pages 972–986, July 2000.

[4] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal. Mining frequent patterns with counting inference. In *Proceeding of the 6th ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining, Boston, Massachusetts, USA*, volume 2(2), pages 66–75, 20-23 August 2000.

[5] S. BenYahia, Y. Slimani, and J. Rezgui. A divide and conquer approach for deriving partially ordered sub-structures. In *Proceedings of the International 9th Pacific-Asia Conference on Knowledge Data Discovery (PAKDD 2005), LNAI, volume 3518, Springer-Verlag, Hanoi, Vietnam*, pages 91–96, 18-20 May 2005.

[6] J.-F. Boulicaut, A. Bykowski, and C. Rigotti. Free-sets: A condensed representation of boolean data for the approximation of frequency queries. *In Jounal of Data Mining and Knowledge Discovery (DMKD)*, 7 (1):5–22, 2003.

[7] T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In T. Elomaa, H. Mannila, and H. Toivonen, editors, *Proceedings of the 6th European Conference on Principles of Knowledge Discovery and Data Mining (PKDD 2002), LNCS, volume 2431, Springer-Verlag, Helsinki, Finland*, pages 74–85, 19-23 August 2002.

[8] T. Calders, C. Rigotti, and J.-F. Boulicaut. A survey on condensed representations for frequent sets. In *Constraint Based Mining, Springer-Verlag, LNAI, volume 3848*, pages 64–80, 2006.

[9] A. Casali, R. Cicchetti, and L. Lakhal. Essential patterns: A perfect cover of frequent patterns. In A Min Tjoa and J. Trujillo, editors, *Proceedings of the 7th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2005), Springer-Verlag, LNCS, volume 3589, Copenhagen, Denmark*, pages 428–437, 22-26 August 2005.

[10] G. Cong, K. H. Tung, X. Xu, F. Pan, and J. Yang. FARMER: finding interesting rule groups in microarray datasets. In *Proceedings of the 2004 ACM SIGMOD International conference on Management of data, Paris, France*, pages 143–154, 2004.

[11] M. El-Hajj and O. Zaiane. Finding all frequent patterns starting from the closure. In *the International Conference on Advanced Data Mining and Applications, Wuhan, China*, pages 67–74, July 2005.

[12] F. Flouvat, F. De Marchi, and J-M. Petit. A thorough experimental study of datasets for frequent itemsets. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), New Orleans, USA*, pages 162–169, November 2005.

[13] H. Fu and E. Mephu Nguifo. Partitioning large data to scale up lattice-based algorithm. In *Proceedings of IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2003), Sacramento, California, USA*, pages 537–541, November 2003.

[14] J. Galambos and I. Simonelli. *Bonferroni-type inequalities with applications.* Springer-Verlag, 2000.

[15] B. Ganter and R. Wille. *Formal Concept Analysis.* Springer-Verlag, 1999.

[16] B. Goethals and M. J. Zaki. FIMI'03: Workshop on frequent itemset mining implementations. In B. Goethals and M. J. Zaki, editors, *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI 2003)*, volume 90 of *CEUR Workshop Proceedings, Melbourne, Florida, USA*, 19 November 2003.

[17] G. Grahne and J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. In B. Goethals and M. J. Zaki, editors, *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI 2003)*, volume 90 of *CEUR Workshop Proceedings, Melbourne, Florida, USA*, 19 November 2003.

[18] T. Hamrouni, S. BenYahia, and Y. Slimani. Avoiding the itemset closure computation "pitfall". In *Proceedings of the 3rd International Conference on Concept Lattices and their Applications (CLA 2005), Olomouc, Czech Republic*, pages 46–59, 7–9 September 2005.

[19] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data (SIGMOD'00), Dallas, Texas, USA*, pages 1–12, May 2000.

[20] J. Hipp, U. Gntzer, and G. Nakhaeizadeh. Algorithms for association rule mining - a general survey and comparison. In *ACM-SIGKDD Explorations*, volume 2(1), pages 58–64, July 2000.

[21] M. Kryszkiewicz. Concise representation of frequent patterns based on disjunction-free generators. In *Proceedings of the 1st IEEE International Conference on Data Mining (ICDM 2001), San Jose, California, USA*, pages 305–312, 2001.

[22] S. O. Kuznetsov and S. A. Ob"edkov. Comparing performance of algorithms for generating concept lattices. In *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, volume 14(2-3), pages 189–216, 2002.

[23] L. Lhote, F. Rioult, and A. Soulet. Average number of frequent and closed pattern in random databases. In *Proceedings of the 7th Conference Francophone d'Apprentissage Automatique (CAp 2005), Presses Universitaires de Grenoble, Nice, France*, pages 345–360, 30 May - 03 June 2005.

[24] C. Lucchese, S. Orlando, P. Palmerini, R. Perego, and F. Silvestri. kDCI: A multi-strategy algorithm for mining frequent sets. In B. Goethals and M. J. Zaki, editors, *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI 2003)*, volume 90 of *CEUR Workshop Proceedings, Melbourne, Florida, USA*, 19 November 2003.

[25] C. Lucchese, S. Orlando, and R. Perego. Fast and memory efficient mining of frequent closed itemsets. *In IEEE Journal Transactions on Knowledge and Data Engineering (TKDE)*, 18 (1):21–36, January 2006.

[26] C. Lucchesse, S. Orlando, and R. Perego. DCI-Closed: A fast and memory efficient algorithm to mine frequent closed itemsets. In B. Goethals, M. J. Zaki, and R. Bayardo, editors, *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI 2004)*, volume 126 of *CEUR Workshop Proceedings, Brighton, UK*, 1 November 2004.

[27] E. Mephu Nguifo. Gallois lattice: A framework for concept learning, design, evaluation and refinement. In *Proceedings of IEEE International Conference on Tools with Artificial Intelligence (ICTAI 1994), New-Orleans, USA*, pages 461–467, November 1994.

[28] F. Pan, G. Cong, K. H. Tung, J. Yang, and M. J. Zaki. Carpenter: finding closed patterns in long biological datasets. In *Proceedings of the 9th ACM SIGKDD International conference on Knowledge discovery and data mining*, pages 637–642, 2003.

[29] N. Pasquier. Datamining : Algorithmes d'extraction et de réduction des règles d'association dans les bases de données. Thèse de doctorat, École Doctorale Sciences pour l'Ingénieur de Clermont Ferrand, Université Clermont Ferrand II, France, Janvier 2000.

[30] N. Pasquier. Mining association rules using formal concept analysis. In *Proceedings of the 8th International Conference on Conceptual Structures (ICCS 2000), Springer-Verlag, LNAI, volume 1867, Darmstadt, Germany*, pages 259–264, August 2000.

[31] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Journal of Information Systems*, 24(1):25–46, 1999.

[32] N. Pasquier, Y. Bastide, R. Touil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In C. Beeri and P. Buneman, editors, *Proceedings of 7th International Conference on Database Theory (ICDT 1999), LNCS, volume 1540, Springer-Verlag, Jerusalem, Israel*, pages 398–416, January 1999.

[33] J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In *Proceedings of the ACM-SIGMOD International Workshop on Data Mining and Knowledge Discovery (DMKD 2000), Dallas, Texas, USA*, pages 21–30, 2000.

[34] R. Srikant. *Fast algorithms for mining association rules and sequential patterns*. Ph. D dissertation, University of Wisconsin, Madison, USA, 1996.

[35] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal. Computing iceberg concept lattices with Titanic. *Journal on Knowledge and Data Engineering (KDE)*, 2(42):189–222, 2002.

[36] T. Uno, T. Asai, Y. Uchida, and H. Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. *Proceedings of the 7th International Conference on Discovery Science, Padova, Italy*, pages 16–31, 2-5 October 2004.

[37] T. Uno, M. Kiyomi, and H. Arimura. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In B. Goethals, M. J. Zaki, and R. Bayardo, editors, *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI 2004)*, volume 126 of *CEUR Workshop Proceedings, Brighton, UK*, 1 November 2004.

[38] P. Valtchev, R. Missaoui, and P. Lebrun. A fast algorithm for building the Hasse diagram of a Galois lattice. In *Proceedings of the Colloque LaCIM 2000, Montréal, Canada*, pages 293–306, September 2000.

[39] J. Wang, J. Han, and J. Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the 9th ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2003), Washington D. C., USA*, pages 236–245, 24-27 August 2003.

[40] R. Wille. Restructuring lattices theory: An approach based on hierarchies of concepts. I. Rival, editor, Ordered Sets, Reidel, Dordrecht-Boston, p. 445-470, 1982.

[41] M. J. Zaki. Parallel and distributed association mining: A survey. *In IEEE Journal Concurrency*, 7 (4):14–25, October 1999.

[42] M. J. Zaki and C. J. Hsiao. ChARM: An efficient algorithm for closed itemset mining. In *Proceedings of the 2nd SIAM International Conference on Data Mining, Arlington, Virginia, USA*, pages 34–43, April 2002.

[43] Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms (long version). Available at http://ai.stanford.edu/users/ronnyk/realworldassoclong-paper.pdf. Accessed on February 15th 2006.

[44] Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms. In F. Provost and R. Srikant, editors, *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM Press*, pages 401–406, August 2001.

[45] J. Zhu. *Efficiently mining frequent itemsets from very large databases*. Ph. D thesis, University of Concordia, Montréal, Québec, Canada, September 2004.