Mining Frequent Patterns by Differential Refinement of Clustered Bitmaps *

Jianwei Li Alok Choudhary Nan Jiang Wei-keng Liao
Department of Electrical and Computer Engineering
Northwestern University
{jianwei, choudhar, jiangjf, wkliao}@ece.northwestern.edu

Abstract

Existing algorithms for mining frequent patterns are facing challenges to handle databases (a) of increasingly large sizes, (b) consisting of variable-length, irregularly-spaced data, and (c) with mixed or even unknown properties. In this paper, we propose a novel self-adaptive algorithm *D-CLUB* that thoroughly addresses these issues by progressively clustering the database into condensed association bitmaps, applying a differential technique to digest and remove dense patterns, and then mining the remaining tiny bitmaps directly through fast aggregate bit operations. The bitmaps are well organized into rectangular two-dimensional matrices and adaptively refined in regions that necessitate further computation. We show that this approach not only drastically cuts down the original database size but also largely reduces and simplifies the mining computation for a wide variety of datasets and parameters. We compare *D-CLUB* with various state-of-the-art algorithms and show significant performance improvement in all cases.

1 Introduction

Mining frequent patterns is a fundamental data mining task. It is widely used in Association Rule Mining [3, 4] and other corelation data analysis. Given a large database ($\mathcal{D}$) of variable-length transactions over a set of items (I), the problem is to find all frequent itemsets (FI's), each occurring at more than a minimum frequency (*minsup*).

To efficiently solve the problem, traditional algorithms like *Apriori* [4] and its variants [20, 26, 6] focus on reducing the number of database scans as well as cutting down the enumeration space of candidate itemsets (CI's). By the downward closure property of FI's - *all subsets of a frequent itemset must also be frequent* - they can iteratively enumerate, prune, and test 1-extension supersets of existing FI's and end up with much reduced number of CI's. The major problem with these algorithms is that their repeated search of profuse itemsets against the huge amount of transactions turns out to be very time-consuming and wasteful. For dense databases with enormous long FI's, the breadth-first search of the itemset lattice and subset testing to prune CI's by existing FI's are both space and time consuming.

In simplifying the itemset search computation, *Partition* [24] and *Eclat* [33] lay the foundation of a vertical mining technique to count itemset supports by intersection (merge-sort) of vertical tid-lists of corresponding items/itemsets,which can largely reduce the computation time and gets prevalent adoption in [14, 8, 25] and others. However, for large dense databases where both the amount and the lengths of the tid-lists turn out to be huge, the amount of computation and memory requirement for those algorithms grow rapidly through iterations, easily hurting the performance and scalability. Diffset optimization [31] has been proposed to track only the changes in tidsets instead of keeping the entire tid-lists through iterations so that it can significantly reduce the size of intermediate data. It also cuts down the initial vertical data sizes for dense databases by using complementary sets of tidsets but this benefit can not efficiently migrate to sparse databases.

Another innovative algorithm targeting dense databases is *FP-growth* [12]. It compacts the repetitive transactions into some concise FP-tree/trie. Transaction itemsets are organized in that frequency-ordered prefix tree so that they share common prefix part as much as possible. This approach can cut down the database sizes and reduce repeated computation for dense databases and is well adopted in [22, 16, 30, 15, 11] and others. For sparse databases, however, building/mining the FP-tree for all itemsets without step-by-step pruning would be wasteful and could suffer great penalty in both data sizes and search of the tree.

All above algorithms work on ID-based data that is either organized as variable-length records or linked by complicated structures. The tedious one-by-one search/match operations and the irregular layout of data easily become the hurdle for higher performance

*This work was supported in part by Argonne National Laboratories under contract number 3F-02201, DOE's SciDAC program with award number DE-FC02-01ER25485, NSF grants CNS-0406341, IIS-0536994, CCF-0444405, and Intel Corporation.

which can otherwise be achieved as in fast scientific computing over well-organized matrices or multi-dimensional arrays. Bitmap algorithms *HBM* [9] and *MAFIA* [7] seem to address this issue by translating the whole database into a bitmap and mine itemsets by bit-wise *AND* of corresponding columns in the bitmap. However, without addressing the inherent sparseness, the benefit of using bitmaps will be very limited or even negative because of the intuitive inefficiency of their representations for irregularly spaced data. And for dense databases, they can not reduce the huge amount of computation either.

We also refer to various algorithms for mining Maximal Frequent Itemsets (MFI's) [13, 5, 2, 7, 10, 34] or Closed Frequent Itemsets (CFI's) [21, 23, 32, 29, 19, 27, 17]. They reduce the computation and memory requirement by mining only a representative set of FI's. Note that CFI's can infer all FI's with exact supports, while MFI's can not. Both techniques work well with very dense databases with extremely long patterns, by orders of magnitude reduction of itemset mining space. However, for general databases with normal density/length of patterns, the benefit targeting long patterns is very limited, as is shown in FIMI'04 experimental results [1] where mining CFI's takes almost the same time as mining all FI's in retails, kosarak and webdocs datasets.

In this paper, we present a fundamentally new algorithm *D-CLUB* that efficiently mines all FI's by adaptive refinement of clustered bitmaps using a novel differential mining technique. Our major contributions include:

(1) A dynamic data clustering technique that progressively clusters the database into clustered bitmaps with most sparse bits removed;

(2) A differential mining technique that accumulatively removes most dense bits from clustered bitmaps, turning them into partial supports for itemsets to be mined;

(3) Organization of ragged data in rectangular two-dimensional matrices of integers that can be computed by fast aggregate bit operations in arrays;

(4) An adaptive bitmap refinement model that only computes in bitmap regions of interest to FI's.

By extensive experiments over a wide variety of datasets and parameters, we show that the size of our bitmaps is generally orders of magnitude smaller than the original database so that our algorithm can handle very large databases and is self-adaptive to various or even unknown database properties. Most importantly, with the fundamental data representation changed and improved, our mining computation is also substantially reduced and simplified, resulting in significant overall performance improvement over other state-of-the-art algorithms.

In the rest of this paper, Section 2 introduces our new concepts to cluster both itemsets and databases in clustered bitmaps. Section 3 describes our differential technique that adaptively refines the clustered bitmaps to mine all FI's with exact supports. Section 4 summarizes our *D-CLUB* algorithm. Section 5 shows our experimental results. Section 6 points out important differences between our work and related research. Section 7 draws conclusions and points out the potentials of our work.

2 Mining by Clustered Bitmaps

In this section, by using a dynamic itemset clustering approach, we hierarchically cluster the database as well as the itemsets so that each itemset cluster has its own clustered database, which then can be mined and optimized independently. Each clustered database is organized as a rectangular clustered bitmap that directly reflects the association nature of FI's and is much more condensed than the direct bitmap translation of the original entire database. In notation, we use $s(X)$ or $S(X)$ for the support or support count of an itemset X, and prefix "k-" for the itemset length.

2.1 Dynamic Itemset Clustering Itemset clustering is first introduced by Zaki *et al* [33] to facilitate the traversing of the itemset lattice. While existing approach uses fixed lexicographic ordering, hence static clustering, we propose a dynamic clustering approach to organize and enumerate all frequent itemsets with better flexibility and self-adaptivity.

DEFINITION 2.1. (FI-CLUSTER) *A* FI-cluster *is an ordered set of itemsets* $(\mathcal{C}, <)$ *recursively defined by:*

(1) The set of all 1-FI's in some specific order forms an initial FI-cluster $(\mathcal{C}_0, <)$*;*

(2) Given a FI-cluster $(\mathcal{C}, <)$*,* $\forall X \in \mathcal{C}$*,* $g(\mathcal{C}, X) \stackrel{\text{def}}{=} \{X \cup Y \mid Y \in \mathcal{C} \wedge X < Y \wedge s(X \cup Y) \geqslant \text{minsup}\}$ *also forms a FI-cluster if nonempty and ordered.*

The order in each FI-cluster is to be defined independently.

We note that the itemset order is crucial to clustering and will be dynamically determined for each FI-cluster. Based on Definition 2.1, we can generate a *FI-cluster tree* by starting from $(\mathcal{C}_0, <)$ and recursively applying the generation function $g(\mathcal{C}, X)$ on each FI-cluster and all its elements, such that each FI-cluster

Database

TID	Items
100	f d b e
200	f e b
300	a d b
400	a e f c
500	a d e
600	a c f e

Frequent Itemsets (minsup = 33%)

k	Frequent Itemsets : support
1	c:33% b:50% d:50% a:67% f:67% e:83%
2	ca:33% cf:33% ce:33% bd:33% bf:33% be:33% da:33% de:33% af:33% ae:50% fe:67%
3	caf:33% cae:33% cfe:33% bfe:33% afe:33%
4	cafe:33%

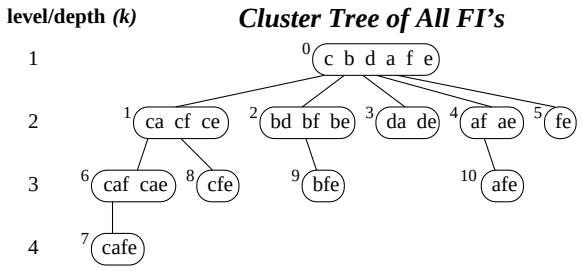


Figure 1: FI-cluster Tree for a Given DB and *minsup*.

forms a node of the tree rooted at $(\mathcal{C}_0, <)$ (level-1) and the connection between two FI-clusters denotes the generation relationship. Thereafter, we use XY in abbreviation for $X \cup Y$ if $X, Y \in \mathcal{C}$ and $X < Y$.

LEMMA 2.1. *Each level-k FI-cluster is a set of k-FI's with a common (k-1)-subset that is called the* kernel *of the cluster.*

Proof. This can be proved by induction over k.

LEMMA 2.2. *The FI-cluster tree covers all FI's exactly once.*

Proof. Since we already have Lemma 2.1, we can prove by induction the equivalent statement that all level-k FI-clusters cover all k-FI's exactly once.

The relationship between the FI-cluster tree and all FI's is disclosed by Lemma 2.1 and 2.2. Figure 1 illustrates an example FI-cluster tree for a given database and *minsup*. Here FI's in each FI-cluster are ordered by supports. We can see, for example, that cluster 6 is generated from cluster 1 by the join of "ca" with "cf" and "ce", respectively. The whole FI-cluster tree covers all FI's exactly once.

2.2 Clustered Bitmaps Once FI's are clustered, we can use the FI-clusters to cluster the database by Definition 2.2 and 2.3.

DEFINITION 2.2. (BIT-VECTOR) *Given a transactional database* $\mathcal{D} = \{\langle \text{TID}, T_i \rangle \mid T_i \subseteq I, i = 1..d\}$, *the* bit-vector *of an itemset* X *is defined to be* $bitvec(X) = [\ b_1\ b_2\ ...\ b_d\ ]^T$, *where* $b_i \in \{0, 1\}$ *and* $b_i = 1$ *iff* $X \subseteq T_i$ *for* $i = 1..d$.

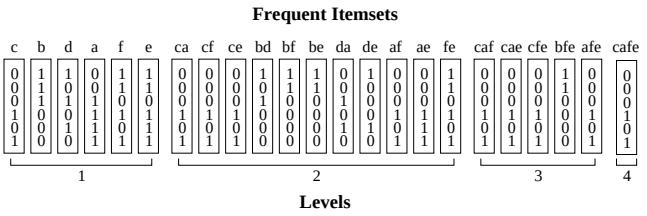


Figure 2: Bit-vectors for the Example Database.

DEFINITION 2.3. (CLUB) *Given a FI-cluster* $\mathcal{C}$ *with itemsets* $X_1 < X_2 < ... < X_n$, *the clustered bitmap of* $\mathcal{C}$ *is defined to be* $club(\mathcal{C}) = [\ bitvec(X_1)\ bitvec(X_2)\ ...\ bitvec(X_n)\ ]$, *where* $[\ X_1\ X_2\ ...\ X_n\ ]$ *serves as the horizontal coordinates.*

By definition, the support count of any itemset X is equal to the number of bit-1's contained in its bit-vector, i.e., $S(X) = population(bitvec(X))$. So the clustered bitmap (*club*) is adequate to mine the corresponding FI-cluster. Thereafter, we collectively call the combination of a FI-cluster and its *club* a *cluster*, and all such clusters are organized into a cluster tree with the same topology as FI-cluster tree. The join of FI's from a parent cluster to a child by $g(\mathcal{C}, X)$ directly maps to the bit-wise *AND* of the corresponding bit-vectors, which is proved by Lemma 2.3. Figure 2 shows the bit-vectors of FI's in the example database. We can see, for example, that the bit-vector of "$cafe$" can be directly derived by *ANDing* those of "caf" and "cae".

LEMMA 2.3. *For any two itemsets* X *and* Y, *we have* $bitvec(X \cup Y) = bitvec(X)\ \&\ bitvec(Y)$.

Proof. $\beta_1 = bitvec(X \cup Y)$, $\beta_2 = bitvec(X)\ \&\ bitvec(Y)$.
$\forall i \in 1..d$, $\beta_1[i], \beta_2[i] \in \{0, 1\}$, and
$\beta_1[i] = 1 \Leftrightarrow X \cup Y \subseteq T_i \Leftrightarrow X \subseteq T_i$ and $Y \subseteq T_i$
$\Leftrightarrow bitvec(X)[i] = 1$ and $bitvec(Y)[i] = 1$
$\Leftrightarrow (bitvec(X)[i] \wedge bitvec(Y)[i]) = 1$
$\Leftrightarrow \beta_2[i] = 1$.
So $\forall i \in 1..d, \beta_1[i] = \beta_2[i]$, which gives $\beta_1 = \beta_2$.

Given the top level cluster, we can recursively derive the whole cluster tree by joining itemsets and *ANDing* their bit-vectors following the rules indicated by $g(\mathcal{C}, X)$. That way, all FI's can be mined with exact supports calculated. We note that reordering/partitioning rows of each *club* will also reorder/partition those of its child *clubs* but does not change the total number of bit-1's in the partitions of each vertical bit-vector in the parent or child. So each *club* can be mined in horizontal partitions, where in each partition the numbers of bit-1's in the vertical bit-vectors corresponds to the support counts of the coordinate itemsets contributed by that

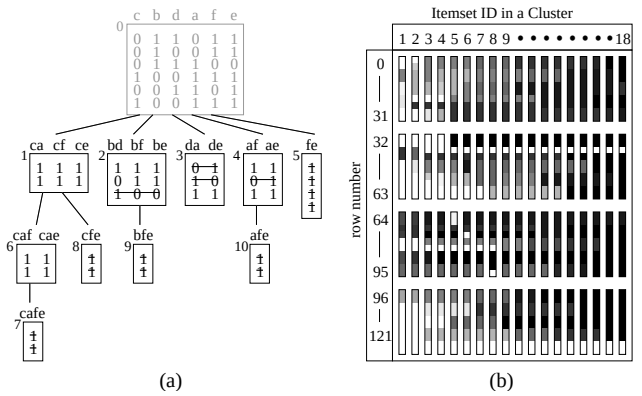


Figure 3: Mining by Clustered Bitmaps: (a) *CLUB's* corresponding to the FI-clusters for the example database; (b) organization of a prototype bitmap.

partition. By this observation, we can cluster rows of a *club* into multiple groups of which some are hard to mine but most others may be obviously easy without even mining. For example, empty rows (*e-rows*, each with all bit-0's) clearly belong to one group that can be erased as they contribute no support count for the subtree. The benefit from this idea is disclosed by Lemma 2.4, which can be directly inferred from Lemma 2.1 and the definition of support.

LEMMA 2.4. *Given a FI-cluster $\mathcal{C}$ with kernel K, the fraction of non-empty rows in club($\mathcal{C}$) is no more than support(K).*

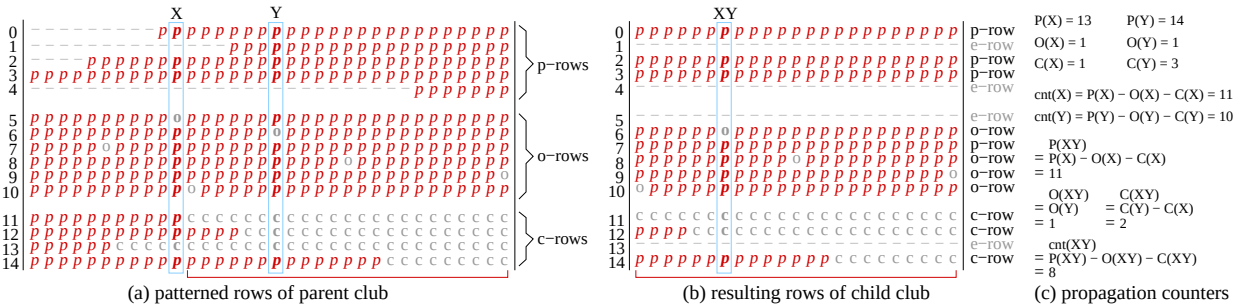
To eliminate single bit-1 noises, we also consider *a-rows*, each with only one bit-1 alone. Although they contribute support counts in current cluster, the bitwise *AND* operation will result in empty rows for all its children clusters. So after collecting the support counts, we can safely erase all those rows. With all *e-rows* and *a-rows* removed, the remaining rows are clustered into a condensed clustered bitmap (*CLUB*). Figure 3(a) shows the cluster tree with optimized *CLUB's* for the example database. We note that the level-1 bitmap is gray because we do not start from it but directly generate all level-2 *CLUB's* as our initial bitmaps instead. The number on the upper left corner of each cluster denotes the mining order that is essentially depth-first except level-2. At each level, our *CLUB's* are much more condensed and reduced than the whole bit-vectors in Figure 2. For example, at level-2, the bit-vectors have 66 bits with 25 bit-1's but our *CLUB's* have only 18 bits with 17 bit-1's. Compared to the original database, the total size of our *CLUB's* at level-2 or on each depth-first traversing path is also much smaller. The original database has 672 bits for the 21 item ID's (assumed to be 32-bit integers) while our *CLUB's* on the left-most path have only 10 bits in total.

In representation, we organize each clustered bitmap into a two-dimensional matrix by grouping each bit column into a number of 32-bit integers. This matrix can be laid out either ROW-wise or COLUMN-wise or hybrid. Thereafter, we use capital ROW/COLUMN for this integer array, with lowercase row/column still referring to bit-row/bit-column of the bitmap. We lay out and mine this integer matrix ROW-wise for better cache locality because we will mine one cluster at a time by computing ROWs of integers. To get a child clustered bitmap, for each ROW of the parent bitmap, we just *AND* the integer in the designated COLUMN with all integers on the right in that ROW. Figure 3(b) shows an example of one prototype clustered bitmap, with 18 columns and 122 rows organized into a 4×18 ROW-majored matrix of integers. We note that the last 6 bits of each integer at the bottom of the bitmap are not used and empty. In the bitmap, the level of gray represents the density of bit-1's, the darker the denser. Since we tend to order FI's by supports, there is a clear trend of increasing darkness from left to right.

3 Hierarchical Differential Refinement

As we have seen, the clustering approach can effectively handle the sparseness of bitmaps by removing most of the bit-0's so that the sizes of the resulting *CLUB's* are drastically reduced. However, for dense databases where the bitmaps are inherently dense, that technique alone will be helpless. Can we also remove most of the bit-1's? In this section, we introduce a differential mining technique that can directly attack this problem. And besides, this technique can also largely reduce the number of FI's to be actually mined while still able to infer all un-mined FI's with exact supports.

3.1 Differential Optimization In each *club*, besides *e-rows* and *a-rows*, we are also building a significant number of *p-rows*, each with zero or more leading 0's followed by trailing 1's, if we try to order the coordinate itemsets by supports. We also consider *o-rows*, each with only one 0, and *c-rows*, each with zero or more leading 1's followed by trailing 0's, to eliminate bit-0 noises or exceptional cases. For example, "0011111" is a *p-rows*, "1111011" is an *o-rows*, and "1110000" is a *c-rows*. For overlapping boundary cases of these rows, we take them as equivalences. For an itemset X, by indexing its bit in a *row* by *row*$[X]$, we say that X is covered by a *p-row* if *p-row*$[X] = 1$, by an *o-row* if *o-row*$[X] = 0$, and by a *c-row* if *c-row*$[X] = 0$. All these rows represent common itemset patterns that can propagate to children *club's*, and the results can be determined with-



(a) patterned rows of parent club (b) resulting rows of child club (c) propagation counters

Figure 4: Propagation of Common Patterns. In the bitmaps, 'p' means bit-1 and '-'/'o'/'c' means bit-0.

out even mining. For example, a full row with all bit-1's in the parent *club's* will always result in full rows in all its children. Definition 3.1 and Lemma 3.1 formalize this idea.

DEFINITION 3.1. *Given a* club *with coordinate itemsets* $[\ X_1\ X_2\ \ldots\ X_n\ ]$*, for each* X_i *where* $i = 1..n$*, we define*

$$\begin{aligned} p(X_i) &= \textit{number of}\ \text{p-rows}\ \textit{that cover}\ X_i, \\ O(X_i) &= \textit{number of}\ \text{o-rows}\ \textit{that cover}\ X_i, \\ C(X_i) &= \textit{number of}\ \text{c-rows}\ \textit{that cover}\ X_i, \\ P(X_i) &= p(X_i) + \textstyle\sum_{j=1}^{n} O(X_j) + C(X_n). \end{aligned}$$

LEMMA 3.1. *Given a FI-cluster* $\mathcal{C}$*,* $club(\mathcal{C})$*, and one of its coordinate itemsets* X*, when generating its child club by* $bitvec(X)$ & $bitvec(Y)$ *for all* $Y \in \mathcal{C}$ *and* $Y > X$ *without reordering, p-rows in the parent will result in p-rows/e-rows in the child, o-rows result in p-rows/o-rows/e-rows, and c-rows result in c-rows/e-rows. And* $P(XY)$*,* $O(XY)$ *and* $C(XY)$ *contributed by these rows can be derived by*

$$\begin{aligned} P(XY) &= P(X) - O(X) - C(X), \\ O(XY) &= O(Y), \\ C(XY) &= C(Y) - C(X). \end{aligned}$$

Intuitively, in the parent club from left to right: $P(X)$ describes the number of bit-1 *propagations* covering X, supposing all *o-rows* and *c-rows* fully propagate from the leftmost itemset; and $O(X)/C(X)$ describes that of bit-1 *omissions/cancellations* from the supposed *propagations*. These common propagations naturally extend to children club's, if those rows were actually mined. Figure 4 gives a typical example of such propagations. We note that row 11 of the child club is counted as a *c-row* instead of an *e-row* because taking an empty row as a full propagation row with full cancellation will be equivalent. Similar equivalence also applies for row 10.

Since *e-rows*, *p-rows*, *o-rows*, *a-rows* and *c-rows* in the parent club will always result in the same set of patterns in the child with known counting propagations by Lemma 3.1, we can digest those rows once and then remove them for ever. And we can get the support counts of itemsets by summing up digested results and those mined from the remaining bitmaps. Definition 3.2 and Lemma 3.2 summarize this idea. We call this differential mining because we only mine rows of bitmaps with different bits mixed disorderly.

DEFINITION 3.2. (DCLUB) *Given a FI-cluster* $\mathcal{C}$ *and its clustered bitmap* $club(\mathcal{C})$*, the differential clustered bitmap of* $\mathcal{C}$*,* $\text{dCLUB}(\mathcal{C})$*, is defined to be* $club(\mathcal{C})$ *with all* e-rows*,* p-rows*,* o-rows*,* a-rows *and* c-rows *removed.*

LEMMA 3.2. *Given a FI-cluster* $\mathcal{C}$ *and* $dCLUB(\mathcal{C})$*, for any* $X, Y \in \mathcal{C}$ *and* $X < Y$*, we have*

$$S(XY) = P(XY) - O(XY) - C(XY) + \delta(X, Y),$$

where $P(XY)$*,* $O(XY)$*,* $C(XY)$ *are given by Lemma 3.1 and* $\delta(X, Y)$ *is the count of* XY *to be mined in* $dCLUB(\mathcal{C})$*.*

In practice, we incrementally mine clustered bitmaps, remove those pattern rows, and accumulate P, O, C counters level by level. When reordering FI's in child clusters before removing those pattern rows, we need to make sure that the C counters are the same for the FI's to be reordered so that Lemma 3.1 and 3.2 still hold for deeper level propagations.

3.2 Adaptive Bitmap Refinement When refining the bitmaps along the cluster hierarchy, we not only reduce data and computation by bitmap rows, but can also adapt our computation by columns. We determine which two columns for itemsets X and Y are worth *ANDing* by the estimated support count: $E(XY) = P(XY) - O(XY) - C(XY) + min\{\delta(X), \delta(Y)\}$, where $\delta(X)$ and $\delta(Y)$ are the support counts of X and Y remaining in their *dCLUB*. Obviously $S(XY) \leqslant E(XY)$. So if $E(XY) < minsup\_count$, itemset XY can not be frequent. As the mining process goes on, the size of *dCLUB* shrinks sharper and sharper such that the support count contributed by δ becomes smaller and

smaller. As a result, the estimated support count will become more and more accurate. Particularly, when the *dCLUB* is empty, we have $E(XY) = S(XY) = P(XY) - O(XY) - C(XY)$. Actually, once there is no bits left in the bitmap, we are done with mining current subtree of FI-clusters and can simply infer all remaining FI's in that subtree by Lemma 3.3.

LEMMA 3.3. *For itemsets $X_1 < X_2 < ... < X_n$ from the same FI-cluster, if $\delta(X_i) = 0$ for all $i = 1..n$, we have*

$$S(X_1X_2...X_n) = P(X_1) - \sum_{i=1}^{n} O(X_i) - C(X_n).$$

Proof. This can be inferred from the following formula which we can prove by induction over n:

$$P(X_1X_2...X_n) = P(X_1) - \sum_{i=1}^{n-1} O(X_i) - C(X_{n-1}),$$
$$O(X_1X_2...X_n) = O(X_n),$$
$$C(X_1X_2...X_n) = C(X_n) - C(X_{n-1}).$$

When $O(X_i) = 0$ and $C(X_i) = 0$ for all $i = 1..n$, we simply have $S(X_1X_2...X_n) = P(X_1)$, which also gives an efficient way to detect MFI's with $O(1)$ complexity for any length. In Figure 3(a), for example, the *dCLUB's* for all bitmaps starting from level-2 would have zero sizes! To infer the left most subtree, where we initially have $P(ca) = P(cf) = P(ce) = 2$ and all O and C counters being 0s in cluster 1, we simply compute the support counts of joined itemsets by $S(caf) = S(cae) = S(cafe) = P(ca) = 2$ and $S(cfe) = P(cf) = 2$.

4 Proposed Algorithm to Mine All FI's

4.1 The *D-CLUB* Algorithm Combining the clustering and differential techniques discussed in Section 2 and 3, we can mine all FI's by adaptively refining the clustered bitmaps by rows and columns along the cluster tree hierarchy until all bitmaps are empty. Figure 5 summarizes our *D-CLUB* algorithm to mine all FI's in the cluster tree, using our differential clustered bitmaps. Although we configure this algorithm to start from each of the level-2 *dCLUB's* that are generated from the original database at the beginning of our mining process, this algorithm applies to the level-1 bitmap as well. We mine each subtree of clusters in a depth-first way, one cluster at a time. Columns of bitmaps can be indexed by their coordinate itemsets, e.g. $dCLUB[X]$ and $dCLUB[Y]$ in line 7. All counters are associated with their itemsets in the context. The ordering policy in line 15 will be discussed in next subsection. The function FI_infer in line 18 will generate all inferred FI's with support counts directly evaluated by Lemma 3.3, without any further mining computation.

4.2 Ordering and Self-adaptivity We note that the high efficiency of our algorithm will be greatly

```
Algorithm D-CLUB(k, C, dCLUB, P, O, C, δ)
1   foreach X ∈ C begin
2     C' = ∅, and initialize dCLUB' to be empty
3     foreach Y ∈ C ∧ Y > X begin
4       Compute P(XY), O(XY), C(XY) by Lemma 3.1
5       S(XY) = P(XY) − O(XY) − C(XY)
6       if S(XY) + min(δ(X), δ(Y)) ⩾ minsup_count then begin
7         dCLUB'[XY] = dCLUB[X] & dCLUB[Y]
8         S(XY) += population(dCLUB'[XY])
9         if S(XY) ⩾ minsup_count then C' = C' ∪ {XY}
10        else Discard column dCLUB'[XY]
11      end
12    end
13    Output frequent itemsets in C' with support counts in S
14    if |C'| ⩾ 2 then begin
15      Reorder each group of FI's (with equal C counters) in C' by
          their bitmap supports
16      Remove all e/p/o/a/c-rows from dCLUB'
17      Accumulate P, O, C and summarize new δ counters for C'
18      if dCLUB' = empty then FI_infer(k+1, C', P, O, C)
19      else D-CLUB(k+1, C', dCLUB', P, O, C, δ)
20    end
21  end
```

Figure 5: The *D-CLUB* Algorithm

supported if most of the bitmap rows turn out to be those removable pattern rows. As the FI's in each level-k FI-cluster share a common (k-1)-subset, their bit-vectors are expected to be so similar as to form a large number of *e-rows* and full rows in the clustered bitmap. However, since the bit-vectors are not always identical after all, the formation of those pattern rows suffers noises! The wider the clustered bitmap is, the more noises will become possible. We can properly define the order of FI's to bring the widths of clustered bitmaps under control and to absorb as many noises as possible by *p-rows*, *o-rows*, *a-rows* and *c-rows*.

We first order 1-FI's by their supports. In generation of level-2 FI-clusters by $g(\mathcal{C}, X)$ from the initial FI-cluster $(\mathcal{C}_0, <)$, both the less and greater 1-FI's will have limited few 1-FI's to join to form children FI-clusters of 2-FI's. For the less 1-FI's, since they have less supports, they will have less probabilities to be joined with too many 1-FI's into 2-FI's; for the greater 1-FI's, although they have greater chances to be joined with more 1-FI's into 2-FI's, they will automatically have less 1-FI's available (on the right) for join. So the sizes of level-2 FI-clusters will be very limited and evenly balanced in rough, so will the widths of level-2 clustered bitmaps. For example, in Figure 1, 1-FI's in cluster 0 $\{c, b, d, a, f, e\}$ are ordered by supports, and the sizes of its children clusters are no more than 3. However, if it happened to be reversely ordered as $\{e, f, a, d, b, c\}$, the largest child cluster $\{ef, ea, ed, eb, ec\}$ would have size of 5, making it harder to form narrow pattern rows in the clustered bitmap.

For lower level FI-clusters, the FI's will be automatically ordered when they are generated by $g(\mathcal{C}, X)$. However, we will adjust the order dynamically. For each level-2 FI-cluster, we will reorder the 2-FI's by their ac-

Table 1: Statistics of Databases and Parameters. Last 3 columns show ranges.

Dataset	DB Size (MB)	#Trans.	#Items	Avg / Max Trans. Size	*minsup* (%)	Max Length of FI	# All FI's
gazelle	1.25	60K	498	3 / 267 (mixed)	1 - 0.1	2 - 6 (short)	78 - 4K
T10I4D100K	5.00	100K	1K	10 / 29 (narrow)	1 - 0.1	3 - 10 (short)	385 - 27.5K
kosarak	41.92	990K	41K	8 / 2.5K (mixed)	1 - 0.1	5 - 18 (mid-long)	383 - 765K
webdocs	1163.34	1692K	5.27M	177 / 71.5K (wide)	30 - 5	5 - 17 (mid-long)	172 - 166M
chess	0.49	3K	76	37 / 37 (mid-wide)	90 - 10	7 - 25 (long)	628 - 4.6B
pumsb	14.41	49K	7.12K	74 / 74 (mid-wide)	90 - 30	8 - 34 (long)	2.6K - 228B

tual supports. That way, not only are the widths of deeper-level clustered bitmaps further limited, but also will more *p-rows* be automatically formed. For deeper levels, we adjust the order of FI's with equal C counters by their bitmap supports before we apply the differential technique, so that more *p-rows* will be self-adaptively formed. The remaining noises for *e-rows* will be largely absorbed by *a-rows*, and those for *p-rows* will be mostly covered by *o-rows* and *c-rows*.

With the widths limited and rows reduced, the sizes of clustered bitmaps are also self-adaptive to the database density. For sparse databases, where support is very small, Lemma 2.4 gives much less rows after all those *e-rows* are removed. For dense databases, the dense bit-1's are incrementally refined and organized into clustered bitmaps where more and more rows will become *p-rows*, *e-rows*, *o-rows*, *a-rows* and *c-rows*, which can be permanently removed, resulting in much less rows. So the differential technique combined with the clustering technique can not only remove most bit-0's for sparse databases, but also remove most bit-1's for dense databases, both resulting in extraordinarily reduced sizes in our *dCLUB's*.

5 Experimental Results

In this section, by comparing *D-CLUB* with various state-of-the-art Frequent Pattern Mining algorithms in the literature, we demonstrate the self-adaptivity of our algorithm in overall performance for a wide variety of parameters, and on various databases spanning dense and sparse, real and synthetic, small and large, and so on. We attribute the significant performance improvement to our algorithmic features that we will analyze in more detail. We choose 6 datasets with increasing densities and various other properties outlined in Table 1. Most of the datasets are obtained from FIMI repository [1], where detailed descriptions can be found. The *gazelle* dataset comes from real click-stream and purchase data and was used in KDD Cup 2000 competition. All datasets are in horizontal binary format. All the experiments were run on a LINUX PC with 3 GHz Intel Xeon CPU and 1 GB memory, unless otherwise noted.

5.1 Overall Performance Comparison For each of the datasets at different support levels, we mine all FI's with exact supports computed and compare the total execution time, including all preprocessing time, for various algorithms. We also measure their peak memory usage for a thorough comparison. In order to save experiment time, in all algorithms, we did not output the FI's to files (hundreds of GB in size for some cases). We use the implementation of *Apriori* by Borgelt and *FP-growth** by Zhu *et al* from FIMI'04 [1], *Eclat* with diffset optimization by Zaki *et al*, and *MAFIA* version 1.4 by Burdick *et al*. We also compare *D-CLUB* with two hybrid algorithms, *LCM* [28] and *kDCI* [18], that were recently proposed and recognized as best implemented ones in FIMI'04.

Figure 6 shows that our algorithm performs best in all cases. For the first 3 sparse datasets, *MAFIA* performs much worse than *Apriori* and *Eclat* because of its inefficiency in sparse bitmaps, *FP-growth** shows little advantage either, but *D-CLUB* handles the sparseness particularly well and gains 2-10 times performance improvement instead. In the last 2 dense datasets, *FP-growth** prevails over the traditional algorithms by its compact and sophisticatedly-designed FP-tree, but *D-CLUB* still out-performs it by up to 6 times benefiting from much more reduced and simplified bit operations in arrays. In the webdocs dataset, due to the large data size, *Apriori* could not finish in days, *Eclat* crashed due to memory shortage, and even *FP-growth** ran endlessly at supports lower than 15% because of large tree sizes and poor memory locality. *MAFIA* originally could not run because it did not support that many items, but after we increased that capability, it ran well. It is shown that *D-CLUB* handles this case much better, especially at lower supports, where it gains up to 20 times speedup over *MAFIA* and more than an order of magnitude over *FP-growth**. Compared with *LCM* and *kDCI*, our algorithm also runs times faster than these best implemented algorithms in most cases.

In terms of memory usage, as Figure 7 shows, our algorithm is also the best, mostly 2-8 times better than the second best that varies for cases. This low mem-

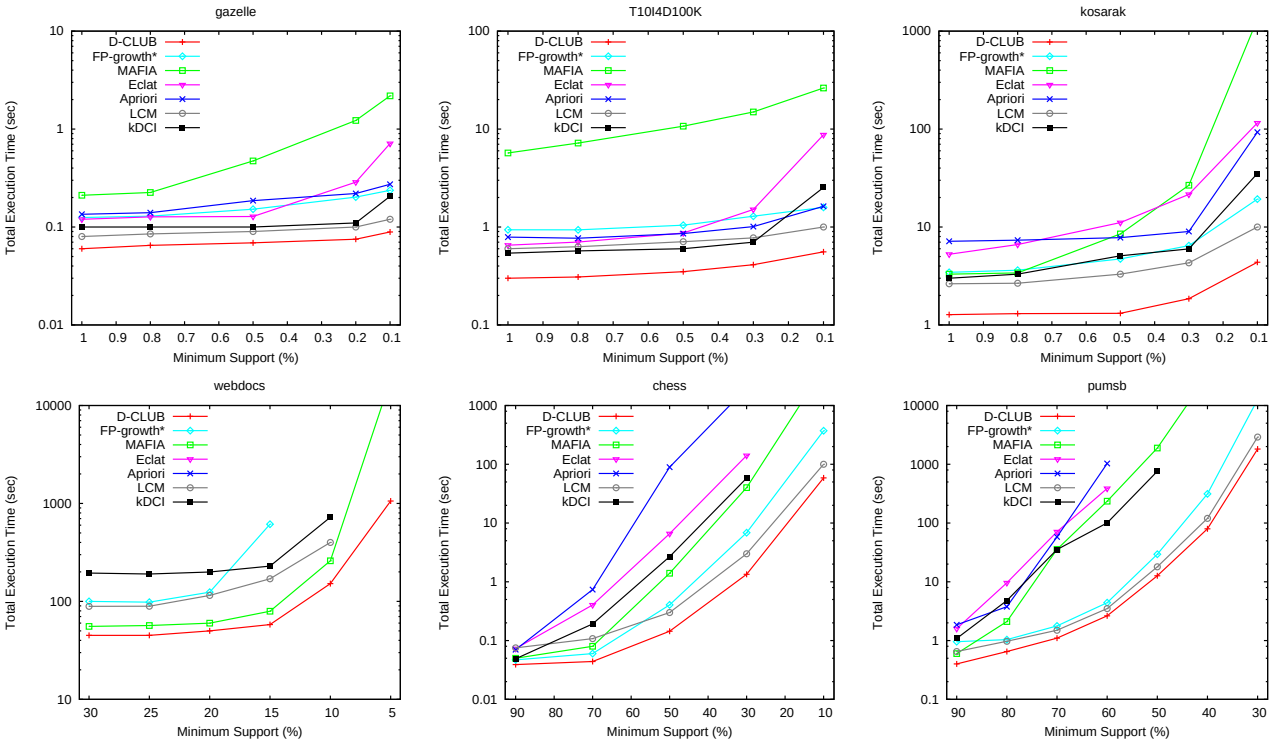


Figure 6: Total Execution Time for Mining All FI's in Different Datasets.

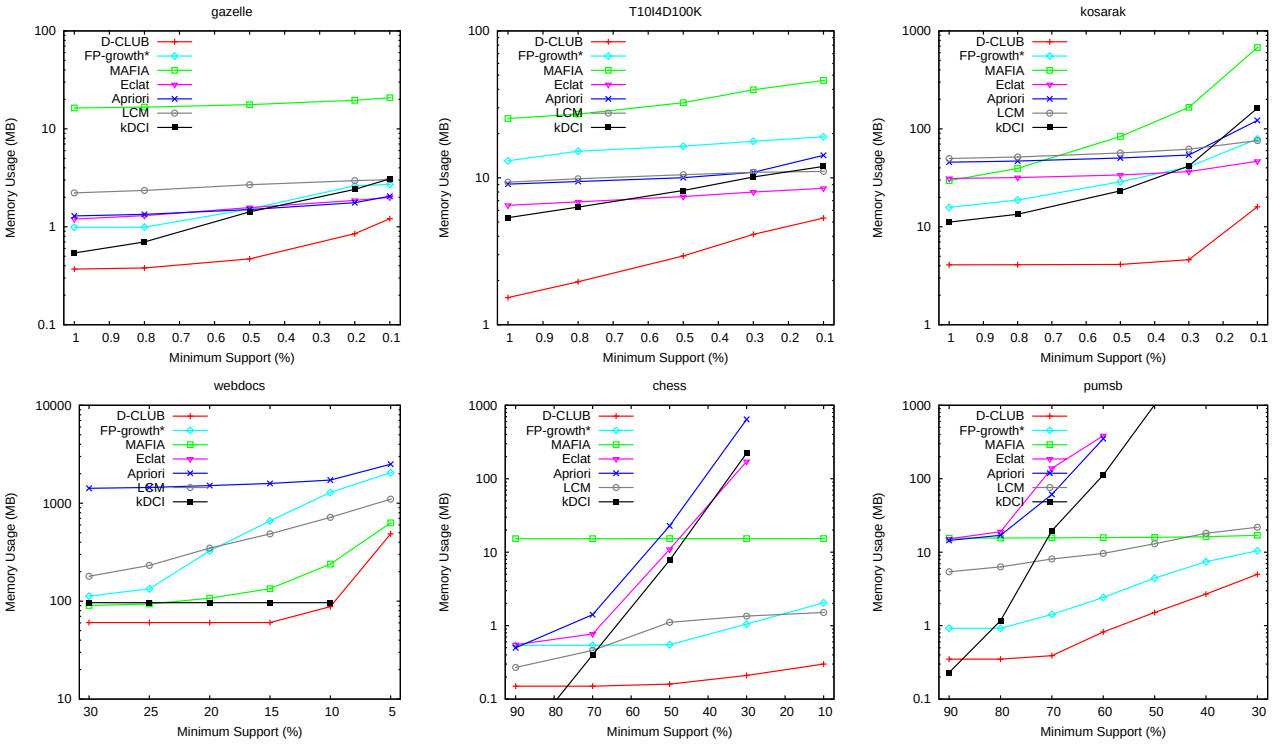


Figure 7: Memory Usage for Mining All FI's in Different Datasets.

Table 2: Data Reduction and Detailed Row Distributions in Level-2 Clustered Bitmaps.

Dataset (*minsup*)	Total *dCLUB* Size (MB)	Avg / Max *club* Width	Row Distributions				
			%e-rows	%p-rows	%o-rows	%a-rows	%c-rows
gazelle (0.2%)	0.011	4 / 14	99.34%	0.10%	0.02%	0.36%	0.00%
T10I4D100K (0.5%)	0.001	2 / 5	99.17%	0.21%	0.04%	0.56%	0.01%
kosarak (0.5%)	0.384	3 / 9	98.15%	0.75%	0.12%	0.64%	0.00%
webdocs (15%)	18.377	7 / 19	76.07%	13.68%	1.90%	1.83%	0.00%
chess (50%)	0.046	15 / 24	20.04%	47.54%	13.73%	2.78%	0.00%
pumsb (60%)	0.587	16 / 28	14.52%	60.10%	9.75%	4.22%	0.00%

ory requirement mainly benefits from the drastically reduced data sizes of our differential clustered bitmaps and the divide-and-conquer characteristic of our algorithm. We note that results well above 1GB were obtained on a machine with 3GB memory (running for days). And in the webdocs dataset, data for *Eclat* is not shown because the algorithm could not allocate enough memory to finish.

5.2 Feature Analysis for *D-CLUB* In our algorithm, the most critical part is to cluster the bitmaps in order to form as many pattern rows as possible. So the widths of bitmaps are very important, the narrower the less noises. Our dynamic clustering by ascending support can self-adaptively balance and narrow down the widths of the clustered bitmaps. As a result, a significant portion of our bitmaps ends up as pattern rows to be removed from bitmap refinement, as shown in Table 2. In sparse datasets, *e-rows* dominate, while in dense datasets, *p-rows* plus *e-rows* and *o-rows* dominate. After those pattern rows are removed, the total sizes of level-2 *dCLUB*'s are 10-5000 times smaller than those of the original databases.

In reducing the amount of computation, our algorithm also shows advantage by reducing the number of FI's to be actually mined and inferring most FI's directly via counter summarization. Figure 8 shows the percentage of FI's that are inferred in each level. From the traces, we can see that the percentage rapidly grows above 50% in the first couple of levels, and soon increases to 100%, i.e., all remaining subtrees of FI's can be fully inferred without further mining and their support counts are calculated by Lemma 3.3.

Figure 9 summarizes the performance impacts from different features of our algorithm. In each case, there are four bars. The first bar, normalized to 1, is for our fully-optimized *D-CLUB*, the second one for *D-CLUB* without clustering (mining the original unclustered bitvectors), the third one for *D-CLUB* without differential optimization (only sparse rows are removed, unaware of dense rows or P, O, C counters), and the fourth one for *D-CLUB* with all optimizations but using static

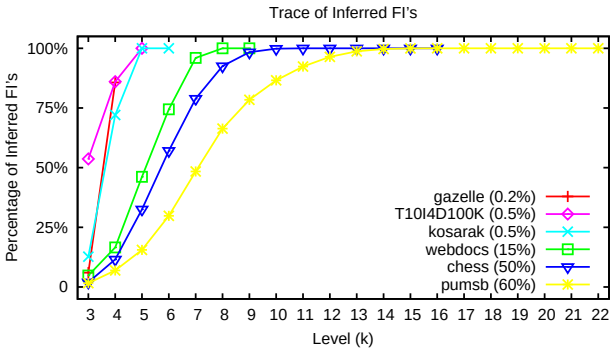


Figure 8: Per-level Percentage of Inferred FI's.

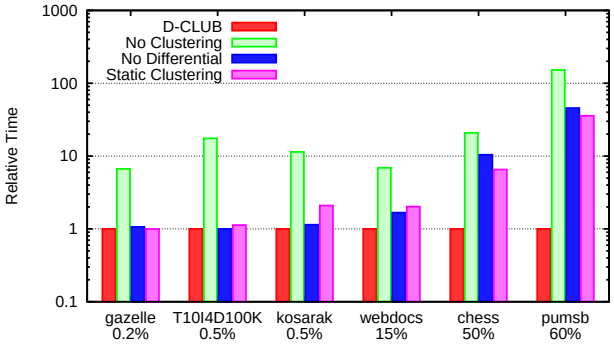


Figure 9: Performance Impacts in *D-CLUB*.

clustering by lexicographic order. We see that the differential mining technique plays an important role in dense databases but only with dynamic clustering ordered by supports. And the bitmap algorithm can not perform well without clustering in all cases.

6 Related Work

6.1 Itemset Trees There are quite a few itemset tree representations in previous work. However, their shapes, concepts and/or usages are quite different from those of the FI-cluster tree in our work. Our FI-cluster tree is graphically different from the lexicographic tree

illustrated in [2], the FP-tree in [12, 11] or AFOPT in [15]. Note that FP-tree and its variants (including AFOPT) are representations of the whole data space instead of the FI space, Also, instead of being clustered, itemsets are mixed in the tree. For example, 1-FI's are all over the tree instead of only in level one, and each 1-FI may has multiple nodes with different counts in the tree. This mixed organization substantially adds the cost to mine the FP-tree.

Our depth-first traversing approach is also different from the traditional ones. First, we traverse one cluster instead of one itemset at a time, i.e., we will finish traversing all the itemsets in a cluster before going deeper. Second, we can avoid traversing a lot of subtrees and infer all those untraversed FI's with exact supports, while other algorithms including those mining MFI's/CFI's have to reach the leaf itemsets if they are MFI's/CFI's or just candidates.

6.2 Bitmap Representation There are a number of algorithms that are also based on bitmap representation of databases. The major differences between them and our algorithm lie in the organization of the bitmaps and in how to mine and optimize the bitmaps. To our knowledge, our work is the first one to cluster the bitmaps into rectangular matrices and to mine them by adaptive differential refinement.

HBM [9], *VIPER* [25] and *MAFIA* [7] use vertical bitmap that is a direct bit translation of the original database. While the initial bitmap is usually sparse, the intermediate bitmaps are even sparser and huge in size, requiring a large amount of processing time and huge storage space. *HBM* avoids unnecessary processing on some grouped bit-0's by using secondary bitmap indices to guide the algorithm to dense bits. *VIPER* uses some generic bit compression algorithm to compress/decompress the bitmap to save storage space, but has to mine every bit no matter 0 or 1. It is also uniquely different in that it does not mine bitmap directly. Instead, it transforms bit-vectors back into tid-lists and performs intersection. This tedious transformation between tid-vectors and tid-lists back and forth adds extra overhead instead of reducing the traditional intersection computation, no need to mention the compression/decompression overhead. *MAFIA* is different from previous two (and *D-CLUB*) in that it mines itemset $X = Pi_1i_2$ by $bitvec(Pi_1)$ & $bitvec(i_2)$, instead of by $bitvec(Pi_1)$ & $bitvec(Pi_2)$, where P is the prefix and i_1, i_2 are the tail items. It conditionally avoids unnecessary operation on some bit-0's by projecting the bit-vector of the *tail* item against that of the *head* itemset on the fly, which is actually a full scan for each sparse bit-vector and turns out to be alternatively expensive. Differently, in *D-CLUB*, we permanently remove most of the bit-0's (and also bit-1's) by using clustering and differential techniques so that we do not waste time processing those unnecessary bits again and again.

In addressing the I/O issue caused by the increasingly large database or its bitmap translation, these three algorithms use quite different approaches than *D-CLUB*. *HBM* is limited to reducing the number of scans of the original database/bitmap, at the cost of mining more CI's. *MAFIA* tries also to reduce the number of full scans of the database/bitmap by traversing the itemset tree in depth-first way, but it has to scan multiple times the bit-vectors for revisited *tail* items. *VIPER* directly attacks the size of bitmap by compression, but the compression ratio is upper-bounded by 32 and practically much lower in average cases [31]. Besides, since it is a breadth-first approach, it will generate intermediate bitmaps in order not to scan and mine the original bitmaps again and again. As mentioned, the total size of the intermediate bitmaps in one level can be huge, and it either suffers this expensive I/O, or skip this stage of bitmaps, at the cost that the next iteration has to start all over again from some earlier stage. However, in *D-CLUB*, we directly and effectively address the sparseness and huge size of the bitmaps through our unique clustering and differential techniques. We reduce the original database into level-2 differential clustered bitmaps as our initial database, and achieve average data reduction ratio of several orders of magnitude. We avoid the exponentially growing width/size of intermediate bitmaps by mining the cluster tree in a depth-first way, so that we only need to handle one clustered bitmap at a time. While a clustered bitmap is guaranteed to be much smaller than the original bitmap and to reduce level by level, it keeps shrinking more sharply as we traverse deeper and incrementally apply clustering and differential optimizations. Also the organization of our bitmaps is very different. While the other three always lay out the whole bitmap into a pile of long vertical bit-vectors, we decouple the original vertical bitmap into multiple smaller clustered bitmaps and organize each of them into a ROW-majored matrix of integers. This ROW-majored organization enables us to optimize and mine the bitmaps both vertically and horizontally! While retaining the advantage of vertical bitmap mining, our ROW-wise mining approach usually results in better cache locality than mining the whole long vertical bit-vectors a pair at a time. We note that *HBM* alleviates this length problem by processing one horizontal partition at a time.

Another algorithm that makes use of bitmap representation is *DepthProject* [2], which is uniquely different in that it uses a projected **horizontal** bitmap

to mine each itemset subtree. The projected bitmap only contains bits for the working set of items and is expected to be dense. However, when traversing the lexicalgraphical tree in depth-first way, the algorithm needs to scan the original database again and again, only to extract the projected portion each time, which is very expensive in both I/O and projection computation. Also, like *MAFIA*, it is a selective projection approach, and when the criteria is not met, it works on the unprojected database. And the definition of the bitmap is slightly different. Instead of itemsets, the bits represent the presence/absence of single items in the projected database.

7 Conclusions and Future Work

In this paper, we presented a novel self-adaptive algorithm *D-CLUB* that is based on our fundamentally new data clustering and differential bitmap refinement techniques. We showed that our techniques directly addressed the major Frequent Pattern Mining issues by organizing the drastically reduced data in rectangular matrices and performing adaptive bitmap refinement through fast aggregate bit operations. With the data size and representation fundamentally improved in our differential clustered bitmaps, the mining computation was also substantially reduced and simplified. Our performance results showed significant improvements over existing algorithms for a wide variety of cases.

The tiny data sizes, extremely fast speed, and very simple instructions, make it very attractive to implement our algorithm on hardware so that even a handheld device can promptly perform practical Frequent Pattern Mining tasks that could only be handled by supercomputers in the old days! On the other hand, the independence between clustered bitmaps enables us to divide and conquer very large scale problems, where the databases can be clustered, distributed and independently mined across a large number of parallel processors with linear scalability.

References

[1] Frequent Itemset Mining Implementations Repository. http://fimi.cs.helsinki.fi/.

[2] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. In *Proc. ACM SIGKDD'00*, August 2000.

[3] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD'93*, May 1993.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. VLDB'94*, September 1994.

[5] R. Bayardo. Efficiently mining long patterns from databases. In *Proc. ACM SIGMOD'98*, June 1998.

[6] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. ACM SIGMOD'97*, May 1997.

[7] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transational databases. In *Proc. IEEE ICDE'01*, April 2001.

[8] B. Dunkel and N. Soparkar. Data organization and access for efficient data mining. In *Proc. IEEE ICDE'99*, March 1999.

[9] G. Gardarin, P. Pucheral, and F. Wu. Bitmap based algorithms for mining association rules. In *BDA'98*, October 1998.

[10] K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In *Proc. IEEE ICDM'01*, November 2001.

[11] G. Grahne and J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. In *IEEE ICDM'03 Workshop FIMI'03*, November 2003.

[12] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD'00*, May 2000.

[13] D.-I. Lin and Z. M. Kedem. Pincer-search: A new algorithm for discovering the maximum frequent set. In *Proc. 6th Int'l. Conf. Extending Database Technology*, March 1998.

[14] J.-L. Lin and M. H. Dunham. Mining association rules: Anti-skew algorithms. In *Proc. ICDE'98*, February 1998.

[15] G. Liu, H. Lu, Y. Xu, and J. X. Yu. Ascending frequency ordered prefix-tree: Efficient mining of frequent patterns. In *Proc. IEEE 8th Int'l. Conf. Database Systems for Advanced Applications*, March 2003.

[16] J. Liu, Y. Pan, K. Wang, and J. Han. Mining frequent item sets by opportunistic projection. In *Proc. ACM SIGKDD'02*, July 2002.

[17] C. Lucchese, S. Orlando, and R. Perego. DCI Closed: A fast and memory efficient algorithm to mine frequent closed itemsets. In *IEEE ICDM'04 Workshop FIMI'04*, November 2004.

[18] S. Orlando, C. Lucchese, P. Palmerini, R. Perego, and F. Silvestri. kDCI: a multi-strategy algorithm for mining frequent sets. In *IEEE ICDM'03 Workshop FIMI'03*, November 2003.

[19] F. Pan, G. Cong, A. K. H. Tung, J. Yang, and M. J. Zaki. Carpenter: Finding closed patterns in long biological datasets. In *Proc. ACM SIGKDD'03*, August 2003.

[20] J. S. Park, M.-S. Chen, and P. S. Yu. An effective hash based algorithm for mining association rules. In *Proc. ACM SIGMOD'95*, May 1995.

[21] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. *Lecture Notes in Computer Science*, 1540:398–416, 1999.

[22] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang. H-Mine: Hyper-structure mining of frequent patterns in large databases. In *Proc. IEEE ICDM'01*, November

2001.

[23] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD'00 Workshop on Research Issues in Data Mining and Knowledge Discovery*, May 2000.

[24] A. Savasere, E. Omiecinski, and S. B. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. VLDB'95*, September 1995.

[25] P. Shenoy, J. R. Haritsa, S. Sundarshan, G. Bhalotia, M. Bawa, and D. Shah. Turbo-charging vertical mining of large databases. In *Proc. ACM SIGMOD'00*, May 2000.

[26] H. Toivonen. Sampling large databases for association rules. In *Proc. VLDB'96*, September 1996.

[27] T. Uno, T. Asai, Y. Uchida, and H. Arimura. LCM: An efficient algorithm for enumerating frequent closed item sets. In *IEEE ICDM'03 Workshop FIMI'03*, November 2003.

[28] T. Uno, M. Kiyomi, and H. Arimura. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *IEEE ICDM'04 Workshop FIMI'04*, November 2004.

[29] J. Wang, J. Han, and J. Pei. CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In *Proc. ACM SIGKDD'03*, August 2003.

[30] Y. Xu, J. X. Yu, G. Liu, and H. Lu. From path tree to frequent patterns: A framework for mining frequent patterns. In *Proc. IEEE ICDM'02*, December 2002.

[31] M. J. Zaki and K. Gouda. Fast vertical mining using diffsets. In *Proc. SIGKDD'03*, August 2003.

[32] M. J. Zaki and C.-J. Hsiao. CHARM: An efficient algorithm for closed itemset mining. In *Proc. 2nd SIAM Int'l. Conf. on Data Mining*, April 2002.

[33] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. Technical Report TR651, University of Rochester, July 1997.

[34] Q. Zou, W. W. Chu, and B. Lu. SmartMiner: A depth first algorithm guided by tail information for mining maximal frequent itemsets. In *Proc. IEEE ICDM'02*, December 2002.