Computing and Informatics, Vol. 37, 2018, 915–945, doi: 10.4149/cai_2018_4_915

MAXPART: AN EFFICIENT SEARCH-SPACE PRUNING APPROACH TO VERTICAL PARTITIONING

Benameur Ziani, Youcef Ouinten, Mustapha Bouakkaz

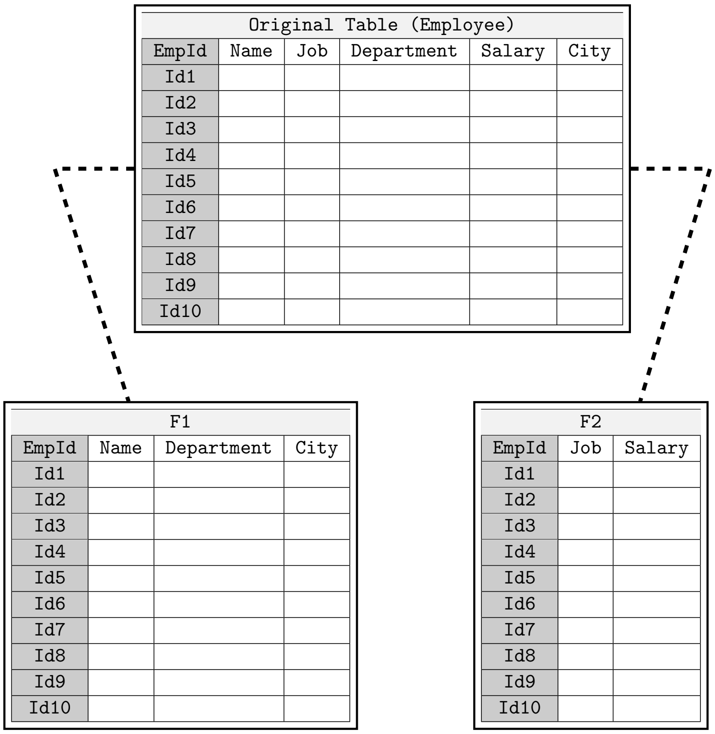
LIM – Department of Informatics
University of Laghouat, BP 37G M'kam 03000, Algeria
e-mail: {bziani, ouinteny, m.bouakkaz}@lagh-univ.dz

Abstract. Vertical partitioning is the process of subdividing the attributes of a relation into groups, creating fragments. It represents an effective way of improving performance in the database systems where a significant percentage of query processing time is spent on the full scans of tables. Most of proposed approaches for vertical partitioning in databases use a pairwise affinity to cluster the attributes of a given relation. The affinity measures the frequency of accessing simultaneously a pair of attributes. The attributes having high affinity are clustered together so as to create fragments containing a maximum of attributes with a strong connectivity. However, such fragments can directly and efficiently be achieved by the use of maximal frequent itemsets. This technique of knowledge engineering reflects better the closeness or affinity when more than two attributes are involved. The partitioning process can be done faster and more accurately with the help of such knowledge discovery technique of data mining. In this paper, an approach based on maximal frequent itemsets to vertical partitioning is proposed to efficiently search for an optimized solution by judiciously pruning the potential search space. Moreover, we propose an analytical cost model to evaluate the produced partitions. Experimental studies show that the cost of the partitioning process can be substantially reduced using only a limited set of potential fragments. They also demonstrate the effectiveness of our approach in partitioning small and large tables.

Keywords: Information systems, knowledge extraction, data mining, maximal frequent itemsets, database design, vertical partitioning

1 INTRODUCTION

Database applications are often characterized by a large volume of data and high demands with regard to query response time and transaction throughput. Vertical partitioning is an effective way of improving performance in the database systems where a significant percentage of query processing time is spent on the full scans of tables. It represents an important aspect of physical database design that have significant impact on performance and manageability [1]. Figure 1 illustrates the principle of vertical partitioning. Consider a classical example of the employee table (original table).



Original Table (Employee)

EmpId	Name	Job	Department	Salary	City
Id1					
Id2					
Id3					
Id4					
Id5					
Id6					
Id7					
Id8					
Id9					
Id10					

F1

EmpId	Name	Department	City
Id1			
Id2			
Id3			
Id4			
Id5			
Id6			
Id7			
Id8			
Id9			
Id10			

F2

EmpId	Job	Salary
Id1		
Id2		
Id3		
Id4		
Id5		
Id6		
Id7		
Id8		
Id9		
Id10		

Figure 1. Principle of vertical partitioning (`EmpId` is the key)

Assume that some employees' information, such as `Name`, `Department` and `City` is frequently required together. When scanning the employee table to fetch this information, other non-relevant information on employee such as `Job` and `Salary` will also be loaded. In such a situation and according to performance expectations, the administrator can split the original table into two fragments $F1$ (`Name`, `Department`, `City`) and $F2$ (`Job`, `Salary`). Such a partitioning is beneficial since it avoids access to non-relevant information and thus significantly reduces the I/O requirements during query processing. In order to minimize the costs of accessing the required data, the relation is partitioned in such a way that each query uses as few fragments as possible. The ideal partitioning is obtained if each query would have to access a single fragment containing exactly the attributes it references, resulting in minimal I/O requirements. Realistically, however, overlapping queries reduce the efficiency of the partitioning. In such a case additional joins between two or more fragments are required to fetch the desired data.

In vertical partitioning, the attributes of a relation $\mathcal{R}$ are clustered into non-overlapping groups and the original relation $\mathcal{R}$ is projected into fragment relations according to these attribute groups. The result of the fragmentation process is a set of fragments defined by a partitioning scheme. The aim is to find a partitioning scheme which minimizes the cost of accessing data during query processing.

However, selecting a suitable partitioning scheme is a difficult problem to solve, since a large space of alternatives must be considered. A table can be vertically partitioned in many different ways. The vertical partitioning problem is computationally intractable. Indeed, if a relation $\mathcal{R}$ has m non-primary key attributes, the possible fragments are given by the Bell number [2] which is approximately $B(m) \approx m^m$. Hence, on the one hand, the complexity of the partitioning problem increases exponentially with the number of attributes. Vertical partitioning, on the other hand, basically stores attributes that are frequently accessed together based on a given workload. But the latter may change over time which implies that the partitioning process may need to be performed very often. Accordingly, finding suitable vertical partitions is a daunting task even for skilled database administrators (DBAs). Thus, database administrators (DBAs) are faced with the challenging task of determining the appropriate choice of partitioned tables and, therefore, there is a practical need for strategies that assist the DBAs in this process.

Studies dealing with the vertical partitioning problem have focused on reducing its complexity and finding approximate solutions using heuristics-based approaches. A basic question related to vertical partitioning is how the attributes are referenced in a given workload? Therefore, most of the proposed algorithms cluster the attributes of a relation according to their *affinity*. Attribute affinity expresses a bond between a pair of attributes. The affinity between two attributes A_i and A_j measures the total number of accesses of queries referencing both attributes A_i and A_j. The core idea of affinity based partitioning is to compute affinities between every pair of attributes and then to cluster them such that high affinity pairs are as close in neighbourhood as possible.

The drawback of affinity based approaches is that the used measure does not reflect the closeness or affinity when more than two attributes are involved. In such approaches, all possible grouping of couples of attributes having high affinity should be examined. During regrouping the attributes are moved between fragments to achieve any possible improvement. This task requires a large number of comparison operations between affinity values of more than two attributes which can be very costly for a large representative workload.

The natural way to reflect the closeness of k attributes of a given relation $\mathcal{R}$ is to measure the accessing frequency of sets of attributes with different size s ($1 \leq s \leq k$). This measure can be achieved by the means of mining frequent itemsets. This technique helps to discover important associations among attributes such that the presence of some attributes in a query will imply the presence of some other attributes. Thus, the partitioning process can be done faster and more accurately with the help of such knowledge discovery technique of data mining. However, the use of all frequent itemsets is limited by the high computational cost as well as the large number of resulting outputs. Such approach generates an enormous number of candidate fragments which leads to high computational overheads.

In this paper we propose `MaxPart` – an approach for finding an optimized solution to vertical partitioning using maximal frequent itemsets. The proposed approach exploits the input workload information to intelligently prune the search space of the optimal solutions. It measures the correlation of attribute sets as naturally expected by the partitioning process. Taking a representative workload as an input, `MaxPart` goes through two major steps

1. enumerating potential fragments, that we call candidate fragments, according to the workload characteristics using maximal frequent itemsets technique;
2. generating possible partitioning schemes exploiting the candidate fragments and selecting the best one according to the workload cost.

We address the complexity of vertical partitioning problem from the perspective of counting a reduced number of optimized possible solutions as efficiently as possible. We are motivated by the desire to achieve computationally simpler, but at least equivalent, solutions for the studied problem. The counting aspect reveals the inherent computational complexity of the partitioning process. We believe that maximal frequent itemsets offer an attractive alternative to achieve this goal.

The reminder of the paper is as follows. We present in Section 2 an example to motivate our proposal. Section 3 provides background information on the studied problem. In Section 4, we investigate related work on vertical partitioning in relational databases and maximal frequent itemsets mining. Section 5 presents the proposed analytical cost model for evaluating generated partitioning schemes. We present the proposed approach to solve the vertical partitioning problem in Section 6. Section 7 deals with the experimental study of the proposed approach. We conclude the paper and present future directions in Section 8.

2 MOTIVATING EXAMPLE

We will first introduce the principle of MaxPart approach through a simple motivating example. An obvious, but important observation, is that the efficiency of the partitioning process depends on the interestingness of the generated fragments in terms of size (length) and frequency. Based on a given workload, expected fragments must, intuitively, store the maximum of attributes that are frequently accessed together.

	Queries	**Freq.**
q_1	Select A, B, E	1
q_2	Select B, E	3
q_3	Select A, D, F	3
q_4	Insert SQL	2
q_5	Insert SQL	1

a)

Extraction Context					
A	B	E			
B	E				
B	E				
B	E				
A	D	F			
A	D	F			
A	D	F			
A	B	C	D	E	F
A	B	C	D	E	F
A	B	C	D	E	F

b)

	AllPart	**MaxPart**
1-Fragment	A, B, D, E, F	–
2-Fragment	AB, AD, AE DF, AF, BE	–
3-Fragment	ADF, ABE	ADF, ABE

c)

Figure 2. Example of a) input workload, b) corresponding extraction context and c) candidate fragments

Example 1. The following example is extracted from [3]. It illustrates the efficiency of using maximal frequent itemsets for the partitioning process. Instead of using all frequent itemsets, our approach, called MaxPart, performs the vertical partitioning exploiting the interesting properties of maximal frequent itemsets. Consider a set of 5 queries $q_1, \ldots, q_5$ referencing the attributes A, B, C, D, E, and F. The queries and their frequencies are illustrated in Figure 2 a). Figure 2 b) shows the corresponding extraction context. For a threshold value % of 40 %, the set of candidate fragments generated using all frequent itemsets, which we call AllPart, and by the means of MaxPart approach are shown in Figure 2 c). AllPart generates 13 candidate fragments, while MaxPart generates only two candidate fragments.

According to the algorithm proposed in [3], the process of generating the possible partitioning schemes is done as follows. AllPart starts by considering the fragment with the maximal length $\{A, D, F\}$ as the first fragment of the partition. It scans successively the 2-itemsets and find that $\{B, E\}$ does not overlap with the existing partition. It forms the second fragment. The remaining attribute $\{C\}$, that is non frequent, forms the third fragment. The result of this iteration is the partitioning scheme $[\{A,D,F\},\{B,E\},\{C\}]$. Similarly, the second large fragment $\{A,B,E\}$ leads to the scheme $[\{A,B,E\},\{D,F\},\{C\}]$. The MaxPart approach performs only two comparisons between the fragments $\{A,D,F\}$ and $\{A,B,E\}$ to generate the same partitioning schemes. Indeed, the maximal fragment $\{A,D,F\}$ is firstly compared to the fragment $\{A,B,E\}$ to deduce the second fragment $\{B,E\}$. Similarly, the comparison between the fragments $\{A,B,E\}$ and $\{A,D,F\}$ generates the second fragment $\{D,F\}$. In both cases, the remaining attribute $\{C\}$, which is non frequent, forms the third fragment.

The above example clearly shows that our approach prunes significantly the search space for the partitioning process. When processing a potential fragment F_k, which has a maximal length, AllPart approach needs to compare it to the fragments $F_{k-1}, F_{k-2}, \ldots, F_1$ resulting in a larger total computation cost. However, most of the fragments $F_{k-1}, \ldots, F_1$ will not be used any more since they are included in the maximal ones. As the number of attributes increases using large representative workloads, the computational cost grows exponentially. Obviously, the efficiency of the partitioning process depends on the number of enumerated fragments. Indeed, the selection of an optimized partitioning scheme becomes computationally more complex when there is a huge number of candidate fragments to choose from. As the number of the candidate fragments becomes longer, the number of possible comparisons becomes larger, thus the pruning effect of our approach is sharper. Hence, our proposal seems a good approach to cope with scalability issues. We believe that the way partitioning schemes are achieved defines the approach's ability to scale. The objective of our approach is to reduce the computational cost of the partitioning process without compromising its correctness.

3 BACKGROUND

3.1 Workload-Based Vertical Partitioning

Fragmentation is a design technique to divide a single database into two or more partitions such that the combination of the partitions yields the original database without any loss or addition of information [4]. The result of the fragmentation process is a set of fragments defined by a partitioning scheme. The objective is to create vertical fragments of a relation so as to minimize the cost of accessing data during transaction processing. In vertical partitioning, attributes of a relation $\mathcal{R}$ are clustered into groups and the relation $\mathcal{R}$ is projected into fragment relations according to these attribute groups.

A general formulation of the vertical partitioning problem is as follows: Given a relation $\mathcal{R}$ of k attributes $\mathcal{R} = \{A_1, A_2, \ldots, A_k\}$ and a representative workload $\mathcal{W}$ of n queries $\{q_1^{f_1}, q_2^{f_2}, \ldots, q_n^{f_n}\}$, where each query $q_i (1 \leq i \leq n)$ has an access frequency f_i, the vertical partitioning problem involves selecting a partitioning scheme, $\mathcal{F} = \{F_1, \ldots, F_m\}$ among all possible partitioning schemes such that:

1. Every fragment $F_i \subseteq \mathcal{F}$ is composed of a subset of the attributes of $\mathcal{R}$ plus the identifier column (primary key) which is used for join operations to reconstruct the original data.
2. $\forall F_i \in \mathcal{F}, \forall F_j \in \mathcal{F}, i \neq j : F_i \cap F_j = \emptyset$ (except for the primary key).
3. $\mathcal{R} = \bowtie_{i=1}^{m} F_i$.
4. The cost of processing the workload $\mathcal{W}$ using the partitioning scheme $\mathcal{F}$ is minimum.

In order to minimize the costs of accessing the required data, the relation is partitioned in a way that each query uses as few fragments as possible. The ideal partitioning scheme is obtained if each query q_i in the workload $\mathcal{W}$ would have to access a single fragment containing exactly the attributes it references, resulting in minimal I/O requirements. Realistically, however, the workload will contain overlapping between queries which reduces the efficiency of vertical partitioning. In such a case additional joins between generated fragments are required to answer some of the queries in the workload.

3.2 Basic Concepts on Frequent Itemsets Mining

To facilitate the understanding of our approach, we briefly sketch, in this section, the key notions on frequent itemsets mining.

Definition 1 (Extraction context)**.** An extraction context (or a formal 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 attributes (or items) and $\mathcal{R}$ is a binary relation (i.e., $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{I}$). Each pair $(o, i) \in \mathcal{R}$ expresses that the object $o \in \mathcal{O}$ contains the item $i \in \mathcal{I}$.

	A	B	C	D	E
1			x	x	
2		x	x		
3	x				x
4		x			x
5		x	x		

Table 1. Example of extraction context

An example is illustrated in Table 1. Transactions are denoted by numbers and items by letters. We have

$$\mathcal{I} = \{A, B, C, D, E\}, \quad \mathcal{O} = \{1, 2, 3, 4, 5\}$$

and

$$\mathcal{R} = \{(1, C), (1, D), (2, B), (2, C), (3, A), (3, E), (4, B), (4, E), (5, B), (5, C)\}.$$

Definition 2 (Itemset)**.** An itemset (or k-itemset) is a subset $I \subseteq \mathcal{I}$ that contains k items ($|I| = k$).

Definition 3 (Support of an itemset)**.** Let $\mathcal{K} = (\mathcal{O}, \mathcal{I}, \mathcal{R})$ be an extraction context and $I \subseteq \mathcal{I}$ be an itemset. The support of I, denoted Support(I), is the number of transactions containing all the items of I, divided by the total number of transactions:

$$\text{Support}(I) = \frac{|\{o \in \mathcal{O} / (\forall i \in I, (o,i) \in \mathcal{R}|}{|\mathcal{O}|}.$$

Definition 4 (Frequent itemset)**.** Let $\mathcal{K} = (\mathcal{O}, \mathcal{I}, \mathcal{R})$ be an extraction context and $I \subseteq \mathcal{I}$ be an itemset. The itemset I is said to be frequent if Support$(I) \geq$ minsup where minsup is a user-defined support threshold.

Definition 5 (Maximal frequent itemset)**.** Let $\mathcal{K} = (\mathcal{O}, \mathcal{I}, \mathcal{R})$ be an extraction context and $I \subseteq \mathcal{I}$ be an itemset. The itemset I is said to be maximal frequent itemset if I is frequent and no super-set of I is frequent:

$$\{(\text{Support}(I) \geq \text{minsup}) \wedge (\forall J \subseteq \mathcal{I} : I \subset J \Rightarrow \text{Support}(J) < \text{minsup})\}.$$

Definition 6 (Frequent itemset mining problem)**.** Let $\mathcal{K} = (\mathcal{O}, \mathcal{I}, \mathcal{R})$ be an extraction context. The frequent itemsets mining problem requires to discover all frequent itemsets given a user-defined minimum support minsup.

4 RELATED WORK

4.1 Vertical Partitioning in Relational Databases

The idea of data partitioning was proposed in the early days of databases as a means to increase I/O performance and it has been studied in different contexts. It is worth pointing that the first works on the vertical partitioning have been proposed in the context of centralized relational databases. With the emergence of new models of data, the existing approaches have been widely adopted by taking into account the characteristics of each of the emerging models. For example, in a centralized context, the fragmentation aims at reducing the query processing costs whereas in a distributed context it aims also at achieving a better distribution of data over distant sites so as to achieve a parallelized treatment of queries. Thus, several vertical partitioning approaches have been proposed for centralized databases [3, 5, 6, 7, 8, 9, 10, 11], distributed databases [12, 13, 14, 15, 16, 17, 18], object-oriented databases [19, 20, 21, 22], data warehouse design [23, 24, 25] and XML database systems [26, 27, 28, 29]. Due to space constraints, we cannot possibly list all the related references here. Instead, we will restrict our discussion to related work on vertical partitioning in centralized relational databases and to used techniques that are directly related to our work.

The NP-hard nature of the vertical partitioning was pointed pretty early [2]. Therefore, studies dealing with this problem have focused on reducing its complexity and finding approximate solutions using heuristics-based approaches. In the literature, most of the proposed algorithms cluster attributes of a relation according to their affinity [5, 6, 7, 8, 9]. The core idea of affinity based partitioning is to compute affinities between every pair of attributes and then to cluster them such that high affinity pairs are as close in neighbourhood as possible. The proposed approaches measure the affinity between pairs of attributes and try to cluster attributes according to their pairwise affinity by using the bond energy algorithm(BEA) [30]. They started from constructing an attribute usage matrix (AUM) to construct the attribute affinity matrix (AAM) on which clustering is performed. The affinity matrix is an $n \times n$ matrix for the n-attribute problem where the (i, j) element equals the *between-attributes* affinity. Affinity between attributes measures the total number of accesses of queries referencing both attributes i and j. The attribute affinity matrix helps to perform a first clustering of attributes. A binary partitioning is repetitively applied in a second step. The authors in [9] developed an algorithm based on graphical technique. It considers the attribute affinity matrix as a complete graph called the *affinity graph*. Each edge value in the affinity graph represents the affinity between two attributes. A linearly connected spanning tree is constructed from the affinity graph and all cycles of the spanning tree form fragments of the relation.

Despite their simplicity, affinity based approaches, however, have the following shortcomings.

1. The metric used for clustering the attributes does not reflect the closeness or affinity when more than two attributes are involved. Thus, it complicates the comparison process between affinity values of more than two attributes which can be very costly for a large workload. In such approaches, all possible grouping of couples of attributes having high affinity should be examined. During regrouping, attributes are moved between fragments to achieve any possible improvement.
2. The proposed approaches performs a binary partitioning. Realistically, however, this is not always an optimal solution. In real cases, the attributes can be grouped into more than two fragments especially for a table having a very large number of attributes.
3. Most of the proposed vertical partitioning algorithms do not have an objective function to evaluate the *goodness* of partitions that they produce.

Some approaches use Genetic Algorithms to perform the partitioning [10, 11]. A binary string is used as the genetic representation of the partitioning. As an example, suppose a relation to be partitioned consists of ten attributes. A binary string (solution or chromosome) representing a binary fragmentation scheme is 1110001010. This solution specifies that attributes 1, 2, 3, 7 and 9 constitute one fragment, while attributes 4, 5, 6, 8 and 10 constitute the other fragment. The genetic algorithm starts with an initial population P_0 that is usually chosen at random. A process of selection-crossover-mutation is repeated to form a final optimized solution. The binary partitioning performed, however, is not always an optimal solution since the attributes can be grouped into more than two fragments as mentioned above. Furthermore, despite their theoretical success, Genetic Algorithms still suffer from their parameters setting challenge. Indeed, finding the best parameter values is not a trivial task for the DBAs and it is difficult to understand the effect of every parameter. Many parameters have effects on other parameters which makes the problem even more complex.

As vertical partitioning aims at grouping the attributes that are frequently referenced together, frequent itemsets mining appears as a natural solution to this problem. In [3] a vertical partitioning approach using this data mining technique was proposed. The proposed approach exploits the Apriori algorithm [31, 32] to generate all possible partitioning schema (called candidate fragments). Although the proposed approach is suitable for the vertical partitioning, the use of all frequent itemsets is limited by the high computational cost as well as the large number of resulting fragments. It is well known that frequent itemsets often generate a huge number of fragments that is very costly to handle. Furthermore, most of generated fragments are redundant resulting in a larger total computation cost. This cost increases exponentially with the number of the attributes in the workload. Thus, frequent itemsets is not the best choice for solving the partitioning problem. However, as we have been motivated in Section 2, the partitioning process can be greatly improved by the use of maximal frequent itemsets. Alluding to the closure principle of frequent itemsets which means that every subset of a frequent itemset is also frequent, the maximal frequent itemsets can imply and include all information of the frequent itemsets. Because of this, we believe that mining maximal frequent itemsets provides an interesting alternative since it generates the largest and the most frequent fragments which match the partitioning problem requirements.

4.2 Maximal Frequent Itemsets Mining

The Knowledge Discovery from Database (KDD) means the non-trivial process of trawling through data to find previously unknown relationships among the data that are interesting to the user of the data [33]. Data mining is the core step of the KDD process. Since its conception in the late 1980s, data mining has achieved tremendous success. Many new problems have emerged and have been solved by data mining researchers [34]. Due to its rich variety of important and challenging problems, data mining is proving to be a fruitful research arena for knowledge and information engineering [35]. Data mining is defined as the set of intelligent, complex, and highly sophisticated data processing techniques used to extract knowledge. Knowledge can take several forms depending on the purpose of the user and the data mining algorithm. Mining Frequent itemsets is a data mining task which consists of finding meaningful relationships between objects (items) of a database. It leads to the discovery of associations and correlations among objects in data sets which can help in many decision-making processes. It is one of the most widely used techniques in data mining and knowledge discovery. It was originally proposed in [31] with the Apriori algorithm. The drawback of mining all frequent itemsets is that if there is a large frequent itemset with size s, then almost all 2^s candidate subsets of the itemset might be generated and tested. Furthermore, the number of frequent itemsets grows very quickly when the minimum support threshold is decreased. Consequently, the complexity of the mining task becomes rapidly intractable. Moreover, the huge size of the output complicates the task of the analyst (final user), who has to extract useful knowledge from a very large amount of results. This drawback is known as pattern explosion. Often an unwieldy number of results is produced, comprising strongly redundant information.

Maximal frequent itemsets are a well known solution to the shortcomings described above. A frequent itemset is called maximal if it has no superset that is frequent. Maximal frequent itemsets are a small subset of frequent itemsets, but they represent exactly the same knowledge in a more succinct way. Using this condensed representation, it is straightforward to derive the set of all frequent itemsets. Hence, the problem of mining frequent itemsets can be reduced to mining maximal frequent itemsets. Moreover, many practical data mining applications only require mining maximal frequent itemsets rather than mining all frequent itemsets [36]. Interestingly, this problem has a strong connection to Formal Concept Analysis (FCA) [37]. FCA and frequent itemsets mining are two research fields that are closely related to each other [38, 39] and several research works showed that FCA provides a strong theory for improving both performance and results of frequent itemsets mining algorithms [40, 41, 42, 43, 44, 45].

A workshop dedicated to different implementation methods of frequent itemsets mining (FIMI) is reported in [46]. Several algorithms [47, 48, 49, 50, 51] have been tested and an analysis of each algorithm is performed, highlighting its performance. More information on the implemented methods and experimental data can be found in [46]. Within the category of mining maximal frequent itemsets, FPMAX [51] is stated to be the best algorithm presented at the cited Workshop. Thus, we have used this algorithm to validate our approach. Of course, this does not guarantee that a more efficient implementation cannot be found. Our choice is motivated by the results published in [46]. FPMAX is as an extension of the FPGrowth method [52]. It builds a special data structure called MFI-Tree (Maximal Frequent Itemsets Tree) to store all maximal frequent itemsets discovered. The MFI-Tree is similar to an FP-Tree used by the FPGrowth method. An FP-Tree is a compact representation of all relevant frequency information in the original database. Every branch of the FP-Tree represents a frequent itemset. The nodes along the branches store, in decreasing order, the frequencies of the corresponding items. In [53] we have presented the principle and a java implementation of FPMAX algorithm.

5 ANALYTICAL COST MODEL

The number of disk accesses, and thus the amount of the data transferred, have been the most commonly used parameters to evaluate the cost of query processing on a given database. While the table is the basic unit in the relational databases, we firstly provide an analytical cost model to evaluate the workload cost on a single partitioned table. Then we broaden our view to the data warehouse context where the queries involve several tables.

In the context of a single partitioned table, we assume that the cost of processing a query q on a partitioning scheme is the sum of

1. the cost of joins between fragments needed to answer the query q and
2. the cost of processing q on the resulted portion of the original data.

Notations	**Meaning**
$\|\|X\|\|$	Total number of tuples in a table X
$\|X\|$	Total size, in bytes, of the attributes of a table (or a fragment) X
DBS	Database bloc size in bytes
B_X	Number of blocks needed to store a table (or a fragment) X
B_q^F	Number of blocks to be accessed by the query q in a fragment F
t_q	Number of tuples satisfying a query q

Table 2. Cost model notations.

Based on the number of I/Os needed for joining two tables X and Y given in [54] and the number of blocks accessed for processing a query q given in [55], we propose an analytical cost model to evaluate the workload cost on a partitioning scheme. Table 2 summarizes the notations used in our cost model. For the join operations between two tables X and Y, we assume that all joins are achieved by the hash-join method. The join attributes are used as hash keys in both tables X and Y. The join operation can be viewed as consisting of two phases:

1. Hash phase: Each table is read/written once. The number of I/Os needed is then: $2 \times (B_X + B_Y)$, where B_X and B_Y are the number of disk blocks needed to store the tables X and Y, respectively.
2. Merge phase: Each table is read once. Consequently the cost of this phase is $B_X + B_Y$. The total cost for joining X and Y is then [54]:

$$C_{\bowtie} = 3 \times (B_X + B_Y). \tag{1}$$

In the partitioning scheme, we apply this formula for each join performed between the fragments needed to answer a given query q.

As in [3], we use the number of blocks estimate as the cost of the second step of query processing. Let R be a relation of n tuples and m the number of blocks needed to store R. Assume that k tuples satisfy a given query q and are distributed uniformly among the m blocks. Then the number of blocks accessed to process the query q is given by [55]:

$$B_q^R = m \times \left(1 - \left(1 - \frac{1}{m}\right)^k\right). \tag{2}$$

To evaluate the workload cost on a partitioning scheme we apply this formula, for each query q, on the portion of data obtained by joining the fragments needed to answer q. Thus, the cost of processing a query q, on a partitioning scheme, is estimated as follows:

1. **Evaluate the cost of joining the fragments required by the query q:** Let R be the original relation. Assume that $F_q = \{F_q^1, F_q^2, \ldots, F_q^N\}$ is the set of fragments required to answer the query q. The portion P_q^{Final} of original data needed to answer q is obtained by:

$$P_q^{Final} = \bowtie_{i=2}^{N} \left(F_q^i, P_q^{i-1}\right) \tag{3}$$

   where $P_q^1 = F_q^1$ and P_q^i represents the intermediate portions of data obtained after the i^{th} join. Using Equation (1), the cost of performing the joins is:

$$\text{Cost}_{\bowtie} = \sum_{i=2}^{N-1} 3 \times \left(B_{F_q^i} + B_{P_q^{i-1}}\right) \tag{4}$$

   where

$$B_X = \frac{||R|| \times |X|}{DBS}.$$

2. **Estimate the number of blocks to be accessed in the result portion:** Let $B_q^{P_q^{Final}}$ be the number of blocks to be accessed in the portion P_q^{Final} required to answer the query q. Using Equation (2), we have:

$$B_q^{P_q^{Final}} = \left\lceil B_{P_q^{Final}} \left(1 - \left(1 - \frac{1}{B_{P_q^{Final}}}\right)^{t_q}\right)\right\rceil \tag{5}$$

   where

$$B_{P_q^{Final}} = \frac{||R|| \times |P_q^{Final}|}{DBS}.$$

   Consequently, the cost of processing a given query q is:

$$\text{Cost}(q) = \left[\sum_{i=2}^{N-1} 3 \times \left(B_{F_q^i} + B_{P_q^{i-1}}\right)\right] + B_q^{P_q^{Final}}. \tag{6}$$

Finally, we have:

$$\begin{aligned}\text{Cost}(q) = &\left[\sum_{i=2}^{N-1} 3 \times \left(\frac{||R|| \times \left(|F_q^i| + |P_q^{i-1}|\right)}{DBS}\right)\right] \\ &+ \left\lceil \frac{||R|| \times |P_q^{Final}|}{DBS} \times \left(1 - \left(1 - \frac{1}{\frac{||R|| \times |P_q^{Final}|}{DBS}}\right)^{t_q}\right)\right\rceil,\end{aligned} \tag{7}$$

$$\mathrm{Cost}(q) = \frac{||R||}{DBS}\left[\left(\sum_{i=2}^{N-1} 3\times\left(|F_q^i| + |P_q^{i-1}|\right)\right) + \left(|P_q^{Final}| \times \left(1 - \left(1 - \frac{1}{\frac{||R||\times|P_q^{Final}|}{DBS}}\right)^{t_q}\right)\right)\right]. \tag{8}$$

In data warehouse environment, the above cost model has to be changed in order to suit the new context. A data warehouse stores a large volume of data and is usually organized in a star schema. A typical star schema consists of a large central fact table linked to multiple dimension tables through primary-foreign key relationships. Dimensions tables are usually much smaller then the fact table. Processing queries over a star schema is expensive. The major bottleneck in evaluating such queries has been the joins of the central (and usually very large) fact table with the surrounding dimension tables (also known as a star joins). The proposed partitioning reduces the size of the fact table tuples participating in a sequence of joins. This way, the joins are performed on a much smaller size tuples. This will be obviously more efficient than the original tuples with much less I/O cost.

In such a context, we assume that the cost of processing a query q on a partitioning scheme is the sum of the two following costs:

1. $\mathrm{Cost}^1_{\bowtie}$: the cost of joins between the fact table fragments needed to answer the query q. The ideal partitioning is obtained if each query would have to access a single fragment containing exactly the attributes it references, This way, additional joins are avoided.

2. $\mathrm{Cost}^2_{\bowtie}$: the cost of joins between the resulted portion of the original fact table and the dimension tables involved by q.

Thus, the cost of processing a query q, on a partitioning scheme, is estimated as follows:

1. **Evaluation of the cost of joining the fragments of the fact table required by the query** q**:** Let F be the original fact table. Assume that $F_q = \{F_q^1, F_q^2, \ldots, F_q^N\}$ is the set of fragments required to answer the query q. The cost of performing the joins is obtained in precisely the same manner as in the preceding cost model (Equation (4)):

$$\mathrm{Cost}^1_{\bowtie} = \sum_{i=2}^{N-1} 3\times\left(B_{F_q^i} + B_{P_q^{i-1}}\right) \tag{9}$$

where

$$B_X = \frac{||F|| \times |X|}{DBS}, \tag{10}$$

$$\mathrm{Cost}^1_{\bowtie} = \sum_{i=2}^{N-1} 3 \times \left(\frac{||F|| \times |F_q^i|}{DBS} + \frac{||F|| \times |P_q^{i-1}|}{DBS}\right), \tag{11}$$

$$\mathrm{Cost}^1_{\bowtie} = \frac{3 \times ||F||}{DBS} * \left[\sum_{i=2}^{N-1} \left(|F_q^i| + |P_q^{i-1}|\right)\right]. \tag{12}$$

2. **Evaluation of the cost of joining the result portion with the dimension tables involved by q:** Let F_q^{Final} be the portion of the original fact table needed to answer q and $\{D_1, D_2, \ldots, D_d\}$ the set of the dimension tables involved by q. We have:

$$\mathrm{Cost}^2_{\bowtie} = 3 \times \left(B_{F_q^{Final}} + B_{D_1}\right) + \cdots + 3 \times \left(B_{F_q^{Final}} + B_{D_d}\right), \tag{13}$$

$$\mathrm{Cost}^2_{\bowtie} = 3 \times \left(d * B_{F_q^{Final}} + B_{D_1} + \cdots + B_{D_d}\right), \tag{14}$$

$$\mathrm{Cost}^2_{\bowtie} = 3 \times \left(\frac{d * ||F|| * |F_q^{Final}|}{DBS} + \frac{||D_1|| * |D_1|}{DBS} + \cdots + \frac{||D_d|| * |D_d|}{DBS}\right), \tag{15}$$

$$\mathrm{Cost}^2_{\bowtie} = \frac{3}{DBS} \times \left(d * ||F|| * |F_q^{Final}| + \sum_{i=1}^{d} \left(||D_i|| * |D_i|\right)\right). \tag{16}$$

Finally, we have:

$$\begin{aligned}
\mathrm{Cost}(q) &= \mathrm{Cost}^1_{\bowtie} + \mathrm{Cost}^2_{\bowtie}, \\
\mathrm{Cost}(q) &= \frac{3 \times ||F||}{DBS} * \left[\sum_{i=2}^{N-1} \left(|F_q^i| + |P_q^{i-1}|\right)\right] \\
&\quad + \frac{3}{DBS} * \left[d \times ||F|| \times |F_q^{Final}| + \sum_{i=1}^{d} \left(||D_i|| * |D_i|\right)\right], \qquad (17) \\
\mathrm{Cost}(q) &= \frac{3}{DBS} * \left[||F|| \times \left(\sum_{i=2}^{N-1} \left(|F_q^i| + |P_q^{i-1}|\right) + d \times |F_q^{Final}|\right)\right. \\
&\quad \left. + \sum_{i=1}^{d} \left(||D_i|| * |D_i|\right)\right]. \qquad (18)
\end{aligned}$$

In both cases, the cost of processing the workload $\mathcal{W}$ is:

$$\text{Cost}(\mathcal{W}) = \sum_{q_i \in \mathcal{W}} (\text{Cost}(q_i) \times f_i).$$

6 THE MAXPART APPROACH

6.1 MaxPart Overview

This section describes the MaxPart approach outlined in Algorithm 1. MaxPart takes as input i) a relation $\mathcal{R}$, ii) a workload $\mathcal{W}$ and iii) a predefined threshold value σ and returns an optimized partitioning scheme $\mathcal{F}_{opt}$ which minimizes the cost of the workload $\mathcal{W}$. The MaxPart approach goes through the following main steps:

1. **Construction of the extraction context:** Given a representative workload $\mathcal{W} = \{q_1^{f_1}, q_2^{f_2}, \dots, q_n^{f_n}\}$, where each query $q_i (1 \leq i \leq n)$ has an access frequency f_i, we build the extraction context for mining maximal frequent itemsets. It expresses the access patterns of queries to attributes. Accesses to attribute by queries are represented by a text file where each row represents a query $q_i (1 \leq i \leq n)$ and each column a non-key attribute A_j involved in the corresponding query. Each row i corresponding to the query q_i is duplicated f_i times which corresponds to the frequency of the query q_i. The extraction context has $\sum_{i=1}^{n} f_i$ rows. The number of columns in each row depends on the number of attributes involved in the considered query. For retrieval transactions, the set of attributes in the `SELECT` clause are considered. For an `INSERT/DELETE` transaction, all the attributes in the relation are used.
2. **Generation of candidate fragments:** For a given value of threshold, we generate the corresponding sets of maximal frequent itemsets using the FPMAX algorithm. Each generated itemset corresponds to a candidate fragment. The generated fragments are then clustered into classes. Each class contains the fragments with the same length (number of attributes). The fragments in each class are sorted in descending order according to their support.
3. **Generation of possible partitioning scheme:** Let $\mathcal{F} = \{\mathcal{F}_k, \dots, \mathcal{F}_1\}$ be the set of classes of fragments generated in the previous step. Each class $\mathcal{F}_i$ corresponds to the fragments having length i. For all $\mathcal{F}_j \in \mathcal{F}_k (1 \leq j \leq size(\mathcal{F}_k))$, we construct a possible partitioning scheme as follows:

   - The itemset $\mathcal{F}_j$ is taken as the first fragment.
   - The classes $\mathcal{F}_k, \mathcal{F}_{k-1}, \dots, \mathcal{F}_1$ are successively examined to construct non-overlapping fragments having the maximal length and the highest support.
   - Finally, each non-frequent single attribute is considered as a fragment.

   The result of this step is a set $\{P_1, P_2, \dots, P_p\}$ of the possible partitioning schemes.

4. **Evaluation of the generated partitioning scheme:** The Partitioning schemes generated in the previous step closely match the requirements of the workload provided. They are evaluated using the cost model proposed in Section 5. The partitioning scheme with the smallest cost is recommended.

6.2 Illustrative Example

The purpose of this section is to illustrate the working of the MaxPart approach in the context of a single table. Consider the workload example given in Section 2. Assume the input considerations summarized in Figures 3 a) and 3 b). Consider that the number of tuples of the relation R to be partitioned is $||R|| = 150$. The database block size is assumed to be 100 bytes. The MaxPart approach goes through the following steps:

Query	Attributes	Frequency	# of Tuples
q_1	A, B, E	1	2
q_2	B, E	3	60
q_3	A, D, F	3	20
q_4	A, B, C, D, E, F	2	10
q_5	A, B, C, D, E, F	1	8

a)

Attribute	Size (byte)
A	1
B	4
C	8
D	2
E	1
F	2

b)

Figure 3. Characteristics of a) queries and b) attributes

1. **Construction of the extraction context:** Figure 4 illustrates the extraction context for our example.
2. **Generation of candidate fragments:** Assuming, for example, a minimum support value of 40 %, generated maximal frequent itemsets are $\{A, D, F\}(6)$ and $\{A, B, E\}(4)$. The number in brackets represents the frequency of the itemset. Each generated itemset corresponds to a candidate fragment.
3. **Generation of possible partitioning scheme:** The maximal itemset $\{A, D, F\}$, having the highest frequency, is taken as the first fragment. Considering the itemset $\{A, B, E\}$, we construct the fragment $\{B, E\}$ that does not overlap with the existing partition. The remaining attribute $\{C\}$, which is infrequent, is taken

Algorithm 1 MaxPart algorithm

Require: Workload $\mathcal{W} = \{q_1^{f_1}, q_2^{f_2}, \ldots, q_n^{f_n}\}$, predefined threshold σ.
Ensure: Optimized partitioning scheme $\mathcal{F}_{opt} = \{F_1, F_2, \ldots, F_k\}$

```
1: Begin
2: $\mathcal{EC} \longleftarrow \emptyset$;    ▷ Construction of the extraction context $\mathcal{EC}$
3: Row ← ””;
4: for all ($q_i^{f_i} \in \mathcal{W}$) do
5:     Row ← Candidate_Attributes($q_i$);
6:     for ($i \leftarrow 1, f_i$) do
7:         Writeln($\mathcal{EC}$, Row);
8:     end for
9: end for
10:         ▷ Mining maximal fragments using the extraction context $\mathcal{EC}$
11: $\mathcal{F}_{\mathcal{W}}[..] \leftarrow FPMAX(\mathcal{EC}, \sigma)$;    ▷ $\mathcal{F}_{\mathcal{W}}$ Set of generated fragments
12: $\mathcal{F}_{\mathcal{W}}[..] \leftarrow Sort(\mathcal{F}_{\mathcal{W}}[..], length, frequency)$;  ▷ Sorting the fragments in $\mathcal{F}_{\mathcal{W}}$
13:         ▷ $\mathcal{F}_{\mathcal{W}}[..] = \{\mathcal{F}_k, \mathcal{F}_{k-1}, \ldots, \mathcal{F}_1\}$
14:         ▷ Generating possible partitioning scheme using $\mathcal{F}_{\mathcal{W}}[..]$
15: $\mathcal{PPS} \leftarrow \emptyset$;    ▷ $\mathcal{PPS}$ Set of possible partitioning scheme
16: for ($i \leftarrow 1, size(\mathcal{F}_k)$) do
17:     $P \leftarrow \mathcal{F}_k[i]$;
18:     $P_{temp}[..] \leftarrow \emptyset$;
19:     for ($p \neq P, p \in \mathcal{F}_k, \mathcal{F}_{k-1}, \ldots, \mathcal{F}_1$) do
20:         $P_{temp}[..] \leftarrow p - (P \cap p)$;    ▷ Construct non-overlapping fragments
21:     end for
22:     while $P_{temp}[..] \neq \emptyset$ do
23:         $P_1 \leftarrow$ Fragment $\in P_{temp}[..]$ with maximal size and highest support;
24:         $P \leftarrow P \cup P_1$;
25:         $P_{temp}[..] \leftarrow P_{temp}[..] - P_1$;
26:     end while
27:     $\mathcal{PPS} \leftarrow \mathcal{PPS} \cup P$;
28: end for
29:         ▷ Find optimal partitioning scheme
30: $\mathcal{F}_{opt} \leftarrow \mathcal{PPS}[0]$;
31: for ($i \leftarrow 1, size(\mathcal{PPS})$) do
32:     if Cost($\mathcal{W}, \mathcal{F}_{opt}$) > Cost($\mathcal{W}, \mathcal{PPS}[i]$) then
33:         $\mathcal{F}_{opt} \leftarrow \mathcal{PPS}[i]$;
34:     end if
35: end for
36: Return $\mathcal{F}_{opt}$;
37: End.
```

A	B	E			
B	E				
B	E				
B	E				
A	D	F			
A	D	F			
A	D	F			
A	B	C	D	E	F
A	B	C	D	E	F
A	B	C	D	E	F

Figure 4. Example of an extraction context

as a fragment, resulting in the partitioning scheme $\{[A,D,F],[B,E],[C]\}$. Similarly, the second maximal fragment $\{A,B,E\}$ leads to the partitioning scheme $\{[A,B,E],[D,F],[C]\}$.

4. **Evaluation of the generated partitioning scheme:** After constructing the possible partitioning schemes, we now apply our cost model to recommend the one having the least cost.

- **Using the partitioning scheme $\{[A,D,F],[B,E],[C]\}$:** Consider the query q_1 which references the attributes A,B and E. To answer q_1, we need a join between the fragments $[A,D,F]$ and $[B,E]$ resulting in the final portion $[A,B,D,E,F]$. We have $|ABDEF| = 10$. Using Equation (8), we obtain:

$$\text{Cost}(q_1) = \frac{150}{100}\left[(3 \times (5+5)) + \left(10 \times \left(1 - \left(1 - \frac{1}{\frac{150\times 10}{100}}\right)^2\right)\right)\right] = 46.95.$$

  The query q_3, which references the attributes A,D and F, is answered using only the fragment $[A,D,F]$. We have:

$$\text{Cost}(q_3) = \frac{150}{100}\left[\left(5 \times \left(1 - \left(1 - \frac{1}{\frac{150\times 5}{100}}\right)^{20}\right)\right)\right] = 7.50.$$

  The query q_2, which does not need joins, is treated in the same manner as q_3 and the queries q_4 and q_5 are treated in the same manner as q_1 because they require additional joins. Table 3 lists the costs for all the considered queries.

- **Using the partitioning scheme $\{[A,B,E],[D,F],[C]\}$:** Similarly, queries costs are summarized in Table 4. Consequently, the partitioning scheme $\{[A,D,F],[B,E],[C]\}$ is recommended.

Query	**Frequency**	**Cost**	**Cost*Frequency**
q_1	1	46.95	46.95
q_2	3	00.46	01.38
q_3	3	07.50	22.50
q_4	2	134.37	268.74
q_5	1	133.02	133.02
Workload cost			**472.59**

Table 3. Queries costs using the first partitioning scheme

Query	**Frequency**	**Cost**	**Cost*Frequency**
q_1	1	01.89	01.89
q_2	3	08.99	26.97
q_3	3	56.25	168.75
q_4	2	134.37	268.74
q_5	1	133.02	133.02
Workload cost			**599.37**

Table 4. Queries costs using the second partitioning scheme

7 EXPERIMENTAL STUDY

7.1 Description of the Experiments

The goal of the experiments is to show the efficiency of the MaxPart approach. Our comparative analysis is quantified in terms of the number of candidate fragments generated and the total number of comparisons needed to generate a possible partitioning scheme. Obviously, the time cost of the partitioning process is proportional to the number of needed comparisons to generate a possible partitioning scheme. We also study the workload cost improvement using the recommended partitioning scheme. The datasets used in our experiments are commonly found in the literature.

The proposed approach improves the one presented in [3]. In order to perform a fair comparison with the cited work, we, naturally, conducted a set of experiments on the same tables. However, given their relatively small size, the used tables are unfortunately only of limited value for the experimental study. We, therefore extend our experiments to another important context: data warehouses. Such decisional databases often manipulate huge amount of data. This extension is used to further demonstrate the capability and effectiveness of MaxPart in partitioning large tables. As in [3], we generate the partitioning schemes using predefined threshold values 20 %, 30 %, 40 %, 50 % and 60 %. To generate the candidate fragments, we have implemented the FPMAX algorithm in Java [53]. All experiments were carried out on a PC with 3.4 GHz Intel® Xenon^{TM} and 1 024 MB of memory running Linux Ubuntu 12.04.

7.2 Experiments on TAE and ADULT Tables

For comparative purposes, the first experiments have been conducted using the same tables, workloads and parameters as in the most closely related work [3].

Table	# of Attributes	# of Tuples	Attributes	
			Attribute	**Size (byte)**
TAE	6	161	A: Speaker	19
			B: Cours_instructor	2
			C: Course	2
			D: Semester	7
			E: Class_size	2
			F: Class_attribute	6
ADULT	15	30 162	A: Age	1
			B: WorkClass	16
			C: Final-weight	4
			D: Education	12
			E: Education-num	1
			F: Marital-status	21
			G: Occupation	17
			H: Relationship	14
			I: Race	18
			J: Sex	6
			K: Capital-gain	3
			L: Capital-loss	2
			M: Hours-per-week	1
			N: Native-country	26
			O: Class	2

Table 5. TAE and ADULT tables characteristics

The proposed approach is tested on two real datasets: Teaching Assistant Evaluation (TAE) and ADULT, which were taken from the UCI Machine Learning Repository [56]. TAE and ADULT are small databases containing 161 and 30 162 tuples respectively. Table 5 lists a summary of the two databases. The used workloads [3] involve 12 and 20 queries for TAE and ADULT datasets respectively. In these experiments, the first cost model, e.g. Equation (8), will be used as a basis to perform our comparisons.

7.2.1 Experiment 1: Computational Study

The first experimental results concerning the number of candidate fragments are shown in Tables 6 and 7. We can note that for most values of threshold, MaxPart significantly reduces the space of candidate fragments. For TAE dataset, the ratio of the number of candidate fragments generated by AllPart to the one generated by MaxPart varies from 1.5 to 17. Using ADULT dataset the improvement is more important. That same ratio varies from 3 to 1 638.

	TAE Dataset		
Threshold (%)	**AllPart**	**MaxPart**	**Reduction Rate**
20	51	3	**94.11 %**
30	29	4	**86.20 %**
40	16	5	**68.75 %**
50	7	5	**28.57 %**
60	2	2	**00.00 %**

Table 6. Number of candidate fragments

Table 8 and Table 9 illustrate the number of comparisons between the fragments to be processed. The performed comparisons are required to generate possible partitioning schemes. As predictable, it could be seen that AllPart needs to perform much more comparisons to achieve this task. In summary, MaxPart shows better performances in terms of computational complexity for low threshold values. The performance rate decreases along with the increase of the threshold value. The reason is that for high values of threshold there are very few or no generated fragments for both AllPart and MaxPart. This leads to almost the same number of candidate fragments. As discussed in Section 2, AllPart is a computationally expensive approach.

	ADULT Dataset		
Threshold (%)	**AllPart**	**MaxPart**	**Reduction Rate**
20	32 767	20	**99.94 %**
30	269	17	**93.68 %**
40	21	7	**66.66 %**
50	5	5	**00.00 %**
60	3	3	**00.00 %**

Table 7. Number of candidate fragments

	TAE Dataset		
Threshold (%)	**AllPart**	**MaxPart**	**Reduction Rate**
20	100	4	**96.00 %**
30	56	6	**89.29 %**
40	30	8	**73.33 %**
50	6	4	**33.33 %**
60	2	2	**00.00 %**

Table 8. Number of comparisons for generating possible partitioning schemes

	ADULT Dataset		
Threshold (%)	**AllPart**	**MaxPart**	**Reduction Rate**
20	622 554	19	**99.99 %**
30	268	16	**94.03 %**
40	40	12	**70.00 %**
50	4	4	**00.00 %**
60	2	2	**00.00 %**

Table 9. Number of comparisons for generating possible partitioning schemes

7.2.2 Experiment 2: Performance Study

The costs of the workload exploiting the produced partitioning schemes compared with the baseline case where no partitioning is performed are shown in Tables 10 and 11, respectively. From Table 10, it could be observed that the best partitioning scheme for TAE dataset is obtained at a threshold of 30 % resulting in an improvement of 11.60 % over unpartitioned scheme. The best partitioning scheme for ADULT dataset is obtained at threshold of 40 % (Table 11) resulting in an improvement of 36.05 % over unpartitioned scheme. It could be seen that, for both datasets, as the threshold value increases, there are many more small fragments. This can be explained by the fact that higher threshold values result in fewer maximal (largest) fragments. In such a case, queries require many fragments, what multiplies join operations resulting in a high workload cost.

Threshold (%)	Partitioning Scheme	Cost	Reduction Rate
20	ABDEF C	2 448	0.32
30	ACDF BE	2 171	11.60
40	ACF BE D	2 420	1.46
50	AF B C D E	2 600	−5.86
60	A B C D E F	2 720	−10.74
Without partitions	ABCDEF	2 456	

Table 10. Best partitioning scheme (TAE dataset)

Threshold (%)	Partitioning Scheme	Cost	Reduction Rate
20	ABCDEFGHIJKLMNO	454 236	0.00
30	ABCEIKM FO DH G J L N	331 520	27.01
40	ABO EFK M G D N C H I J L	290 451	36.05
50	A B C D E F G H I J K L M N O	501 283	−10.35
60	A B C D E F G H I J K L M N O	501 283	−10.35
Without partitions	ABCDEFGHIJKLMNO	454 236	

Table 11. Best partitioning scheme (ADULT dataset)

7.3 Experiments on TPC-H Benchmark

In order to evaluate our approach in data warehouse context, we use as experimental data the TPC-H benchmark with a sequence of 21 queries [57]. The decision-support benchmark TPC-H contains a fact table `Lineitem` (6 000 000 tuples) and 7 dimensions tables: `Orders` (1 500 000 tuples), `Part` (200 000 tuples), `Partsupp` (800 000 tuples) `Supplier` (10 000 tuples), `Customer` (150 000 tuples), `Nation` (25 tuples) and `Region` (5 tuples). Table 12 lists a summary of the fact table attributes.

Table	**# of Attributes**	**# of Tuples**	**Attributes**	
			Attribute	**Size(byte)**
			A: LineNumber	4
			B: Quantity	8
Lineitem	13	6 000 000	C: ExtendedPrice	8
			D: Discount	8
			E: Tax	8
			F: ReturnFlag	1
			G: LineStatus	1
			H: ShipDate	7
			I: CommiDate	7
			J: ReceipDate	7
			K: ShipInStruct	25
			L: ShipMode	10
			M: Comment	44

Table 12. TPC-H fact table characteristics

7.3.1 Experiment 1: Computational Study

Table 13 and Table 14 show respectively the number of candidate fragments and the number of comparisons required to generate possible partitioning schemes for different values of threshold. As it is predictable, we can note that MaxPart significantly reduces the space of candidate fragments. This can be explained by the fact that MaxPart minimizes the cost of enumerating candidate fragments by restricting the output set to only the most relevant fragments. As a consequence (Table 14), AllPart needs to perform much more comparisons to generate possible partitioning schemes.

7.3.2 Experiment 2: Performance Study

In these experiments, the second cost model, e.g. Equation (18), is used as a basis to perform our comparisons. The workload cost using the produced partitioning schemes can be seen in Table 15. The best partitioning scheme is obtained at a threshold value of 30 %. The results clearly show that the performance of queries

		TPC-H	
Threshold (%)	**AllPart**	**MaxPart**	**Reduction Rate**
20	2 621	23	**99.12 %**
30	251	19	**92.43 %**
40	32	6	**81.25 %**
50	7	4	**42.85 %**
60	2	2	**00.00 %**

Table 13. Number of candidate fragments

		TPC-H	
Threshold (%)	**AllPart**	**MaxPart**	**Reduction Rate**
20	100	4	**96.00 %**
30	56	6	**89.29 %**
40	30	8	**73.33 %**
50	6	4	**33.33 %**
60	2	2	**00.00 %**

Table 14. Number of comparisons for generating possible partitioning schemes

is greatly enhanced. The workload cost in approximately half the cost where no partitioning is performed.

Threshold (%)	**Partitioning Scheme**	**Cost**	**Reduction Rate**
20	BCD AEFGHIJKLM	172 849 442.32	39.59
30	BCD FG AEHIJKLM	145 251 636.64	49.23
40	CD ABEFGHIJKLM	306 179 690.01	−6.99
50	CD ABEFGHIJKLM	306 179 690.01	−6.99
60	CD ABEFGHIJKLO	306 179 690.01	−6.99
Without partitions	ABCDEFGHIJKLO	286 149 243.01	

Table 15. Best partitioning scheme (TPC-H)

8 CONCLUSIONS AND PERSPECTIVES

Partitioning is a common method used for improving the performance and the scalability of data bases systems. The database is divided into smaller pieces called partitions. Partitions are then managed independently increasing the system throughput. For a given workload, vertical partitioning in databases is highly dependent on the number of attributes. The number of choices that can be made is very large making the partitioning process quite tedious and difficult even for skilled DBAs. Therefore, there is a practical need for strategies that assist the DBAs in this process.

In this paper we have proposed a maximal frequent itemsets based approach to vertical partitioning. We believe that the input data, available as a workload, can be turned into useful information and knowledge that are previously unknown. In particular, in this work we are dealing with knowledge in the form of maximal frequent itemsets. The information and knowledge gained can be used for identifying an optimized partitioning scheme. We have particularly optimized the partitioning process by the means of the downward-closure property of the set of maximal frequent itemsets which are inherently scalable and far less numerous. We use only an extremely small percentage of the possibly huge search space required by similar approaches.

Experimental study have shown that our approach does not only reduce the search space of the studied problem, but it also improves the system performance. The experiments confirm the theoretical prediction, showing that the performance improvement increases along with the increase of the volume of the data. The workload cost improvement is less noticeable for the smaller dataset (having 161 tuples), but for the other datasets (having 30 162 and 6 000 000 tuples, respectively), the improvement is more important.

The work presented in this paper can be extended in the two following directions. First, the system performance can be improved by coupling vertical partitioning with other optimization techniques such as indexing. It will be interesting to study the impact of combining the two optimization techniques on the system performances. Second, due to the interdependencies between partitioning and distributed query optimization, our approach can easily be extended to distributed, or cloud, context design according to their special requirements. The technique presented in this work would also be beneficial for those systems. Each compute node exploits our technique to split the data that are locally stored. In such cases, our cost model will be updated including network transit fees.

REFERENCES

[1] Agrawal, S.—Narasayya, V.—Yang, B.: Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design. Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, Paris, France, 2004, pp. 359–370, doi: 10.1145/1007568.1007609.

[2] Hammer, M.— Niamir, B.: A Heuristic Approach to Attribute Partitioning. Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, 1979, pp. 93–101, doi: 10.1145/582095.582110.

[3] Gorla, N.—Pang, W. Y. B.: Vertical Fragmentation in Databases Using Data-Mining Technique. In: Taniar, D., Rusu, L. I. (Eds.): Strategic Advancements in Utilizing Data Mining and Warehousing Technologies: New Concepts and Developments. IGI Global, 2010, pp. 178–197.

[4] Ramakrishnan, R.—Gehrke, J.: Database Management Systems. McGraw-Hill, Inc., New York, NY, USA, 2003.

[5] Hoffer, J. A.—Severance, D. G.: The Use of Cluster Analysis in Physical Data Base Design. Proceedings of the 1st International Conference on Very

Large Data Bases (VLDB '75), Framingham, Massachusetts, 1975, pp. 69–86, doi: 10.1145/1282480.1282486.

[6] Navathe, S.—Ceri, S.—Wiederhold, G.—Dou, J.: Vertical Partitioning Algorithms for Database Design. ACM Transactions on Database Systems (TODS), Vol. 9, 1984, No. 4, pp. 680–710, doi: 10.1145/1994.2209.

[7] Cornell, D. W.—Yu, P. S.: A Vertical Partitioning Algorithm for Relational Databases. Proceedings of the Third International Conference on Data Engineering (ICDE), IEEE Computer Society, 1987, pp. 30–35.

[8] Cornell, D. W.—Yu, P. S.: An Effective Approach to Vertical Partitioning for Physical Design of Relational Databases. IEEE Transactions on Software Engineering, Vol. 16, 1990, No. 2, pp. 248–258, doi: 10.1109/32.44388.

[9] Navathe, S. B.—Ra, M.: Vertical Partitioning for Database Design: A Graphical Algorithm. ACM SIGMOD Record, Vol. 18, 1989, No. 2, pp. 440–450, doi: 10.1145/67544.66966.

[10] Song, S. K.—Gorla, N.: A Genetic Algorithm for Vertical Fragmentation and Access Path Selection. The Computer Journal, Vol. 43, 2000, No. 1, pp. 81-93.

[11] Cheng, C. H.—Lee, W. K.—Wong, K. F.: A Genetic Algorithm-Based Clustering Approach for Database Partitioning. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), Vol. 32, 2002, No. 3, pp. 215–230.

[12] Muthuraj, J.—Chakravarthy, S.—Varadarajan, R.—Navathe, S. B.: A Formal Approach to the Vertical Partitioning Problem in Distributed Database Design. Proceedings of the Second International Conference on Parallel and Distributed Information Systems, San Diego, California, USA, 1993, pp. 26–35, doi: 10.1109/PDIS.1993.253076.

[13] March, S. T.—Rho, S.: Allocating Data and Operations to Nodes in Distributed Database Design. IEEE Transactions on Knowledge and Data Engineering, Vol. 7, 1995, No. 2, pp. 305–317.

[14] Bellatreche, L.—Simonet, A.—Simonet, M.: Vertical Fragmentation in Distributed Object Database Systems with Complex Attributes and Methods. Proceedings of 7th International Conference and Workshop on Database and Expert Systems Applications (DEXA 96), 1996, pp. 15–21, doi: 10.1109/DEXA.1996.558266.

[15] Özsu, M.—Valduriez, P.: Principles of Distributed Database Systems. Prentice-Hall, 1999.

[16] Ezeife, C. I.—Barker, K.: Distributed Object Based Design: Vertical Fragmentation of Classes. Distributed and Parallel Databases, Vol. 6, 1998, No. 4, pp. 317–350.

[17] Barker, K.—Bhar, S.: A Graphical Approach to Allocating Class Fragments in Distributed Object Base Systems. Distributed and Parallel Databases, Vol. 10, 2001, No. 3, pp. 207–239.

[18] Son, J. H.—Kim, M. H.: An Adaptable Vertical Partitioning Method in Distributed Systems. Journal of Systems and Software, Vol. 73, 2004, No. 3, pp. 551–561.

[19] Gorla, N.: An Object-Oriented Database Design for Improved Performance. Data and Knowledge Engineering, Vol. 37, 2001, No. 2, pp. 117–138.

[20] Fung, C. W.—Karlapalem, K.—Li, Q.: An Evaluation of Vertical Class Partitioning for Query Processing in Object-Oriented Databases. IEEE Transactions on Knowledge and Data Engineering, Vol. 14, 2002, No. 5, pp. 1095–1118.

[21] Fung, C. W.—Karlapalem, K.—Li, Q.: Cost-Driven Vertical Class Partitioning for Methods in Object Oriented Databases. The VLDB Journal, Vol. 12, 2003, No. 3, pp. 187–210.

[22] Schewe, K. D.: Fragmentation of Object Oriented and Semistructured Data. Proceedings of the Baltic Conference, BalticDB & IS, 2002, Vol. 1, 2002, pp. 253–266.

[23] Labio, W.—Quass, D.—Adelberg, B.: Physical Database Design for Data Warehouses. Proceedings of the Thirteenth International Conference on Data Engineering, Birmingham, U.K., 1997, pp. 277–288, doi: 10.1109/ICDE.1997.581802.

[24] Golfarelli, M.—Maio, D.—Rizzi, S.: Vertical Fragmentation of Views in Relational Data Warehouses. SEBD, 1999, pp. 19–33.

[25] Furtado, C.—Lima, A. B.—Pacitti, E.—Valduriez, P.—Mattoso, M.: Physical and Virtual Partitioning in OLAP Database Clusters. 17th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD '05), Rio de Janeiro, Brazil, 2005, pp. 143–150, doi: 10.1109/CAHPC.2005.32.

[26] Kling, P.—Özsu, M. T.—Daudjee, K.: Generating Efficient Execution Plans for Vertically Partitioned XML Databases. Proceedings of the VLDB Endowment, Vol. 4, 2010, No. 1, pp. 1–11, doi: 10.14778/1880172.1880173.

[27] Andrade, A.—Ruberg, G.—Baião, F. A.—Braganholo, V. P.—Mattoso, M.: Efficiently Processing XML Queries over Fragmented Repositories with PartiX. Proceedings of the 2006 International Conference on Current Trends in Database Technology (EDBT '06), Munich, Germany, 2006, pp. 150–163, doi: 10.1007/11896548_15.

[28] Mahboubi, H.—Darmont, J.: Data Mining-Based Fragmentation of XML Data Warehouses. Proceedings of the ACM 11th International Workshop on Data Warehousing and OLAP, Napa Valley, California, USA, 2008, pp. 9–16, doi: 10.1145/1458432.1458435.

[29] Bose, S.—Fegaras, L.: XFrag: A Query Processing Framework for Fragmented XML Data. Proceedings of the Eighth International Workshop on the Web and Databases (WebDB 2005), Baltimore, Maryland, USA, collocated with ACM SIGMOD/PODS, 2005, pp. 97–102.

[30] McCormick, W. T.—Schweitzer, P. J.—White, T. W.: Problem Decomposition and Data Reorganisation by a Clustering Technique. Journal of Operations Research, Vol. 20, 1972, No. 5, pp. 993–1009.

[31] Agrawal, R.—Imieliński, T.—Swami, A.: Mining Association Rules Between Sets of Items in Large Databases. ACM SIGMOD Record, Vol. 22, 1993, No. 2, pp. 207–216, doi: 10.1145/170035.170072.

[32] Agrawal, R.—Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94), San Francisco, CA, USA, 1994, pp. 487–499.

[33] Fayyad, U. M.—Piatetsky-Shapiro, G.—Smyth, P.: Knowledge Discovery and Data Mining: Towards a Unifying Framework. Proceedings of the Second Interna-

tional Conference on Knowledge Discovery and Data Mining (KDD '96), AAAI Press, 1996, pp. 82–88.

[34] Yang, Q.—Wu, X.: 10 Challenging Problems in Data Mining Research. International Journal of Information Technology and Decision Making, Vol. 5, 2006, No. 4, pp. 597–604.

[35] Glover, F. W.—Kochnberger, G.: New Optimization Models for Data Mining. International Journal of Information Technology and Decision Making, Vol. 5, 2006, No. 4, pp. 605–609, doi: 10.1142/S0219622006002143.

[36] Han, J.—Cheng, H.—Xin, D.—Yan, X.: Frequent Pattern Mining: Current Status and Future Directions. Data Mining and Knowledge Discovery, Vol. 15, 2007, No. 1, pp. 55–86.

[37] Ganter, B.—Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer-Verlag New York, Inc., 1999.

[38] Martin, B.—Eklund, P. W.: From Concepts to Concept Lattice: A Border Algorithm for Making Covers Explicit. In: Medina, R., Obiedkov, S. (Eds.): Formal Concept Analysis (ICFCA 2008). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4933, 2008, pp. 78–89.

[39] Pisková, L.—Horváth, T.: Comparing Performance of Formal Concept Analysis and Closed Frequent Itemset Mining Algorithms on Real Data. Proceedings of the Tenth International Conference on Concept Lattices and Their Applications, 2013, pp. 299–304.

[40] Pasquier, N.—Bastide, Y.—Taouil, R.—Lakhal, L.: Efficient Mining of Association Rules Using Closed Itemset Lattices. Information Systems, Vol. 24, 1999, No. 1, pp. 25–46, doi: 10.1016/S0306-4379(99)00003-4.

[41] Zaki, M. J.—Ogihara, M.: Theoretical Foundations of Association Rules. Proceedings of SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 1998, pp. 1–8.

[42] Kuznetsov, S. O.—Obiedkov, S. A.: Comparing Performance of Algorithms for Generating Concept Lattices. Journal of Experimental and Theoretical Artificial Intelligence, Vol. 14, 2002, No. 2-3, pp. 189–216, doi: 10.1080/09528130210164170.

[43] Poelmans, J.—Ignatov, D. I.—Viaene, S.—Dedene, G.—Kuznetsov, S. O.: Text Mining Scientific Papers: A Survey on FCA-Based Information Retrieval Research. In: Perner, P. (Ed.): Advances in Data Mining. Applications and Theoretical Aspects (ICDM 2012). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7377, 2012, pp. 273–287.

[44] Kuznetsov, S. O.—Poelmans, J.: Knowledge Representation and Processing with Formal Concept Analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, Vol. 3, 2013, No. 3, pp. 200–215.

[45] Andrews, S.: A 'Best-of-Breed' Approach for Designing a Fast Algorithm for Computing Fixpoints of Galois Connections. Information Science, Vol. 295, 2015, pp. 633–649, doi: 10.1016/j.ins.2014.10.011.

[46] Goethals, B.—Zaki, M.: An Introduction to Workshop on Frequent Itemset Mining Implementations. Proceeding of the ICDM 03 International Workshop on Frequent Itemset Mining Implementations, 2003, pp. 1–13.

[47] Bayardo Jr., R. J.: Efficiently Mining Long Patterns from Databases. ACM SIGMOD Record, Vol. 27, 1998, No. 2, pp. 85–93.

[48] Agarwal, R. C.—Aggarwal, C. C.—Prasad, V. V. V.: Depth First Generation of Long Patterns. Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00), 2000, pp. 108–118, doi: 10.1145/347090.347114.

[49] Burdick, D.—Calimlim, M.—Flannick, J.—Gehrke, J.—Yiu, T.: MAFIA: A Maximal Frequent Itemset Algorithm. IEEE Transactions on Knowledge and Data Engineering, Vol. 17, 2005, No. 11, pp. 1490–1504, doi: 10.1109/TKDE.2005.183.

[50] Gouda, K.—Zaki, M. J.: Efficiently Mining Maximal Frequent Itemsets. IEEE International Conference on Data Mining, IEEE Computer Society, 2001.

[51] Grahne, G.—Zhu, J.: High Performance Mining of Maximal Frequent Itemsets. Sixth SIAM International Workshop on High Performance Data Mining, 2003.

[52] Han, J.—Pei, J.—Yin, Y.: Mining Frequent Patterns Without Candidate Generation. ACM SIGMOD Record, Vol. 29, 2000, No. 2, pp. 1–12.

[53] Ziani, B.—Ouinten, Y.: Mining Maximal Frequent Itemsets: A Java Implementation of FPMAX Algorithm. Proceedings of the 6th International Conference on Innovations in Information Technology (IIT), IEEE Press, Piscataway, NJ, USA, 2009, pp. 11–15, doi: 10.1109/IIT.2009.5413790.

[54] Mishra, P.—Eich, M. H.: Join Processing in Relational Databases. ACM Computing Surveys (CSUR), Vol. 24, 1992, No. 1, pp. 63–113, doi: 10.1145/128762.128764.

[55] Yao, S. B.: Approximating Block Accesses in Database Organizations. Communications of the ACM, Vol. 20, 1977, No. 4, pp. 260–261.

[56] UCI (Machine Learning Repository) Web Site. Available at: `http://archive.ics.uci.edu/ml`, University of California, Irvine, School of Information and Computer Sciences.

[57] TPC (Transaction Performance Council) Web Site. Available at: `http://www.tpc.org`.

Benameur Ziani received his Engineer degree in computer science from Sidi Belabbes University (Algeria) and Ph.D. degree in computer science from the University of Laghouat (Algeria). He is currently Associate Professor in computer science at the Department of Computer Science of the University of Laghouat. Prior to joining the Department of Computer Science he served as Engineer in computer science at the computing center of the University of Laghouat during 1992–2012. His current research interests include knowledge discovery, data mining and machine learning with applications in various areas: database and data warehouse design optimisation, big data and data networks analytics.

Youcef Ouinten received his M.Sc. degree and Ph.D. degree in operational research from the University of Southampton (UK), in 1984 and 1988, respectively. He received his graduation degree (Diplôme d'Etudes Superieures) in mathematics, option operational research, from the University of Science and Technology – Houari Boumediene of Algiers, Algeria, in 1981. He served as Head of the Computing Center at the University Amar Telidji of Laghouat, from 1999 to 2012. He is currently Senior Lecturer at the Department of Mathematics and Computer Science of the University Amar Telidji of Laghouat, Algeria. His research interests include data mining, text mining, information retrieval and optimization.

Mustapha Bouakkaz is Associate Professor in computer science at the Department of Computer Science of the University of Laghouat Algeria. He received his Ph.D. degree in computer science from the University of Laghouat in 2017. He carries out research on OLAP and Data Mining. He is more interested about data coming from documents or social networks. His current work focuses on graph OLAP, text mining and social networks analysis.