Clustering Cubes with Binary Dimensions in One Pass

Carlos Garcia-Alvarado
Pivotal Inc.
San Mateo, CA 94403, USA

Carlos Ordonez
University of Houston
Houston, TX 77204, USA

ABSTRACT

Finding aggregations of records with high dimensionality in large data warehouses is a crucial and costly task. These groups of similar records are the result of partitions obtained with GROUP BYs. In this research, we focus on obtaining aggregations of groups of similar records by turning the problem into efficient binary clustering of a fact table as a relaxation of a GROUP BY clause. We present an efficient window-based Incremental K-Means algorithm in a relational database system implemented as a user-defined function. This variant is based on the Incremental K-Means algorithm. The speed up is achieved through the computation of sufficient statistics, multithreading, efficient distance computation and sparse matrix operations. Finally, the performance of our algorithm is compared against multiple variants of the K-Means algorithm. Our experiments show that our incremental K-Means algorithm achieves similar or even better results more quickly than the traditional K-Means algorithm.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Relational databases*

Keywords

OLAP; Clustering; DBMS

1. INTRODUCTION

Processing a large amount of records in one-pass is similar to dealing with stream data, which is characterized by the data's volume and velocity of arrival which makes storage for later processing unfeasible [3, 10]. Examples of these environments are network traffic, transactions in retail stores (e.g. market basket data), sensor data, weather monitoring, telephone communications, stock trading, and web logs, among many others [8, 5]. The simplest form of representation of these data streams is binary, which can be used to characterize categorical data. Clustering binary data presents advantages since it does not contain the noise of quantitative data and it has lean storage requirements.

Despite this, most clustering algorithms work with numerical data and the problem of clustering categorical data is not as obvious as the problem of clustering continuous dimensions [9].

The main motivation for this research is the computation of efficient aggregations (e.g. SUM, MAX, MIN, AVG) of groups of tuples from a fact table, where the aggregation is the result of using clustering as a relaxation of a GROUP BY clause. The relaxation of GROUP BY as clustering is important for efficient processing by reducing the number of groups to a desired quantity when the number of groups is large. This paper introduces several improvements to the Incremental K-Means algorithm for clustering binary data streams, which was presented initially in [15]. The K-Means variants studied in this paper include the well-known Standard K-Means, On-line K-Means, the incremental K-Means, and an incremental K-Means version that exploits multithreading. In addition to this, we introduce a sliding window incremental K-Means algorithm to deal with time decay [2, 20]. This paper also presents K-Means improvements to cluster binary data streams. These include simple sufficient statistics for binary data, efficient distance computation, and sparse matrix operations. We examine how to incorporate these changes into several variants of the K-Means algorithm in a database management system (DBMS). In order to do so, we exploit user-defined functions to extend the DBMS functionalities. Particular attention is paid to the improvements made to the Incremental K-Means algorithm.

The paper is organized as follows: Section 2 introduces definitions, the K-Means algorithm, and a detailed example. Section 3 presents the proposed improvements to K-Means and how to incorporate them into a DBMS. Section 4 shows an experimental evaluation with real and synthetic data sets. Section 5 discusses related work. Finally, Section 6 contains the conclusions and directions for future work.

2. PRELIMINARIES

Given a fact table D having n d-dimensional records and m measurements, it is desired to partition it into k clusters and aggregate the measurements. A characteristic of the fact table is that all the dimensions are binary, which requires transforming complex data sets. The partitioning results are a matrix with the cluster centroids C, a matrix with the weights of the clusters, W, and an array of variance diagonal matrices, R. As a result, matrices C and R are of size $d \times k$ and W is a $k \times 1$ matrix. Notice that the array of matrices R is managed as a matrix because only the main diagonal of each variance matrix is stored. These matrices use the following convention for subscripts: Let transactions t be identified by the i subscript, where $i \in$

http://dx.doi.org/10.1145/2513190.2513192

Table 1: Computing aggregation example.

clusterID	W_j	D_1	D_2	D_3	A_1	A_2
1	66%	0.9	0.6	0.1	10	200
2	33%	0.2	1.0	0.8	7	320

$\{1, 2, \ldots, n\}$. In addition, each transaction i contains a set of m measurements. The cluster number is given by j, where $j \in \{1, 2, \ldots, k\}$. Therefore, the j subscript is used to refer to the columns of C or R. As a result, C_j, R_j, and W_j refer to the j^{th} cluster centroid, the j^{th} variance diagonal matrix, and the j^{th} cluster weight, respectively. Dimensions are mapped by h where $h \in \{1, 2, \ldots, d\}$. Moreover, let $\{D_1, D_2, \ldots, D_k\}$ be the k subsets of D given C such that $D_j \cap D_{j'} = \emptyset$ for $j \neq j'$. C_{hj} represents the fraction of records in cluster j that have dimension h equal to one. Similarly, W_j is the fraction of the n records that belong to cluster j. These fractions of records in C_{hj} and W_j can also be seen as percentages which will be useful for interpreting the result. Finally, A_{je} represents the aggregation (e.g. SUM, MAX, MIN) of the corresponding measurement e per cluster j. An example is presented in Table 1.

We define a generic operator `diag[]` to ease matrix manipulation. This operator obtains a diagonal matrix from a vector or converts the diagonal of a matrix into a vector. In this work, we use the Euclidean similarity measure, in which a 0/0 match is as important as a 1/1 match on some dimension. Thus, the Euclidean distance from t_i to C_j is given by $\delta(t_i, C_j) = (t_i - C_j)^t(t_i - C_j)$.

Since transactions are sparse vectors, we found that $|t_i| \ll d$. This fact will be exploited for efficient distance computations. We will use T to denote average transaction size ($T = \sum_{i=1}^{n} |Ti|/n$). In this work, we focus on data sets with $d \ll n$.

2.1 The K-Means Algorithm

The K-Means algorithm can be initialized from a random or approximate solution. Each iteration assigns each point to its nearest cluster (using any distance measure) and then records belonging to the same cluster are averaged to get new cluster centroids. Each iteration successively improves cluster centroids until they become stable (the solution converges). Formally, the problem of clustering is defined as finding a partition of D into k subsets such that the distance between a transaction and a centroid is minimized. When using the Euclidean distance, the quality of a clustering model $q(C)$ is measured by the sum of squared distances from each point to the cluster where it was assigned [21, 14, 4]. This quantity is proportional to the average quantization error, also known as distortion. The quality of a solution is obtained by the squared error measured as $q(C) = \frac{1}{n}\sum_{i=1}^{n} \delta(t_i, C_j)$, which can be computed from R and W as $q(C) = q(R, W) = \sum_{j=1}^{k} W_j \sum_{h=1}^{d} R_{hj}$.

If a different measure such as the Jaccard distance is used, the convergence of the clustering is obtained when the records do not change cluster memberships. The evaluation of the clustering solution can be done using other quality measurements. Additional information from the quality of the clusters, regardless of the distance measure used, can be obtained by computing the entropy E of the resulting clusters C if class labels are given. Since clustering is an unsupervised model, having a class label g is not expected. However, we complement a fact table with this information for quality validation purposes.

The entropy of each cluster $E(C_j)$ is computed by $E(C_j) = -\sum_{\forall g \in C_j} \frac{N_{jg}}{N_j} \log \frac{N_{jg}}{N_j}$, where N_{jg} represents the number of points in cluster j of class g and N_j is the number of points in cluster j. Notice that for the purpose of computing entropy, N is extended to count the number of points of a certain class in every cluster. The total entropy of model $E(C)$ is obtained by the weighted sum of all the previously computed cluster entropies. $E(C)$ is computed as $\sum_j^k W_j\ E(C_j)$.

2.2 Stream Data

Binary stream data is a continuous ordered sequence of transactions. The arrival time can either be implicit by arrival time or explicit by timestamp [11, 12]. The arrival interval allows physical window models based on the number of transactions that can be held in memory at a time to be analyzed. Hence, let τ represent the window in which the transactions are contained. This window is defined as $\tau = \tau_1, \ldots, \tau_m$, where each sliding non-overlapping subwindow τ_γ can aggregate at most ω-transactions. Notice that there is no difference if the window is assumed to have a time interval. However, the number of transactions that can be held within a window can be uneven. Also, if the streaming environment has a constant flow of transactions and the timestamp is given during arrival time, a physical approach is the most efficient solution.

2.3 User-defined Functions

Data mining algorithms are difficult to implement in relational databases, due to the fact that DBMSs lack matrix and vectors native data types and operations that would ease their implementation [17]. Despite this limitation, multiple proposals have shown that user-defined functions (UDFs) can be used to extend the functionality of the DBMS [6]. A UDF is a compiled piece of code that is attached to the DBMS to be run within the SELECT statement. UDFs are executed in a memory pool assigned by the DBMS. User-defined functions are scalar, aggregate or table-valued. Scalar UDFs are routines executed record-by-record. The isolation level of a scalar UDF is that the memory assigned to the UDF is not carried between records. The output of a scalar UDF is a single data type. Aggregate UDFs maintain the allocated memory through the data stream resulting from the SELECT statement. Aggregate UDFs process a set of records in independent parallel runs which are merged into a final global aggregation. Aggregate UDFs return a single row per group. Table-valued functions (TVFs) are a type of user-defined function that, unlike aggregate UDFs, are able to return a table as the final result of the function. In addition to this, a TVF cannot implicitly manage parallelism. Despite this lack of 'out-of-the-box' parallelism, it is common that database systems allow the user to implement routines that support parallelism. TVFs read and process an input data set as a single data stream. Table-valued functions allow the user to manage arrays and matrices within the memory space assigned to the UDFs.

3. CLUSTERING BINARY DATA STREAMS

Given a data stream X, we want to cluster and aggregate the incoming transactions from a fact table using the Standard K-Means. The main problem with the traditional

K-Means is that iterating through the data stream is unfeasible. As a result, we will have to modify the algorithm in order to obtain a model without revisiting the points. Despite this huge limitation, the binary nature of the items gives us the opportunity to accelerate the computation of the model. In this section, we explore optimizations within the K-Means algorithm for the distance computation and model update. We also present our variants for the K-Means algorithm for clustering data streams.

3.1 Sufficient Statistics and Sparse Operations

We first explore possible optimizations to accelerate the computations in the K-Means algorithm. Sparse distance computation and simpler sufficient statistics are our main objectives. When D is a sparse matrix and d is high, the distance formula is costly to compute (especially because this computation is performed between each point and all of the centroids). Sparse Euclidean distance computation is optimized, without modifying the result, by precomputing the distance from every C_j to the null vector $\vec{0}$. In order to do so, a k-dimensional vector $\Delta : \delta_j = (\vec{0}, C_j)$ is defined. Thus each $\delta(t_i, C_j)$ can be computed as $\delta(t_i, C_j) = \Delta_j + \sum_{h=1,(t_i)_h \neq 0}((t_i)_h - C_{hj})^2 - C_{hj}^2$. A similar optimization is obtained for metrics such as the Jaccard distance by precomputing the norm of each centroid. Here, each Δ_j value is obtained from the $\sum_{h=1}^{d} C_{hj}^2$.

K-Means has been proven to use sufficient statistics to compute the model [4, 21], which are summaries of $D_1, D_2, \ldots, D_k$ represented by the three matrices N,L and Q. These matrices contain the sum of points, the sum of squared points, and the number of points per cluster, respectively. The idea of using sufficient statistics is well known to accelerate the computations [4, 7]. However, due to the fact that we are working with binary data, it is possible to simplify these computations. The following lemma states that the sufficient statistics for the problem of clustering binary vectors are simpler than the ones required for clustering numeric data.

LEMMA 1. *Let D be a set of n transactions of binary data and $D_1, D_2, \ldots, D_k$ be a partition of D. Then the sufficient statistics required for computing C, R, W are only N and L.*

PROOF. In order to compute W, k counters are needed for the k subsets of D that are stored in the $k \times 1$ matrix N and then $W_j = N_j/n$. For the computation of C, we use $L_j = \sum_{i=1}^{n} t_i, \forall t_i \in D_j$ and then $C_j = L_j/N_j$. To compute Q the following formula must be computed: $Q_j = \sum_{i=1}^{n} diag[t_i t_i^t] = \sum_{t=1}^{n} t_i = L_j, \forall t_i \in D_j$. Note that $\texttt{diag}[t_i t_i^t] = t_i$ because $x = x^2$ if x is binary and t_i is a vector of binary numbers. Elements off of the diagonal are ignored for diagonal matrices. Since we know that $Q = L$, we conclude that only N and L are required for obtaining sufficient statistics from binary transactions. □

Thus R can be computed from C without scanning D or storing Q. Even further, Lemma 1 makes it possible to reduce storage to one half. However, it is not possible to reduce storage further because when K-Means determines cluster membership, it needs to keep a copy of C_j to compute distances and a separate matrix with L_j to accumulate the point to cluster j. Therefore, both C and L are needed for an Incremental version, but not for an On-line version. The On-line K-Means algorithm, which corresponds to a variant of K-Means that updates the model with every new transaction, could keep a single matrix for centroids and the sum of points. For the remainder of this article, L is a $d \times k$ matrix and $L_j = \sum_{\forall t_i \in D_j} t_i$ and N is $k \times 1$ matrix and $N_j = |D_j|$. The update formulas for C, R, W are $C_j = \frac{1}{N_j} L_j$, $R_j = diag[C_j] - C_j C_j^t$ and $W_j = \frac{N_j}{\sum_{j'=1}^{k} N_{j'}}$.

3.2 K-Means Variants for Binary Data Streams

We now introduce the variants of K-Means (KM) for binary data streams based on the optimizations explained previously. These variants from the KM include the Incremental K-Means (IKM), Multithreaded Incremental K-Means (IKM-MT), Incremental Window K-Means (Inc.WinKM), Multithreaded Incremental Window Means (Inc.WinKM-MT) and the K-Means with multithreading (KM-MT). Consider the binary data points given as transactions $T_1, T_2, \ldots, T_n$ and k clusters. Observe that the points are assumed to arrive as lists of integers as defined in Section 2. The order of the dimensions within each transaction does not affect the algorithm as long as transactions do not occur sorted by cluster. The output is the clustering model (C, R, W), a partition of D into $D_1, D_2, \ldots, D_k$, the aggregated measurements, and a metric of cluster quality. Let the nearest neighbor function be defined as $NN(C, t_i) = J$, such that $\delta(t_i, C_J) \leq \delta(t_i, C_j)$ for every cluster where $J \neq j$.

Let $\oplus$ be a sparse addition of vectors where only nonzero entries are added. The $L_j \oplus t_i$ operation has complexity $O(T)$. Similar to this operation, the $t_i \cdot C_j^T$ also takes $O(T)$ and is equivalent to a sparse aggregation given by $\sum_{\forall h \in T_i} C_{hj}$. Initialization is based on a sample of k different points to seed C. The weights W_j are initialized to $1/k$ to avoid early re-seeding (see [14]).

3.3 Incremental K-Means

The Incremental K-Means algorithm is our proposed variant of K-Means. This version finds a middle ground between the On-line K-Means algorithm and the Standard K-Means algorithm by doing only one iteration. A fundamental difference with Standard K-Means is that Incremental K-Means does not iterate until convergence and that the model is updated every $\sqrt(n)$ transactions, each time touching the entirety of C and W. This algorithm is detailed and fully explained in [15].

3.3.1 Multithreaded Incremental K-Means

The Incremental K-Means algorithm does not update the model for a batch of transactions. As a result, during this batch it is possible to compute in parallel the membership and measurements of each transaction. Figure 1 shows the pseudocode of our proposal of a one-pass incremental multithreaded K-Means algorithm. In the pseudocode, variable r represents a random number between [0, 1], μ is the global mean and $\sigma = diag[\sqrt(R_j)]$ represents a vector of global standard deviations. The initialization of the algorithm proceeds as in the Incremental K-Means. However, a set I is initialized to hold a set of transactions to process per thread (load) and a load size I_s is given by the user.

In summary, the expectation step (Estep) is performed for a set of transactions in the stream in every update batch (`ThreadPoolAdd` function). During this step, the Nearest Neighbor computation is obtained for every point, and the

Input: $\{T_1, T_2, \ldots, T_n\}$,$k$,$\{M_1, \ldots, M_m\}$,$i_s$
Output: $C, R, W, q(C), \{A_1, \ldots, A_m\}$

```
1  for j ← 1 to k do
2  |  C_j ← μ ± σr/d ;
3  |  N_j ← 0 ;
4  |  L_j ← 0⃗ ;
5  |  W_j ← 1/k ;
6  |  I ← ∅
7  end
8  for i ← 1 to n do
9  |  I ← I ∪ T_i ;
   |  /* E Step */
10 |  if ( |I| = i_s or (i mod(√n) = 0) ) then
   |  |  /* A_1, ..., A_m are aggregated during the
   |  |  EStep.  */
11 |  |  ThreadPoolAdd(EStep(I));
12 |  |  I ← ∅
13 |  end
14 |  if (i mod(√n) = 0) then
   |  |  /* M Step */
15 |  |  Join(ThreadPool) ;
16 |  |  for j ← 1 to k do
17 |  |  |  C_j ← L_j/N_j ;
18 |  |  |  R_j ← diag[C_j] − C_j C_j^t ;
19 |  |  |  W_j ← N_j/i ;
20 |  |  |  if W_j = 0 then
21 |  |  |  |  Reseed(j);
22 |  |  |  end
23 |  |  end
24 |  end
25 end
26 q(C) ← q(R, W) ;
```

Figure 1: Multithreaded Incremental K-Means.

sparse addition of $L_J \oplus T_i$ and N_j are computed. An important consideration during the L and N update is that these arrays must hold a lock during the update process to avoid unexpected results. The model update (Mstep) is performed every $\sqrt{(n)}$ steps. However, this step must wait until all the threads in the thread pool finish their loads to update the model (`Join` function). In contrast to the Estep, the operations during the maximization phase cannot be performed in a sparse manner. Fortunately, these operations are computed efficiently with the N and L vectors. If empty clusters are found, the `Reseed` function can re-seed clusters with the furthest neighbors of non-empty clusters as proposed in [4]. The re-seeding points are extracted from the outliers list stored in a summary table. This summary table consists of two arrays held in memory for storing the most frequent item per cluster and the farthest transactions assigned to a cluster. An additional optimization is that the R and W matrices can be computed only at the end of the iteration if no On-line $q(C)$ or re-seeding is desired. In Figure 2, we exemplify the incremental K-Means process. Notice that the figure's color code is related to the transactions that were taken into consideration for updating the corresponding model. The Estep is shown in the table on the right (we omit the L and N arrays). During this step, the nearest cluster is computed for every transaction (using the $\vec{\Delta}$) and the sparse addition between $L_j \oplus T_i$. We compare the obtained distance to update our summary table for obtaining outliers and top items. In this example, we use $\psi = \sqrt{(23)}$, where ψ represents the number of model updates. As a result, the model is updated in batches of five transactions. In practice, a rounding function should be used. During the model update, C, R and W are updated. Notice that the Δ vector needs to be recomputed, with the new centroids. During each E step the top items and outliers' transactions are computed.

3.4 Sliding Windows over Stream Data

We propose a variation of the incremental K-Means algorithm that works within a window of transactions of the data stream X. The version is similar to the incremental K-Means in the fact that it does not iterate until convergence. Similar to this, the model is updated every $\sqrt{(n)}$ transactions. On the other hand, Window Incremental K-Means (IncWinKM) only considers the transactions in a current window for estimating the output matrices C, R and W. The Incremental Window K-Means is shown in Figure 3. A multithreaded version of this algorithm is possible by adding the `ThreadPoolAdd` and `Join` functions to the proposed algorithm (adding an additional lock to the L_τ and N_τ data structures is required).

The window is managed as a fixed number of transactions that can be held within an interval τ. The sufficient statistics of the window are kept in N and L. However, s subwindows with smaller number of transactions w are used to displace the window in the current time. As a result, there exists a global array with the sufficient statistics of the whole window and a set of partial subwindows containing only a few transactions. The proposed algorithm initially obtains all the sufficient statistics of the incoming transactions and stores them in the available subwindow. When all the subwindows have been filled with transactions, the subwindow with the smallest average transaction arrival time is reset and used to accumulate new transactions. The `SW` function in Figure 3 obtains the window with the smallest average transaction time. Note that prior to the reset of the selected subwindow, the values of the transactions accumulated in such an interval are discarded from the global window. Once the subwindow with the oldest transactions has been subtracted from the global aggregation, the new transaction is aggregated to the empty subwindow and to the general aggregation. The final result of this process is a set of overlapping windows. The model then will be evaluated in a manner similar to the incremental K-Means algorithm. However, the C,R and W matrices will be obtained with the transactions in the current window. The aggregated measurements are global. If re-seeding is required, this should be done with transactions in the current window.

A variation of the Standard KM algorithm was developed for comparison purposes. The algorithm iterates until reaching convergence in an equal-sized fixed window. In contrast with the Incremental Window KM, the Window KM does not have a set of overlapping windows. Thus, the model and the global sufficient statistics are updated every ωs transactions. The purpose of this variation was to avoid the update of the model until convergence was reached with every new transaction added (and removed) from the current window. The result of this algorithm is similar to that obtained from the implementation of Scalable K-Means. Nevertheless, the

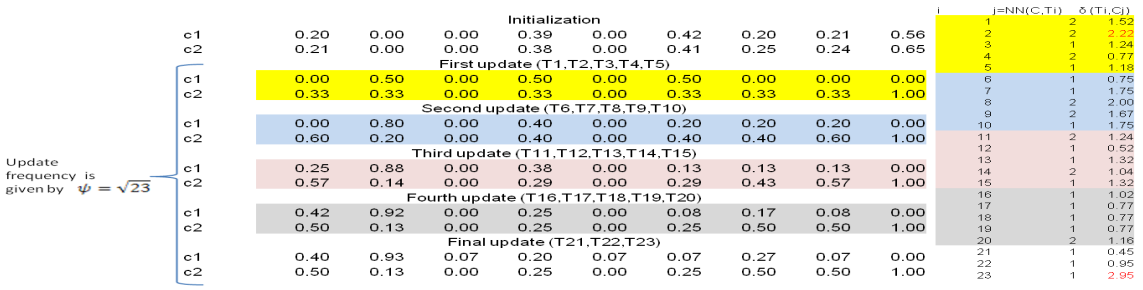


Figure 2: Example: IKM Process.

algorithm computes the updated model with only the sufficient statistics of the current model.

Input: $\{T_1, T_2, \ldots, T_n\}$,$k$,$\{M_1, \ldots, M_m\}$,$\omega$
Output: $C, R, W, q(C), \{A_1, \ldots, A_m\}$

for $j \leftarrow 1$ **to** k **do**
  $C_j \leftarrow \mu \pm \sigma r/d$;
  $N_j \leftarrow 0$;
  $L_j \leftarrow \vec{0}$;
  $W_j \leftarrow 1/k$;
  for $\gamma \leftarrow 1$ **to** s **do**
    $N_{\tau j\gamma} \leftarrow 0$;
    $L_{\tau j\gamma} \leftarrow \vec{0}$;
  end
end
for $i \leftarrow 1$ **to** n **do**
  `/* E Step */`
  $J \leftarrow NN(C, T_i)$;
  $p \leftarrow SW(N_\tau, J, \omega, f)$;
  if $(f = true)$ **then**
    $N \leftarrow N - N_{\tau p}$;
    $L \leftarrow L - L_{\tau p}$;
    $reset(N_{\tau p}, L_{\tau p})$;
  end
  $L_{\tau pJ} \leftarrow L_{\tau pJ} \oplus T_i$;
  $N_{\tau pJ} \leftarrow N_{\tau pJ} + 1$;
  $L_J \leftarrow L_J \oplus T_i$;
  $N_J \leftarrow N_J + 1$;
  for $e \leftarrow 1$ **to** m **do**
    $A_{Je} \leftarrow M_e$
  end
  if $(i \ mod(\sqrt{n}) = 0)$ **then**
    `/* M Step */`
    for $j \leftarrow 1$ **to** k **do**
      $C_j \leftarrow L_j/N_j$;
      $R_j \leftarrow diag[C_j] - C_j C_j^t$;
      $W_j \leftarrow N_j/i$;
      if $W_j = 0$ **then**
        Reseed(j);
      end
    end
  end
end
$q(C) \leftarrow q(R, W)$;

Figure 3: Incremental Window KM.

3.5 DBMS integration

The incremental KM algorithm, the window variant, and the multithreaded versions can be integrated into the DBMS to be processed in one-pass by exploiting user-defined functions. The data set X is indexed in the DBMS by i and h guaranteeing that all the dimensions of a sparse transaction are contiguous with each other. As a result, X is being read by the user-defined function as a data stream of sparse transactions. The best fit of a user-defined function for implementing the K-Means variants in one-pass is a table valued-function (TVF). This TVF will perform a single scan through the data and maintain the central data structures in main memory. An aggregate user-defined function is also an alternative. However, it is not possible to synchronize the threads and update the model without partitioning the data.

3.6 Time Complexity Analysis

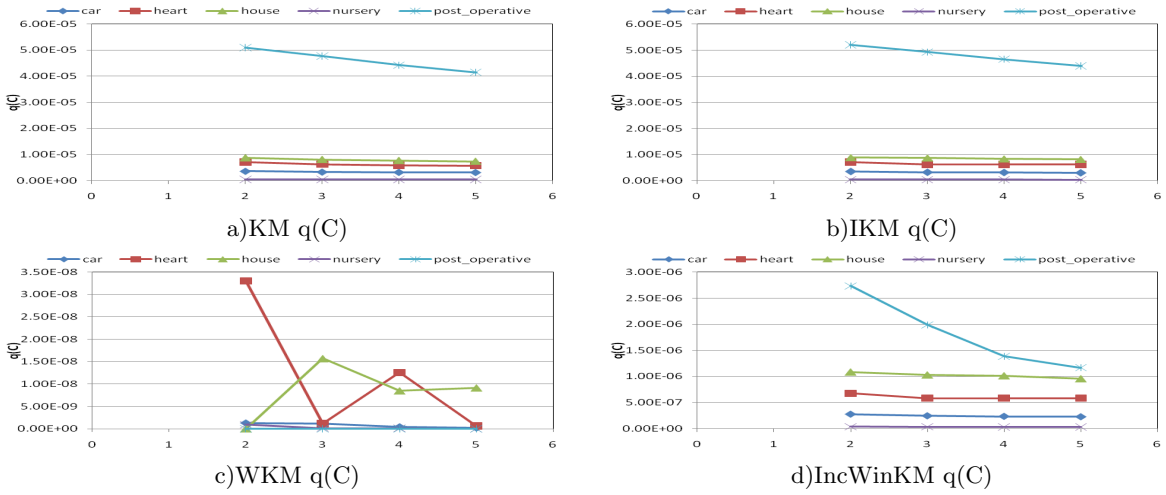
Clustering data streams has become a popular field of research [13]. All the variants introduced above read each data point from a disk just once. On-line K-Means and Incremental K-Means can keep up with the incoming flow of transactions since they do not iterate.

The proposed K-Means variants have $O(Tkn)$ complexity where T is the average transaction size; recall that $T \ll d$. Re-seeding using outliers, if desired, can be done in time $O(k)$. The top items' and outlier's data structures can be updated in $O(1)$ by using fixed intervals. Matrices L, C require $O(dk)$ space and N,W require $O(k)$. Since R is derived from C, that space is saved. In short, space requirements are $O(dk)$ and time complexity is $O(kn)$ for sparse binary vectors. Note that the space requirements for the Incremental Window K-Means vary from the Incremental K-Means. The window management requires an additional $O(dks)$ space for managing the transactions accumulated within the current window. Notice that s represents the total amount of subwindows that accumulate a maximum of ω-transactions. The time complexity is still $O(kn)$. Despite this, the algorithm contains a larger hidden constant than the Incremental K-Means for the time that is required to manage the window.

4. EXPERIMENTAL EVALUATION

A thorough examination of the proposed algorithms is contained in this section. The algorithms were benchmarked

Table 3: Real data sets results (squared error).



a)KM q(C) b)IKM q(C)

c)WKM q(C) d)IncWinKM q(C)

Table 2: Real data sets.

Data set	n	d	T	Classes
car	1728	21	7	4
heart	655	22	18	2
house	435	45	16	2
nursery	12960	27	9	3
post operative	90	24	9	3

on the quality of the results and performance. The quality was evaluated using $q(C)$ for the squared error. However, we also used entropy for the real data sets to compare the distance approaches. We decided to incorporate all the sufficient statistics and sparse matrix optimizations in all the algorithms for a fair performance evaluation. All the times are presented in seconds. The experiments were run on an Intel Xeon E3110 server with 3.0GHz 6MB L2 Cache Dual-Core Processor, 4 GB of main memory and 1 TB of hard disk. All the algorithms were implemented as user-defined functions in MSSQL Server using the C# language. In general, all the algorithms require the k parameter. The Standard K-Means distance-based algorithm also requires a tolerance error for $q(C)$ which was set to $\epsilon = 1.0e-4$. The incremental algorithms also require setting the ψ value, which defines the number of times the model gets updated. Finally, the window algorithm requires setting the number of subwindows and the amount of transactions to aggregate per subwindow.

4.1 Accuracy Evaluation

We initially study the quality of the results of the presented algorithms. In order to do so, we use five datasets taken from the UCI Data Set repository from different domains. From these data sets, the model with the smallest squared error is chosen from a set of 30 runs.

In Table 3, we ran all algorithms several times to find a good value for k. In the car data set, it is clear that the squared error decreases as the number of clusters grow in all the algorithms. This data set has the characteristic of being unbalanced for one of the classes (most of the transactions of the unbalanced class appear at the end of the stream). As a result, the Incremental K-Means and the window versions tend to obtain a better squared error than the Standard K-Means. The rationale behind this is that the model gets updated with the most current data, lowering the squared error for the incoming transactions. On the other hand, the standard K-Means re-evaluates old transactions and is less sensitive to the latest transactions. Notice that the standard window K-Means tends to overfit the data in the small window. As a result, it obtains the lowest squared error. In the heart data set, it is observed that as the number of clusters increases, the squared error is smaller. The best solution is found by the Standard K-Means algorithm. However, the solution of the incremental K-Means is within our tolerance value. The WKM found the best solution, although the IncWinKM is within our tolerance value, too. It is important to point out that the IncWinKM produces a model that is less sensitive to new data because of the sliding window approach. On the other hand, the Window KM is highly sensitive to the window because of the disjoint windows approach. Notice that even though the incremental KM algorithm is sensitive to recent data (when calculating the current model), the window versions of the standard KM, and the incremental KM are more sensitive to the transactions in the window. The nursery data set contains the largest number of rows of the real data sets. An interesting result of this data set is that the squared error improves as the number of clusters increases. This behavior is also noticed in the window versions of the algorithm. The rationale behind this is that the transactions are not skewed, which delivers ideal results. In addition, the Incremental KM reports equal or better results than the Standard KM. The last data set is also the smallest one. The post operative data set has a smaller error when the amount of clusters increases, except in the Standard WinKM. This behavior is explained by the small window size ($\omega = 1$). In general, the standard algorithm seems to obtain better solutions than the incremental K-Means algorithm. Despite this, the squared difference is quite small. Also, the IncWinKM is less sensitive to new transactions in the current stream. Even though we did not include entropy plots (due to space constraints), we noticed that the best squared error solution did not obtain the best E(C) model. We also noticed that the total model entropy is slightly better for the incremental variant. As a result IncWinKM produces good quality clusters.

4.2 Performance evaluation

In addition to the real data sets, we prepared some large fact tables using the IBM data generator in order to obtain some performance measurements. The defaults we used were as follows: The number of transactions was $n = 100k$. The average transaction size T was 4, 8, 10 and 20 and a single aggregation. Pattern length (I) was one half of transaction length. Dimensionality d was 10, 100 and 1000. The rest of the parameters were kept at their defaults (average rule confidence= 0.25, correlation=0.75). These data sets represent very sparse matrices and very high dimensional data. We did not expect to find any significant clusters, but we wanted to try the algorithms in order to observe their scalability.

The set of plots in Table 4 shows experiments for ten computed parallel models when varying the average transaction size, the data set size, the number of dimensions and the number of clusters for KM and all the variants. It is important to notice that even though the comparison is made between equivalent algorithms (e.g. WKM and IncWinKM), we included the eight trends in the same plot. Plot (a) shows that all the algorithms have a linear increase when the average transaction size increases. However, this increase does occur at a high rate due the optimization for sparse transactions. Despite this optimization, this increase is not insignificant and cannot be neglected. Plot (b) presents the trends when increasing the number of transactions in the data set. Notice that all the algorithms present linear scalability. In addition, all the incremental versions perform at least half the time of their corresponding KM algorithm. Plot (c) shows the trend close to linear when increasing the number of dimensions. The trend of both algorithms KM and incremental is similar for the window and regular versions. This is due to the fact that the update of the model takes $O(d)$ and it takes the same time in all the algorithms. However, the trend of the Standard K-Means algorithm seems to remain constant when $d = 1000$, and this is due to the fact that the squared error of the Standard K-Means in the data set is already within the tolerance value and is converging in the first iteration. As a result, the traditional K-Means seems to be faster due to the fact that it only performs an update operation of the model at the end of the execution. The last plot (d) shows the linear scalability with respect to the number of clusters. Notice that the incremental algorithm always performs faster than the standard KM. Also notice that in all the plots, the multithreaded version has a 25% to 30% speed up with respect to the non-threaded version. Also, the equivalent multithreaded versions (e.g. IKM-MT and IncWinKM-MT) take almost the same time. As a result, the multithreaded versions significantly reduce window management time. The improvement is from 66% to 75% of the non-threaded version's performance.

5. RELATED WORK

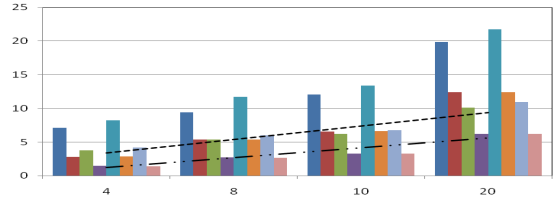
Specialized database systems, such as [1] and [19], have been developed to manage stream data. Nevertheless, these systems require modifying the system model and creating an SQL extension to monitor and manage data streams. In contrast, our research relies on traditional DBMS, where we have implemented and deployed our proposed algorithms using user-defined functions. In a similar manner, STREAM is a system that tries to extend SQL to support stream-oriented primitives [3]. Like our research, the authors included a sliding window approach to perform queries on evolving data streams. However, as in [1] and [19], the authors had to extend the functionality by creating a specialized stream system. In previous research (see [17]), we have proposed the usage of user-defined functions in order to compute several data mining models in a DBMS. However, we have not extended these functionalities to deal with stream data. Likewise, the authors in [16] and [18] propose optimization for distance computations within the DBMS. However, they are not focused on binary data. Finally, this paper extends the previous work in [15] by incorporating the presented optimizations within the DBMS in order to compute efficient clustering of binary data streams in a DBMS using user-defined functions. Moreover, this paper contributes even further to OLAP aggregation using binary stream clustering with the proposal of a multithreaded Incremental Window KM algorithm for analyzing intervals of data streams.

6. CONCLUSIONS

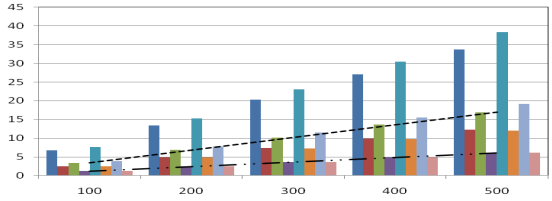
This paper proposed a window-based incremental k-means algorithm for computing aggregations in a fact table of binary data streams. Some optimizations included computing sufficient statistics, efficient nearest cluster computation with a precomputed Δ vector for the Euclidean distance, sparse operations, and multithreading. It is important to notice that this IKM one-pass algorithm was implemented within a traditional DBMS using user-defined functions. Moreover, the distance computation is optimized for sparse binary vectors. The proposed improvements are fairly easy to incorporate in the standard K-Means. The WIKM algorithm and some variants were compared with real and synthetic data sets. The proposed Incremental K-Means showed faster performance than the standard K-Means and gives solutions of comparable quality. In a similar manner, the Incremental Window K-Means was found to outperform the Standard Window K-Means with a similar quality in the results. In addition to this, both incremental algorithms showed linear scalability in the number of points, number of dimensions, number of clusters and average transaction size. However, sparse matrix addition reduces the effect of increasing the number of clusters and the average transaction size. It was also found that a smaller window size will decrease the quality of the results due to the fact that the centroids will be updated with only a few transactions. Despite this, there is a trade-off between the size of the window, the amount of memory required for holding the sufficient statistics, and the amount of time the algorithm requires to manage the subwindows. Using multithreading also showed an improvement of 25% to 30% in the Incremental K-Means. However, the real improvement was in the window version of the algorithm, due to the fact that maintaining multithreading balances the load of the processor. As a result, the multithreaded window version had a performance similar to that of the non-window versions.

Future work on this research includes incorporating a variation of the triangle inequality for speeding up the selection of the nearest cluster. Additional improvements can be made in the selection of the best models, in which a mixed evaluation between the squared error and cluster entropy can be used to select the best resulting models. More complex initialization approaches can be used to initialize C. Possible variations of the incremental algorithm may in-

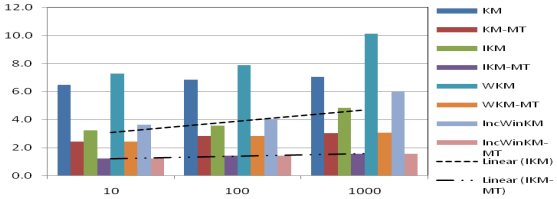
Table 4: Transactional data sets experiments for ten models when modifying T, n, d and k (time in seconds).



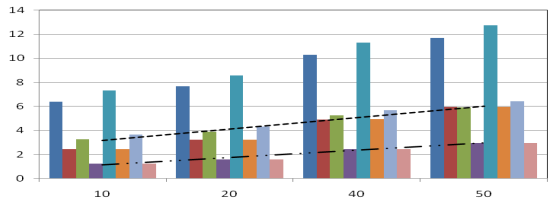
a) Varying average transaction size $d = 100, k = 10, n = 100k$.



b) Varying data set size $T = 4, d = 10, k = 10$.



c) Varying number of dimensions $T = 4, n = 100k, k = 10$.



b) Varying number of clusters $T = 4, d = 10, d = 10$.

clude different reseeding strategies for outlier transactions. We believe a further acceleration of Incremental K-Means is possible by using approximation, heuristics for cluster selection, randomization, or sampling.

7. REFERENCES

[1] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.

[2] C. Aggarwal, J. Han, J. Wang, and P. Yu. A framework for clustering evolving data streams. In *VLDB Conference*, pages 81–92, 2003.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. of ACM PODS*, pages 1–16, 2002.

[4] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proc. ACM KDD Conference*, pages 9–15, 1998.

[5] Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *Proc. of ACM SIGKDD*, pages 133–142, 2007.

[6] J. Cohen, B. Dolan, M. Dunlap, J. Hellerstein, and C. Welton. MAD skills: New analysis practices for big data. In *Proc. VLDB Conference*, pages 1481–1492, 2009.

[7] F. Fanstrom, J. Lewis, and C. Elkan. Scalability for clustering algorithms revisited. *SIGKDD Explorations*, 2(1):51–57, June 2000.

[8] M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *ACM Sigmod Record*, 34(2):18–26, 2005.

[9] V. Ganti, J. Gehrke, and R. Ramakrishnan. Cactus-clustering categorical data using summaries. In *ACM KDD Conference*, pages 73–83, 1999.

[10] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: You only get one look: A tutorial. In *Proc. of ACM SIGMOD*, page 635, 2002.

[11] L. Golab and M. Özsu. Issues in data stream management. *ACM Sigmod Record*, 32(2):5–14, 2003.

[12] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, pages 515–528, 2003.

[13] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *FOCS Conference*, page 359, 2000.

[14] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high quality clustering. In *IEEE ICDE Conference*, 2002.

[15] C. Ordonez. Clustering binary data streams with K-means. In *Proc. ACM SIGMOD Data Mining and Knowledge Discovery Workshop*, pages 10–17, 2003.

[16] C. Ordonez. Integrating K-means clustering with a relational DBMS using SQL. *IEEE Transactions on Knowledge and Data Engineering*, 18(2):188–201, 2006.

[17] C. Ordonez. Statistical model computation with UDFs. *IEEE Transactions on Knowledge and Data Engineering*, 22(12), 2010.

[18] S. Pitchaimalai, C. Ordonez, and C. Garcia-Alvarado. Efficient distance computation using SQL queries and UDFs. In *IEEE HPDM*, 2008.

[19] H. Wang, C. Zaniolo, and C. Luo. ATLaS: A small but complete SQL extension for data mining and data streams. In *Proc. VLDB Conference*, pages 1113–1116, 2003.

[20] J. Yang. Dynamic clustering of evolving streams with a single pass. In *Proc. of IEEE ICDE*, pages 695–697, 2004.

[21] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proc. ACM SIGMOD Conference*, pages 103–114, 1996.