CORD: A HYBRID APPROACH FOR EFFICIENT CLUSTERING OF ORDINAL DATA USING FUZZY LOGIC AND SELF-ORGANIZING MAPS

Natascha Hoebel and Stanislav Kreuzer
Department of Computer Science and Mathematics, Database and Information Systems
Goethe University Frankfurt, Germany

Keywords: User profile analysis, Clustering, Ordinal data, Optimization, Web mining.

Abstract: This paper presents CORD, a hybrid clustering system, which combines modifications of three modern clustering approaches to create a hybrid solution, that is able to efficiently process very large sets of *ordinal data*. The *Self-organizing Maps* algorithm for categorical data by Chen and Marques is hereby used for a rough preclustering for finding the initial position and number of centroids. The main clustering task utilizes a k-modes algorithm and its fuzzy set extension described by Kim et al. for categorical data using fuzzy centroids. Finally in dealing with large amounts of data, the BIRCH algorithm described by Zhang et al. for efficient clustering of very large databases (VLDBs) is adapted to ordinal data. BIRCH can be used as a preliminary phase for both Fuzzy Centroids and NCSOM. Both algorithms profit from this symbiosis as their iterative computations can be done on data, that is fully held in main memory. Combining these approaches, the resulting system is able to extract significant information even from very large datasets efficiently. The presented reference implementation of the hybrid system shows good results. The aim is clustering and visual analyzing large amounts of user profiles. This should help in understanding Web user behavior and personalize advertisement.

1 INTRODUCTION

Clustering algorithms have evolved greatly since the development of data-mining technologies in terms of significance, as well as performance and quality. Especially when taking into account the constantly growing number of algorithms, research of the usability and adaptivity seems inevitable in a practical environment. It is obvious that an efficient solution to all clustering problems cannot be concentrated into one algorithm. Therefore it is necessary to expand the field of research to a set of partial solutions. Some of these solutions are adaptable to changing practical requirements exploiting synergies of their members.

This paper introduces CORD (**C**lustering of **Or**dinal **D**ata), a hybrid clustering algorithm system. Main contribution of our work presented herein is:

- Definition of a more accurate distance measure for ordinal data (section 4.1).
- Extension of the known algorithms to work with ordinal data by using the described measure.
- Optimization of the time complexity of FuzzyCentroids (Kim et al., 2004) and showing its implementation, which allows the results to be saved and therefore refined later (section 4.2).
- Improvement of the result of the clustering task by helping in the decision process of choosing how many centroids should be used; and where should the k-centroids be placed. This is possible through our hybrid approach with NCSOM.
- Improvement of the clustering task so that it works with very large datasets. This is possible through our hybrid approach with BIRCH.
- Definition of the normalization for the sum of $w_{l,t}$ for FuzzyCentroids (assumption in 3.1).

1.1 Problem Formulation

Clustering of elements is a common problem and has been broadly discussed. All algorithms have to deal with the issue of distance computation for the observed elements. Most of the known approaches focus on *numerical data* that allows measuring the distance by arithmetic operations, for example Euclidean or chi-squared distance metric. Others focus on *categorical data* ((Gan et al., 2005), (Huang, 1997), and

Hoebel N. and Kreuzer S.
CORD: A HYBRID APPROACH FOR EFFICIENT CLUSTERING OF ORDINAL DATA USING FUZZY LOGIC AND SELF-ORGANIZING MAPS.
DOI: 10.5220/0002795402970306
In *Proceedings of the 6th International Conference on Web Information Systems and Technology (WEBIST 2010)*, page
ISBN: 978-989-674-025-2

(Parmar et al., 2007)) and measure the distance e.g. by counting the number of different attribute values. This is known as Hamming distance.

The focus of the CORD approach is to efficiently cluster large amounts of *ordinal data*. Ordinal data, unlike pure categorical data, possesses an inherent order. Though this order can not be utilized to improve a distance metric in general, there are specific cases, where this kind of information can be a valuable asset and enhance the quality of the cluster analysis.

1.2 Problem Application: User Profiles

The motivation in the design of CORD arose in the *Gugubarra* project (Zicari et al., 2006). This project lacks an appropriate solution for clustering and visual analyzing large amounts of user profiles. The project focuses on calculating different kinds of user profiles of Web site visitors and understanding the user behavior. The non-obvious user profiles (NOPs) represent the interests of users in different topics. It is a vector per user with numerical values between 0 and 1 for each topic. While a numerical value is not very meaningful, these values can be mapped to a so called *scale of interest*. Sometimes these profiles are stored in this *scale*, right from the start. The scale of interest classifies the values in a human understandable form and is ordinal. An example of a scale of interest is: *no*, *little*, *medium*, *high*, *absolute* interest. Another kind of profile is the *user feedback profile*. Several Web sites allow or ask their users to give a feedback on their interests, hobbies or opinions about a product or the site itself. A user feedback can hereby be given on a scale of interest over a number of topics. The reply can then be stored in a feedback profile. To understand users better, it can be of interest to cluster across this feedback (i.e. what the user tells about his interest) and the NOP (i.e. what he is supposed to be interested in, by analyzing his click streams).

The paper shows the design and implementation of a clustering solution in the area of clustering *user profiles* and in general for clustering VLDBs of *ordinal data* having the same cardinality and scale.

The rest of the paper is structured as follows: Section 2 introduces the necessary definitions. Theoretical foundations are presented in section 3. Section 4 describes the design and implementation of CORD. Section 5 presents the experimental results. Related work is compared in section 6.

2 DEFINITIONS

This section defines some of the concepts that will be used in the rest of the paper.

Let N be the entire set of categorical records X:

$$N = \{X_1, ..., X_n\},\ |N| = n \tag{1}$$

with a relational schema $R = \{A_1, ..., A_r\}$ with $|R| = r$. A_i, $i = 1, ..., r$ is a *categorical attribute* with its *domain* $D_i = DOM(A_i)$ respectively:

$$R = \{A_1, ..., A_r\},\ D_i = DOM(A_i),\ i = 1, ..., r \tag{2}$$

A *domain* D_i for attribute A_i is a set of *categorical values* $a_{i,j}$:

$$D_i = \{a_{i,1}, ..., a_{i,n_i}\},\ j = 1, ..., n_i \tag{3}$$

A *record* $X \in N$ is a tuple of r categorical values $x_i \in D_i$ of the attributes $A_1, ..., A_r$, that belong to the relational schema R of N. The attribute value of X for an attribute $A_i \in R$ is defined as $A_i(X)$.

$$X = (x_1, ..., x_r),\ x_i \in D_i,\ A_i(X) = x_i,\ i = 1, ..., r \tag{4}$$

The *universe* $U = D_1 \times D_2 \times ... \times D_r$ of all valid assignments for records of N is defined as the Cartesian product of the domains D_i of all attributes A_i belonging to R of N. Based on this definition, a *distance measure* $d(X,Y)$ refers to a mapping given as: $d : U \times U \longrightarrow \mathbb{R}$, $U = D_1 \times D_2 \times ... \times D_r$ that fulfills for every two records X, Y of U the following constraints:

- $d(X,X) = d_0,\ X \in U,\ d_0 \in \mathbb{R}$
- $d(X,Y) > d_0,\ X, Y \in U,\ X \neq Y$
- $d(X,Y) = d(Y,X),\ X, Y \in U$

A *Cluster partition* C of N, is defined as $C = \{c_j \subseteq N\},\ j = 1, ..., k$ where k is the number of clusters and where for every two clusters c_i, c_j applies: $\forall\ c_i, c_j \in C,\ i \neq j,\ c_i \cap c_j = \emptyset$. The *centroid* Z of a cluster c_j is a record, that is the best representative for the average of all records in the cluster.

3 THEORETICAL FOUNDATIONS

As CORD is a hybrid clustering algorithm based on the following three algorithms, these algorithms are first described in the next subsections in more detail.

3.1 Fuzzy Centroids

The work done by Kim et al. describes a Fuzzy clustering algorithm of categorical data using fuzzy centroids (Kim et al., 2004), and is named *FuzzyCentroids* in the following. This algorithm is an extension of *Fuzzy k-Modes* (Huang, 1997), and needs the number of clusters k, a-priori. The fundamental idea of this approach is using fuzzy sets as centroids to overcome the information loss between iterations in *Fuzzy k-Modes*. The fuzzy sets allow to build on information from the previous iterations when choosing the centroids in the current iteration. In this way the center of a cluster can be calculated more accurately. A *fuzzy (set) centroid* $\widetilde{Z}$ is defined as follows:

$$\widetilde{Z} = (\widetilde{z}_1, ..., \widetilde{z}_r), \widetilde{z}_l = \{(a_{l,1}, \omega_{l,1}), ..., (a_{l,n_l}, \omega_{l,n_l})\} \quad (5)$$

with $a_{l,t} \in A_l$, $|A_l| = n_l$, $0 \leq \omega_{l,t} \leq 1$, $1 \leq l \leq r$, $1 \leq t \leq n_l$ and $\sum_{t=1}^{n_l} \omega_{l,t} = 1$, $1 \leq l \leq r$. The values $\omega_{l,t}$ build for each cluster the confidence degree matrix $W_i = [\omega_{l,t}]$, $1 \leq i \leq k$. Each $\widetilde{z}_l \in \widetilde{Z}$ is determined by the category distribution of attribute A_l of the records belonging to the cluster. $\omega_{l,t}$ indicates the confidence degree (or resonance) with which the attribute value $a_{l,t}$ contributes to $\widetilde{z}_l$.

In the *first step*, initial centroids are chosen by selecting the values $\omega_{l,t}$. In the *second step* the membership matrix M, formula (6), of the values $\mu_{i,j}$ are calculated. The fuzziness modifier m and the distance measure $d\left(\widetilde{Z}, X\right) = \sum_{l=1}^{r} \delta(\widetilde{z}_l, x_l)$, that takes into account the $\omega_{l,t}$ are both described in (Kim et al., 2004).

$$\mu_{i,j}(t+1) = \left(\sum_{l=1}^{k} \left(\frac{d\left(\widetilde{Z}_i, X_j\right)}{d\left(\widetilde{Z}_l, X_j\right)} \right)^{\frac{1}{m-1}} \right)^{-1} \quad (6)$$

In *step three*, the $\omega_{l,t}$ of all centroids will be updated, using the following stop condition, i.e. as long as there is an improvement in minimizing the function:

$$J_m\left(M, \widetilde{Z}\right) = \sum_{i=1}^{k} \sum_{j=1}^{n} (\mu_{i,j})^m \, d\left(\widetilde{Z}_i, X_j\right) \quad (7)$$

subject to

$$\sum_{i=1}^{k} \mu_{i,j} = 1 \wedge 0 < \sum_{j=1}^{n} \mu_{i,j} < n \quad (8)$$

In case the stop condition is not fulfilled, step two is repeated. Otherwise it stops. For further details, please refer to (Kim et al., 2004).

It should be noted that $\omega_{l,t}$, as defined in (Kim et al., 2004), can have values between 0 and 1 only. The sum of $\omega_{l,t}$ for $t = 1, ..., n_l$ has to be equal 1. However, because the sum is taken over $\mu_{i,j}^m$ it is possible that the result is higher than 1. This problem is not mentioned in (Kim et al., 2004) and therefore the following assumption is made: *The $\omega_{l,t}$ will be normalized after the summation. Therefore, the sum $\sum_{t=1}^{n_l} \omega_{l,t} = 1$, $1 \leq l \leq r$ is calculated. Afterwards the $\omega_{l,t}$, that are part of the summation, are normalized with the result of the summation.*

3.2 BIRCH

BIRCH, introduced in (Zhang et al., 1996), is a well known clustering method for VLDBs and broadly studied. The whole dataset is iterated once at maximum. A *cluster feature tree* is built as representation of the data. Each leaf of the CF tree is a set of *cluster features* (CF). A CF saves information about the centroid and the amount of records in the cluster. This summary is small enough to be held in the fast main memory, that is allocated by BIRCH. Using this summary, the distance measurements for numerical data (Zhang et al., 1996) can be used between cluster features and records. In a further phase a clustering algorithm can process the leaves and is therefore faster than working on the original dataset. A *cluster feature* of a cluster c_i for numerical data is a triple CF_i

$$CF_i = \left(N_i, \vec{LS}_i, SS_i\right) \quad (9)$$

with N_i, the number of records in c_i, $\vec{LS}_i = \sum_{X \vec{\in} c_j} \vec{X}$, is the linear sum of the attribute values and $SS_i = \sum_{X \vec{\in} c_j} \vec{X}^2$ the according square sum.

BIRCH is more or less an abstract instruction for building an algorithm for VLDBs. Zhang et al. describes an implementation of this approach for numerical data, using 4 distance measurements and an agglomerative hierarchical clustering algorithm. For further details, please refer to (Zhang et al., 1996).

3.3 NCSOM

The NCSOM algorithm (Chen and Marques, 2005) is an extension of the ***S**elf **O**rganized **M**ap* (SOM) algorithm for *categorical data* and builds upon artificial neural networks, described by *Teuvo Kohonen*. The data structure of the NCSOM algorithm is a two dimensional network of K neurons. The network is square and the sides have all the same number of neurons. Each neuron has a randomly initialized reference vector $m_j = [m_{j,1}, ..., m_{j,r}]$, where r is the number of attributes as defined in section 2. During the iteration, these reference vectors change dynamically, since they adapt to the records already pro-

cessed and likewise to their neighbor neurons. Thus a map of reference vectors forms, whereby similar vectors lie closer to each other. The algorithm demands no knowledge of the data and is able to determine a global optimum of the cluster partition. For further details of NCSOM, see (Chen and Marques, 2005).

4 CORD HYBRID ALGORITHM AND IMPLEMENTATION

(Cheu et al., 2004) describes several possibilities to use a *hybrid algorithm*. The word *hybrid* has its origin from the Greek language and means bundling, crossing or mixture. A *hybrid* is the combination of two or more different things, aimed at achieving a particular objective. In this case, a hybrid concept is used to cluster large amounts of ordinal data. The **first phase** of the concept processes the whole dataset once to create a summary of its most distinctive features being small enough to be held in main memory. A modification of BIRCH (Zhang et al., 1996), described in subsection 4.3, is used in this phase. The clustering task of the **main phase** uses the BIRCH summary as input. Therefore the algorithm in (Kim et al., 2004) was modified as described in subsection 4.2. A **preliminary data analysis phase** can be placed before the main clustering task to review the structure of the summary and to manually optimize its initialization by choosing the initial cluster centers. The modification of the NCSOM algorithm (Chen and Marques, 2005) is used here for creating a similarity map, where best places for initial cluster centers can be seen and picked visually. The results of the main clustering phase, consisting of k matrices $[\omega_{l,t}]$ of weights for each of the k clusters, where l is the number of attributes and t is the number of categories for the attributes, can then be stored efficiently. Moreover, the ability to save the weight matrices creates the possibility to pause and resume the main clustering task. A refinement of an already achieved result can then be done in the same way, allowing the use of the old result as initialization, so that the clustering does not have to be restarted completely.

4.1 Distance Measurement

We define in the following a distance measure for ordinal data. The motivation is as follows: The attributes of the user profiles as described in subsection 1.2 have the same cardinality and scale of attribute values. Using the *scale of interest* as domain, results in comparable attributes with the same context and symbolism, namely the *user interest* in different topics. This data is ordinal.

The *measurements for text* distances are not appropriate, because the label letters itself do not play any role. The *measurements for numerical data* are also not appropriate, as the ordinal data can not be transformed to numbers. For example, if the feedback is given on an ordinal scale, then this data cannot easily be transformed into numerical values, because ordinal data only offers information of the ordering of some elements. The distance between these elements is not defined. This is the central matter for this research field and is discussed in statistics. For example, Podani has discussed this issue for Braun-Blanquet dominance (Braun-Blanquet et al., 1932): "it is inappropriate to analyse Braun-Blanquet abudance/dominance data by methods assuming that Euclidean distance is meaningful" (Podani, 2005).

The *measurements for categorical data* could be applied, as by the clustering algorithms CACTUS, ROCK or STIRR. However it seams insufficient to us, as ordinal data can offer more information (i.e. the *ordering*) than simple categorical data. The attribute values of the *scale of interest* are ordered and quasi equidistant. Therefore the number of categories between two attribute values is taken into account. We extended the distance metric to include this *ordering* into the computation, see formula (10). We accept a small error, as it is only a quasi-equidistance, but the distance measure is more precise with these modifications rather than without them. An example: Using Hamming distance for categorical data, the distance between *small* and *high* is equal to the distance between *small* and *medium*. With our proposed distance measure in formula (10), the first distance is higher than the second, which makes sense.

The *distance measure* $d(X,Y)$ for two records X and Y of dataset N is given as follows:

$$d(X,Y) = \sum_{l=1}^{r} \delta(x_l, y_l), \tag{10}$$

with r is the number of attributes as defined in sec. 2 and:

$$\delta(x_l, y_l) = \begin{cases} 0, & \text{if } (x_l = y_l) \\ |Rank(x_l) - Rank(y_l)|, & \text{otherwise} \end{cases} \tag{11}$$

$Rank(x_l)$ is the position of x_l in the order of the scale. Given the distance $d(x,y) = 1$ of two records means, that only one categorical attribute value differs, and the categories are ordered next to each other. This is the smallest distance, that two ordinal records can have. The next upper distance is 2, thus the distance concept is discrete.

```
1  getMatrixRow(W, X∈N, cost){
2      μ=(μ1,...,μk);
3
4      for (Wj∈W, j=1,...,k){
5          dj = getDistance(Wj, X);}
6      for (Wj∈W, j=1,...,k){
7          μ[j] = 1/power(dj/sum(dj), 1/(m−1));
8          cost += μj*getDistance(Wj, X);}
9      return μ;
10 }
```

Listing 1: Membership calculation of one record.

```
1  newFuzzyCentroids(k){
2      W1,W2={Wj}, j=1,...,k;
3      M=[μx,y], x=1,...,n, j=1,...,k;
4      cost1 = 0;
5      cost0 = ∞;
6      iN=0;
7
8      for (Wj∈W1, j=1,...,k){
9          Wj = initializeFuzzyCentroid(j);}
10     while ((cost0 < cost1 || iN < 1) && iN < max)
11     {
12         cost1 = cost0;
13         for(Xi∈N, i=1,...,n){
14             μi = getMatrixRow(W1, Xi, cost0);
15             W2 = updateCentroids(μi, W2, Xi);}
16         switchPlaces(W1, W2);
17         iN++;}
18 }
```

Listing 2: Implementation of modified FuzzyCentroids algorithm (main phase of CORD).

4.2 Optimizing FuzzyCentroids for Implementation

The steps of FuzzyCentroids as defined in (Kim et al., 2004) are not optimized. The time complexity is given as $O(kn(r+M)s)$ where $M=\sum_{l=1}^{r} n_l$ is the total number of categories of all attributes. However, for our problem of clustering very large databases, it makes sense to have a closer look at the real runtime. Implementing the algorithm as described by Kim et al. would need 5 iterations through the whole dataset. Our first aim is minimizing the iteration cycles.

For the minimization, we changed the steps of the algorithm. The calculation of ω_{l_t} and $\mu_{i,j}$ requires each an iteration through the dataset. However it is possible to perform both steps successively for a record. In addition the calculation of the cost function can be done in parallel (listing 1, line 8). The modifications do not change the functionality of FuzzyCentroids, but decrease the computation time. As a result the modified version of FuzzyCentroids iterates only once through the whole dataset in the main method. All steps are done in the `for loop` for each x_i, see listing 2, line 13. The confidence degree matrices W_l for all clusters are saved twice in the third order tensors $W1$ and $W2$. This allows in each iteration accessing the result from the last iteration and save the current result. The method `switchPlaces` changes the role of the tensors in-between.

```
1  getDistance(X∈N, Wl) {
2      d = 0;
3
4      for (i=1,...,r){
5          for (j=1,...,ni){
6              if (xi∈X != ai,j∈Di){
7                  d += |xi−ai,j|*Wl[i][j];}}}
8      return d;
9  }
```

Listing 3: New distance measure record-to-centroid.

```
1  updateCentroids(μ, W, X∈N){
2      for (Wj∈W, j=1,...,k){
3          for (i=1,...,r, xi∈X){
4              ωi,xi += Math.pow(μj, m);}}
5      return W;
6  }
```

Listing 4: Confidence degree (resonance) calculation.

The main method, shown in listing 2, executes two more methods. The first `getMatrixRow`, listing 1, computes the membership μ_j for the current x_i and the cost function. This is different to (Kim et al., 2004), where the membership matrix is computed for all $X \in N$ in one step and the cost function in another step. The computation for membership and cost both rely on the measurement of distance.

The `getDistance` method is shown in listing 3. For our needs, the distance calculation of (Kim et al., 2004) is modified according to the considerations in subsection 4.1 and formula (10).

Listing 4 shows the update of centroids. The new resonance of $\omega_{l,t}$ is calculated incrementally by taking into account the membership degrees of one record after another. The new tensor $W2$ results at the end of an iteration. As described in section 3.1, we perform a normalization of $\omega_{l,t}$ at the end of each iteration.

4.3 Increasing Speed with BIRCH

(Chiu et al., 2001) introduces a procedure developing on BIRCH, using an hierarchical clustering algorithm. It works, with the help of a distance measure based on *maximum likelihood estimation*, on mixed numerical and categorical data. As the main goal of our work is not on building a cluster hierarchy, but on a partition and only on ordinal data, further improvements for the selection of the distance measure and the algorithm can be realized. The distance measurement described in 4.1 ensures an optimal and useful distance concept, that can be applied here, too.

As described in 3.2, BIRCH builds the so called *cluster features*. Each CF is initialized with one

record as centroid. Other records will be absorbed, as long as they are similar in a certain distance interval. The parameter T controls the distance interval size (*radius*). Using the described distance measure for ordinal values, the parameter T defines how many attribute values can be different from the centroid of the CF, to be still part of the sub set, represented by the CF. Thereby the information stored for each CF is simplified too. Contrary to (Zhang et al., 1996) only the centroid has to be held in the memory.

If the size of the CF tree borders the provided memory during the execution, the tree must be reduced. Here the T parameter is used. T is increased and the CFs of the old tree leaves are inserted into a new tree. Due to the larger radius, some of the CFs of the old tree absorb other CFs in the new tree. The tree becomes smaller and, in addition, more diffused. This is a good side-effect, since it leads to the fact that outliers have marginal influence on the result, as they disappear in the crowd.

The optimal choice of the parameter T is a difficult task. In (Zhang et al., 1996) a heuristic is used, to determine T_{i+1} from the previous value T_i, the quantity of already processed records N_i, and the size of the whole dataset N. For our application (see 1.1), we define and use the following alternative heuristic:

$$T_{i+1} = 1 + T_i + \log_2\left(\frac{N}{N_i}T_i\right) \qquad (12)$$

This heuristic should provide some advantages: At the beginning, when the CF tree is too big, the lower the ratio processed records per dataset, the more the radius increases. At the end, when the most records are processed, the increase of the radius should indeed flatten but still remain high enough to ensure, that no excessive reconstruction of the CF tree takes place. At the start $T_0 = 0$ and $T_1 = 1$.

The first phase of CORD is based on BIRCH and shown in listing 5. *CFTree* is the data structure of the CF tree and CF_i is the current processed cluster feature. Each record $X \in N$ is first transformed into a cluster feature with `createClusterFeature`, listing 5, line 8. If the tree is not too big, the method `addToNode` adds the new CF into the tree by finding the correct position. If the CF tree borders the provided memory, the processing of cluster features is paused and `updateRadius` is called to increase the radius by using the heuristic, formula (12). After this step the method `rebuildCFTree` adds the CF leaves of the old tree leaves into a new tree. Afterwards the processing continues with the next record. Finally the set S of records, i.e. the set of CF members of all leaves in the CF tree, is returned.

The method `addToNode` is shown in listing 6. A

```
1  S = {center(CF_1), ..., center(CF_ns)},
2  N = {X_1, ..., X_n}
3
4  birch(N){
5     T = 0;
6
7     for (∀ X ∈ N){
8        CF_i = createClusterFeature(X);
9        if (size(CFTree) <= maxSize–size(CF_i)){
10          CFTree =
11          addToNode(CFTree, CF_i, T);}
12       else{
13          T = updateRadius(T);
14          CFTree =
15          rebuildCFTree(CFTree, T);
16          addToNode(CFTree, CF_i, T);}}
17    return S = buildSummary(CFTree);
18 }
```

Listing 5: Implementation of the first phase of CORD, based on BIRCH.

```
1  addToNode(CFTree, CF, T){
2     node = min_z(node|node ∈ root(CFTree), z = d(CF, node));
3
4     while(!isLeaf(node)){
5        node = min_z(Child|Child ∈ node, z = d(CF, Child));}
6
7     if (size(node) < L){
8        for (∀ CF_i ∈ node){
9           if (d(CF_i, CF) < T)
10             return;}
11       node.add(CF);
12       return;}
13    else{
14       M = {CF_i ∈ node} = splitLeaf(node);
15       for (∀ CF_i ∈ M ∪ CF)
16          CFTree =
17          addToNode(CFTree, CF_i, T);}
18    return CFTree;
19 }
```

Listing 6: Addition of cluster feature to a CF tree node.

CF tree node is denoted by *node*. In case of a leaf node, it is composed of a set of L CFs, otherwise a set of B CFs. First, the appropriate leaf for adding *CF* is searched recursively. The *CF* is added taking into account the restrictions. Therefore the distance function `d` checks if the CF to be inserted will be absorbed by one of the CFs of it's appropriate leaf. `d` uses the distance measurement according to subsection 4.1.

If the appropriate leaf is full, it has to be split into two pieces with `splitLeaf`. The method `addToNode` is recursively executed, for each of them. If there is no space in the leaf, the recursion continues splitting the CF tree nodes, until a leaf is reached with enough space. If the procedure splits the tree root into two nodes, these nodes are carried by a new root, that is built. Thereby tree depth grows by one level.

This implementation of BIRCH is used for the **first phase** of CORD. In the **main phase**, the result from BIRCH is clustered by the modified FuzzyCen-

troids described in 4.2. The result from BIRCH can also be processed by a phase in-between, as described in the following section. This new phase is used to increase quality by finding adequate initial centroids.

4.4 Increasing Quality with NCSOM

Before the dataset summary from BIRCH can be clustered by FuzzyCentroids, the number k of centroids and their values have to be selected. This phase is crucial for the quality of the final result. For this issue the NCSOM algorithm can help by discovering the clustering structure, which is visualized as follows: The NCSOM forms a map of reference vectors, whereby similar vectors lie closer to each other, as described in section 3.3. The distance between two neighboring vectors can be illustrated on a color palette of gray tones. For small distances bright colors are used and with increasing distance darker colors. A similarity map of the data builds up as shown in figure 1. Groups of similar neurons can be identified as bright areas on the similarity map. If the users then selects a neuron from the center on this bright area, this reference vector will be a quite good choice as centroid of the records represented by this area. This representation is thus not only an assistance for finding the *number of centroids*, that exist in the clustering structure. It can additionally help very well to specify the initial centroids allocation for a following cluster analysis.

We modified the distance measure of NCSOM, to use it with ordinal data only, see formula (13) and (14). The modifications are done accordingly to 4.1.

$$d(X_i, m_j) = \sum_{l=1}^{r} \delta\left(x_{i,l}, m_{j,l}\right), \tag{13}$$

with

$$\delta(x_{i,l}, m_{j,l}) = \begin{cases} 0, & \text{if } (x_{i,l} = m_{j,l}) \\ |x_{i,l} - m_{j,l}|, & \text{otherwise} \end{cases} \tag{14}$$

The implementation of NCSOM is shown in listing 7. Parameter $\delta(t)$ gives a monotonically decreasing radius and s_i the vector space coordinates of the neuron in the network. This leads to a fast move of the neurons to their approximate place. In the further process they adapt slightly until they arrive in their final place. A good choice of $\delta(t)$ is the diagonal network length.

Step 1 of NCSOM is implemented in line 2 and 3, listing 7. `initializeDelta` initializes the parameter $\delta(t)$ and `initializeArtificialNetwork` the reference vectors randomly. The data structure of the $p \times p$ neuron network is a matrix $M_{p\times p} = [m_{i,j}]$. $ref(m_{i,j})$ is the reference vector of the respective neuron, $pos(m_{i,j})$ the vector of it's position in the network. $nsum(m_{i,j})$ contains in each case a matrix $V_{r\times n_r} = [v_{s,t}]$ for $s = 1,...,r$ attributes and $t = 1,...,n_r$

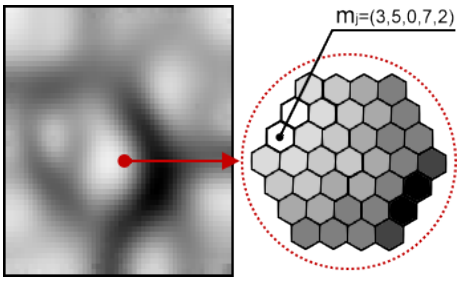

Figure 1: Similarity map (left) and a reference vector m_j of one neuron (right).

```
1  NCSOM(size, maxIterations){
2      δt = initializeDelta(size, maxIterations);
3      initializeArtificialNetwork(M, size);
4
5      while (δt>0){
6          for (Xl ∈ N, l=1,...,n){
7              mi,j = findBestMatchingUnit(M, Xl);
8              saveNeighbourSums(Xl, mi,j, M, δt);}
9          for (i=1,...,p, j=1,...,p){
10             batchUpdateUnit(mi,j);}
11         δt−−;}
12 }
```

Listing 7: Implementation of the second phase of CORD, based on NCSOM.

categories. This matrix saves the summations that are used in *step 3* (see below).

The distance measurement is used during the selection of the best matching neuron (*Step 2*). The best matching neuron for a record is searched in each iteration with the method `findBestMatchingUnit`, line 7, listing 7. This is the neuron with the reference vector that has the smallest distance to the record according to formula (13) and (14). The implementation of the distance measurement is shown in listing 8. After the winner neuron is found, the necessary information for the update, has to be collected. Listing 9 shows the calculation of the matrices $V_{r\times n_r} = [v_{s,t}]$. This computes the occurrence frequency of the categorical values for the r attributes of the neighborhood records, using the Gaussian function defined in (Chen and Marques, 2005). The computational complexity of this step is $O\left(n^2\right)$, as for each record and neuron a matrix of sums has to be processed.

The update of the reference vectors (*Step 3*) is

```
1  getDistance(X∈N, mi,j) {
2      d = 0;
3      for (i=1,...,r){
4          for (j=1,...,ni){
5              if (xi∈X != aij∈Di){
6                  d += |xi−aij|;}}}
7      return d;
8  }
```

Listing 8: New distance measure between record an neuron.

done in line 9 to 10, listing 7, at the end of each iteration. The details of the implementation is shown in listing 10. The neurons are updated according to formula 15 and 17 in (Chen and Marques, 2005). The summations are used in line 5. This update requires one iteration through the whole dataset. This is relevant in the case of VLDBs, as the computation time is linear in $O(maxIterations * n)$ on the dataset size n.

An important factor is the network size. The calculation in listing 9 and 10 respectively is quadratic $(O(p^2))$ on the neuron number p of one side. Summarizing, the best case computation time is approximately at least $maxIterations * n$, as the algorithm always executes the maximum number of steps. This is necessary, since the number of steps is attached to the function δ_t. However, also in this case, the dataset must be read only once per iteration. This *batch algorithm* has a better accuracy than an *online algorithm* but requires memory for an additional data structure; the matrix $V_{r \times n_r}$ for the neurons.

5 EXPERIMENTAL RESULTS

To evaluate CORD, we performed tests with two different datasets. Subsection 5.1 describes the tests with a dataset of 100,000 records and 5.2 the tests with 14 million records. As with work of this nature, i.e a *hybrid approach*, a comparison is difficult, especially as the proposed solution is tailored to suit a specific use case. In our solution, the three phases of CORD as described in section 4 can be processed independently. For example one can select an automatic solution without using NCSOM in-between. Therefore we decided to evaluate each one separately.

In general algorithms are dependent on the read-

```
1 saveNeighbourSums(X ∈ N, m_z, M, δ_t){
2     for (k = 1,...,p, l = 1,...,p){
3         for (i = 1,...,r){
4             for (j = 1,...,n_i){
5                 nsum(m_{k,l})[i][x_i] +=
6                 exp(−||pos(m_{k,l}) − pos(m)||^2 / 2δ_t^2);}}}
7 }
```

Listing 9: Calculation of categorical value frequency of the records in the neighborhood.

```
1 batchUpdateUnit(m){
2     for (i = 1,...,r){
3         double s = 0.0;
4         for (j = 1,...,n_i){
5             s += j * nsum(m) / sum(nsum(m)[k]);}
6         ref(m)[k] = round(s);}
7 }
```

Listing 10: Update of the neuron reference vector at the end of each iteration.

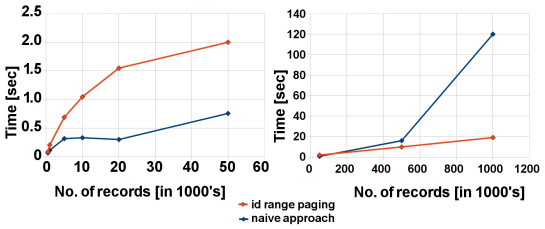

Figure 2: Reading time for naive approach and *paging*. Results for small record no. (left) and large data sets (right).

ing time of the data. Therefore we performed an optimization of the reading process, we named as *id range paging* (results shown in figure 2). A data base query loads intervals of records (as page) on the basis of the primary key from the data base.

The *id range paging* processes approximately 50 records in one millisecond. It is clearly observable, that the *id range paging* exceeds the naive approach, for which an *out-of-memory exception* occurred after 2 minutes and one million records. Further we tested the influence of different page sizes on the reading time. The size of 500 provided the best results.

5.1 Dataset User Profiles

Input for the test was a dataset of approx. 100,000 records (Gugubarra, 2009) according to the *problem application user profiles* (see 1.2). The test was done on a PC with Intel Core 2 Duo T7500 processor with 2.20GHz and 2GB DDR2 SDRAM main memory.

The module for the **FuzzyCentroids** was tested with the result of BIRCH as input. Therefore the CF tree is held in the main memory and the FuzzyCentroids can be executed quite fast. The computation time depends linearly on the number of attributes. The average of three test runs is shown in figure 3(b) for 10, 15, 20, and 100 attributes and 1,000 records. The number 100 is normally not exceeded for the described user profiles (see subsection 1.2) and supports the regression line built out of the results for the other attribute numbers. For 100 attributes the computation time was 150 milliseconds. Similarly are the results for the number of records. The computation time depends linearly on it as shown in figure 3(a) for 1,000, 2,000, 3,000 and 105,691 records and 10 attributes. For 105,691 records the computation time was about 2 seconds and supports the regression line that is shown.

For testing the computation time of the **BIRCH** module, we focused on the most interesting criteria. This is when the CF tree is too big for the main memory. At this point the CF tree has to be rebuilt which

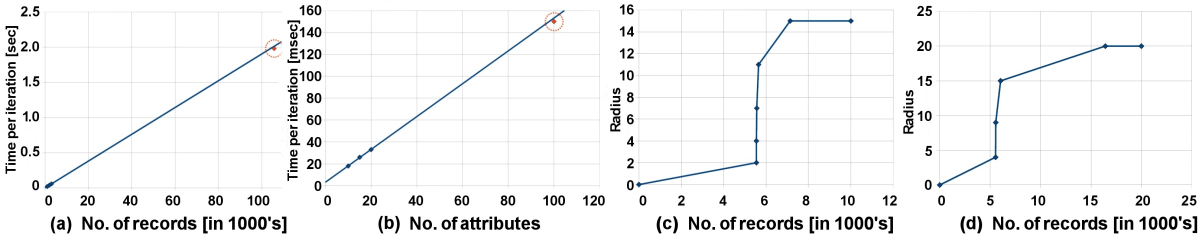

Figure 3: FuzzyCentroids evaluation for number of records (a) and attributes (b). BIRCH evaluation: Increase in the radius for 10000 records (c) and 20000 records (d).

takes the most time. In order to ensure, that no excessive reconstruction of the CF tree takes place we proposed a heuristic in 4.3. It's functionality was proven empirically by repeated runs of the module for different number of records and CF tree parameters. The average of 3 test runs is shown in figure 3(c),(d) for 10,000 and 20,000 records.

The parameters were set as follows: The maximum no. of nodes as 5, maximum no. of CFs in one leaf as 15, and the maximum size of the CF tree was $5,000,000$ bytes. The result shows that at the end, when the most records are processed, the increase in the radius is less. Most important, the steps are big enough to ensure a fast inclusion of the whole data set into the CF tree. That is for 20,000 records after 5 enlargements only. It is not a perfect logarithmic function, because the dataset used is uniform.

Similar to the FuzzyCentroids, the computation time of **NCSOM** depends linearly on the number of records and attributes. A more interesting evaluation would be regarding the influence of the neural network size, that we report here instead. The time complexity is quadratic. The average of three test runs is shown in figure 4 for 1,000 records with different network sizes. The quadratic dependency is clearly observable. Further the running time depends on the number of neighbor neurons used for the computation of the similarity map. Since this computation is implemented at the end of each iteration and not for each record, the effect is small compared with other steps.

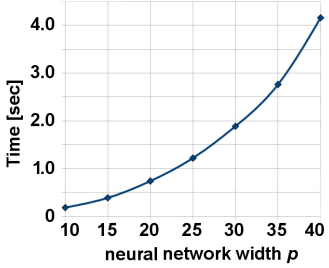

Figure 4: NCSOM evaluation for neural network size.

5.2 Dataset MSNBC.COM Web Data

The second performance test was done to prove the scalability of the system. It was done on a Windows Server with two Intel Xeon E5420 processors with 2.50GHz and 8GB main memory.

The dataset *MSNBC.COM Data Set* was taken from the *UCI ML Repository* (D.J. Newman, 2007). The data comes from the Server logs for msnbc.com and news-related portions of msn.com. Each sequence in the dataset corresponds to page views of a user during one day. Each event in the sequence corresponds to a user's page request. Requests are not recorded at the page level but rather, they are recorded at the level of page topics. The fourteen topics we used are "frontpage", "news", "tech", "local", "opinion", "on-air", "misc", "weather", "health", "living", "business", "sports", "summary", and "bbs". The full dataset consists of approx. one million users (i.e. 989,818).

This data is very useful, as it represents clickstreams associated to topics. From this data, it is very easy to generate profiles that are similar to the NOPs like described in subsection 1.2. We mapped the number of page visits on an ordinal scale, representing which topics the user has more or less "touched". In the Gugubarra Project the number of topics (attributes) changes dynamically. Therefore each attribute value is saved in a separate record. For the dataset MSNBC.COM we chose the same domain specific database schema to have a realistic and comparable situation. As a result, CORD had to work on *attribute* $\times$ *user* records, that is 14 million records.

The average of 3 test runs for FuzzyCentroids using BIRCH was a computation time of approx. 3,300 seconds. This is a good result compared with (Kim et al., 2004). The average of 3 test runs for NCSOM using BIRCH was a computation of approx. 3,400 seconds. The tree size decreased from 9,200 KBytes to 1,800 KBytes, when the radius increased to 3. The results are similar because each of the tests had to read the records from the database. This was the slowest part as a lot of disk reads had to be done.

6 RELATED WORK

In the area of *clustering Web users*, it is a new approach to take into account the non-obvious profiles, described in (Zicari et al., 2006). Clustering non-obvious-profiles with CORD takes indirectly into account the *time* spent by the user on a page and the *content topics* of this page. CORD clusters users based on their supposed interest in these topics. With the fuzzy centroids, it gives an interpretation to the clusters as a value of a predefined scale of interest. This is a nameable advantage. Several algorithms have been proposed in the area of *clustering large datasets*, as BIRCH (Zhang et al., 1996) and CLARANS (Clustering Large Applications based on RANdom Search) by Ng and Han (Ng and Han, 1994). There are *specialized fields*, e.g. multi-relational data clustering. Yin and Han proposed here CrossClus (Yin et al., 2007), that clusters data stored in multiple relational tables based on user guidance and multi-relational features. This algorithm requires as CORD the help of the user, that is here the person who wants to cluster the elements. Clustering can be applied to various domains and issues, e.g. in (Aggarwal et al., 2006) the k-anonymity (a technique to preserve privacy in data) is treated as a special clustering problem, called r-cellular clustering. (Aggarwal et al., 2006) handle categorical attributes by the representation as n equidistant points in a metric space. *Hybrid Systems* are used in various research fields, e.g. in the area of *Web* Burke (Burke, 2002) has defined Hybrid Recommender Systems, that combine information filtering and collaborative filtering techniques. Helmer proposed in (Helmer, 2007) a hybrid approach to measure the similarity of semistructured documents based on entropy. (Kossmann et al., 2002) use a hybrid approach to find the Skyline, i.e. a set of interesting points from a potentially large set of data.

REFERENCES

Aggarwal, G., Feder, T., and Kenthapadi, K. (2006). Achieving anonymity via clustering. In *Proc. of the* 25^{th} *ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 153–162, NY, USA.

Braun-Blanquet, J., Conard, H. S., and Fuller, G. D. (1932). *Plant sociology*. McGraw-Hill book company. http://www.biodiversitylibrary.org/bibliography/7161.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370.

Chen, N. and Marques, N. C. (2005). An extension of self-organizing maps to categorical data. In *EPIA*, Portugal.

Cheu, E. Y., Kwoh, C. K., and Zhou, Z. (2004). On the two-level hybrid clustering algorithm. *Nanyang Technological University*.

Chiu, T., Fang, D., Chen, J., Wang, Y., and Jeris, C. (2001). A robust and scalable clustering algorithm for mixed type attributes in large database environment. In *Proceedings of the* 7^{th} *ACM SIGKDD*, pages 263–268, NY, USA.

D.J. Newman, A. A. (2007). UCI machine learning repository. http://archive.ics.uci.edu/ml/.

Gan, G., Yang, Z., and Wu, J. (2005). A genetic k-modes algorithm for clustering categorical data. In *ADMA*, pages 195–202.

Gugubarra (2009). Data set user profiles. www.dbis.cs.uni-frankfurt.de/downloads/research/data.zip.

Helmer, S. (2007). Measuring the structural similarity of semistructured documents using entropy. In *Proc. of the* 33^{rd} *Int. Conf. on VLDBs*, pages 1022–1032.

Huang, Z. (1997). A fast clustering algorithm to cluster very large categorical data sets in data mining. In *In Research Issues on Data Mining and Knowledge Discovery*, pages 1–8.

Kim, D.-W., Lee, K. H., and Lee, D. (2004). Fuzzy clustering of categorical data using fuzzy centroids. *Pattern Recogn. Lett.*, 25(11):1263–1271.

Kossmann, D., Ramsak, F., and Rost, S. (2002). Shooting stars in the sky: an online algorithm for skyline queries. In *Proc. of the* 28^{th} *Int. Conf. on VLDBs*, pages 275–286.

Ng, R. T. and Han, J. (1994). Efficient and effective clustering methods for spatial data mining. In *Proc. of the* 20^{th} *Int. Conf. on VLDBs*, pages 144–155, San Francisco, CA, USA. Morgan Kaufmann Pub. Inc.

Parmar, D., Wu, T., and Blackhurst, J. (2007). Mmr: An algorithm for clustering categorical data using rough set theory.

Podani, J. (2005). Multivariate exploratory analysis of ordinal data in ecology: Pitfalls, problems and solutions. *Journal of Vegetation Science*, 16(5):497–510.

Yin, X., Han, J., and Yu, P. S. (2007). Crossclus: user-guided multi-relational clustering. *Data Min. Knowl. Discov.*, 15(3):321–348.

Zhang, T., Ramakrishnan, R., and Livny, M. (1996). Birch: An efficient data clustering method for vldbs. In *Proc. of the ACM SIGMOD*, pages 103–114, Montreal, Canada.

Zicari, R. V., Hoebel, N., Kaufmann, S., and Tolle, K. (2006). The design of gugubarra 2.0: A tool for building and managing profiles of web users. In *Proc. of the IEEE/WIC/ACM Int. Conf. on Web Intelligence*, pages 317–320, Washington, DC, USA.