Distributed Clustering Based on Sampling Local Density Estimates

Matthias Klusch
Deduction and Multiagent Systems
German Research Centre
for Artificial Intelligence
Stuhlsatzenhausweg 3
66123 Saarbruecken, Germany
klusch@dfki.de

Stefano Lodi
Department of Electronics,
Computer Science and Systems
University of Bologna
Viale Risorgimento 2
I-40136 Bologna BO, Italy
slodi@deis.unibo.it

Gianluca Moro
Department of Electronics,
Computer Science and Systems
University of Bologna
Via Rasi e Spinelli, 176
I-47023 Cesena FC, Italy
gmoro@deis.unibo.it

Abstract

Huge amounts of data are stored in autonomous, geographically distributed sources. The discovery of previously unknown, implicit and valuable knowledge is a key aspect of the exploitation of such sources. In recent years several approaches to knowledge discovery and data mining, and in particular to clustering, have been developed, but only a few of them are designed for distributed data sources. We propose a novel distributed clustering algorithm based on non-parametric kernel density estimation, which takes into account the issues of privacy and communication costs that arise in a distributed environment.

1 Introduction

Knowledge discovery is a process aiming at the extraction of previously unknown and implicit knowledge out of large databases, which may potentially be of added value for some given application [Fayyad *et al.*, 1996].

Data mining, which is devoted to the automated extraction of unknown patterns from given data, is a central element among the steps of the overall knowledge discovery process; the steps include preparation of the data to be analyzed as well as evaluation and visualization of the discovered knowledge. The large variety of data mining techniques which have been developed over the past decade include: methods for pattern-based similarity search, cluster analysis, decision-tree based classification, generalization taking the data cube or attribute-oriented induction approach, and mining of association rules [Chen *et al.*, 1996]

The increasing demand to scale up to massive data sets which are inherently distributed over networks with limited bandwidth and computational resources has led to methods for parallel and distributed knowledge discovery [Kargupta *et al.*, 2000]. The related pattern extraction problem in distributed knowledge discovery is referred to as *distributed data mining*. Distributed data mining is expected to perform partial analysis of data at individual sites and then to send the outcome as partial result to other sites where it is sometimes aggregated to the global result.

One of the most common approaches of business applications to perform distributed data mining is to centralize distributed data into a *data warehouse* on which to apply the usual data mining techniques. Data warehousing is a popular technology which integrates data from multiple data sources into a single repository in order to efficiently execute complex analysis queries [Moro and Sartori, 2001]. However, despite its commercial success, this approach may be impractical or even impossible for some business settings, for instance:

- when huge amounts of data are (frequently) produced at different sites and the cost for their centralization cannot scale in terms of communication, storage and computation;
- whenever data owners cannot or do not want to release information, for instance to protect privacy or because disclosing such information may result in a competitive advantage or a considerable commercial added value.

One of the most studied data mining techniques in centralized environments is *data clustering*. The goal of this technique is to decompose or partition a data set into groups such that both intra-group similarity and inter-group dissimilarity are maximized. Despite the success of data clustering in centralized environments, only a few approaches to the problem in a distributed environment are available to date.

In this work we present KDEC, a novel approach to distributed data clustering based on sampling density estimates. In KDEC each data source transmits an estimate of the probability density function of its local data to a helper site, and then executes a density based clustering algorithm that is driven by the overall density estimate, which is built by the helper from the samples of the local densities.

The paper is organized as follows. In Section 2 we describe related work and highlight differences with respect to our approach. Section 3 and 4 present the KDEC scheme to distributed data clustering. Finally, Section 5 concludes the paper and outlines ongoing and future research work.

2 Related work

In [Johnson and Kargupta, 1999] a tree clustering approach is taken to build a global dendrogram from individual dendrograms that are computed at local data sites subject to a given set of requirements. In contrast to the approach presented in this paper, the distributed data sets are assumed to be heterogeneous, therefore every site has access only to a subset of the features of an object. The proposed solution implements

a distributed version of the single-link clustering algorithm which generates clusters that are substantially different from the ones generated by density-based methods. In particular, it suffers from the so-called chaining effect, by which any of two well separated and internally homogeneous groups of objects connected only by a dense sequence of objects are regarded as a single cluster. [Kargupta *et al.*, 2001] proposes a technique for distributed principal component analysis, Collective PCA. It is shown that the technique satisfies efficiency and data security requirements and can be integrated with existing clustering methods in order to cluster distributed, high-dimensional heterogeneous data. Since the dimensionality of the data is reduced prior to clustering by applying PCA, the approach is orthogonal to ours. Another related research direction deals with incremental clustering algorithms. The BIRCH [Zhang *et al.*, 1996] and related BUBBLE method [Ganti *et al.*, 1999], compute the most accurate clustering, given the amount of memory available, while minimizing the number of I/O operations. It uses a dynamic index structure of nodes that store synthetic, constant-time maintainable summaries of sets of data objects. The method is sufficiently scalable requiring $O(N \log N)$ time and linear I/O. However, since it uses the centroid to incrementally aggregate objects, the method exhibits a strong bias towards globular clusters. IncrementalDBSCAN [Ester *et al.*, 1998] is a dynamic clustering method supporting both insertions and deletions, which is shown to be equivalent to the well-known static DBSCAN algorithm. Since in turn DBSCAN can be shown to be equivalent to a method based on density estimation when the kernel function is the square pulse and the clusters are density-based, IncrementalDBSCAN is less general than methods based on kernel density estimates. Its time complexity is $O(N \log N)$.

3 Data Clustering

3.1 The cluster analysis problem

Cluster analysis is a a descriptive data mining task which aims at decomposing or partitioning a usually multivariate data set into groups such that the data objects in one group are similar to each other and are different as possible from those in other groups. Therefore, a clustering algorithm $\mathcal{A}(\cdot)$ is a mapping from any data set S of objects to a *clustering* of S, that is, a collection of pairwise disjoint subsets of S. Clustering techniques inherently hinge on the notion of distance between data objects to be grouped, and all we need to know is the set of interobject distances but not the values of any of the data object variables. Several techniques for data clustering are available but must be matched by the developer to the objectives of the considered clustering task [Grabmeier and Rudolph, 2002]. In partition-based clustering, for example, the task is to partition a given data set into multiple disjoint sets of data objects such that the objects within each set are as homogeneous as possible. Homogeneity here is captured by an appropriate cluster scoring function. Another option is based on the intuition that homogeneity is expected to be high in densely populated regions of the given data set. Consequently, searching for clusters may be reduced to searching for dense regions of the data space which are more likely to be populated by data objects. That leads us to the approach of density estimation based clustering.

3.2 Density estimation based clustering

In *density estimation* (DE) based clustering the search for densely populated regions is accomplished by estimating a so-called probability density function from which the given data set is assumed to have arisen. Many techniques for DE-based clustering are available from the vast KDD literature [Ankerst *et al.*, 1999; Ester *et al.*, 1996; Schikuta, 1996; Hinneburg and Keim, 1998] and statistics [Silverman, 1986]. In both areas, the proposed clustering methods require the computation of a non-parametric estimation of the density function from the data. One important family of non-parametric estimates is known as *kernel estimators*. The idea is to estimate a density function by defining the density at any data object as being proportional to a weighted sum of all objects in the data set, where the weights are defined by an appropriately chosen kernel function. In the following we introduce kernel-based density estimation [Parzen, 1962; Silverman, 1986] and our approach to density estimation based clustering.

Let us assume a set $S = \{\vec{x}_i \mid i = 1, \ldots, N\} \subseteq \mathbb{R}^n$ of data points or objects. Kernel estimators originate from the intuition that the higher the number of neighbouring data objects $\vec{x}_i$ of some given object $\vec{x} \in \mathbb{R}^n$, the higher the density at this object $\vec{x}$. However, there can be many ways of capturing and weighting the influence of data objects. When given the distance between one data object $\vec{x}$ and another $\vec{x}_i$ as an argument, the influence of $\vec{x}_i$ may be quantified by using a so called kernel function. A *kernel function* $K(x)$ is a real-valued, non-negative function on $\mathbb{R}$ which has finite integral over $\mathbb{R}$. When computing a kernel-based density estimation of the data set S, any element $\vec{x}_i$ in S is regarded as to exert more influence on some $\vec{x} \in \mathbb{R}^n$ than elements which are farther from $\vec{x}$ than the element. Accordingly, kernel functions are often non-increasing with $|x|$. Prominent examples of kernel functions are the square pulse function $\frac{1}{4}(\text{sign}(x+1) - \text{sign}(x-1))$, and the Gaussian function $\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}x^2)$. [1]

A *kernel-based density estimate* $\hat{\varphi}_{K,h}[S](\cdot) \colon \mathbb{R}^n \to \mathbb{R}_+$ is defined, modulo a normalization factor, as the sum over all data objects $\vec{x}_i$ in S of the distances between $\vec{x}_i$ and $\vec{x}$, scaled by a factor h, called *window width*, and weighted by the kernel function K:

$$\hat{\varphi}_{K,h}[S](\vec{x}) = \sum_{i=1}^{N} K\left(\frac{d(\vec{x}, \vec{x}_i)}{h}\right). \qquad (1)$$

The influence of data objects and the smoothness of the estimate is controlled by both the window width h and the shape of kernel K: h controls the smoothness of the estimate, whereas K determines the decay of the influence of a data object according to the distance. Even if the number N of data objects is very large, in practice it is not necessary to compute N distances for calculating the kernel density estimate at a given object $\vec{x}$. In fact, the value of commonly used kernel functions is negligible for distances larger than a few h units;

[1] Where $\forall x \in \mathbb{R} \setminus \{0\} : \text{sign}(x) = x/|x|$, and $\text{sign}(0) = 0$.

it may even be zero if the kernel has bounded support, as it is the case, for example, for the square pulse. Using kernel-based density estimation, it is straightforward to decompose the clustering problem into three phases as follows.

1. Choose a window width h and a kernel function K.
2. Compute the kernel-based density estimate $\hat{\varphi}_{K,h}[S](\vec{x})$ from the given data set.
3. Detect regions of the data space where the value of the estimate is high and group all data objects of space regions into corresponding clusters.

In the literature, many different definitions of cluster have been proposed formalizing the clusters referred to in step 3 above. A *density-based* cluster [Ester *et al.*, 1996] collects all data objects included in a region where density exceeds a threshold. *Center-defined* clusters [Hinneburg and Keim, 1998] are based on the idea that every local maximum of $\hat{\varphi}$ corresponds to a cluster including all data objects which can be connected to the maximum by a continuous, uphill path in the graph of $\hat{\varphi}$. Finally, an *arbitrary-shape* cluster [Hinneburg and Keim, 1998] is the union of center-defined clusters having their maxima connected by a continuous path whose density exceeds a threshold.

Algorithm 1 (DE-cluster) implements the computation of center-defined clusters by a climbing procedure driven by the density estimate. The main procedure is *DECluster*, taking as inputs an instance S of the class of data objects, the kernel function K, the window width h, and returning a clustering represented by C, which stores a mapping from each $\vec{x}_i$ to the unique integer label of $\vec{x}_i$'s cluster. It is assumed that S is an instance of a class which provides the following methods: *get(i)* to access object $\vec{x}_i$ given index i, $NQ(k,\vec{x})$ and $Radius(k,\vec{x})$ to retrieve, given $\vec{x} \in \mathbb{R}$, the indexes and maximum distance of $\vec{x}$'s k nearest neighbours. *Uphill* computes the steepest direction on the graph of the estimated density as the versor of its gradient, computed by function *DEGradient* (cf. [Hinneburg and Keim, 1998]). *Uphill* then moves in that direction a fraction $\delta < 1$ of the distance $S.Radius(k,\vec{x})$ of the kth nearest neighbour of $\vec{x}$, and finally returns the index of the nearest neighbour in S of the reached position. Every nested call to *FixedPoint* marks the current object $\vec{x}_i$ as visited and calls *Uphill* to get the index of the next data object $\vec{x}_j$. If such object has already been visited, the proximity of a local maximum has been reached and j is taken as new cluster label. Otherwise $\vec{x}_j$ is inductively assumed to lie at the bottom end of a path leading to the proximity of a local maximum, and to be already labeled accordingly. (If $\vec{x}_j$ is not marked as clustered, a recursive call ensures that the assumption holds.)

The complexity of the DE-cluster algorithm is that of calling $N = S.count$ times *FixedPoint*. At the beginning of every iteration in *DECluster*, the sets of clustered and visited objects are equal. *FixedPoint* is never called with a clustered object as argument, and visits unclustered objects at most once. Therefore, even if the number of visited data objects in one call of *FixedPoint* is bounded only by N, the number of visited data objects in all calls is still only N. For each visited object a single k-nearest neighbour query suffices to compute the gradient and the next uphill object. The methods $NQ(\cdot,\cdot)$ and $Radius(\cdot,\cdot)$ can be efficiently implemented by equipping the class of S with a spatial access method like the KD-, or MVP-, or M-tree. Therefore, the time complexity of *DECluster* is $O(Nq(N))$, where $q(N)$ is the cost of a k nearest neighbour query in any such access method. Note that in many practical cases, $q(N)$ is very close to $\log N$.

Algorithm 1 DE-cluster: Clustering based on density estimation

```
funct DEGradient(x⃗, S, K(), h) ≡
  foreach i ∈ S.NQ(k, x⃗) do
    g := g + (x⃗_i − x⃗) * K(d(x⃗, x⃗_i)/h) od
  g.
funct Uphill(i, S, K(), h) ≡
  x⃗ := S.get(i)
  v⃗ := DEGradient(x⃗, S, K, h) / ||DEGradient(x⃗, S, K, h)||
  S.NQ(1, x⃗ + δ * S.Radius(k, x⃗) * v⃗).
proc FixedPoint(i, S, K(), h, C)
  C.setVisited(i)
  j := Uphill(i, S, K, h)
  if C.clustered(j)
    then C.setLabel(i, C.getLabel(j))
    else if C.visited(j) then C.setLabel(i, j)
                         else FixedPoint(j, S, K, h, C)
                              C.setLabel(i, C.getLabel(j))
         fi
  fi
  C.setClustered(i).
proc DECluster(S, K(), h, k, C) ≡
  for i := 1 to S.count do
    if ¬C.clustered(i) then FixedPoint(i, S, K, h, C) fi
  od.
```

4 Distributed Data Clustering

The body of work on applications of data clustering in distributed environments, the problem of so called *distributed data clustering* (DDC), is comparatively small. In this section we adopt the kernel density estimation based clustering approach presented above for the distributed case assuming homogeneous data, which means that a data object cannot be split across two sites.

4.1 The DDC Problem

We define the problem of *homogeneous distributed data clustering* for a clustering algorithm $\mathcal{A}$ as follows. Let $S = \{\vec{x}_i \mid i = 1,\ldots,N\} \subseteq \mathbb{R}^n$ be a data set of objects. Let $L_j, j = 1,\ldots,M$, be a finite set of *sites*. Each site L_j stores one data set D_j, and it will be assumed that $S = \bigcup_{j=1}^{M} D_j$. The DDC problem is to find a site clustering $\mathcal{C}_j$ residing in the data space of L_j, for $j = 1,\ldots,M$, such that

(i). $\mathcal{C}_j = \{C \cap D_j : C \in \mathcal{A}(S)\}$ (*correctness requirement*)

(ii). Time and communications costs are minimized (*efficiency requirement*)

(iii). At the end of the computation, the size of the subset of S which has been transferred out of the data space of any site L_j is minimized (*privacy requirement*).

The traditional solution to the homogeneous distributed data clustering problem is to simply collect all the distributed data sets D_j into one centralized repository where their union S is computed, and the clustering $\mathcal{C}$ of the union S is computed and transmitted to the sites. Such approach, however, does not satisfy our problem's requirements both in terms of privacy and efficiency. We therefore propose a different approach yielding a kernel density estimation based clustering scheme, called KDEC.

4.2 The KDEC Scheme for DDC

The key idea of the KDEC scheme is based on the following observation: Although the density estimate computed on each local data set gives information on the distribution of the objects in the data set, it conceals the objects themselves. Moreover, the local density estimate can be coded to provide a more compact representation of the data set for the purpose of transmission. In the sequel, we tacitly assume that all sites L_j agree on using a global kernel function K and a global window width h. We will therefore omit K and h from our notation, and write $\hat{\varphi}[S](\vec{x})$ for $\hat{\varphi}_{K,h}[S](\vec{x})$. Density estimates in the form of Equation (1) are additive, i.e. the global density estimate $\hat{\varphi}[S](\vec{x})$ can be decomposed into the sum of the site density estimates, one estimate for every data set D_j:

$$\hat{\varphi}[S](\vec{x}) = \sum_{j=1}^{M} \sum_{\vec{x}_i \in D_j} K\left(\frac{d(\vec{x},\vec{x}_i)}{h}\right) = \sum_{j=1}^{M} \hat{\varphi}[D_j](\vec{x}). \qquad (2)$$

Thus, the local density estimates can be transmitted to and summed up at a distinguished *helper site* yielding the global estimate which can be returned to all sites. Each site L_j may then apply, in its local data space, the hill-climbing technique of Algorithm 1 (DE-cluster) to assign clusters to the local data objects. There is nevertheless a weakness in such a plan: the definition of a density estimate explicitly refers to all the data objects $\vec{x}_i$. Hence, knowing how to manipulate the estimate entails knowing the data objects, which contradicts the privacy requirement. However, only an intensional, algebraic definition of the estimate includes knowledge of the data objects. Multidimensional sampling theory provides the basis for an alternative extensional representation of the estimate which makes no explicit reference to the data objects.

The theoretical idea of sampling is to represent a function f by a *sampling series*, that is, a summation of suitable expansion functions weighted by the values of f at a discrete subset of its domain [Higgins, 1996]. In the following, let the ith coordinate of $\vec{x} \in \mathbb{R}^n$ be denoted by $\vec{x}^{(i)}$, and let $Diag[\vec{u}]$, $\vec{u} = [u^{(1)}, \ldots, u^{(n)}]^T \in \mathbb{R}^n$, denote the $n \times n$ diagonal matrix having diagonal $\vec{u}$, i.e., defined by $Diag[\vec{u}]_{ij} = 0$ if $i \neq j$, $Diag[\vec{u}]_{ii} = u^{(i)}$. Further let $\vec{\tau} = [\tau^{(1)}, \ldots, \tau^{(n)}]^T \in \mathbb{R}^n$ a vector of *sampling periods*. The *sampled form* of $\hat{\varphi}[S](\vec{x})$ at intervals $\vec{\tau}$ is the sequence $\{\hat{\varphi}_{\vec{z}}[S]\}$, $\vec{z} \in \mathbb{Z}^n$, defined by

$$\hat{\varphi}_{\vec{z}}[S] = \hat{\varphi}[S](Diag[\vec{z}] \cdot \vec{\tau}), \qquad (3)$$

where $\cdot$ is the inner product between vectors. Therefore, $\hat{\varphi}_{\vec{z}}[S]$ is the sequence of the values of $\hat{\varphi}[S](\vec{x})$ at all the real, n-dimensional vectors whose ith coordinates are spaced by a multiple of the ith sampling period $\tau^{(i)}$, $i = 1, \ldots, n$. The sampled forms of the local density estimates are defined in a similar way by $\hat{\varphi}_{\vec{z}}[D_j] = \hat{\varphi}[D_j](Diag[\vec{z}] \cdot \vec{\tau})$, for $j = 1, \ldots, M$. It is immediate to see by (2) that addivity holds for the sampled forms:

$$\hat{\varphi}_{\vec{z}}[S] = \sum_{j=1}^{M} \hat{\varphi}_{\vec{z}}[D_j] \qquad j = 1, \ldots, M. \qquad (4)$$

Therefore, after receiving the sampled forms $\hat{\varphi}_{\vec{z}}[D_j]$ of the M density estimates, the helper site can compute by (4) the sampled form of the overall estimate and transmit it to the sites L_j. Sites L_j can then cluster local data with respect to the overall density estimate, using the gradient of the sampling series

$$\sum_{\vec{z} \in \mathbb{Z}^n} \hat{\varphi}_{\vec{z}}[S] \, \mathrm{Sinc}(Diag[\vec{\tau}]^{-1} \cdot (\vec{x} - Diag[\vec{z}] \cdot \vec{\tau})), \qquad (5)$$

where $\mathrm{Sinc}(\vec{x}) = \prod_{i=1}^{n} \mathrm{sinc}(x^{(i)})$, and

$$\mathrm{sinc}(x) = \begin{cases} 1 & \text{if } x = 0, \\ \frac{\sin \pi x}{\pi x} & \text{otherwise,} \end{cases}$$

as needed in the hill-climbing function.

We briefly discuss the extent to which the series (5) can be used to represent $\hat{\varphi}[S](\vec{x})$. It is well known that, under mild conditions, sampling a function $g(\vec{x})$ is an invertible transformation if, for every coordinate $i = 1, \ldots, n$, there is a frequency $f^{(i)}$ such that the Fourier transform of g differs from zero only in the interval $[-f^{(i)}, f^{(i)}]$, and the samples are computed with a period not greater than $\tau^{(i)} = 1/(2f^{(i)})$ (cf. [Higgins, 1996]). Under these assumptions, the value of the sampling series computed at $\vec{x}$ equals $g(\vec{x})$. Unfortunately, most popular kernel functions (hence summations of kernel functions) do not satisfy these hypotheses since the support of their Fourier transform is unbounded. Consequently sampling density estimates yields an information loss. However, it can be shown that the Fourier transform of a kernel density estimate is negligible everywhere except for $|f^{(i)}|$ not greater than $1/h$. Therefore, the global density estimate can be reconstructed from its samples by (5) introducing only a small error if $\tau^{(i)} < h/2$.

It is worth noting that the infinite series (5) need not be approximated, if it has finitely many nonzero terms. The latter case holds if the used kernel function has a bounded support, since the density estimate will also have bounded support. If, however, the kernel function has unbounded support, like the Gaussian kernel, then the density estimate can be approximated by regarding its value to be zero everywhere except inside an appropriately chosen bounded region.

According to this approach, we propose the following algorithmic KDEC scheme for computing the kernel density estimation based clusters for local data spaces at M distributed data sites L_j (see Algorithm 2). Every local site runs the procedure *SiteDECluster* whereas the helper site runs *Helper*. *SiteDECluster* is passed a reference H to the helper and the local data set D, and returns a clustering in a class instance C. *Helper* is passed a list of references L to the local sites. The procedure *SiteNegotiate* carries out a negotiation with the other local sites through *HelperNegotiate* at the helper site to determine the sampling periods $\vec{\tau}$, the bounding corners of the sampling rectangle $\vec{z}_1, \vec{z}_2 \in \mathbb{Z}^n$, the kernel K, and

Algorithm 2 KDEC: distributed clustering based on density estimation

funct $Sample(D,\vec{\tau},\vec{z}_1,\vec{z}_2,K(),h) \equiv$
  $s.setCorners(\vec{z}_1,\vec{z}_2)$
  foreach $\vec{z} \in Rectangle(s.getCorners())$
    do $s.Add(\vec{z}, DE(Diag[\vec{z}]\cdot\vec{\tau}, D, K, h))$ **od**
  s.
funct $Uphill(\vec{x},\vec{\tau},GlobalSam) \equiv$
  if $\|SeriesGradient(\vec{x},\vec{\tau},GlobalSam)\| > \varepsilon$
    then $\vec{x} + \delta * SeriesGradient(\vec{x},\vec{\tau},GlobalSam)$
    else $\vec{x}$ **fi**.
funct $FixedPoint(\vec{x},D,GlobalSam,C)$
  $\vec{y} := Uphill(\vec{x},\vec{\tau},GlobalSam)$
  $q := D.RangeQ(\delta,\vec{y})$
  if $\exists i \in q : C.clustered(i)$ **then** $Id := C.getLabel(i)$
  elsif $\vec{y} = \vec{x}$
      then $Id := D.Add(\vec{x})$
      else $Id := FixedPoint(\vec{y},D,GlobalSam,C)$
  fi
  foreach $i \in D.RangeQ(\delta,\vec{x})$
    do $C.setLabel(i,Id)$; $C.setClustered(i)$ **od**
  Id.
proc $SiteDECluster(D,H,C) \equiv$
  $SiteNegotiate(H,\vec{\tau},\vec{z}_1,\vec{z}_2,K,h)$
  $Send(H, Sample(D,\vec{\tau},\vec{z}_1,\vec{z}_2,K,h))$
  $GlobalSam := Receive(H)$
  for $i := 1$ **to** $D.count$ **do**
    if $\neg C.clustered(i)$
      then $FixedPoint(\vec{x}_i,D,GlobalSam,C)$ **fi**
  od.
proc $Helper(L) \equiv$
  $HelperNegotiate(L)$
  for $j := 1$ **to** $L.count$ **do**
    $Sam := Receive(L.get(j))$
    foreach $\vec{z} \in Rectangle(Sam.getCorners())$
      do $GlobalSam.Sum(\vec{z}, Sam.get(\vec{z}))$ **od**
  od
  for $j := 1$ **to** $L.count$ **do** $Send(GlobalSam, L.get(j))$ **od**.

the window width h. The negotiation procedures contain appropriate handshaking primitives to ensure that all sites participate and exit negotiations only if an agreement has been reached. Each local site computes the sampled form of the estimate of D, by calling function *Sample*, and sends it to the helper. (Function *DE* computes the density estimate and is omitted for brevity.) The helper receives the sampled estimates, sums them by sampling indexes into a global sample, and returns it to all sites. Procedures *Send* and *Receive* implement appropriate blocking and handshaking to ensure the transmission takes place. Each local site uses the global sample in functions *FixedPoint* and *Uphill* to compute the values of the gradient of the global density estimate. Function *SeriesGradient* can be easily derived from (5). Local sites perform a DE-cluster algorithm to compute the corresponding local data clusters. The details of the hill-climbing strategy are however different from Algorithm 1 because the sites are allowed access to local data objects only. *Uphill* advances a fraction δ of the gradient in its direction, if the gradient's norm exceeds a threshold ε. If a δ-neighbourhood of the object returned by *Uphill* contains an already clustered data object, the current cluster label *Id* is set from that object's label. Otherwise, checking whether *Uphill* returned the same space object signals to *FixedPoint* that the proximity of the local maximum has been reached. The maximum is marked by adding the current space object $\vec{x}$ as a dummy object to the local data set; this ensures that subsequent paths converging to the same local maximum will use the same cluster label as the current path. Method $D.Add(\cdot)$ returns the identifier of the added object, which is used as current cluster label. If neither case holds, the label *Id* is obtained by a recursive call. Finally, all objects in a small neighbourhood of the current object are labeled by *Id*. Note that adding dummy objects has effect only on the range queries, and does not modify the density estimate.

4.3 Complexity of the KDEC scheme

In terms of the complexity in computation and communication one crucial point of the KDEC scheme is how many samples have to be computed and transferred among the sites. In most cases, to obtain good density estimates, h must not be less than a small multiple of the smallest object distance. As $\tau^{(i)} \simeq h/2$, the number of samples should rarely exceed the number of objects, if only space regions where the density estimate is not negligible are sampled. Since the size of a sample is usually much smaller than the size of a data object, the overall communication costs of our DDC approach will be in most cases significantly lower than in a centralized approach. Of course, the precise number of samples depends on the bounding region that is being sampled by every site. In Algorithm 2 the site L_j determines autonomosly the rectangle that contains the computed samples.

The computational costs of the KDEC scheme in terms of used CPU cycles and I/O do not exceed the one in the centralized approach where clustering is performed on data collected in a single repository. The computational complexity is linear in the number of samples. The precise cost of computation of any KDEC-based DDC algorithm as an instance of the proposed scheme largely depends also on the used kernel function and local clustering algorithm. The DE-cluster algorithm we developed for the KDEC scheme in Section 3.2 is of complexity $O(Nq(N))$, where $q(N)$ is the cost of a nearest neighbour query (which in practical cases is close to $\log N$). Algorithm 2 implements a slightly different approach in the hill-climbing function than Algorithm 1, since the function does not use data objects to direct the uphill path. However, preliminary results of experiments conducted on a prototype implementation show good scalability of the approach, in terms of number of executed range queries.

5 Conclusion

Due to the explosion in the number of autonomous data sources there is a growing need for effective approaches to distributed knowledge discovery and data mining. In this paper we have presented KDEC, a novel scheme for distributed data clustering which computes the density estimation, to per-

form the clustering, from sampled forms of local densities at each data source site.

The approach exploits statistical density estimation and information theoretic sampling to minimize communications between sites. Moreover, the privacy of data is preserved to a large extent by never transmitting data values but kernel based density estimation samples outside the site of origin. The approach does not require CPU and I/O costs significantly higher than a similar centralized approach and its communication costs may be lower. Ongoing research focuses in particular on implementations of a multiagent system for KDEC-based distributed data clustering in a peer-to-peer network, and investigation on methods to mitigate the risk of security and privacy violations in distributed data mining environments.

References

[Ankerst *et al.*, 1999] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: Ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 49–60, Philadephia, PA, June 1999.

[Chen *et al.*, 1996] Ming-Syan Chen, Jiawei Han, and Philip S. Yu. Data mining: an overview from a database perspective. *IEEE Trans. On Knowledge And Data Engineering*, 8:866–883, 1996.

[Ester *et al.*, 1996] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231, Portland, OR, 1996.

[Ester *et al.*, 1998] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, and Xiaowei Xu. Incremental clustering for mining in a data warehousing environment. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB'98)*, pages 323–333, New York City, NY, August 1998.

[Fayyad *et al.*, 1996] Usama Fayyad, Gregory Piatetsky-Shapiro, Padrhaic Smyth, and Ramasamy Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. AAAI Press/MIT Press, 1996.

[Ganti *et al.*, 1999] Venkatesh Ganti, Raghu Ramakrishnan, Johannes Gehrke, Allison L. Powell, and James C. French. Clustering large datasets in arbitrary metric spaces. In *Proceedings of the 15th International Conference on Data Engineering (ICDE 1999)*, pages 502–511, Sydney, Australia, March 1999.

[Grabmeier and Rudolph, 2002] J. Grabmeier and A. Rudolph. Techniques of cluster algorithms in data mining. *Data Mining and Knowledge Discovery*, 6:303–360, 2002.

[Higgins, 1996] J. R. Higgins. *Sampling Theory in Fourier and Signal Analysis*. Clarendon Press, Oxford, 1996.

[Hinneburg and Keim, 1998] Alexander Hinneburg and Daniel A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pages 58–65, New York City, New York, USA, 1998. AAAI Press.

[Johnson and Kargupta, 1999] Erik Johnson and Hillol Kargupta. Collective, hierarchical clustering from distributed heterogeneous data. In M. Zaki and C. Ho, editors, *Large-Scale Parallel KDD Systems*, Lecture Notes in Computer Science, pages 221–244. Springer-Verlag, 1999.

[Kargupta *et al.*, 2000] H. Kargupta, B. Park, D. Hershberger, and E. Johnson. *Advances in Distributed and Parallel Knowledge Discovery*, chapter 5, Collective Data Mining: A New Perspective Toward Distributed Data Mining, pages 131–178. AAAI/MIT Press, 2000.

[Kargupta *et al.*, 2001] H. Kargupta, W. Huang, S. Krishnamoorthy, and E. Johnson. Distributed clustering using collective principal component analysis. *Knowledge and Information Systems Journal*, 3(4):422–448, 2001.

[Moro and Sartori, 2001] Gianluca Moro and Claudio Sartori. Incremental maintenance of multi-source views. In M. E. Orlowska and J. Roddick, editors, *Proceedings of the 12th Australasian conference on Database technologies, ADC 2001*, ACM International Conference Proceeding, pages 13–20, Queensland, Australia, 2001. IEEE Computer Society Press.

[Parzen, 1962] E. Parzen. On estimation of a probability density function and mode. *Ann. Math. Statist.*, 33:1065–1076, 1962.

[Schikuta, 1996] Erich Schikuta. Grid-clustering: An efficient hierarchical clustering method for very large data sets. In *Proceedings of the 13th International Conference on Pattern Recognition*, pages 101–105. IEEE, 1996.

[Silverman, 1986] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London, 1986.

[Zhang *et al.*, 1996] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An efficient data clustering method for very large darabases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103–114, Montreal, Canada, June 1996.