Incremental Clustering for Mining in a Data Warehousing Environment

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, Xiaowei Xu
Institute for Computer Science, University of Munich
Oettingenstr. 67, D-80538 München, Germany
email: {ester | kriegel | sander | wimmerm | xwxu}@informatik.uni-muenchen.de

Abstract

Data warehouses provide a great deal of opportunities for performing data mining tasks such as classification and clustering. Typically, updates are collected and applied to the data warehouse periodically in a batch mode, e.g., during the night. Then, all patterns derived from the warehouse by some data mining algorithm have to be updated as well. Due to the very large size of the databases, it is highly desirable to perform these updates incrementally. In this paper, we present the first incremental clustering algorithm. Our algorithm is based on the clustering algorithm DBSCAN which is applicable to any database containing data from a metric space, e.g., to a spatial database or to a WWW-log database. Due to the density-based nature of DBSCAN, the insertion or deletion of an object affects the current clustering only in the neighborhood of this object. Thus, efficient algorithms can be given for incremental insertions and deletions to an existing clustering. Based on the formal definition of clusters, it can be proven that the incremental algorithm yields the same result as DBSCAN. A performance evaluation of IncrementalDBSCAN on a spatial database as well as on a WWW-log database is presented, demonstrating the efficiency of the proposed algorithm. IncrementalDBSCAN yields significant speed-up factors over DBSCAN even for large numbers of daily updates in a data warehouse.

1 Introduction

Many companies have recognized the strategic importance of the knowledge hidden in their large databases and, therefore, have built data warehouses. A *data warehouse* is a collection of data from multiple sources, integrated into a common repository and extended by summary information (such as aggregate views) for the purpose of analysis [MQM 97]. When speaking of a data warehousing environment, we do not anticipate any special architecture but we address an environment with the following two characteristics:

(1) Derived information is present for the purpose of analysis.
(2) The environment is dynamic, i.e. many updates occur.

In such an environment, either manual analyses supported by appropriate visualization tools or (semi)automatic data mining may be performed. *Data mining* has been defined as the application of data analysis and discovery algorithms that - under acceptable computational efficiency limitations - produce a particular enumeration of patterns over the data [FPS 96]. Several data mining tasks have been identified [FPS 96], e.g., clustering, classification and summarization. Typical results of data mining are as follows:

- Clusters of items which are typically bought together by some set of customers (clustering in a data warehouse storing sales transactions).
- Symptoms distinguishing disease A from disease B (classification in a medical data warehouse).
- Description of the typical WWW access patterns (summarization in the data warehouse of an internet provider).

The task considered in this paper is *clustering* [KR 90], i.e. grouping the objects of a database into meaningful subclasses. Recently, several clustering algorithms for mining in large databases have been developed [NH 94], [ZRL 96], [EKSX 96].

Typically, a data warehouse is not updated immediately when insertions and deletions on the operational databases occur. Updates are collected and applied to the data warehouse periodically in a batch mode, e.g., each night [MQM 97]. Then, all patterns derived from the warehouse by data mining algorithms have to be updated as well. This update must be efficient enough to be finished when the warehouse has to be available for users again, e.g., the next morning. Due to the very large size of the databases, it is highly desirable to perform these updates incrementally ([FAAM 97], [Huy 97]), so as to consider only the old clus-

---

Proceedings of the 24th VLDB Conference
New York, USA, 1998

ters and the objects inserted or deleted during the day, instead of applying the clustering algorithm to the (very large) updated database.

Maintenance of derived information such as views and summary tables has been an active area of research [MQM 97], [Huy 97]. The problem of incrementally updating mined patterns on changes of the database, however, has just recently started to receive more investigation. [CHNW 96] and [FAAM 97] propose efficient methods for incrementally modifying a set of association rules mined from a database. [EW 98] introduces generalization algorithms for incremental summarization in a data warehousing environment.

In this paper, we present the first incremental clustering algorithm. Our algorithm is based on DBSCAN [EKSX 96], [SEKX 98] which is an efficient clustering algorithm for metric databases (that is, databases with a distance function for pairs of objects) for mining in a data warehousing environment. Due to the density-based nature of DBSCAN, the insertion or deletion of an object affects the current clustering only in the neighborhood of this object. We demonstrate the high efficiency of incremental clustering on a spatial database [Gue 94] as well as on a WWW access log database [MJHS 96].

The rest of this paper is organized as follows. We discuss related work on clustering algorithms in section 2. In section 3, we briefly introduce the clustering algorithm DBSCAN. The algorithms for incrementally updating a clustering on insertions and deletions of the database are presented in section 4 and an extensive performance evaluation is reported in section 5. Section 6 concludes with a summary and some directions for future research.

2 Related Work

The problem of incrementally updating mined patterns after making changes to the database has just recently started to receive more attention.

The task of mining association rules has been introduced by [AS 94]. An *association rule* is a rule $I_1 \Rightarrow I_2$ where I_1 and I_2 are disjoint subsets of a set of items I. For a given database *DB* of transactions (i.e. each record contains a set of items bought by some customer in one transaction), all association rules should be discovered having a support of at least *minsupport* and a confidence of at least *minconfidence* in *DB*. The subsets of I that have at least *minsupport* in *DB* are called *frequent sets*.

[FAAM 97] describes two typical scenarios for mining association rules in a dynamic database. For example, in a medical database, one may seek associations between treatments and results. The database is constantly updated and at any given time, the medical researcher is interested in obtaining the current associations. In a database containing news articles, e.g., patterns of co-occurrence amongst the topics of articles may be of interest. An economic analyst receives a lot of new articles every day and he would like to find relevant associations based on all current articles.

[CHNW 96] proposes to apply a non-incremental algorithm for mining association rules to the newly inserted database objects, i.e. to the increment of the database, and then to combine the frequent sets of both the database and the increment. The incremental algorithms presented in [FAAM 97] are based on information about the frequency of attribute pairs and border sets respectively. While the space overhead for keeping track of these frequencies is small, the incremental algorithms yield a speed-up of several orders of magnitude compared to the non-incremental algorithm.

Summarization, e.g., by generalization, is another important task of data mining. A*ttribute-oriented generalization* [HCC 93] of a relation is the process of replacing the attribute values by a more general value, one attribute at a time, until the number of tuples of the relation becomes less than a specified threshold. The more general value is taken from a concept hierarchy which is typically available for most attributes in a data warehouse.

[EW 98] presents algorithms for incremental attribute-oriented generalization with the conflicting goals of good efficiency and minimal overly generalization. The algorithms for incremental insertions and deletions are based on the materialization of a relation at an intermediate generalization level, i.e. the anchor relation. Experiments demonstrate that incremental generalization can be performed efficiently at a low degree of overly generalization.

This paper focuses on the data mining task of clustering and, in the following, we review clustering algorithms from a data mining perspective.

Partitioning algorithms construct a partition of a database *DB* of *n* objects into a set of *k* clusters where *k* is an input parameter. Each cluster is represented by the center of gravity of the cluster (*k*-means) or by one of the objects of the cluster located near its center (*k*-medoid) [KR 90] and each object is assigned to the cluster with its representative closest to the considered object. Typically, partitioning algorithms start with an initial partition of *DB* and then use an iterative control strategy to optimize the clustering quality, e.g., the average distance of an object to its representative.

[NH 94] explores partitioning algorithms for mining in spatial databases. An algorithm called *CLARANS (Clustering Large Applications based on RANdomized Search)* is introduced which is more effective and more efficient than previous partitioning algorithms.

Hierarchical algorithms create a hierarchical decomposition of *DB*. The hierarchical decomposition is represented by a *dendrogram*, a tree that iteratively splits *DB* into smaller subsets until each subset consists of only one object. In such a hierarchy, each level of the tree represents a clustering of *DB*.

The basic hierarchical clustering algorithm works as follows ([Sib 73], [Bou 96]). Initially, each object is placed in a unique cluster. For each pair of clusters, some value of dissimliarity or distance is computed. For instance, the distance may be the minimum distance of all pairs of points from the two clusters (*single-link method*). [Bou 96] discusses alternative definitions of the distance and shows that, in general, no one approach outperforms any other in terms of clustering quality. In every step, the clusters with the minimum distance in the current clustering are merged until all points are contained in one cluster.

None of the above algorithms is efficient on large databases. Therefore, some focusing techniques have been proposed to increase the efficiency of clustering algorithms.

[EKX 95] presents an R*-tree based focusing technique (1) creating a sample of the database that is drawn from each R*-tree data page and (2) applying the clustering algorithm only to that sample. [ZRL 96] proposes a special data structure to condense information about subclusters of points. A *Clustering Feature (CF)* is a triple that contains the number of points, the linear sum and the square sum of all points in the cluster. Clustering features are organized in a height balanced tree, i.e. the CF-tree. *BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies)* [ZRL 96] is a CF-tree based multiphase clustering method. First, the database is scanned to build an initial in-memory CF-tree. In an optional second phase, this CF-tree can be further reduced until a desired number of leaf nodes is reached. In phase 3 an arbitrary clustering algorithm is used to cluster the CF-values stored in the leaf nodes of the CF-tree. Note that the CF-tree is an incremental structure but phase 3 of BIRCH is non-incremental.

Recently, a new type of single scan clustering algorithms has been introduced. The basic idea of a *single scan algorithm* is to group neighboring objects of the database into clusters based on a local cluster condition, thus performing only one scan through the database. Single scan clustering algorithms are very efficient if the retrieval of the neighborhood of an object is efficiently supported by the DBMS. Different cluster conditions yield different cluster definitions and algorithms. For instance, *DBSCAN (Density Based Spatial Clustering of Applications with Noise)* [EKSX 96] [SEKX 98] relies on a density-based notion of clusters.

We use DBSCAN as a base for our incremental clustering algorithm due to the following reasons. First, DBSCAN is one of the most efficient algorithms on large databases. Second, whereas BIRCH is applicable only to spatial databases (Euclidean vector space), DBSCAN can be applied to any database containing data from a metric space (only assuming a distance function).

3 The Algorithm DBSCAN

The key idea of density-based clustering is that for each object of a cluster the neighborhood of a given radius (*Eps*) has to contain at least a minimum number of objects (*MinPts*), i.e. the cardinality of the neighborhood has to exceed some threshold.

We will first give a short introduction to DBSCAN including the definitions which are required for incremental clustering. For a detailed presentation of DBSCAN see [EKSX 96].

Definition 1: (directly density-reachable) An object p is *directly density-reachable* from an object q wrt. *Eps* and *MinPts* in the set of objects D if

1) $p \in N_{Eps}(q)$ ($N_{Eps}(q)$ is the subset of D contained in the *Eps*-neighborhood of q.)

2) $Card(N_{Eps}(q)) \geq MinPts$.

Definition 2: (density-reachable) An object p is *density-reachable* from an object q wrt. *Eps* and *MinPts* in the set of objects D, denoted as $p >_D q$, if there is a chain of objects $p_1, ..., p_n$, $p_1 = q$, $p_n = p$ such that $p_i \in D$ and p_{i+1} is directly density-reachable from p_i wrt. *Eps* and *MinPts*.

Density-reachability is a canonical extension of direct density-reachability. This relation is transitive, but it is not symmetric. Although not symmetric in general, it is obvious that density-reachability is symmetric for objects o with $Card(N_{Eps}(o)) \geq MinPts$. Two "border objects" of a cluster are possibly not density-reachable from each other because there are not enough objects in their *Eps*-neighborhoods. However, there must be a third object in the cluster from which both "border objects" are density-reachable. Therefore, we introduce the notion of density-connectivity.

Definition 3: (density-connected) An object p is *density-connected* to an object q wrt. *Eps* and *MinPts* in the set of objects D if there is an object $o \in D$ such that both p and q are density-reachable from o wrt. *Eps* and *MinPts* in D.

Density-connectivity is a symmetric relation. Figure 1 illustrates the definitions on a sample database of objects from a 2-dimensional *vector* space. Note however, that the above definitions only require a distance measure and will also apply to data from a metric space.

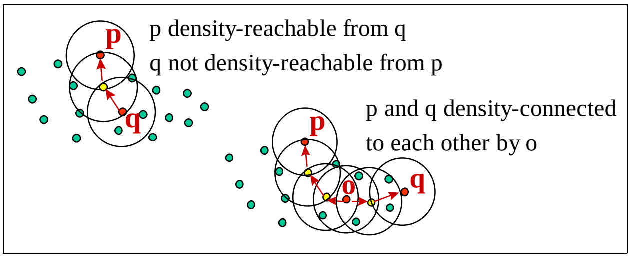

Figure 1: : density-reachability and density-connectivity

A *cluster* is defined as a set of density-connected objects which is maximal wrt. density-reachability and the *noise* is the set of objects not contained in any cluster.

Definition 4: (cluster) Let D be a set of objects. A *cluster* C wrt. *Eps* and *MinPts* in D is a non-empty subset of D satisfying the following conditions:

1) Maximality: $\forall p,q \in D$: if $p \in C$ and $q >_D p$ wrt. *Eps* and *MinPts*, then also $q \in C$.

2) Connectivity: $\forall p,q \in C$: p is density-connected to q wrt. *Eps* and *MinPts in D*.

Definition 5: (noise) Let $C_1, \ldots, C_k$ be the clusters wrt. *Eps* and *MinPts* in D. Then, we define the *noise* as the set of objects in the database D not belonging to any cluster C_i, i.e. $noise = \{p \in D \mid \forall\ i: p \notin C_i\}$.

We omit the term "wrt. *Eps* and *MinPts*" in the following whenever it is clear from the context. There are two different kinds of objects in a clustering: *core objects* (satisfying condition 2 of definition 1) and *non-core objects* (otherwise). In the following, we will refer to this characteristic of an object as the *core object property* of the object. The non-core objects in turn are either *border objects* (not a core object but density-reachable from another core object) or *noise objects* (not a core object and not density-reachable from other objects).

The algorithm DBSCAN was designed to efficiently discover the clusters and the noise in a database according to the above definitions. The procedure for finding a cluster is based on the fact that a cluster is uniquely determined by any of its core objects:

- First, given an arbitrary object *p* for which the core object condition holds, the set $\{o \mid o >_D p\}$ of all objects *o* density-reachable from *p* in *D* forms a complete cluster *C* and $p \in C$.
- Second, given a cluster *C* and an arbitrary core object $p \in C$, *C* in turn equals the set $\{o \mid o >_D p\}$ (c.f. lemma 1 and 2 in [EKSX 96]).

To find a cluster, DBSCAN starts with an arbitrary object *p* in *D* and retrieves all objects of *D* density-reachable from *p* with respect to *Eps* and *MinPts*. If *p* is a core object, this procedure yields a cluster with respect to *Eps* and *MinPts*. If *p* is a border object, no objects are density-reachable from *p* and *p* is assigned to the noise. Then, DBSCAN visits the next object of the database *D*.

The retrieval of density-reachable objects is performed by successive region queries. A *region query* returns all objects intersecting a specified query region. Such queries are supported efficiently by spatial access methods such as R*-trees [BKSS 90] for data from a vector space or M-trees [CPZ 97] for data from a metric space.

The algorithm DBSCAN is sketched in figure 2.

```
Algorithm DBSCAN (D, Eps, MinPts)
// Precondition: All objects in D are unclassified.
  FORALL objects o in D DO:
    IF o is unclassified
      call function expand_cluster to construct a cluster wrt.
       Eps and MinPts containing o.

FUNCTION expand_cluster (o, D, Eps, MinPts):
  retrieve the Eps-neighborhood N_Eps(o) of o;
  IF | N_Eps(o) | < MinPts    // i.e. o is not a core object
      mark o as noise and RETURN;
  ELSE // i.e. o is a core object
      select a new cluster-id and mark all objects in N_Eps(o)
      with this current cluster-id;
      push all objects from N_Eps(o)\{o} onto the stack seeds;
      WHILE NOT seeds.empty() DO
          currentObject := seeds.top();
          retrieve the Eps-neighborhood N_Eps(currentObject)
          of currentObject;
          IF | N_Eps(currentObject) | ≥ MinPts
              select all objects in N_Eps(currentObject) not yet
              classified or are marked as noise,
              push the unclassified objects onto seeds
               and mark all of these objects with current
               cluster-id;
          seeds.pop();
  RETURN
```

Figure 2: : Algorithm DBSCAN

4 IncrementalDBSCAN

DBSCAN, as introduced in [EKSX 96], is applied to a static database. In a data warehouse, however, the databases may have frequent updates and thus may be rather dynamic. For example, in a WWW access log database, we may want to find and monitor groups of similar access patterns by clustering the access sequences of different users. These patterns may change over time because each day new log-entries are added to the database and old entries (past a user-supplied expiration date) are deleted. After insertions and deletions to the database, the clustering discovered by DBSCAN has to be updated. In section 4.1, we examine which part of an existing clustering is affected by an update of the database. We present algorithms for incremental updates of a clustering after insertions (section 4.2) and deletions (section 4.3). Based on the formal notion of clusters, it can be proven that the incremental algorithm yields the same result as the non-incremental DBSCAN algorithm. This is an important advantage of our approach.

4.1 Affected Objects

We want to show that changes of some clustering of a database *D* are restricted to a neighborhood of an inserted or deleted object *p*. Objects contained in $N_{Eps}(p)$ can change their core object property, i.e. core objects may become non-core objects and vice versa. The objects contained in $N_{2Eps}(p) \setminus N_{Eps}(p)$ keep their core object property, but non-core objects may change their connection status, i.e. border objects may become noise objects or vice versa, because their *Eps*-neighborhood may contain objects with a changed core object property. For all objects outside of $N_{2Eps}(p)$, it holds that neither these objects themselves nor objects in their *Eps*-neighborhood change their core object property. Therefore, the connection status of these objects is unchanged.

After the insertion of some object *p*, non-core objects (border objects or noise objects) in $N_{Eps}(p)$ may become core objects implying that new density connections may be established, i.e. chains $p_1, ..., p_n$, $p_1 = r$, $p_n = s$ with p_{i+1} directly density-reachable from p_i for two objects *r* and *s* may arise which were not density-reachable from each other before the insertion. Then, one of the p_i *for* $i < n$ must be contained in $N_{Eps}(p)$.

When deleting some object *p*, core objects in $N_{Eps}(p)$ may become non-core objects implying that density connections may be removed, i.e. there may no longer be a chain $p_1, ..., p_n$, $p_1 = r$, $p_n = s$ with p_{i+1} directly density-reachable from p_i for two objects *r* and *s* which were density-reachable from each other before the deletion. Again, one of the p_i for $i < n$ must be contained in $N_{Eps}(p)$.

Figure 3 illustrates our discussion using a sample database of 2D objects and an object *p* to be inserted or to be deleted. The objects *a* and *b* are density connected wrt. *Eps* as depicted and *MinPts* = 4 without using one of the elements of $N_{Eps}(p)$. Therefore, *a* and *b* belong to the same cluster independently from *p*. On the other hand, the objects *d* and *e* in $D \setminus N_{Eps}(p)$ are only density-connected via *c* in $N_{Eps}(p)$ if the object p is present, so that the cluster membership of d and e is affected by p.

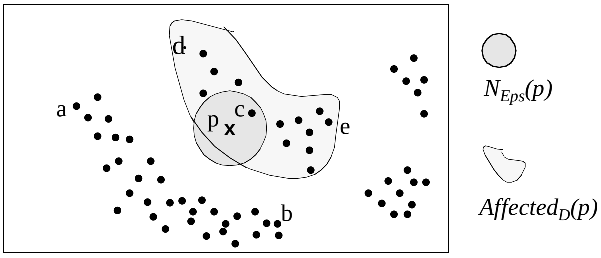

Figure 3: : Affected objects in a sample database

In general, on an insertion or deletion of an object p, the set of *affected objects*, i.e. objects which may potentially change cluster membership after the update, is the set of objects in $N_{Eps}(p)$ plus all objects density-reachable from one of these objects in $D \cup \{p\}$. The cluster membership of all other objects not in the set of *affected objects* will not change. This is the intuition of the following definition and lemma. In particular, the lemma states that a cluster c in the database is independent of an insertion or deletion of an object p if a core object of the cluster is outside the set $Affected_D(p)$. Note that a cluster is uniquely determined by any of its core objects. Therefore, by definition of $Affected_D(p)$ it follows that if one core object of a cluster is outside (inside) $Affected_D(p)$ then *all* core objects of the cluster are outside (inside) the set $Affected_D(p)$.

Definition 6: (affected objects) Let D be a database of objects and p be some object (either in or not in D). We define the set of objects in D affected by the insertion or deletion of p as

$$Affected_D(p) = N_{Eps}(p) \cup \{q \mid \exists\, o \in N_{Eps}(p) \wedge q >_{D \cup \{p\}} o\}.$$

Lemma 1: Let D be a set of objects and p be some object. Then $\forall\, o \in D$: $o \notin Affected_D(p) \Rightarrow \{q \mid q >_{D \setminus \{p\}} o\} = \{q \mid q >_{D \cup \{p\}} o\}$.

Proof (sketch): 1) $\subseteq$: because $D \setminus \{p\} \subseteq D \cup \{p\}$. 2) $\supseteq$: if $q \in \{q \mid q >_{D \cup \{p\}} o\}$, *then there is some chain* $q_1, ..., q_n$, $q_1 = o$, $q_n = q$, $q_{i+1} \in N_{Eps}(q_i)$ and q_i is a core object in $D \cup \{p\}$ for all $i < n$ and, for all i, it holds that $q_i >_{D \cup \{p\}} o$. Because q_i is a core object for all $i < n$ and density-reachability is symmetric for core objects, it also holds that $o >_{D \cup \{p\}} q_i$. If there existed an $i < n$ such that $q_i \in N_{Eps}(p)$, then $q_i >_{D \cup \{p\}} p$ implying also $o >_{D \cup \{p\}} p$ due to the transitivity of density-reachability. By definition of the set $Affected_D(p)$ it now follows that $o \in Affected_D(p)$, in contrast to the assumption. Thus, $q_i \notin N_{Eps}(p)$ for all $i < n$ implying that all the objects q_i, $i < n$, are core objects independent of p and also $q_n \neq p$ because otherwise $q_{n-1} \in N_{Eps}(p)$. Thus, the chain $q_1, ..., q_n$ exists also in the set $D \setminus \{p\}$ and then $q \in \{q \mid q >_{D \setminus \{p\}} o\}$. □

Due to lemma 1, after inserting or deleting an object p, it is sufficient to reapply DBSCAN to the set $Affected_D(p)$ in order to update the clustering. For that purpose, however, it is not necessary to retrieve the set first and *then* apply the clustering algorithm. We simply have to start a restricted version of DBSCAN which does not loop over the whole database to start expanding a cluster but only over certain "seed"-objects which are all located in the neighborhood of p. These "seed"-objects are core objects *after* the update operation which are located in the *Eps*-neighborhood of a core object in $D \cup \{p\}$ which in turn is located in $N_{Eps}(p)$. This is the content of the next lemma.

Lemma 2: Let D be a set of objects. Additionally, let $D^* = D \cup \{p\}$ after insertion of an object p or $D^* = D \setminus \{p\}$ after deletion of p and let c be a core object in D^*.
$C = \{o \mid o >_{D^*} c\}$ is a cluster in D^* and $C \subseteq Affected_D(p) \Leftrightarrow \exists\, q, q'$: $q \in N_{Eps}(q')$, $q' \in N_{Eps}(p)$, $c >_{D^*} q$, q is core object in D^* and q' is core object in $D \cup \{p\}$.

Proof (sketch): If $D^* = D \cup \{p\}$ or $c \in N_{Eps}(p)$, the lemma is obvious by definition of $Affected_D(p)$. Therefore, we consider only the case $D^* = D \setminus \{p\}$ and $c \notin N_{Eps}(p)$.
"=>": $C \subseteq Affected_D(p)$ and $C \neq \emptyset$. Then, there exists $o \in N_{Eps}(p)$ and $c >_{D \cup \{p\}} o$, i.e. there is a chain of directly density-reachable objects from o to c. Now, because $c \notin N_{Eps}(p)$ we can construct a chain $o = o_1, \ldots, o_n = c$, $o_{i+1} \in N_{Eps}(o_i)$ with the property that there is $j \leq n$ such that for all k, $j \leq k \leq n$, $o_k \notin N_{Eps}(p)$ and for all k, $1 \leq k < j$, $o_k \in N_{Eps}(p)$. Then $q = o_j \in N_{Eps}(o_{j-1})$, $q' = o_{j-1} \in N_{Eps}(p)$, $c >_{D^*} o_j$, o_j is a core object in D^* and o_{j-1} is a core object in $D \cup \{p\}$.
"<=": obviously, $C = \{o \mid o >_{D^*} c\}$ is a cluster (see the comments on the algorithm after definition 5). By assumption, c is density-reachable from a core object q in D^* and q is density-reachable from an object $q' \in N_{Eps}(p)$ in $D \cup \{p\}$. Then also c and hence all objects in C are density-reachable from q' in $D \cup \{p\}$. Thus, $C \subseteq Affected_D(p)$. □

Due to lemma 2, the general strategy for updating a clustering would be to start the DBSCAN algorithm only with core objects that are in the *Eps*-neighborhood of a (previous) core object in $N_{Eps}(p)$. However, it is not necessary to rediscover density-connections which are known from the previous clustering and which are not changed by the update operation. For that purpose, we only need to look at core objects in the *Eps*-neighborhood of those objects that change their core object property as a result of the update. In case of an insertion, these objects may be connected after the insertion. In case of a deletion, density connections between them may be lost. In general, this information can be determined by using very few region queries. The remaining information needed to adjust the clustering can be derived from the cluster membership before the update. Definition 7 introduces the formal notions which are necessary to describe this approach. Remember: objects with a changed core object property are all located in $N_{Eps}(p)$.

Definition 7: (seed objects for the update) Let D be a set of objects and p be an object to be inserted or deleted. Then, we define the following notions:

$UpdSeed_{Ins} = \{q \mid q$ is a core object in $D \cup \{p\}$,
$\exists\, q'$: q' is core object in $D \cup \{p\}$ but not in D
and $q \in N_{Eps}(q')\}$

$UpdSeed_{Del} = \{q \mid q$ is a core object in $D \setminus \{p\}$,
$\exists\, q'$: q' is core object in D but not in $D \setminus \{p\}$
and $q \in N_{Eps}(q')\}$

We call the objects $q \in UpdSeed$ "seed objects for the update". Note that these sets can be computed rather efficiently if we additionally store for each object the *number* of objects in its neighborhood when initially clustering the database. Then, we need only to perform a single region query for the object *p* to be inserted or deleted to detect all objects *q'* with a changed core object property (i.e. objects in $N_{Eps}(p)$ with *number* = *MinPts*-1 in case of an insertion, objects in $N_{Eps}(p)$ with *number* = *MinPts* in case of a deletion). Only for these objects *q'* (if there are any) do we have to retrieve $N_{Eps}(q')$ to determine all objects *q* in the set *UpdSeed*. Since at this point of time the *Eps*-neighborhood of *p* is still in main memory we first check this set for neighbors of *q'* and perform an additional region query only if there are more objects in the neighborhood of *q'* than already contained in $N_{Eps}(p)$. Our experiments, however, indicate that objects with a changed core object property after an update (different from the inserted or deleted object *p*) are not very frequent (see section 5). Therefore, in most cases we just have to perform the *Eps*-neighborhood query for *p* and to change the counter for the number of objects in the neighborhood of the retrieved objects.

4.2 Insertions

When inserting a new object *p*, new density-connections may be established, but none are removed. In this case, it is sufficient to restrict the application of the clustering procedure to the set $UpdSeed_{Ins}$. If we have to change cluster membership for an object from *C* to *D* we perform the same change of cluster membership for all other objects in *C*. Changing cluster membership of these objects does not involve the application of the clustering algorithm but can be handled by simply storing the information about which clusters have been merged.

When inserting an object *p* into the database *D*, we can distinguish the following cases:

(1) (*Noise*)
$UpdSeed_{Ins}$ is empty, i.e. there are no "new" core objects after insertion of *p*. Then, *p* is a noise object and nothing else is changed.

(2) (Creation)
$UpdSeed_{Ins}$ contains only core objects which did not belong to a cluster before the insertion of *p*, i.e. they were noise objects or equal to *p*, and a new cluster containing these noise objects as well as *p* is created.

(3) (Absorption)
$UpdSeed_{Ins}$ contains core objects which were members of exactly one cluster *C* before the insertion. The object *p* and possibly some noise objects are absorbed into cluster *C*.

(4) (Merge)
$UpdSeed_{Ins}$ contains core objects which were members of several clusters before the insertion. All these clusters and the object *p* are merged into one cluster.

Figure 4 illustrates the most simple forms of the different cases when inserting an object *p* into a sample database of 2D points, using parameters *Eps* as depicted and *MinPts*=3.

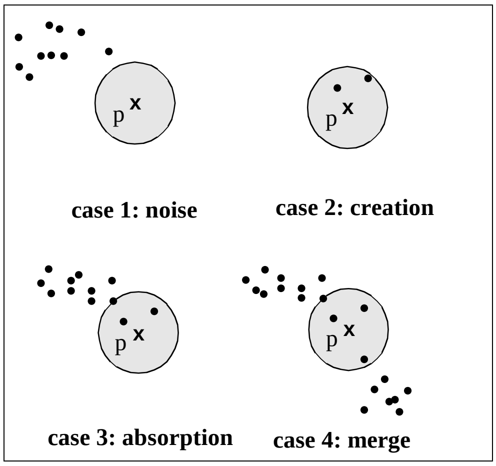

Figure 4: : The different cases of the insertion algorithm

Figure 5 presents a more complicated example of merging clusters when inserting an object *p*. In this example the value for *Eps* is as depicted and *MinPts* = 6. Then, the inserted point *p* is not a core object, but o_1, o_2, o_3 and o_4 are core objects after the update. The previous clustering can be adapted by analyzing only the *Eps*-neighborhood of these objects: cluster A is merged with cluster B and C because o_1 and o_4 as well as o_2 and o_3 are mutual directly density-reachable, implying the merge of B and C. The changing of cluster membership for objects in case of merging clusters can be done very efficiently by simply storing the information about the clusters that have been merged. Note that this kind of "transitive" merging can only occur if *MinPts* is larger than 5, because otherwise *p* would be a core object and then all objects in $N_{Eps}(p)$ would already be density-reachable from *p*.

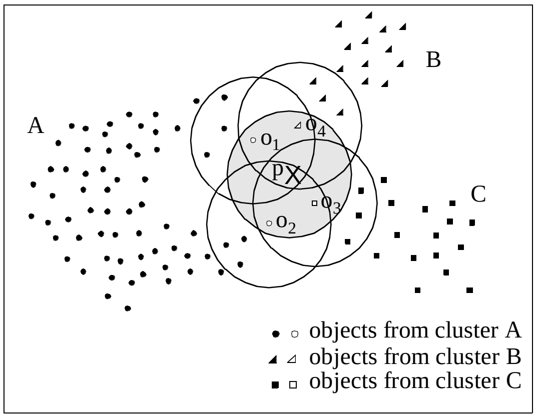

Figure 5: : "Transitive" merging of clusters A, B, C by the insertion algorithm

4.3 Deletions

As opposed to an insertion, when deleting an object *p*, density-connections may be removed, but no new connections are established. The difficult case for deletion occurs when the cluster *C* of *p* is no longer density-connected via (previous) core objects in $N_{Eps}(p)$ after deleting *p*. In this case, we do not know in general how many objects we have to check before it can be determined whether *C* has to be split or not. In most cases, however, this set of objects is very small because the split of a cluster is not very frequent and in general a non-split situation will be detected in a small neighborhood of the deleted object *p*.

When deleting an object *p* from the database *D* we can distinguish the following cases:

(1) (*Removal*)
$UpdSeed_{\mathrm{Del}}$ is empty, i.e. there are no core objects in the neighborhood of objects that may have lost their core object property after the deletion of *p*. Then *p* is deleted from *D* and eventually other objects in $N_{Eps}(p)$ change from a former cluster *C* to noise. If this happens, the cluster *C* is completely removed because then *C* cannot have core objects outside of $N_{Eps}(p)$.

(2) (*Reduction*)
All objects in $UpdSeed_{\mathrm{Del}}$ are *directly* density-reachable from each other. Then *p* is deleted from *D* and some objects in $N_{Eps}(p)$ may become noise.

(3) (*potential Split*)
The objects in $UpdSeed_{\mathrm{Del}}$ are *not* directly density-reachable from each other. These objects belonged to exactly one cluster *C* before the deletion of *p*. Now we have to check whether or not these objects are density-connected by other objects in the former cluster *C*. Depending on the existence of such density-connections, we can distinguish a *split* and a *non-split* situation.

Figure 6 illustrates the different cases when deleting *p* from a sample database of 2D points using parameters *Eps* as depicted and *MinPts* = 3. Note that the situations described in case 3 may occur simultaneously.

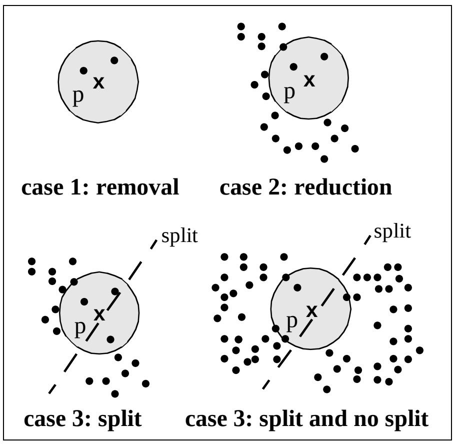

Figure 6: : The different cases of the deletion algorithm

If case (3) occurs, then the clustering procedure must also consider objects outside of $UpdSeed_{\mathrm{Del}}$, but it stops in case of a non-split situation as soon as the objects from the set $UpdSeed_{\mathrm{Del}}$ are density-connected to each other.

Case (3) is implemented by a procedure similar to the function *expand_cluster* in algorithm DBSCAN (see figure 2) starting in parallel from the elements of the set $UpdSeed_{\mathrm{Del}}$. The main difference is that the candidates for further expansion are managed in a queue instead of a stack. Thus, a breadth-first search for the missing density-connections is performed which is more efficient than a depth-first search due to the following reasons:

- In a non-split situation, we stop as soon as all members of $UpdSeed_{\mathrm{Del}}$ are found to be density-connected to each other. The breadth-first search implies that density-connections with the minimum number of objects (requiring the minimum number of region queries) are detected first.
- A split situation is in general the more expensive case because the parts of the cluster to be split actually have to be discovered. The algorithm stops when all but the last part have been visited. Usually, a cluster is split only into two parts and one of them is relatively small. Using breadth-first search we only have to visit the smaller part and a small percentage of the larger one.

5 Performance Evaluation

In this section, we evaluate the efficiency of IncrementalDBSCAN versus DBSCAN. We present an experimental evaluation using a 2D spatial database as well as a WWW access log database. For this purpose, we implemented both algorithms in C++ based on implementations of the R*-tree [BKSS 90] (for the 2D spatial database) and the M-tree [CPZ 97] (for the WWW log database) respectively. Furthermore, we present an analytical comparison of both algorithms and derive the speed-up factors for typical parameter values depending on the database size and the number of updates.

For the first set of experiments, we used a synthetic database of 1,000,000 2D points with $k = 40$ clusters of similar sizes. 21.7% of all points are noise, uniformly distributed outside of the clusters, and all other points are uniformly distributed inside the clusters with a significantly higher density than the noise. In this database, the goal of clustering is to discover groups of neighboring objects. A typical real world application for this type of database is clustering earthquake epicenters stored in an earthquake catalog. Earthquake epicenters occur along seismically active faults, and are measured with some errors, so that over time observed earthquake epicenters should be clustered along such seismic faults [AF 96].

In this type of application, there are only insertions. The Euclidean distance was used as distance function and an R*-tree [BKSS 90] as an index structure. *Eps* was set to 4.48 and *MinPts* was set to 30. Note that the *MinPts* value had to be rather large due to the high percentage of noise. We performed experiments on several other synthetic 2D databases with *n* varying from 100,000 to 1,000,000, *k* varying from 7 to 40 and with the noise percentage varying from 10% up to 20%. Since we always obtained similar results, we restrict the discussion to the above database.

```
romblon.informatik.uni-muenchen.de lopa - [04/Mar/1997:01:44:50 +0100] "GET /~lopa/ HTTP/1.0" 200 1364
romblon.informatik.uni-muenchen.de lopa - [04/Mar/1997:01:45:11 +0100] "GET /~lopa/x/ HTTP/1.0" 200 712
fixer.sega.co.jp unknown - [04/Mar/1997:01:58:49 +0100] "GET /dbs/porada.html HTTP/1.0" 200 1229
scooter.pa-x.dec.com unknown - [04/Mar/1997:02:08:23 +0100] "GET /dbs/kriegel_e.html HTTP/1.0" 200 1241
```

Figure 7: : Sample WWW access log entries

For the second set of experiments, we used a WWW access log database of the Institute for Computer Science of the University of Munich. This database contains 1,400,000 entries following the *Common Log Format* specified as part of the HTTP protocol [Luo 95]. Figure 7 depicts some sample log entries.

All log entries with identical IP address and user id within a given maximum time gap are grouped into a *session* and redundant entries, i.e. entries with filename suffixes such as "gif", "jpeg", and "jpg" are removed [MJHS 96]. A session has the following structure:

session::= <ip_address, user_id, [url_1, . . ., url_k]>

In this application, the goal of clustering is to discover groups of similar sessions. A WWW provider may use the discovered clusters as follows:

- The users associated with the sessions of a cluster form some kind of user group which may be used to develop marketing strategies.
- The URLs of the sessions contained in a cluster seem to be logically correlated and should be made easily accessible from each other via appropriate links.

Entries are deleted from the WWW access log database after six months. Assuming a constant daily number of WWW accesses, the numbers of insertions and deletions are the same. We used the following *distance function* for pairs of sessions s_1 and s_2:

$$dist(s_1, s_2) = \frac{Cardinality(s_1 \backslash s_2) + Cardinality(s_2 \backslash s_1)}{Cardinality(s_1) + Cardinality(s_2)}$$

The domain of *dist* is the interval [0 . . 1], *dist*(s,s) = 0, *dist* is symmetric and it fulfills the triangle inequality. Other distance functions may use the hierarchy of the directories to define the degree of similarity between two URLs. The database was indexed by an M-tree [CPZ 97]. *Eps* was set to 0.4 and *MinPts* to 2.

In the following, we compare the performance of IncrementalDBSCAN versus DBSCAN. Typically, the number of page accesses is used as a cost measure for database algorithms because the I/O time heavily dominates CPU time. In both algorithms, region queries are the only operations requiring page accesses. Since the number of page accesses of a single region query is the same for DBSCAN and for IncrementalDBSCAN, we only have to compare the number of region queries. Thus, we use the number of region queries as the cost measure for our comparison. Note that we are not interested in the absolute performance of the two algorithms but only in their relative performance, i.e. in the speed-up factor as defined below. To validate this approach, we performed a set of experiments on our test databases and found that the experimental speed-up factor always was slightly larger than the analytically derived speed-up factor (experimental value 1.6 times the expected value in all experiments).

DBSCAN performs exactly one region query for each of the *n* objects of the database (see algorithm in figure 2), i.e. the cost of DBSCAN for clustering *n* objects, denoted by $Cost_{DBSCAN}(n)$, is

$$Cost_{DBSCAN}(n) = n$$

The number of region queries performed by IncrementalDBSCAN depends on the application and, therefore, it must be determined experimentally. In general, a deletion affects more objects than an insertion. Thus, we introduce two parameters r_{ins} and r_{del} denoting the average number of region queries for an incremental insertion resp. deletion. Let f_{ins} and f_{del} denote the percentage of insertions resp. deletions in the number of all incremental updates. Then, the cost of IncrementalDBSCAN for performing *m* incremental updates, denoted by $Cost_{IncrementalDBSCAN}(m)$, is as follows:

$$Cost_{IncrementalDBSCAN}(m) = m \times (f_{ins} \times r_{ins} + f_{del} \times r_{del})$$

Table 1 lists the parameters of our performance evaluation and the values obtained for the 2D spatial as well as for the WWW-log database. To determine the average values

Table 1: Parameters of the performance evaluation

Parameter	Meaning	Value for 2D spatial	Value for WWW-log
n	number of database objects	1,000,000	69,000
m	number of (incremental) updates	varying	varying
r_{ins}	average number of region queries for an incremental insertion	1.58	1.1
r_{del}	average number of region queries for an incremental deletion	6.9	6.6
f_{del}	relative frequency of deletions in the number of all updates	0	0.5
f_{ins}	relative frequency of insertions in the number of all updates (1- f_{del})	1.0	0.5

(r_{ins} and r_{del}), the whole databases were incrementally inserted and deleted, although f_{del} = 0 for the 2D spatial database.

Now, we can calculate the speed-up factor of IncrementalDBSCAN versus DBSCAN. We define the *speed-up factor* as the ratio of the cost of DBSCAN (applied to the database after all insertions and deletions) and the cost of *m* calls of IncrementalDBSCAN (once for each of the insertions resp. deletions), i.e.:

$$SpeedupFactor = \frac{Cost_{DBSCAN}(n + f_{ins} \times m - f_{del} \times m)}{Cost_{IncrementalDBSCAN}(m)}$$

$$= \frac{(n + f_{ins} \times m - f_{del} \times m)}{m \times (f_{ins} \times r_{ins} + f_{del} \times r_{del})}$$

Figure 8 and figure 9 depict the speed-up factors depending on *n* for several values of *m*. For relatively small numbers of daily updates, e.g., *m* = 1,000 and *n* = 1,000,000, we obtain speed-up factors of 633 for the 2D spatial database and 260 for the WWW-log database. Even for rather large numbers of daily updates, e.g., *m* = 25,000 and *n* = 1,000,000, IncrementalDBSCAN yields speed-up factors of 26 and 10 for the 2D spatial as well as for the WWW-log database.

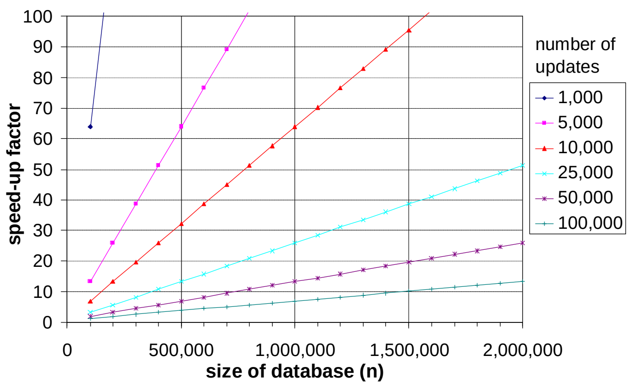

Figure 8: : Speed-up factors for 2D spatial database

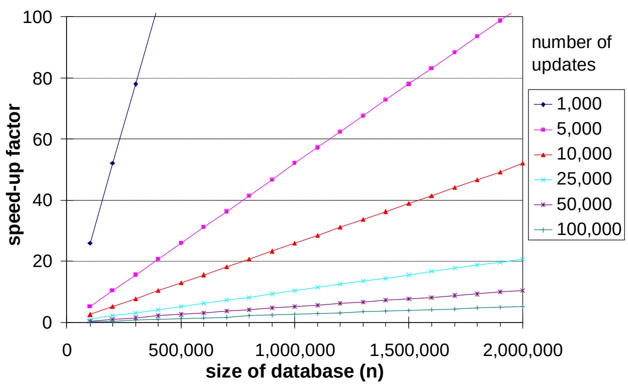

Figure 9: : Speed-up factors for WWW-log database

When setting the speed-up factor to 1.0, we obtain the number of updates (denoted by *MaxUpdates*) up to which the multiple application of IncrementalDBSCAN for each update is more efficient than the single application of DBSCAN to the whole updated database. Figure 10 depicts the values of *MaxUpdates* depending on *n* for f_{del} values of up to 0.5 which is the maximum value to be expected in most real applications. This figure was derived by setting r_{ins} to 1.34 and r_{del} to 6.75 which are the averages over the respective values obtained for our test databases. Note that - in contrast to the significant differences of other characteristics of the two applications - the differences of both r_{ins} and r_{del} are rather small indicating that the average values are a realistic choice for many applications. The *MaxUpdates* values obtained are much larger than the actual numbers of daily updates in most real databases. For databases without deletions (that is, f_{del} = 0), *MaxUpdates* is approximately 3 * *n*, i.e. the cost for 3 * *n* updates on a database of *n* objects using IncrementalDBSCAN is the same as the cost of DBSCAN on the updated database containing 4 * *n* objects. Even in the worst case of f_{del} = 0.5, *MaxUpdates* is approximately 0.25 * *n*. These results clearly emphasize the relevance of incremental clustering.

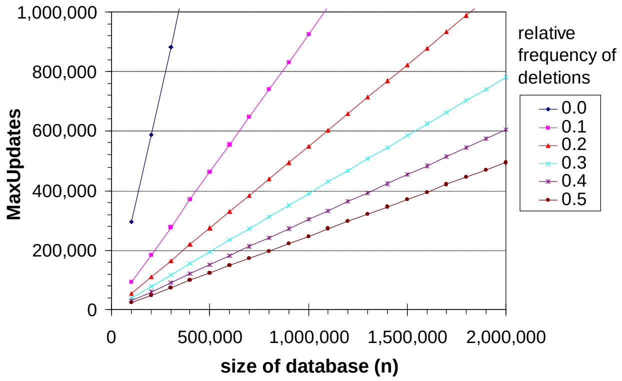

Figure 10: *MaxUpdates* depending on database size for different relative frequencies of deletions

6 Conclusions

Data warehouses provide a great deal of opportunities for performing data mining tasks such as classification and clustering. Typically, updates are collected and applied to the data warehouse periodically in a batch mode, e.g., during the night. Then, all patterns derived from the warehouse by some data mining algorithm have to be updated as well.

In this paper, we presented the first incremental clustering algorithm - based on DBSCAN - for mining in a data warehousing environment. DBSCAN requires only a distance function and, therefore, it is applicable to any database containing data from a metric space. Due to the density-based nature of DBSCAN, the insertion or deletion of an object affects the current clustering only in a small neighborhood of this object. Thus, efficient algorithms could be given for incremental insertions and deletions to a clustering. Based on the formal definition of clusters, it was proven that the incremental algorithm yields the same result as DBSCAN.

A performance evaluation of IncrementalDBSCAN versus DBSCAN using a spatial database as well as a WWW-log database was presented, demonstrating the efficiency of the proposed algorithm. For relatively small numbers of daily updates, e.g., 1,000 updates in a database of size 1,000,000, IncrementalDBSCAN yielded speed-up factors of several hundred. Even for rather large numbers of daily updates, e.g., 25,000 updates in a database of 1,000,000 objects, we obtained speed-up factors of more than 10 versus DBSCAN.

In this paper, we assumed that the parameter values *Eps* and *MinPts* of DBSCAN do not change significantly when inserting and deleting objects. However, there may be applications where this assumption does not hold, i.e. the parameters may change after many updates of the database. In our future work, we plan to investigate this case. In this paper, sets of updates are processed one at a time without considering the relationships between the single updates. In the future, bulk insertions and deletions will be considered to further improve the efficiency of IncrementalDBSCAN.

Acknowledgments

We thank Marco Patella for the M-tree implementation and Franz Krojer for providing us with the WWW access log database.

References

[AF 96] Allard D. and Fraley C.:*"Non Parametric Maximum Likelihood Estimation of Features in Saptial Point Process Using Voronoi Tessellation"*, Journal of the American Statistical Association, December 1997. [also http://www.stat.washington.edu/tech.reports/tr293R.ps].

[AS 94] Agrawal R., Srikant R.: *"Fast Algorithms for Mining Association Rules"*, Proc. 20th Int. Conf. on Very Large Data Bases, Santiago, Chile, 1994, pp. 487-499.

[BKSS 90] Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: *"The R*-tree: An Efficient and Robust Access Method for Points and Rectangles"*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 322-331.

[Bou 96] Bouguettaya A.: *"On-Line Clustering"*, IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 2, 1996, pp. 333-339.

[CHNW 96] Cheung D. W., Han J., Ng V. T., Wong Y.: *"Maintenance of Discovered Association Rules in Large Databases: An Incremental Technique"*, Proc. 12th Int. Conf. on Data Engineering, New Orleans, USA, 1996, pp. 106-114.

[CPZ 97] Ciaccia P., Patella M., Zezula P.: *"M-tree: An Efficient Access Method for Similarity Search in Metric Spaces"*, Proc. 23rd Int. Conf. on Very Large Data Bases, Athens, Greece, 1997, pp. 426-435.

[EKSX 96] Ester M., Kriegel H.-P., Sander J., Xu X.: *"A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise"*, Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, Portland, OR, 1996, pp. 226-231.

[EKX 95] Ester M., Kriegel H.-P., Xu X.: *"Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification"*, Proc. 4th Int. Symp. on Large Spatial Databases, Portland, ME, 1995, in: Lecture Notes in Computer Science, Vol. 951, Springer, 1995, pp. 67-82.

[EW 98] Ester M., Wittmann R.: *"Incremental Generalization for Mining in a Data Warehousing Environment"*, Proc. 6th Int. Conf. on Extending Database Technology, Valencia, Spain, 1998, in: Lecture Notes in Computer Science, Vol. 1377, Springer, 1998, pp. 135-152.

[FAAM 97] Feldman R., Aumann Y., Amir A., Mannila H.: *"Efficient Algorithms for Discovering Frequent Sets in Incremental Databases"*, Proc. ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, Tucson, AZ, 1997, pp. 59-66.

[FPS 96] Fayyad U., Piatetsky-Shapiro G., and Smyth P.: *"Knowledge Discovery and Data Mining: Towards a Unifying Framework"*, Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, Portland, OR, 1996, pp. 82-88.

[Gue 94] Gueting R. H.: *"An Introduction to Spatial Database Systems"*, The VLDB Journal, Vol. 3, No. 4, October 1994, pp. 357-399.

[HCC 93] Han J., Cai Y., Cercone N.: *"Data-driven Discovery of Quantitative Rules in Relational Databases"*, IEEE Transactions on Knowledge and Data Engineering, Vol.5, No. 1, 1993, pp. 29-40.

[Huy 97] Huyn N.: *"Multiple-View Self-Maintenance in Data Warehousing Environments"*, Proc. 23rd Int. Conf. on Very Large Data Bases, Athens, Greece, 1997, pp. 26-35.

[KR 90] Kaufman L., Rousseeuw P. J.: *"Finding Groups in Data: An Introduction to Cluster Analysis"*, John Wiley & Sons, 1990.

[Luo 95] Luotonen A.: *"The common log file format"*, http://www.w3.org/pub/WWW/, 1995.

[MJHS 96] Mombasher B., Jain N., Han E.-H., Srivastava J.: *"Web Mining: Pattern Discovery from World Wide Web Transactions"*, Technical Report 96-050, University of Minnesota, 1996.

[MQM 97] Mumick I. S., Quass D., Mumick B. S.: *"Maintenance of Data Cubes and Summary Tables in a Warehouse"*, Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 100-111.

[NH 94] Ng R. T., Han J.: *"Efficient and Effective Clustering Methods for Spatial Data Mining"*, Proc. 20th Int. Conf. on Very Large Data Bases, Santiago, Chile, 1994, pp. 144-155.

[SEKX 98] Sander J., Ester M., Kriegel H.-P., Xu X.: *"Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Applications"*, will appear in: Data Mining and Knowledge Discovery, Kluwer Acedemic Publishers, Vol. 2, 1998.

[Sib 73] Sibson R.: *"SLINK: an optimally efficient algorithm for the single-link cluster method"*, The Computer Journal, Vol. 16, No. 1, 1973, pp. 30-34.

[ZRL 96] Zhang T., Ramakrishnan R., Linvy M.: *"BIRCH: An Efficient Data Clustering Method for Very Large Databases"*, Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 103-114.