In Proc. 5th IEEE International Conference on Data Mining (ICDM), Houston, TX, 2005

Online Hierarchical Clustering in a Data Warehouse Environment

Elke Achtert, Christian Böhm, Hans-Peter Kriegel, Peer Kröger

Institute for Computer Science, University of Munich
{achtert,boehm,kriegel,kroegerp}@dbs.ifi.lmu.de

Abstract

Many important industrial applications rely on data mining methods to uncover patterns and trends in large data warehouse environments. Since a data warehouse is typically updated periodically in a batch mode, the mined patterns have to be updated as well. This requires not only accuracy from data mining methods but also fast availability of up-to-date knowledge, particularly in the presence of a heavy update load. To cope with this problem, we propose the use of online data mining algorithms which permanently store the discovered knowledge in suitable data structures and enable an efficient adaptation of these structures after insertions and deletions on the raw data. In this paper, we demonstrate how hierarchical clustering methods can be reformulated as online algorithms based on the hierarchical clustering method OPTICS, using a density estimator for data grouping. We also discuss how this algorithmic schema can be specialized for efficient online single-link clustering. A broad experimental evaluation demonstrates that the efficiency is superior with significant speed-up factors even for large bulk insertions and deletions.

1 Introduction

Many companies store terabytes of corperate data in large-scale data warehouses. This tremendous amount of data can only be fully exploited and utilized by efficient and effective data mining tools. Industrial applications such as decision support systems in a data warehouse environment require not only high accuracy from data analysis methods but also fast availability of up-to-date knowledge. The challenge is to recall current knowledge from anywhere at any time with a delay of no more than a few minutes — a prohibitive demand for many data mining algorithms which are able to gain knowledge only from scratch using highly complex operations.

To cope with this problem of updating mined patterns in a data warehouse environment, we propose the use of data mining algorithms which permanently store the acquired knowledge in suitable data structures and facilitate an efficient adaptation of this stored knowledge whenever the raw data from which the knowledge is drawn changes. We call such a method an *online* data mining algorithm, and its conventional counterpart is called *offline*. When speaking of a data warehouse environment, we do not anticipate a certain architecture, but address an environment in which changes in the transactional database are collected over some period (e.g. daily) and the data warehouse is updated using batch operations, i.e. bulk insertions and deletions. Therefore, it is important that online data mining algorithms efficiently support insertions, deletions and updates not only of single data items but also for larger sets. Depending on the size of the bulk to be inserted, deleted, or updated, the response of an online data mining algorithm is much faster compared to its offline competitor. Let us note that the concepts presented in this paper are independent of the physical process of incremental data warehouse maintainance (cf. e.g. [11] for a discussion on that topic).

One of the major data mining tasks is clustering which aims at partitioning the data set into groups (clusters) of similar objects. Hierarchical algorithms determine a hierarchy of nested clusters. Widespread methods such as single-link, complete-linkage, average linkage [8] or OPTICS [1] represent and visualize cluster hierarchies by dendrograms or reachability diagrams. A dendrogram (cf. Figure 1 (right)) is a (usually unbalanced) tree where the root represents the complete data set, each leaf node represents a single object and the internal nodes represent the cluster structure. A reachability diagram (cf. Figure 1 (left)) can be regarded as a linearization of this tree by a depth-first traversal, depicting only the degree of similarity of the objects in a cluster. Both structures visualize the

hierarchical cluster structure of the data set and can be transferred into each other [12].

Both the dendrogram as well as the reachability diagram are exactly those knowledge representations that need to be stored permanently and need to be updated accordingly when reformulating a hierarchical clustering algorithm from the conventional offline version into the new online version. In this paper, we focus on reachability diagrams, mainly because the online maintainance is easier to handle as we will see later. By doing this we do not sacrifice generality, due to the equivalence of dendrograms and reachability diagrams. We show how this structure can be efficiently updated whenever the data set changes due to insertions, deletions or update operations.

The rest of the paper is organized as follows: In Section 2 we discuss related work and different methods to represent the hierarchical clustering structure. Section 3 proposes algorithms for hierarchical online clustering. Section 4, presents a broad experimental evaluation. Section 5 concludes the paper.

2 Related Work

2.1 Online Clustering Methods

Clustering is one of the primary data mining tasks. A huge amount of clustering algorithms have been proposed in the past years (see e.g. [8] for an overview). Since we tackle the problem of hierarchical clustering, we focus on discussing recent work on online hierarchical clustering in this paper, rather than online algorithms for flat partitioning clustering algorithms.

Several methods have been proposed for online hierarchical clustering, e.g. SLINK [13] for single-linkage, CLINK [6] for complete-linkage, GRIN [5] based on the gravity theory in physics, and IHC [14]. All thesse approaches suffer from at least one of the following drawbacks: (1) The algorithms only provide the possibility to handle insertions of data objects. Deletions cannot be handled in an online fashion. (2) The algorithms can only process one update object at a time and are usually difficult to generalize to manage sets of update objects. Thus, they are not efficient for large bulk updates which frequently occur in a data warehouse environment. In this paper, we propose an algorithmic schema that overcomes these limitations, i.e. enables efficient online hierarchical clustering when inserting and deleting huge bulks of data objects.

In [4] a compression technique for hierarchical clustering called "Data Bubbles" is proposed (recently extended in [15]). In [10], the authors propose an incremental summarization method based on Data Bubbles which can be applied to dynamic hierarchical clustering. Since Data Bubbles tend to miss details in the cluster hierarchy when increasing the compression rate, the benefit of improving the runtime is limited by the accuracy. The approach of compressing data by Data Bubbles, however, is orthogonal to the approach proposed in this paper: Online hierarchical clustering can be applied to the original data or to compressed data.

The clustering of a data stream is to some degree related to the online clustering in data warehouses. Both methodologies aim at providing the user with up-to-date information on clusters very quickly in a dynamic environment. However, data streams impose different requirements on clustering algorithms and the entire data mining process (see e.g. [2, 7]). In particular, in a data warehouse, the clustering algorithm has access to all points currently in the database and not necessarily only to the most recently inserted points as usually in case of stream data. In addition, when clustering stream data, the algorithm is restricted to sequential access to newly inserted objects. This restriction does obviously not apply to algorithms for online clustering in a data warehouse environment. Our solutions are therefore different from the data stream clustering context in these two aspects. In particular, we empirically show that our bulk update algorithms are considerably more efficient than a sequential processing of each single object in the update bulk.

In [9] an preliminary incremental version of OPTICS is presented. However, this approach has no bulk update mode and thus, cannot handle large sets of updates.

2.2 Computing and Representing Hierarchical Clustering Structures

Hierarchical clustering algorithms compute a hierarchical decomposition of the data objects. The hierarchical clustering structure is usually visualized using a *dendrogram* (cf. Figure 1 (right)). One leaf of this tree corresponds to one data object. The root represents the entire database. A node in the dendrogram represents a cluster containing all child nodes.

A well-known algorithmic schema constructs a dendrogram in an agglomerative fashion. It starts with each object of the database placed in a unique cluster (leaf nodes) and then merges in each step that pair of clusters having minimal distance until all data objects are contained in one cluster. Algorithms differ in the definition of the distance between clusters, e.g. the single-link method [13] defines the distance between two sets of objects as the distance of the closest pair between both sets. Most popular distance measures be-

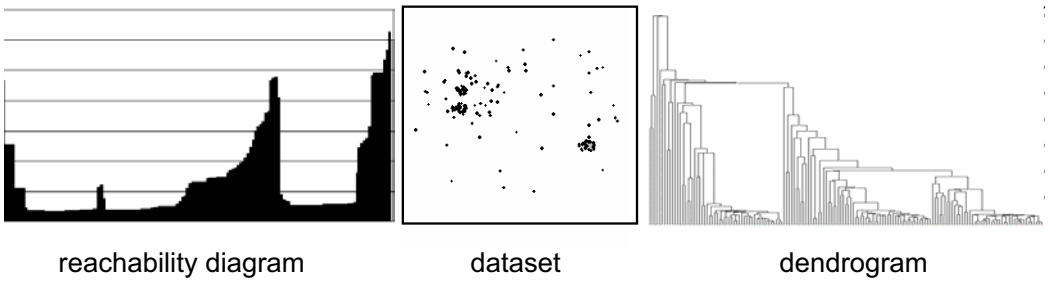

Figure 1. A reachability diagram (left) and a dendrogram (right) for a sample data set (middle).

tween clusters (e.g. single-link, average-link, complete-link) usually favor clusters of certain shapes.

In [1] the hierarchical algorithm called OPTICS is presented which uses a density-based clustering notion. In particular, OPTICS takes the density in the neighborhood of an object into account, using two input parameters, ε and *minPts*. The *core distance* of $o \in \mathcal{D}$, denoted by $\text{CDIST}(o)$, measures the density around o and is computed as $CoreDist(o) = nn\text{-}dist_{minPts}(o)$ if $|\mathcal{N}_\varepsilon(o)| \geq minPts$ else $CoreDist(o) = \infty$. $\mathcal{N}_\varepsilon(o)$ denotes the ε-neighborhood of o and $nn\text{-}dist_k(o)$ denotes the k-nearest neighbor distance of o. The *reachability distance* of an object $p \in \mathcal{D}$ relative from object $o \in \mathcal{D}$ w.r.t. $\varepsilon \in \mathbb{R}$ and $minPts \in \mathbb{N}$ is then defined as $\text{RDIST}(o,p) = \max(\text{CDIST}(o), \delta(o,p))$, where δ is the distance function. The reachability distance of p from o is an asymmetric distance measure that takes the density around o (its core distance) into account. Let us point out that for $\varepsilon{=}\infty$ and $minPts{=}2$ the reachability distance of p from o is $\delta(o,p)$), since $\text{CDIST}(o) = \delta(o,p)$).

The OPTICS algorithm computes a so-called *cluster ordering* of a database w.r.t. the two input parameters ε and *minPts*. In addition, the core distance and a "suitable" reachability distance is stored for each object. This information is enough to extract the hierarchical clustering structure of the data set. The algorithm starts with an arbitrary object $o \in \mathcal{D}$, assigns a reachability distance of ∞ to o and expands the cluster order if the core distance of o is smaller than the input parameter ε. The expansion is performed by inserting the objects $p \in \mathcal{N}_\varepsilon(o)$ into a seed list. This seed list is organized as a heap, storing that object q as first object in the list, having the minimum reachability distance to all the already processed objects. The next object to be inserted in the cluster ordering is always the first object in the seed list. If the core distance of this object is smaller or equal to ε, all objects in the ε-neighborhood are again inserted into or updated in the seed list. If the seed list is empty and there are still some not yet processed objects in $\mathcal{D}$, we have a so-called "jump". OPTICS selects another arbitrary not yet handled object in $\mathcal{D}$ to further expand the cluster ordering CO as described above.

The resulting cluster ordering can be visualized very intuitively and clearly by means of a so-called *reachability diagram*, even for very large datasets. A reachability diagram is a 2D visualization of a cluster ordering, where the objects are plotted according to the cluster ordering along the x-axis, and the reachability distance of each object along the y-axis (cf. Figure 1 (left)). Clusters are "valleys" in the diagram.

In contrast to other methods, OPTICS does not favor clusters of a particular size or shape. Single-link clustering is in fact a special parametrisation of OPTICS. A single-link clustering can be computed using OPTICS with $\varepsilon{=}\infty$ and $minPts{=}2$.

3 Online Bulk Updates of Hierarchical Clustering Structures

As discussed above, usually a reachability diagram can be transformed into a dendrogram and *vice versa* [12]. Due to these relationships, we can develop online algorithms for bulk insertions and deletions using reachability diagrams as a representation of a hierarchical clustering structure without loosing generality. We propose algorithmic schemata called *OnlineOPTICS* that can be used for efficient online clustering in both worlds — agglomerative single-link clustering and density-based clustering.

In the following, we call the insertion or deletion of a bulk of objects an *update operation*. The bulk of update objects is denoted by $\mathcal{U}$. We further assume $\mathcal{D}$ to be a database of n objects and δ to be a metric distance function on the objects in $\mathcal{D}$.

3.1 General Ideas and Concepts

OPTICS computes a cluster ordering. In order to maintain a cluster ordering, we first formalize it.

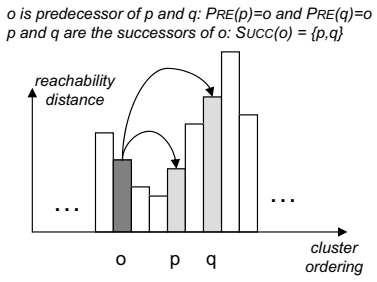

Figure 2. Visualization of predecessor and successors.

Definition 1 (cluster ordering) *Let $minPts \in \mathbb{N}$, $\varepsilon \in \mathbb{R}$, and CO be a permutation of the objects in $\mathcal{D}$. Each $o \in \mathcal{D}$ has additional attributes $o.P$, $o.C$ and $o.R$, where $o.P \in \{1, \ldots, n\}$ symbolizes the position of o in CO. We call CO a* cluster ordering *w.r.t. ε and $minPts$ if the following conditions hold:*

(1) $\forall p \in CO : p.C = \text{CDIST}(p)$

(2) $\forall p \in CO : p.P = i \Rightarrow p.R = \min\{\text{RDIST}(x, y) \mid x.P < i \wedge y.P \geq i\}$, *where* $\min \emptyset = \infty$.

Condition (1) states that $p.C$ is the core distance of object p. Intuitively, condition (2) states that the order is built by starting at an arbitrary object in $\mathcal{D}$ and then selecting at each position i in CO that object p having the minimum reachability distance from any object before i. $p.R$ is this minimum reachability distance assigned to object p during the generation of CO. The object p which is responsible for the choice of object o at position $o.P$ is called the *predecessor* of o in CO, denoted by $\text{PRE}(o)$. We say that o has been "reached" from its predecessor. If o has not been reached from any other object, its reachability distance $o.R$ in CO is ∞, and thus, its predecessor is undefined. Analogously, the *successors* of on object o in CO, denoted by $\text{SUC}(o)$, are those objects having o as their predecessor, i.e. $\text{SUC}(o) = \{p \in CO \mid \text{PRE}(p) = o\}$. The successors of an object o include all objects in the cluster ordering that have been reached from o. Obviously, there may also be objects that do not have any successors. Both concepts of predecessor and successors are visualized in Figure 2.

As stated above, the cluster ordering is exactly that representation we need for an efficient online reconstruction of the hierarchical clustering structure. In particular, for each object o in the cluster ordering, we compute its position ($o.P$), its core distance ($o.C$), the "suitable" reachability distance ($o.R$), and in addition to the original OPTICS algorithm its predecessor ($\text{PRE}(o)$) and its successors ($\text{SUC}(o)$) on the fly during the original OPTICS run or during reorganization.

3.2 Affected Objects

In case of an update operation, condition (2) of Definition 1 may be violated. Recomputing the entire cluster ordering using the offline OPTICS algorithm requires the computation of one range query for each object in the database. However, in most cases, several parts of the cluster ordering remain unchanged. For these parts of the cluster ordering, we do not need to compute range queries. The idea of an online update is to save unnecessary range queries and distance calculations during reconstruction. In the following, we say that if we decide to compute a range query around an object, this object is affected by the update and needs reorganization.

To determine the objects that are affected by update operations, we make the following considerations. Due to an update operation, the core distance of some objects may change. As a consequence, the reachability distances of some objects that were "reached" from these objects in the cluster ordering may also change, causing the above mentioned violation of condition (2) in Definition 1. We call the objects with changing core distances *directly affected objects*. Let $NN_k(q)$ be the set of the k-nearest neighbors of $q \in \mathcal{D}$. The set of *reverse k-nearest neighbors* of an object $p \in \mathcal{D}$ is defined as $\text{REV}_k(p) = \{q \in \mathcal{D} \mid p \in NN_k(q)\}$.

Definition 2 (directly affected objects) *Let $\mathcal{U}$ be the update set and CO be a cluster ordering of $\mathcal{D}$ w.r.t. $\varepsilon \in \mathbb{R}$ and $minPts \in \mathbb{N}$. The set of* directly affected objects *due to an update operation, denoted by* $\text{DAff}(\mathcal{U})$*, is defined as*

$$\text{DAff}(\mathcal{U}) = \{\text{o} \in \text{REV}_{minPts}(\text{u}) \mid \text{u} \in \mathcal{U} \wedge \delta(\text{o}, \text{u}) \leq \varepsilon\}.$$

In fact, not all objects in $\text{DAff}(\mathcal{U})$ must change their core distances. It may happen that an object $o \in \text{DAff}(\mathcal{U})$ has a core distance of already ∞. Then, in case of deletion, the core distance of o obviously remains unchanged. In case of insertion, the change of the core distance of o depends on whether the *minPts*-nearest neighbor distance of o decreases under the limit of ε. Obviously, an object not belonging to $\text{DAff}(\mathcal{U})$ cannot change its core distance due to the update operation.

The set $\text{DAff}(\mathcal{U})$ can be efficiently computed using the following lemma.

Lemma 1 *Let $\mathcal{U}$ be the update set and CO be a cluster ordering of $\mathcal{D}$. Then the following holds:*

$\text{DAff}(\mathcal{U}) = \{o \mid u \in \mathcal{U} \wedge o \in \mathcal{N}_\varepsilon(u) \wedge \delta(u,o) \leq \text{CDist}(o)\}$.

Proof. *Let* $X := \{o \mid u \in \mathcal{U} \wedge o \in \mathcal{N}_\varepsilon(u) \wedge \delta(u,o) \leq \text{CDist}(o)\}$. *We show that* $\text{DAff}(\mathcal{U}) = X$*:*

$\forall o \in \text{DAff}(\mathcal{U}) : u \in \mathcal{U} \wedge \delta(o,u) \leq \varepsilon \wedge o \in \text{Rev}_{minPts}(u)$
$\Leftrightarrow u \in \mathcal{U} \wedge o \in \mathcal{N}_\varepsilon(u) \wedge o \in \text{Rev}_{minPts}(u)$
$\Leftrightarrow u \in \mathcal{U} \wedge o \in \mathcal{N}_\varepsilon(u) \wedge u \in NN_{minPts}(o)$
$\Leftrightarrow u \in \mathcal{U} \wedge o \in \mathcal{N}_\varepsilon(u) \wedge \delta(u,o) \leq nn\text{-}dist_{minPts}(o)$
$\Leftrightarrow u \in \mathcal{U} \wedge o \in \mathcal{N}_\varepsilon(u) \wedge \delta(u,o) \leq \text{CDist}(o) \Leftrightarrow o \in X$

Lemma 1 states that we can filter out a lot of objects not belonging to $\text{DAff}(\mathcal{U})$ by computing only one range query around each update object $u \in \mathcal{U}$. In addition, for all $u \in \mathcal{U}$ we only have to test the objects $o \in \mathcal{N}_\varepsilon(u)$ whether $o \in \text{Rev}_{minPts}(u)$ holds. The idea is that for all $o \in \text{Rev}_{minPts}(u)$, it holds that $\delta(u,o) \leq \text{CDist}(o)$. If the update operation is an insertion, the core distance of o decreases, whereas if the update operation is a deletion, the core distance of o increases.

Due to mutating core distances, reachability distances of some objects may change. We call these objects indirectly affected. A changing reachability distance may cause the violation of condition (2) in Definition 1. If a reachability distance of an object o decreases due to a changed core distance, o may move forward in the cluster ordering, otherwise o may move backwards. In addition, if an object o has changed its postion in the cluster ordering, the successors and the predecessor (if not yet handled) might also be affected because these objects may now be reached earlier or later, i.e. may be placed at a different position in the new cluster ordering.

We call these objects that may be indirectly affected by the reorganization of an object o the *potential successors* of o. The set of potential successors $\text{Suc}(o)$ of an affected object $o \in CO$ is recursively defined:

(1) if $p \in \text{Suc}(o)$ then $p \in \text{Suc}^{pot}(o)$.

(2) if $q \in \text{Suc}^{pot}(o)$ and $p \in \text{Suc}(q)$ and $x \in \text{Suc}^{pot}(o) \wedge x.P = p.P - 1 \Rightarrow x.R \leq p.R$ then $p \in \text{Suc}^{pot}(o)$.

Condition (1) states that all direct successors of o are also candidates for reorganization. The second condition collects recursively the direct successors of potential successors, as long as their assigned reachability distance increases. If the reachability distance decreases, we enter a new cluster where the objects are more densely packed. In this case, the objects can only be affected by the local neighborhood. The intuition behind condition (2) is visualized in Figure 3. The recursive collection of the potential successors of object o stops at object p. All objects in C will be reached from

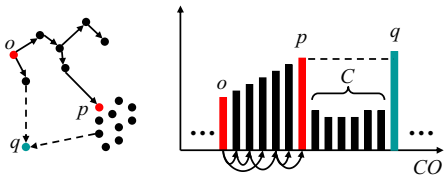

Figure 3. The recursive collection of potential successors of o stops at p and may restart at q.

p or another object in C but not from o. The recursive collection may restart at q if q or any object after q is the successor of p or of any object before p.

Lemma 2 *Let $\mathcal{U}$ be the update set, CO be a cluster ordering of $\mathcal{D}$, and $o, p \in CO$. If $p \notin \text{Suc}^{pot}(o)$, then $o \neq \text{Pre}(p)$.*

Proof. *Let $p \notin \text{Suc}^{pot}(o)$. We show $o \neq \text{Pre}(p)$ by recursion over $\text{Suc}^{pot}(o)$:*

(1) $p \notin \text{Suc}(o) \Rightarrow o \neq \text{Pre}(p)$.

(2) $q \in \text{Suc}^{pot}(o)$ and $p \notin \text{Suc}(q)$ and $x \in \text{Suc}^{pot}(o) \wedge x.P = p.P - 1 \wedge x.R > p.R$:
Since x has been reached earlier than p from the objects before x, $x.P = p.P - 1$, and $x.R > p.R$, p must have been reached from x, i.e. $\text{Pre}(p) = x \Rightarrow o \neq \text{Pre}(p)$.

Lemma 2 states, that only the points in $PotSuc(o)$ can be affected by a reorganization of an object o. Now we are able to develop online versions to handle insertions and deletions efficiently. In the following, CO_{old} denotes the original cluster ordering before the update. OnlineOPTICS will create CO_{new}, the new (valid) cluster ordering after the update, by performing a single pass over CO_{old}.

3.3 Online Bulk Insertions

The pseudo code of the online bulk insertion algorithm is depicted in Figure 4. In the first step of the insertion of all objects in $\mathcal{U}$, the core distances of each $o \in \text{DAff}(\mathcal{U})$ are updated and all objects $u \in \mathcal{U}$ are inserted into the seed list OrderedSeeds with $u.R = \infty$ and $\text{Pre}(u) = \emptyset$. This is because it is not yet clear, from which object the objects in $\mathcal{U}$ are reached in CO_{new}.

After that, the reorganization is performed, imitating the original OPTICS algorithm. We manage the objects that need reorganization in the seed list. In each iteration of the reconstruction loop, we compare the next not yet handled object c in CO_{old} with the

```
bulkInsert(SetOfObjects 𝒰, ClusterOrdering CO_old)
// all o ∈ CO_old and u ∈ 𝒰 marked as not yet handled
// CO_new is an empty cluster ordering
for each u ∈ 𝒰 do
  u.P := n + 1;
  u.C := CDIST(u);
for each o ∈ DAff(𝒰) do
  update the core distance of o;
insert u ∈ 𝒰 into OrderedSeeds with u.R = ∞ and PRE(u) = ∅;
while CO_old contains unhandled objects or OrderedSeeds ≠ ∅
do
  c := first not yet handled object in CO_old;
  s := first not yet handled object in OrderedSeeds;
  if s.R > c.R or (s.R = c.R and s.P > c.P) then
    l:= c; append l to CO_new;
  else
    l:= s; OrderedSeeds.remove(s); append l to CO_new;
  mark l as handled;
  if l is chosen from OrderedSeeds or l ∈ DAff(𝒰) then
    OrderedSeeds.update(SUC^pot(l), l);
  else
    for each u ∈ 𝒰 : u do
      if u is not yet handled and l.C ≤ ε and u ∈ N_ε(l) then
        OrderedSeeds.update({u}, l);
```

Figure 4. Algorithm bulkInsert.

first object s in OrderedSeeds. The object among the two (c and s) which has the smaller reachability distance value is appended to CO_{new}. If the reachability distance values of both objects are equal, we append that object having the smaller position.

After the insertion of an object l into the new cluster ordering CO_{new}, we have to update OrderedSeeds. This is done by the method update which is an adoption of the corresponding method in [1]. If the recently processed object l is derived from the original cluster ordering CO_{old}, we have to test which update objects $u \in \mathcal{U}$ have been already processed. If any $u \in \mathcal{U}$ has not been processed and $l.C \neq \infty$ and $\delta(l,u) \leq \varepsilon$, u would have been inserted/updated in OrderedSeeds in the original OPTICS run. Thus, l may be a potential predecessor of u and OrderedSeeds has to be updated accordingly. Other connections are not affected, since we store the objects that need reorganization in the seed list. If l is derived from OrderedSeeds or from DAff($\mathcal{U}$), some connections may need reorganization. Thus, all not yet processed potential successors $x \in \text{SUC}^{pot}(l)$ have to be inserted/updated in the seed list.

The reorganization stops if the original cluster ordering CO_{old} does not contain unprocessed objects any more and the seed list is empty.

Lemma 3 *The online bulk insert algorithm produces a valid cluster ordering w.r.t. Definition 1.*

Proof. *Due to space limitations, we refer to the long version of this paper for the proof.*

```
bulkDelete(SetOfObjects 𝒰, ClusterOrdering CO_old)
// all o ∈ CO_old marked as not yet handled
// CO_new is an empty cluster ordering
for each u ∈ 𝒰
  mark u as handled;
rs := ∅;
for each o ∈ DAff(𝒰) do
  update the core distance of o;
  insert SUC^pot(o) into rs;
insert r ∈ rs into OrderedSeeds with r.R = ∞ and PRE(r) = ∅;
while CO_old contains unhandled objects or OrderedSeeds ≠ ∅
do
  c := first unhandled object in CO_old not contained in rs;
  s := first unhandled object in OrderedSeeds;
  if s.R ≤ c.R or PRE(c) is not yet handled then
    l:=s; OrderedSeeds.remove(s); append l to CO_new;
  else
    l:=c; append l to CO_new;
  mark l as handled;
  if l.C ≤ ε then
    OrderedSeeds.updateAll(l, ε);
    OrderedSeeds.update(SUC(l), l);
```

Figure 5. Algorithm bulkDelete.

3.4 Online Bulk Deletion

The pseudo code of the online bulk deletion is depicted in Figure 5. In the first step, all objects $u \in \mathcal{U}$ are marked as handled. In addition, for each $o \in \text{DAff}(\mathcal{U})$ the core distance of each o is updated and its potential successors $\text{SUC}^{pot}(o)$ are inserted into OrderedSeeds with a reachability distance of ∞ and (yet) undefined predecessor.

After that, the reorganization is performed, again simulating the original OPTICS algorithm. In each iteration of the reconstruction loop, we compare the reachability distance of the next not yet handled object c in CO_{old} which is not contained in DAff($\mathcal{U}$) with that of the first object s in OrderedSeeds. If the predecessor of c is not yet processed (i.e. inserted into CO_{new}), c cannot be taken from CO_{old} and cannot be appended to CO_{new}. Otherwise, that object of c and s, having the smaller reachability distance value, is appended to CO_{new}.

After the insertion of an object l into the new cluster ordering CO_{new}, we have to update OrderedSeeds. This is done by the methods update and updateAll which are both adoptions of the according method in [1].

The reorganization stops if the original cluster ordering CO_{old} does not contain unprocessed objects any more and the seed list is empty.

Lemma 4 *The online bulk delete algorithm produces a valid cluster ordering w.r.t. Definition 1.*

Proof. *Due to space limitations, we refer to the long version of this paper for the proof.*

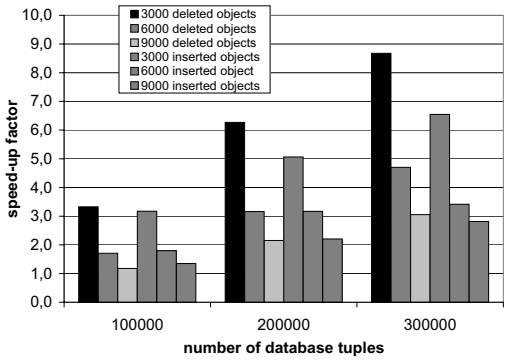

Figure 6. Speed-up w.r.t database size.

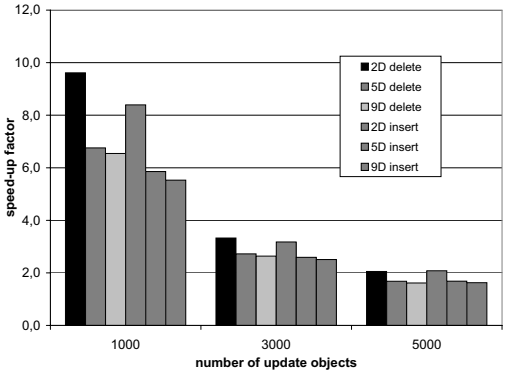

Figure 7. Speed-up w.r.t data dimensionality.

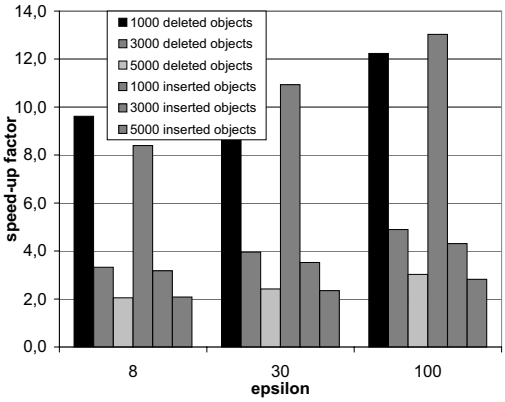

Figure 8. Speed-up w.r.t the choice of ε.

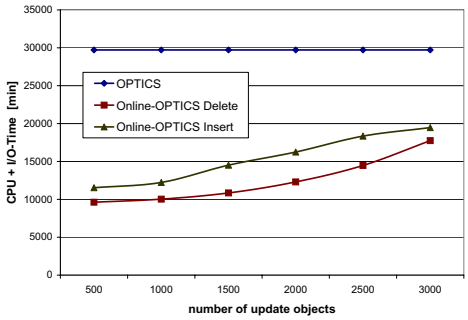

Figure 9. Results on real-world TV data.

4 Experimental Evaluation

We compared OnlineOPTICS with offline OPTICS. We on several synthetic databases and a real-world database containing TV-snapshots represented by 64D color histograms. All tests were run under Linux on a workstation with a 3.2 GHz CPU, and hard disk with a transfer rate of 20 MByte/s for the sequential scan. Random page accesses need 8ms including seek time, latency delay, and transfer time. Unless otherwise specified, we used an X-Tree [3] to speed-up the range-queries required by both algorithms.

The speed-up factors of OnlineOPTICS over the offline version w.r.t. the database size are depicted in Figure 6. As it can be seen, the performance gain achieved by the online update algorithm is growing with an increasing number of data objects both for insertions and deletions. In fact, an online update of nearly 10% of the data objects is still more efficient than an offline update.

We illustrate the speed-up factors of OnlineOPTICS over offline OPTICS w.r.t. the dimensionality of the database in Figure 7. With increasing dimensionality of the database, the speed-up factors slightly decrease due to the performance deterioration of the underlying index.

The only parameter of offline OPTICS that affects the performance is ε. In [1] the authors state, that ε has to be chosen large enough. In particular, ε has to be chosen large enough, such that there is no jump in the ordering. In general, the optimal choice of ε is quite hard to anticipated beforehand, so we copmared OnlineOPTICS to offline OPTICS w.r.t. several choices of ε. The results are depicted in Figure 8 showing that the higher the value for ε, the bigger is the performance gain of the online update. This can be explained by the decreasing selectivity of the underlying index.

We applied offline OPTICS and OnlineOPTICS to the above mentioned real-world dataset of 64D color histograms representing TV snapshots. The results are illustrated in Figure 9, confirming the observations made on the synthetic data. As it can be seen, OnlineOPTICS also achieved impressive speed-up factors over OPTICS for both insertion and deletion of a bulk of objects.

We also tested offline OPTICS and OnlineOPTICS computing all range queries on top of the sequential scan using a synthetic database of 2D feature vectors.

The result is not shown due to space limitations. We observed that for 200,000 data objects, the online update of 10% of the data objects is still more efficient than an offline update.

Our proposed bulk update algorithms can easily be adopted to process the update set sequentially, i.e. processing each object u in the update set separately as it is required for data streams. We compared our bulk schema with such a sequential update processing on a 2D dataset containing 10,000 points. We varied the update load between 1% and 10% of the database size. Figure 10 illustrates the speed-up factor of bulk OnlineOPTICS over sequential Online OPTICS. As it can be seen, the bulk mode clearly outperforms the sequential update processing. The ratio between the speed-up factors of the bulk mode and of the sequential mode is increasing with growing update load, i.e. the bulk mode outperforms the sequential mode the more, the larger the size of the update set is. Obviously, this is because the bulk mode only requires one run through the old cluster ordering while the sequential update processing requires multiple runs over the old cluster ordering.

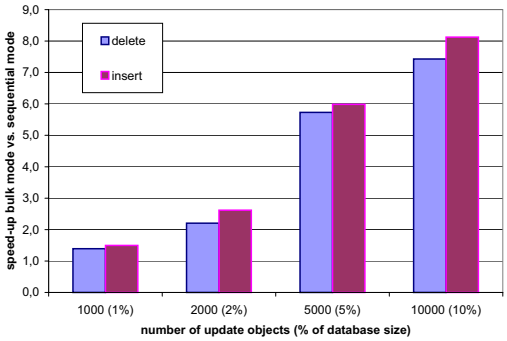

Figure 10. Comparison of bulk update versus sequential update processing.

5 Conclusions

Online algorithms for data mining are important to detect knowledge which is not only accurate but also immediately available and up-to-date. In this paper, we have proposed OnlineOPTICS, an online version of the single-link method and its generalization OPTICS which is able to process large bulk insertions and deletions as occuring in a data warehouse environment. We have shown that the performance of our algorithm is superior to offline algorithms even in the presence of a heavy update load (large bulk insertions and bulk deletions). In addition, we empirically showed that the bulk mode clearly outperforms a sequential processing of the update set. Our solution could also serve as a template for online versions of further data mining algorithms such as single-link.

References

[1] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. "OPTICS: Ordering Points to Identify the Clustering Structure". In *Proc. ACM SIGMOD*, 1999.

[2] D. Barbara. "Requirements for Clustering Data Streams". *SIGKDD Explorations*, 3:23–27, 2002.

[3] S. Berchtold, D. A. Keim, and H.-P. Kriegel. "The X-Tree: An Index Structure for High-Dimensional Data". In *Proc. VLDB*, 1996.

[4] M. M. Breunig, H.-P. Kriegel, P. Kröger, and J. Sander. "Data Bubbles: Quality Preserving Performance Boosting for Hierarchical Clustering". In *Proc. ACM SIGMOD*, 2001.

[5] C. Chen, S. Hwang, and Y. Oyang. "An Incremental Hierarchical Data Clustering Algorithm Based on Gravity Theory". In *Proc. PAKDD*, 2002.

[6] D. Defays. "CLINK: An Efficient Algorithm for the Complete Link Cluster Method". *The Computer Journal*, 20(4):364–366, 1977.

[7] V. Ganti, J. Gehrke, and R. Ramakrishnan. "Mining Data Steams under Block Evolution". *SIGKDD Explorations*, 3:1–10, 2002.

[8] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Academic Press, 2001.

[9] H.-P. Kriegel, P. Kröger, and I. Gotlibovich. "Incremental OPTICS: Efficient Computation of Updates in a Hierarchical Cluster Ordering". In *Proc. DaWaK*, 2003.

[10] S. Nassar, J. Sander, and C. Cheng. "Incremental and Effective Data Summerization for Dynamic Hierarchical Clustering". In *Proc. ACM SIGMOD*, 2004.

[11] P. Ram and L. Do. "Extracting Delta for Incremental Data Warehouse Maintenance". In *Proc. ICDE*, pages 220–229, 2000.

[12] J. Sander, X. Qin, Z. Lu, N. Niu, and A. Kovarsky. "Automatic Extraction of Clusters from Hierarchical Clustering Representations". In *Proc. PAKDD*, 2003.

[13] R. Sibson. "SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method". *The Computer Journal*, 16(1):30–34, 1973.

[14] D. H. Widyantoro, T. R. Ioerger, and J. Yen. "An Incremental Approach to Building a Cluster Hierarchy". In *Proc. ICDM*, pages 705–708, 2002.

[15] J. Zhou and J. Sander. "Data Bubbles for Non-Vector Data: Speeding-up Hierarchical Clustering in Arbitrary Metric Spaces". In *Proc. VLDB*, 2003.