Proc. 5th Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK'03), pp. 224-233, Prague, Czech Rep., 2003

Incremental OPTICS: Efficient Computation of Updates in a Hierarchical Cluster Ordering

Hans-Peter Kriegel, Peer Kröger, and Irina Gotlibovich

Institute for Computer Science
University of Munich, Germany
{kriegel,kroegerp,gotlibov}@dbs.informatik.uni-muenchen.de

Abstract. Data warehouses are a challenging field of application for data mining tasks such as clustering. Usually, updates are collected and applied to the data warehouse periodically in a batch mode. As a consequence, all mined patterns discovered in the data warehouse (e.g. clustering structures) have to be updated as well. In this paper, we present a method for incrementally updating the clustering structure computed by the hierarchical clustering algorithm OPTICS. We determine the parts of the cluster ordering that are affected by update operations and develop efficient algorithms that incrementally update an existing cluster ordering. A performance evaluation of incremental OPTICS based on synthetic datasets as well as on a real-world dataset demonstrates that incremental OPTICS gains significant speed-up factors over OPTICS for update operations.

1 Introduction

Many companies gather a vast amount of corporate data. This data is typically distributed over several local databases. Since the knowledge hidden in this data is usually of great strategic importance, more and more companies integrate their corporate data into a common data warehouse. In this paper, we do not anticipate any special warehousing architecture but simply address an environment which provides derived information for the purpose of analysis and which is dynamic, i.e. many updates occur.

Usually manual or semi-automatic analysis such as OLAP cannot make use of the entire information stored in a data warehouse. Automatic data mining techniques are more appropriate to fully exploit the knowledge hidden in the data.

In this paper, we focus on clustering, which is the data mining task of grouping the objects of a database into classes such that objects within one class are similar and objects from different classes are not (according to an appropriate similarity measure). In recent years, several clustering algorithms have been proposed [1,2,3,4,5].

A data warehouse is typically not updated immediately when insertions or deletions on a member database occur. Usually updates are collected locally and applied to the common data warehouse periodically in a batch mode, e.g. each night. As a consequence, all clusters explored by clustering methods have to be updated as well. The update of the mined patterns has to be efficient because it should be finished when the warehouse has to be available for its users again, e.g. in the next morning. Since a warehouse usually

stores a large amount of data, it is highly desirable to perform updates incrementally [6]. Instead of recomputing the clusters by applying the algorithm to the entire (very large) updated database, only the old clusters and the objects inserted or deleted during a given period are considered.

In this paper, we present an incremental version of OPTICS [5] which is an efficient clustering algorithm for metric databases. OPTICS combines a density-based clustering notion with the advantages of hierarchical approaches. Due to the density-based nature of OPTICS, the insertion or deletion of an object usually causes expensive computations only in the neighborhood of this object. A reorganization of the cluster structure thus affects only a limited set of database objects. We demonstrate the advantage of the incremental version of OPTICS based on a thorough performance evaluation using several synthetic and a real-world dataset.

The remainder of this paper is organized as follows. We review related work in Section 2. Section 3 briefly introduces the clustering algorithm OPTICS. The incremental algorithms for insertions and deletions are presented in Section 4. In Section 5, the results of our performance evaluation are reported. Conclusions are presented in Section 6.

2 Related Work

Beside the tremendous amount of clustering algorithms (e.g. [1,2,3,4,5]), the problem of incrementally updating mined patterns is a rather new area of research. Most work has been done in the area of developing incremental algorithms for the task of mining association rules, e.g. [7]. In [8] algorithms for incremental attribute-oriented generalization are presented.

The only algorithm for incrementally updating clusters detected by a clustering algorithm is IncrementalDBSCAN proposed in [6]. It is based on the algorithm DBSCAN [4] which models clusters as density-connected sets. 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. Based on these observations, IncrementalDBSCAN yields a significant speed-up over DBSCAN [6].

In this paper, we propose `IncOPTICS` an incremental version of OPTICS [5] which combines the density-based clustering notion of DBSCAN with the advantages of hierarchical clustering concepts. Since OPTICS is an extension to DBSCAN and yields much more information about the clustering structure of a database, `IncOPTICS` is much more complex than IncrementalDBSCAN. However, `IncOPTICS` yields an accurate speed-up over OPTICS without any loss of effectiveness, i.e. quality.

3 Density-Based Hierarchical Clustering

In the following, we assume that D is a database of n objects, $dist : D \times D \rightarrow \mathbb{R}$ is a metric distance function on objects in D and $\mathcal{N}_\varepsilon(p) := \{q \in D \mid dist(p,q) \leq \varepsilon\}$ denotes the ε-neighborhood of $p \in D$ where $\varepsilon \in \mathbb{R}$.

OPTICS extends the density-connected clustering notion of DBSCAN [4] by hierarchical concepts. In contrast to DBSCAN, OPTICS does not assign cluster memberships

but computes a *cluster order* in which the objects are processed and additionally generates the information which would be used by an extended DBSCAN algorithm to assign cluster memberships. This information consists of only two values for each object, the *core-level* and the *reachability-distance* (or short: *reachability*).

Definition 1 (core-level). *Let $p \in D$, MinPts $\in \mathbb{N}$, $\varepsilon \in \mathbb{R}$, and MinPts-dist(p) be the distance from p to its MinPts-nearest neighbor. The* core-level *of p is defined as follows:*

$$\mathit{CLev}(p) := \begin{cases} \infty & \text{if} \quad |\mathcal{N}_\varepsilon(p)| < \mathit{MinPts} \\ \mathit{MinPts\text{-}dist(p)} & \text{otherwise.} \end{cases}$$

Definition 2 (reachability). *Let $p, q \in D$, MinPts $\in \mathbb{N}$, and $\varepsilon \in \mathbb{R}$. The* reachability *of p wrt. q is defined as* $\mathit{RDist}(p,q) := \max\{\mathit{CLev}(q), dist(q,p)\}$.

Definition 3 (cluster ordering). *Let MinPts $\in \mathbb{N}$, $\varepsilon \in \mathbb{R}$, and CO be a totally ordered permutation of the n objects of D. Each $o \in D$ has additional attributes $\mathit{Pos}(o)$, $\mathit{Core}(o)$ and $\mathit{Reach}(o)$, where $\mathit{Pos}(o)$ symbolizes the position of o in CO. We call CO a* cluster ordering *wrt. ε and MinPts if the following three conditions hold:*

(1) $\forall p \in CO : \mathit{Core}(p) = \mathit{CLev}(p)$
(2) $\forall o, x, y \in CO : \mathit{Pos}(x) < \mathit{Pos}(o) \wedge \mathit{Pos}(y) > \mathit{Pos}(o) \Rightarrow \mathit{RDist}(y,x) \geq \mathit{RDist}(o,x)$
(3) $\forall p, o \in CO : \mathit{Reach}(p) = \min\{\mathit{RDist}(p,o) \,|\, \mathit{Pos}(o) < \mathit{Pos}(p)\}$, *where* $\min \emptyset = \infty$.

Intuitively, Condition (2) states that the order is built on selecting at each position i in CO that object o having the minimum reachability to any object before i.

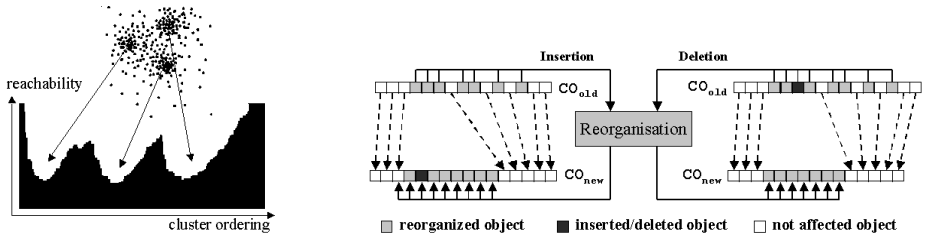
A cluster ordering is a powerful tool to extract flat, density-based decompositions for any $\varepsilon' \leq \varepsilon$. It is also useful to analyze the hierarchical clustering structure when plotting the reachability values for each object in the cluster ordering (cf. Fig. 1(a)).

Like DBSCAN, OPTICS uses one pass over D and computes the ε-neighborhood for each object of D to determine the core-levels and reachabilities and to compute the cluster ordering. The choice of the starting object does not affect the quality of the result [5]. The runtime of OPTICS is actually higher than that of DBSCAN because the computation of a cluster ordering is more complex than simply assigning cluster memberships and the choice of the parameter ε affects the runtime of the range queries (for OPTICS, ε has typically to be chosen significantly higher than for DBSCAN).

4 Incremental OPTICS

The key observation is that the core-level of some objects may change due to an update. As a consequence, the reachability values of some objects have to be updated as well. Therefore, condition (2) of Def. 3 may be violated, i.e. an object may have to move to another position in the cluster ordering. We will have to reorganize the cluster ordering such that condition (2) of Def. 3 is re-established. The general idea for an incremental version of OPTICS is not to recompute the ε-neighborhood for each object in D but restrict the reorganization on a limited subset of the objects (cf. Fig. 1(b)).

Although it cannot be ensured in general, it is very likely that the reorganization is bounded to a limited part of the cluster ordering due to the density-based nature of



(a) (b)

Fig. 1. (a) Visual analysis of the cluster ordering: clusters are valleys in the according reachability plot. (b) Schema of the reorganization procedure.

OPTICS. `IncOPTICS` therefore proceeds in two major steps. First, the starting point for the reorganization is determined. Second, the reorganization of the cluster ordering is worked out until a valid cluster ordering is re-established. In the following, we will first discuss how to determine the frontiers of the reorganization, i.e. the starting point and the criteria for termination. We will determine two sets of objects affected by an update operation. One set called *mutating objects*, contains objects that *may* change its core-level due to an update operation. The second set of affected objects contains objects that move forward/backwards in the cluster ordering to re-establish condition (2) of Def. 3. Movement of objects may be caused by changing reachabilities — as an effect of changing core-levels — or by moving predecessors/successors in the cluster ordering. Since we can easily compute a set of all objects possibly moving, we call this set *moving objects*, containing all objects that *may* move forward/backwards in the cluster ordering due to an update.

4.1 Mutating Objects

Obviously, an object o may change its core-level only if the update operation affects the ε-neighborhood of o. From Def. 1 it follows that if the inserted/deleted object is one of o's *MinPts*-nearest neighbors, *Core*(o) increases in case of a deletion and decreases in case of an insertion. This observation led us to the definition of the set MUTATING(p) of *mutating objects*:

Definition 4 (mutating objects). *Let p be an arbitrary object either in or not in the cluster ordering CO. The set of objects in CO possibly mutating their core-level after the insertion/deletion of p is defined as:* MUTATING$(p) := \{q \mid p \in \mathcal{N}_\varepsilon(q)\}$.

Let us note that $p \in$ MUTATING(p) since $p \in \mathcal{N}_\varepsilon(p)$. In fact, MUTATING$(p)$ can be computed rather easily.

Lemma 1. $\forall p \in D$: MUTATING$(p) = \mathcal{N}_\varepsilon(p)$.

Proof. Since $dist$ is a metric, the following conclusions hold:
$\forall\, q \in \mathcal{N}_\varepsilon(p) \;:\; dist(q,p) \leq \varepsilon \Leftrightarrow dist(p,q) \leq \varepsilon \Leftrightarrow p \in \mathcal{N}_\varepsilon(q) \Leftrightarrow q \in \text{MUTATING}(p)$.

Lemma 2. *Let C be a cluster ordering and $p \in CO$. $\text{MUTATING}(p)$ is a superset of the objects that change their core-level due to an insertion/deletion of p into/from CO.*

Proof. (Sketch)
Let $q \in \text{MUTATING}(p)$: *Core*(q) changes if p is one of q's *MinPts*-nearest neighbors.
Let $q \notin \text{MUTATING}(p)$: According to Lemma 1, $p \notin \mathcal{N}_\varepsilon(q)$ and thus p either cannot be any of q's *MinPts*-nearest neighbors or *Core*$(q) = \infty$ remains due to Def. 1.

Due to Lemma 2, we have to test for each object $o \in \text{MUTATING}(p)$ whether *Core*(o) increases/decreases or not by computing $\mathcal{N}_\varepsilon(o)$ (one range query).

4.2 Moving Objects

The second type of affected objects move forward or backwards in the cluster ordering after an update operation. In order to determine the objects that may move forward or backwards after an update operation occurs, we first define the *predecessor* and the set of *successors* of an object:

Definition 5 (predecessor). *Let CO be a cluster ordering and $o \in CO$. For each entry $p \in CO$ the* predecessor *is defined as*

$$\textit{Pre}(p) = \begin{cases} o & \textit{if} \quad \textit{Reach}(p) = \textit{RDist}(o,p) \\ \text{UNDEFINED} & \textit{if} \quad \textit{Reach}(p) = \infty. \end{cases}$$

Intuitively, *Pre*(p) is the object in CO from which p has been reached.

Definition 6 (successors). *Let CO be a cluster ordering. For each object $p \in CO$ the set of* successors *is defined as* $\textit{Suc}(p) := \{q \in CO \mid \textit{Pre}(q) = p\}$.

Lemma 3. *Let CO be a cluster ordering and $p \in CO$. If Core(p) changes due to an update operation, then each object $o \in$ Suc(p) may change its reachability values.*

Proof. $\forall o \in CO$: $o \in \textit{Suc}(p) \overset{\text{[Def. 6]}}{\Longrightarrow} \textit{Pre}(o) = p \overset{\text{[Def. 5]}}{\Longrightarrow} \textit{Reach}(o) = \textit{RDist}(o,p) \overset{\text{[Def. 2]}}{\Longrightarrow} \textit{Reach}(o) = \max\{\textit{Core}(p), dist(p,o)\}$. Since the value *Core*$(p)$ has changed, *Reach*(o) may also have changed.

As a consequence of a changed reachability value, objects may move in the cluster ordering. If the reachability-distance of an object decreases, this object may move forward such that Condition (2) of Def. 3 is not violated. On the other hand, if the reachability-distance of an object increases, this object may move backwards due to the same reason. In addition, if an object has moved, all successors of this objects may also move although their reachabilities remain unchanged. All such objects that may move after an insertion or deletion of p are called *moving objects*:

Definition 7 (moving objects). *Let p be an arbitrary object either in or not in the cluster ordering CO. The set of objects possibly moving forward/backwards in CO after insertion/deletion of p is defined recursively:*

(1) *If* $x \in \text{MUTATING}(p)$ *and* $q \in \textit{Suc}(x)$ *then* $q \in \text{MOVING}(p)$.
(2) *If* $y \in \text{MOVING}(p)$ *and* $q \in \textit{Suc}(y)$ *then* $q \in \text{MOVING}(p)$.
(3) *If* $y \in \text{MOVING}(p)$ *and* $q \in \textit{Pre}(y)$ *then* $q \in \text{MOVING}(p)$.

Case (1) states, that if an object is a successor of a mutating object, it is a moving object. The other two cases state, that if an object is a successor or predecessor of a moving object it is also a moving object. Case (3) is needed, if a successor of an object o is moved to a position before o during reorganization.

For the reorganization of moving objects we do not have to compute range queries. We solely need to compare the old reachability values to decide whether these objects have to move or not.

4.3 Limits of Reorganization

We are now able to determine between which bounds the cluster ordering must be reorganized to re-establish a valid cluster ordering according to Def. 3.

Lemma 4. *Let CO be a cluster ordering and p be an object either in or not in CO. The set of objects that have to be reorganized due to an insertion or deletion of p is a subset of* $\text{MUTATING}(p) \cup \text{MOVING}(p)$.

Proof. (Sketch)
Let o be an object which has to be reorganized. If o has to be reorganized due to a change of $\textit{Core}(o)$, then $o \in \text{MUTATING}(p)$. Else o has to be reorganized due to a changed reachability or due to moving predecessor/successors. Then $o \in \text{MOVING}(p)$.

Since OPTICS is based on the formalisms of DBSCAN, the determination of the start position for reorganization is rather easy. We simply have to determine the first object in the cluster ordering whose core-level changes after the insertion or deletion because reorganization is only initiated by changing core-levels.

Lemma 5. *Let CO be a cluster ordering which is updated by an insertion or deletion of object p. The object $o \in D$ is the* start object *where reorganization starts if the following conditions hold:*

(1) $o \in \text{MUTATING}(p)$
(2) $\forall q \in \text{MUTATING}(p), o \neq q : \textit{Pos}(o) \leq \textit{Pos}(q)$.

Proof. Since reorganization is caused by changing core-levels, the start object must change its core-level due to the update. (1) follows from Def. 4. According to Def. 7, each $q \in \textit{Suc}(p)$ can by affected by the reorganization. To ensure, that no object is lost by the reorganization procedure, o has to be the first object, whose core-level has changed ($\Rightarrow$(2)). In addition, all objects before o are neither element of $\text{MUTATING}(p)$ nor of $\text{MOVING}(p)$. Therefore, they do not have to be reorganized.

```
WHILE NOT Seeds.isEmpty() DO
  // Decide which object is at next added to CO_new
  IF currObj.reach > Seeds.first().reach THEN
    CO_new.add(Seeds.first());
    Seeds.removeFirst();
  ELSE
    CO_new.add(currObj);
    currObj = next object in CO_old which has not yet been inserted into CO_new
  // Decide which objects are inserted into Seeds
  q = CO_new.lastInsertedObject();
  IF q∈ MUTATING(p) THEN
    FOR EACH o∈N_ε(p) which has not yet been inserted into CO_new DO
      Seeds.insert(o, max{q.core, dist(q,o)});
  ELSE IF q∈ MOVING(p) THEN
    FOR EACH o∈Pre(p) OR o∈Suc(p) and o has not yet been inserted into CO_new DO
      Seeds.insert(o, o.reach);
```

Fig. 2. IncOPTICS: Reorganization of the cluster ordering

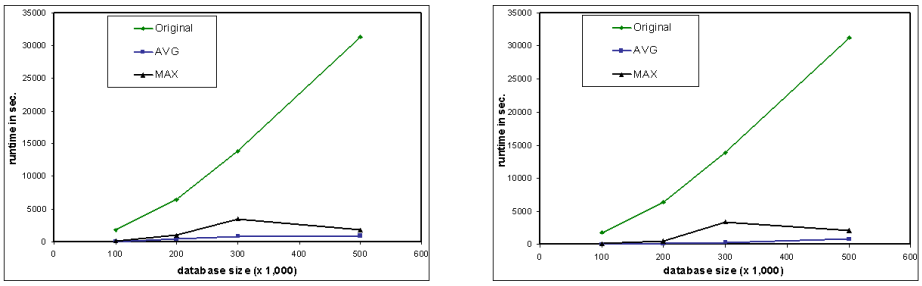
4.4 Reorganizing a Cluster Ordering

In the following, $\mathtt{CO}_{\mathtt{old}}$ denotes the old cluster ordering before the update and $\mathtt{CO}_{\mathtt{new}}$ denotes the updated cluster ordering which is computed by IncOPTICS. After the start object so has been determined according to Lemma 5, all objects $q \in \mathtt{CO}_{\mathtt{old}}$ with *Pos*$(q) <$ *Pos*(so) can be copied into $\mathtt{CO}_{\mathtt{new}}$ (cf. Fig. 1(b)) because up to the position of so $\mathtt{CO}_{\mathtt{old}}$ is a valid cluster ordering.

The reorganization of CO begins at so and imitates OPTICS. The pseudo-code of the procedure is depicted in Fig. 2. It is assumed that each not yet handled $o \in \mathcal{N}_\varepsilon(so)$ is inserted into the priority queue Seeds which manages all not yet handled objects from MOVING$(p) \cup$ MUTATING(p) (i.e. all $o \in$ MOVING$(p) \cup$ MUTATING(p) with *Pos*$(o) \geq$ *Pos*(so)) sorted in the order of ascending reachabilities.

In each step of the reorganization loop, the reachability of the first object in Seeds is compared with the reachability of the current object in $\mathtt{CO}_{\mathtt{old}}$. The entry with the smallest reachability is inserted into the next free position of $\mathtt{CO}_{\mathtt{new}}$. In case of a delete operation, this step is skipped if the considered object is the update object. After this insertion, Seeds has to be updated depending on which object has recently been inserted. If the inserted object is an element of MUTATING(p), all neighbors that are currently not yet handled may change their reachabilities. If the inserted object is an element of MOVING(p), all predecessors and successors that are currently not yet handled may move. In both cases, the corresponding objects are inserted into Seeds using the method Seeds::insert which inserts an object with its current reachability or updates the reachability of an object if it is already in the priority queue. If a predecessor is inserted into Seeds, its reachability has to be recomputed (which means a distance calculation in the worst-case) because *RDist*$(.,.)$ is not symmetric.

According to Lemma 4, the reorganization terminates if there are no more objects in Seeds, i.e. all objects in MOVING$(p) \cup$ MUTATING(p), that have to be processed, are



(a) Insertion (b) Deletion

Fig. 3. Runtime of OPTICS vs. average and maximum runtime of `IncOPTICS`.

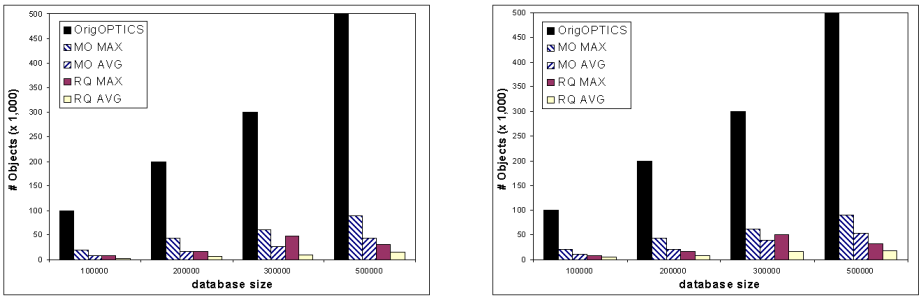
handled. $\texttt{CO}_{\texttt{new}}$ is filled with all objects from $\texttt{CO}_{\texttt{old}}$ which are not yet handled (and thus need not to be considered by the reorganization) maintaining the order determined by $\texttt{CO}_{\texttt{old}}$ (cf. Fig. 1(b)). The resulting $\texttt{CO}_{\texttt{new}}$ is valid according to Def. 3.

5 Experimental Evaluation

We evaluated `IncOPTICS` using four synthetic datasets consisting of 100,000, 200,000, 300,000, and 500,000 2-dimensional points and a real-world dataset consisting of 112,361 TV snapshots encoded as 64-dimensional color histograms. All experiments were run on a workstation featureing a 2 GHz CPU and 3,5 GB RAM. An X-Tree was used to speed up the range queries computed by OPTICS and `IncOPTICS`.

We performed 100 random updates (insertions and deletions) on each of the synthetic datasets and compared the runtime of OPTICS with the maximum and average runtimes of `IncOPTICS` (insert/delete) on the random updates. The results are depicted in Fig. 3. We observed average speed-up factors of about 45 and 25 and worst-case speed-up factors of about 20 and 17 in case of insertion and deletion, respectively. A similar observation, but on a lower level, can be made when evaluating the performance of OPTICS and `IncOPTICS` applied to the real world dataset. The worst ever observed speed-up factor for the real-world dataset was 3. In Fig. 5(a)) the average runtimes of `IncOPTICS` of the best 10 inserted and deleted objects are compared with the runtime of OPTICS using the TV dataset.

A possible reason for the large speed up is that `IncOPTICS` saves a lot of range queries. This is shown in Fig. 4(a) and 4(b) where we compared the average and maximum number of range queries and moved objects, respectively. The cardinality of the set $\text{MUTATING}(p)$ is depicted as "RQ" and the cardinality of the set $\text{MOVING}(p)$ is depicted as "MO" in the figures. It can be seen, that `IncOPTICS` saves a lot of range queries compared to OPTICS. For high dimensional data this observation is even more important since the logarithmic runtime of most index structures for a single range query degenerates to a linear runtime. Fig. 5(b) presenting the average cardinality of



(a) Insertion (b) Deletion

Fig. 4. Comparison of average and maximum cardinalities of MOVING(p) vs. MUTATING(p)

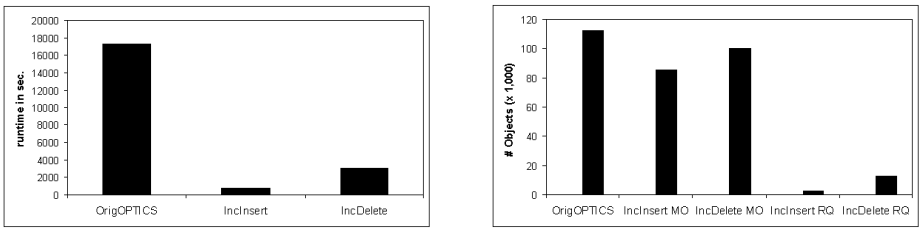
the sets of mutating objects and moving objects of incremental insertion/deletion, illustrates this effect. Since the number of objects which have to be reorganized is rather high in case of insertion or deletion the runtime speed-up is caused by the strong reduction of range queries (cf. bars "IncInsert RQ" and "IncDelete RQ" in Fig. 5(b)).

We separately analyzed the objects o whose insertions/deletions caused the highest runtime. Thereby, we found out that the biggest part of the high runtimes originated from the reorganization step due to a high cardinality of the set MOVING(o). We further observed that these objects causing high update runtimes usually are located between two clusters and objects in MUTATING(o) belong to more than one cluster. Since spatially neighboring clusters need not to be adjacent in the cluster ordering, the reorganization affects a lot more objects. This observation is important because it indicates that the runtimes are more likely near the average case than near the worst case especially for insert operations since most inserted objects will probably reproduce the distribution of the already existing data. Let us note, that since the tests on the TV Dataset were run using unfavourable objects, the performance results are less impressive than the results on the synthetic datasets.

6 Conclusions

In this paper, we proposed an incremental algorithm for mining hierarchical clustering structures based on OPTICS. Due to the density-based notion of OPTICS, insertions and deletions affect only a limited subset of objects directly, i.e. a change of their core-level may occur. We identified a second set of objects which are indirectly affected by update operations and thus they may move forward or backwards in the cluster ordering. Based on these considerations, efficient algorithms for incremental insertions and deletions of a cluster ordering were suggested.

A performance evaluation of `IncOPTICS` using synthetic as well as real-world databases demonstrated the efficiency of the proposed algorithm.



(a) Runtimes (b) Affected objects

Fig. 5. Runtimes and affected objects of `IncOPTICS` vs. OPTICS applied on the TV Data.

Comparing these results to the performance of IncrementalDBSCAN which achieves much higher speed-up factors over DBSCAN, it should be mentioned that incremental hierarchical clustering is much more complex than incremental "flat" clustering. In fact, OPTICS generates considerably more information than DBSCAN and thus `IncOPTICS` is suitable for a much broader range of applications compared to IncrementalDBSCAN.

References

1. McQueen, J.: "Some Methods for Classification and Analysis of Multivariate Observations". In: 5th Berkeley Symp. Math. Statist. Prob. Volume 1. (1967) 281–297
2. Ng, R., J., H.: "Efficient and Affective Clustering Methods for Spatial Data Mining". In: Proc. 20st Int. Conf. on Very Large Databases (VLDB'94), Santiago, Chile. (1994) 144–155
3. Zhang, T., Ramakrishnan, R., M., L.: "BIRCH: An Efficient Data Clustering Method for Very Large Databases". In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'96), Montreal, Canada. (1996) 103–114
4. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'96), Portland, OR, AAAI Press (1996) 291–316
5. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: "OPTICS: Ordering Points to Identify the Clustering Structure". In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'99), Philadelphia, PA. (1999) 49–60
6. Ester, M., Kriegel, H.P., Sander, J., Wimmer, M., Xu, X.: "Incremental Clustering for Mining in a Data Warehousing Environment". In: Proc. 24th Int. Conf. on Very Large Databases (VLDB'98). (1998) 323–333
7. Feldman, R., Aumann, Y., Amir, A., Mannila, H.: "Efficient Algorithms for Discovering Frequent Sets in Incremental Databases". In: Proc. ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, Tucson, AZ. (1997) 59–66
8. Ester, M., Wittman, R.: "Incremental Generalization for Mining in a Data Warehousing Environment". In: Proc. 6th Int. Conf. on Extending Database Technology, Valencia, Spain. Volume 1377 of Lecture Notes in Computer Science (LNCS)., Springer (1998) 135–152