

Efficient Density-Based Clustering of Complex Objects

Stefan Brecheisen, Hans-Peter Kriegel, Martin Pfeifle
Institute for Computer Science, University of Munich, Germany
{brecheisen, kriegel, pfeifle}@dbs.informatik.uni-muenchen.de

Abstract

Nowadays data mining in large databases of complex objects from scientific, engineering or multimedia applications is getting more and more important. In many different application domains complex object representations along with complex distance functions are used for measuring the similarity between objects. Often not only these complex distance measures are available but also simpler distance functions which can be computed much more efficiently. Traditionally, the well known concept of multi-step query processing which is based on exact and lower-bounding approximative distance functions is used independently of data mining algorithms. In this paper, we will demonstrate how the paradigm of multi-step query processing can be integrated into the two density-based clustering algorithms DBSCAN and OPTICS resulting in a considerable efficiency boost. Our approach tries to confine itself to ϵ -range queries on the simple distance functions and carries out complex distance computations only at that stage of the clustering algorithm where they are compulsory to compute the correct clustering result. In a broad experimental evaluation based on real-world test data sets, we demonstrate that our approach accelerates the generation of flat and hierarchical density-based clusterings by more than one order of magnitude.

1. Introduction

Effective data mining in large databases of complex objects, e.g. chemical compounds, CAD drawings, XML data, web sites or images, is a very challenging task, but often cannot be performed due to efficiency problems. An important area where this complexity problem is a strong handicap is that of density-based clustering. Density-based clustering algorithms like DBSCAN [7] and OPTICS [1] are based on ϵ -range queries for each database object. Each range query requires a lot of exact distance calculations, especially when high ϵ -values are used. Therefore, these algorithms are only applicable to large collections of complex objects, if those range queries are supported efficiently. When working with complex objects, the necessary distance calculations are the time-limiting factor. Thus, the ultimate goal is to save as many as possible of these complex distance calculations.

The core idea of our approach is to integrate the multi-step query processing paradigm directly into the clustering algo-

rithm rather than using it “only” for accelerating the single range queries. Our clustering approach itself exploits the information provided by simple distance measures lower-bounding complex and expensive exact distance functions. Expensive exact distance computations are only used when the information provided by simple distance computations, which are often based on simple object representations, is not enough to compute the exact clustering.

The remainder of this paper is organized as follows: In Section 2, we first introduce the basics of density-based clustering before discussing the flat density-based clustering algorithm DBSCAN [7] and the hierarchical density-based clustering algorithm OPTICS [1]. Then, we will present the related work on efficient density-based clustering and describe its limitations. Thereafter, we present our new approach which integrates the multi-step query processing paradigm directly into the clustering algorithms rather than using it independently. Finally, in Section 3, we present a detailed experimental evaluation showing that the presented approach can accelerate the generation of density-based clusterings on complex objects by more than one order of magnitude. We close this paper, in Section 4, with a short summary and a few remarks on future work.

2. Efficient Density-Based Clustering

In this section, we will discuss in detail how we can efficiently compute a flat (DBSCAN) and a hierarchical (OPTICS) density-based clustering. First, in Section 2.1, we present the basic concepts of density-based clustering along with the two algorithms DBSCAN and OPTICS. Then we look in Section 2.2 at different approaches presented in the literature for efficiently computing these algorithms. We will explain why the presented algorithms are not suitable for expensive distance computations if we are interested in the exact clustering structure. In Section 2.3, we will present our new approach which tries to use lower-bounding distance functions before computing the expensive exact distances.

2.1. Density-based Clustering

The key idea of density-based clustering is that for each object of a cluster the neighborhood of a given radius ϵ has to contain at least a minimum number of *MinPts* objects, i.e. the cardinality of the neighborhood has to exceed a given threshold. In the following, we will present the basic definitions of density-based clustering.

Definition 1 (directly density-reachable)

Object p is *directly density-reachable* from object q w.r.t. ϵ and $MinPts$ in a set of objects D , if $p \in N_\epsilon(q)$ and $|N_\epsilon(q)| \geq MinPts$, where $N_\epsilon(q)$ denotes the subset of D contained in the ϵ -neighborhood of q .

The condition $|N_\epsilon(q)| \geq MinPts$ is called the *core object* condition. If this condition holds for an object q , then we call q a *core object*. Other objects can be directly density-reachable only from core objects.

Definition 2 (density-reachable and density-connected)

An object p is *density-reachable* from an object q w.r.t. ϵ and $MinPts$ in the set of objects D , if there is a chain of objects $p_1, \dots, p_n, p_1 = q, p_n = p$ such that $p_i \in D$ and p_{i+1} is directly density-reachable from p_i w.r.t. ϵ and $MinPts$. Object p is *density-connected* to object q w.r.t. ϵ 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 w.r.t. ϵ and $MinPts$ in D .

Density-reachability is the transitive closure of direct density-reachability and does not have to be symmetric. On the other hand, density-connectivity is a symmetric relation.

2.1.1. DBSCAN. A flat density-based cluster is defined as a set of density-connected objects which is maximal w.r.t. density-reachability. Then the noise is the set of objects not contained in any cluster. Thus a cluster contains not only core objects but also objects that do not satisfy the core object condition. These border objects are directly density-reachable from at least one core object of the cluster.

The algorithm DBSCAN [7], which discovers the clusters and the noise in a database, is based on the fact that a cluster is equivalent to the set of all objects in O which are density-reachable from an arbitrary core object in the cluster (cf. lemma 1 and 2 in [7]). The retrieval of density-reachable objects is performed by iteratively collecting directly density-reachable objects. DBSCAN checks the ϵ -neighborhood of each point in the database. If the ϵ -neighborhood $N_\epsilon(q)$ of a point q has more than $MinPts$ elements, q is a so-called *core point*, and a new cluster C containing the objects in $N_\epsilon(q)$ is created. Then, the ϵ -neighborhood of all points p in C which have not yet been processed is checked. If $N_\epsilon(p)$ contains more than $MinPts$ points, the neighbors of p which are not already contained in C are added to the cluster and their ϵ -neighborhood is checked in the next step. This procedure is repeated until no new point can be added to the current cluster C . Then the algorithm continues with a point which has not yet been processed trying to expand a new cluster.

2.1.2. OPTICS. While the partitioning density-based clustering algorithm DBSCAN [7] can only identify a “flat” clustering, the newer algorithm OPTICS [1] computes an ordering of the points augmented by additional information, i.e. the *reachability-distance*, representing the intrinsic hierarchical

Algorithm OPTICS:

```

repeat {
  if the seedlist is empty {
    if all points are marked “done”, terminate;
    choose “not-done” point  $q$ ;
    add ( $q$ , infinity) to the seedlist;
  }
  ( $o_1, r$ ) = seedlist entry with smallest reachability value;
  remove ( $o_1, r$ ) from seedlist;
  mark  $o_1$  as “done”;
  output ( $o_1, r$ );
  update-seedlist( $o_1$ );
}

```

Figure 1. The OPTICS algorithm.

(nested) cluster structure. The result of OPTICS, the cluster-ordering, is displayed by the so-called reachability plot. Thus, it is possible to explore interactively the clustering structure, offering additional insights into the distribution and correlation of the data.

In the following, we will shortly introduce the definitions underlying the OPTICS algorithm, the *core-distance* of an object p and the *reachability-distance* of an object p w.r.t. a predecessor object o .

Definition 3 (core-distance)

Let p be an object from a database D , let ϵ be a distance value, let $N_\epsilon(p)$ be the ϵ -neighborhood of p , let $MinPts$ be a natural number and let $MinPts\text{-}dist(p)$ be the distance of p to its $MinPts$ -th neighbor. Then, the *core-distance* of p , denoted as $core\text{-}dist_{\epsilon, MinPts}(p)$ is defined as $MinPts\text{-}dist(p)$ if $|N_\epsilon(p)| \geq MinPts$ and *UNDEFINED* otherwise.

Definition 4 (reachability-distance)

Let p and o be objects from a database D , let $N_\epsilon(o)$ be the ϵ -neighborhood of o , let $dist(o, p)$ be the distance between o and p , and let $MinPts$ be a natural number. Then, the *reachability-distance* of p w.r.t. o denoted as $reachability\text{-}dist_{\epsilon, MinPts}(p, o)$ is defined as $\max(core\text{-}dist_{\epsilon, MinPts}(o), dist(o, p))$ if $|N_\epsilon(o)| \geq MinPts$ and *UNDEFINED* otherwise.

The OPTICS algorithm (cf. Figure 1) creates an ordering of a database, along with a reachability-value for each object. Its main data structure is a *seedlist*, containing tuples of points and reachability-distances. The seedlist is organized w.r.t. ascending reachability-distances. Initially the seedlist is empty and all points are marked as *not-done*.

The procedure update-seedlist (o_1) executes an ϵ -range query around the point o_1 , i.e. the first object of the sorted seedlist, at the beginning of each cycle. For every point p in $N_\epsilon(o_1)$ it computes $r = reachability\text{-}dist_{\epsilon, MinPts}(p, o_1)$. If the seedlist already contains an entry (p, s) , it is updated to $(p, \min(r, s))$, otherwise (p, r) is added to the seedlist. Finally, the order of the seedlist is reestablished.

2.2. Related Work

DBSCAN and OPTICS determine the local densities by repeated range queries. In this section, we will sketch different approaches from the literature to accelerate these density-based clustering algorithms and discuss their unsuitability for complex object representations.

Multi-Dimensional Index Structures. The most common approach to accelerate each of the required single range queries is to use multi-dimensional index structures. For objects modelled by low-, medium-, or high-dimensional feature vectors there exist several specific R-tree [10] variants. For more detail we refer the interested reader to [9].

Metric Index Structures. Besides feature vectors, there exist quite a few other promising and approved modelling approaches for complex objects, e.g. trees, graphs, and vector sets, which cannot be managed by the index structures mentioned in the last paragraph. Nevertheless, we can use index structures, such as the M-tree [6] for efficiently carrying out range queries as long as we have a metric distance function for measuring the similarity between two complex objects. For a detailed survey on metric access methods we refer the reader to [5].

Multi-Step Query Processing. The main goal of multi-step query processing is to reduce the number of complex and, therefore, time consuming distance calculations in the query process. In order to guarantee that there occur no false drops, the used filter distances have to fulfill a lower-bounding distance criterion. For any two objects p and q , a lower-bounding distance function d_f in the filter step has to return a value that is not greater than the exact object distance d_o of p and q , i.e. $d_f(p, q) \leq d_o(p, q)$. With a lower-bounding distance function it is possible to safely filter out all database objects which have a filter distance greater than the current query range because the exact object distance of those objects cannot be less than the query range. Using a multi-step query architecture requires efficient algorithms which actually make use of the filter step. Agrawal, Faloutsos and Swami proposed such an algorithm for range queries [2] which form the foundation of density-based clustering. For efficiency reasons, it is crucial that $d_f(p, q)$ is considerably faster to evaluate than $d_o(p, q)$, and, furthermore, in order to achieve a high selectivity $d_f(p, q)$ should be only marginally smaller than $d_o(p, q)$.

Using Multiple Similarity Queries. In [3] a schema was presented which transforms query intensive KDD algorithms into a representation using the similarity join as a basic operation without affecting the correctness of the result of the considered algorithm. The approach was applied to accelerate the clustering algorithm DBSCAN and the hierarchical cluster structure analysis method OPTICS by using an R-tree like index structure. In [4] an approach was introduced for efficiently supporting multiple similarity queries for mining in metric databases.

It was shown that many different data mining algorithms can be accelerated by multiplexing different similarity queries.

Summary. Multi-dimensional index structures based on R-tree variants and clustering based on the similarity join are restricted to vector set data. Furthermore, the main problem of all approaches mentioned above is that distance computations can only be avoided for objects located outside the ϵ -range of the actual query object. In order to create, for instance, a reachability plot without loss of information, the authors in [1] propose to use a very high ϵ -value. Therefore, all of the above mentioned approaches lead to $O(|DB|^2)$ exact distance computations for OPTICS.

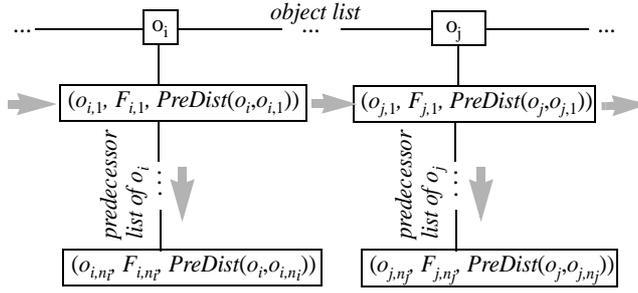
Furthermore, there exist other approaches which do not aim at producing the exact density-based clustering structure, but try to compute efficiently an approximated one. In this paper, we will propose an approach which computes an exact density-based clustering trying to confine itself to simple distance computations lower-bounding the exact distances. Basically, we do not carry out ϵ -range queries on the exact object distances but *MinPts*-nearest-neighbor queries on the exact object distances which are based on ϵ -range queries on the filter information. Further expensive exact distance computations are postponed as long as possible, and are only carried out at that stage of the algorithm where they are compulsory to compute the exact clustering.

2.3. Accelerated Density-Based Clustering

In this section, we will demonstrate how to integrate the multi-step query processing paradigm into the two density-based clustering algorithms DBSCAN and OPTICS. We discuss in detail our approach for OPTICS and sketch how a simplified version of this extended OPTICS approach can be used for DBSCAN.

2.3.1. Basic Idea. DBSCAN and OPTICS are both based on numerous ϵ -range queries. None of the approaches discussed in literature can avoid that we have to compute the exact distance to a given query object q for all objects contained in $N_\epsilon(q)$. Especially for OPTICS, where ϵ has to be chosen very high in order to create reachability plots without loss of information, we have to compute $|DB|$ exact distance computations for each single range query, even when one of the methods discussed in Section 2.2 is used. In the case of DBSCAN, typically, the ϵ -values are much smaller. Nevertheless, if we apply the traditional multi-step query processing paradigm with non-selective filters, we also have to compute up to $|DB|$ many exact distance computations.

In our approach, the number of exact distance computations does not primarily depend on the size of the database and the chosen ϵ -value but rather on the value of *MinPts*, which is typically only a small fraction of $|DB|$, e.g. *MinPts* = 5 is a suitable value even for large databases [1, 7]. Basically, we use



➔ ordered *object list* such that the following conditions hold:
DBSCAN: $(i < j) \wedge (PL(o_i) \neq NIL) \Rightarrow (PL(o_j) \neq NIL \wedge PreDist(o_i, o_{i,1}) \leq PreDist(o_j, o_{j,1}))$
OPTICS: $(i < j) \Rightarrow PreDist(o_i, o_{i,1}) \leq PreDist(o_j, o_{j,1})$

⬇ ordered *predecessor lists* such that the following conditions hold:
DBSCAN: $\forall i: (l < k) \Rightarrow PreDist(o_i, o_l) \leq PreDist(o_i, o_k)$
OPTICS: $\forall i: (l < k) \Rightarrow PreDist(o_i, o_l) \leq PreDist(o_i, o_k)$

Figure 2. Data structure Xseedlist.

MinPts-nearest neighbor queries instead of ϵ -range queries on the exact object representations in order to determine the “core-properties” of the objects. Further exact distance computations are only carried out at that stage of the algorithms where they are compulsory to compute the correct clustering result.

2.3.2. Extended OPTICS. The main idea of our approach is to carry out the range queries based on the lower-bounding filter distances instead of using the expensive exact distances. In order to put our approach into practice, we have to slightly extend the data structure underlying the OPTICS algorithm, i.e. we have to add additional information to the elements stored in the seedlist.

The Extended Seedlist. We do not any longer use a single seedlist as in the original OPTICS algorithm (cf. Figure 1) where each list entry consists of a pair (*ObjectId*, *ReachabilityValue*). Instead, we use a list of lists, called *Xseedlist*, as shown in Figure 2. The *Xseedlist* consists of an ordered list of objects, called *object list*, quite similar to the original seedlist but without any reachability information. The order of this object list, cf. the horizontal arrow in Figure 2, is determined by the first element of the second list anchored at each object of the first list. This second list is called *predecessor list PL*, cf. the vertical arrows in Figure 2.

An entry located at position l of the *predecessor list PL*(o_i) belonging to object o_i consists of the following information:

- **Predecessor ID.** An object $o_{i,l}$ which was already reported throughout the OPTICS run, i.e. $o_{i,l}$ was already added to the reachability plot, which is computed from left to right.

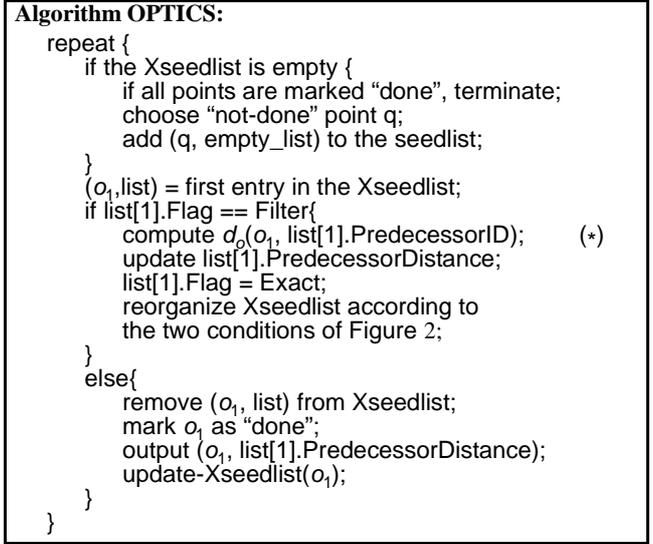


Figure 3. The extended OPTICS algorithm.

- **Filter Flag.** A flag F indicating whether we already computed the exact object distance between o_i and $o_{i,l}$, i.e. $d_o(o_i, o_{i,l})$, or whether we only computed the distance of these two objects based on the lower-bounding filter information, i.e. $d_f(o_i, o_{i,l})$.
- **Predecessor Distance.** $PreDist(o_i, o_{i,l})$ is equal to either $\max(\text{core-dist}_{\epsilon, \text{MinPts}}(o_{i,l}), d_o(o_i, o_{i,l}))$ or to $d_f(o_i, o_{i,l})$ dependent on the fact whether we already computed the exact object distance $d_o(o_i, o_{i,l})$ or only the filter distance $d_f(o_i, o_{i,l})$.

Throughout our new algorithm, the conditions depicted in Figure 2 belonging to this extended OPTICS algorithm are maintained. In the following, we will describe the extended OPTICS algorithm trying to minimize the number of exact distance computations.

Algorithm. The extended OPTICS algorithm exploiting the filter information is depicted in Figure 3. The algorithm always takes the first element o_1 from the sorted *object list*. If it is at the first position due to a filter computation, we compute the exact distance $d_o(o_1, o_{1,1})$ and reorganize the *Xseedlist*. The reorganization might displace $o_{1,1}$ from the first position of $PL(o_1)$. Furthermore, object o_1 might be removed from the first position of the *object list*. On the other hand, if the filter flag $F_{1,1}$ indicates that an exact distance computation was already carried out, we add object o_1 to the reachability plot with a reachability-value equal to $PreDist(o_1, o_{1,1})$. Furthermore, we carry out the procedure *update-Xseedlist*(o_1).

Update-Xseedlist. This is the core function of our extended OPTICS algorithm. First, we carry out a range query around object o_1 based on the filter information. Then we compute the core-level of the current query object o_1 by computing the *MinPts*-nearest neighbors of o_1 as follows:

- We carry out an ε -range query around o_1 based on the filter information, yielding the result set $N_\varepsilon^{filter}(o_1)$.
- We order all elements in $N_\varepsilon^{filter}(o_1)$ in ascending order according to their filter distance to o_1 yielding a $SortList_\varepsilon(o_1)$.
- We walk through $SortList_\varepsilon(o_1)$ starting at the first element. For the first element we compute the exact distance and reorder the $SortList_\varepsilon(o_1)$ which might move o_1 upward in this sorted list. This step is repeated until the first $MinPts$ elements of $SortList_\varepsilon(o_1)$ are at their final position due to an exact distance computation. The core-level of our current query object o_1 is equal to the distance between o_1 and the object stored at the $MinPts$ -th position of the final $SortList_\varepsilon(o_1)$.

Some of the elements $o_j \in N_\varepsilon^{filter}(o_1)$ along with their actual reachability values w.r.t. o_1 are inserted into the $Xseedlist$.

- Elements o_j for which we already computed the exact distance to o_1 and for which $o_j \in N_\varepsilon(o_1)$ holds, are inserted as follows: If there exists no entry in the *object list* for o_j , $(o_j, \langle o_1, Exact, \max(d_o(o_j, o_1), core-dist_{\varepsilon, MinPts}(o_1)) \rangle)$ is inserted into the *object list*. If there already exists an entry in the *object list* belonging to o_j , $(o_j, \langle o_1, Exact, \max(d_o(o_j, o_1), core-dist_{\varepsilon, MinPts}(o_1)) \rangle)$ is inserted into $PL(o_j)$. Note that in both cases the ordering of Figure 2 has to be maintained. On the other hand, if $o_j \notin N_\varepsilon(o_1)$, o_j is not inserted into the $Xseedlist$.
- If we have not yet computed $d_o(o_j, o_1)$, o_j is inserted into the $Xseedlist$. If there exists no entry in the *object list* belonging to o_j , $(o_j, \langle o_1, Filter, d_f(o_j, o_1) \rangle)$ is inserted into the *object list*. If there already exists an entry in the *object list* for o_j , $(o_j, \langle o_1, Filter, d_f(o_j, o_1) \rangle)$ is inserted into $PL(o_j)$. Again, the ordering of Figure 2 has to be maintained.

Note that this approach carries out exact distance computations only for those objects o which are very close to the actual query object q according to the filter information. On the other hand, the traditional multi-step query approach would compute exact distance computations for all objects $o \in N_\varepsilon^{filter}(q)$. As ε has to be chosen very high in order to create reachability plots without loss of information [1], the traditional approach has to compute $|DB|$ exact distance computations, even when one of the approaches discussed in Section 2.2 is used. On the other hand, the number of exact distance computations in our approach does not depend on the size of the database but rather on the value of $MinPts$, which is only a small fraction of the cardinality of the database. Note that our approach only has to compute $|DB| \cdot MinPts$, i.e. $O(|DB|)$, exact distance computations if we assume an optimal filter, in contrast to the $O(|DB|^2)$ distance computations carried out by the original OPTICS run. Only if necessary, we carry out further

additional exact distance computations (cf. line (*) in Figure 3).

2.3.3. Extended DBSCAN. Our extended DBSCAN algorithm is a simplified version of the extended OPTICS algorithm using also the $Xseedlist$ as the main data structure. Again, we carry out an ε -range query for each database object q on the lower-bounding filter distances yielding a result set $N_\varepsilon^{filter}(q)$. Due to the lower-bounding properties of the filters, $N_\varepsilon(q) \subseteq N_\varepsilon^{filter}(q)$ holds. Therefore, if $|N_\varepsilon^{filter}(q)| < MinPts$ holds, q is certainly no core-point. Otherwise, we test whether q is a core-point as follows.

We organize all $o \in N_\varepsilon^{filter}(q)$ in ascending order according to their filter distance $d_f(o, q)$ yielding a $SortList_\varepsilon(q)$. We walk through this sorted list, and compute for each visited object o_i the exact distance $d_o(o_i, q)$ until for $MinPts$ elements $d_o(o_i, q) \leq \varepsilon$ holds or until we reach the end. If we reached the end, we certainly know that q is no core point. Otherwise q is a core point and in the case of DBSCAN this information is enough. The main difference to the extended OPTICS algorithm is that we do not have to reorder $SortList_\varepsilon(q)$, as we do not have to compute the core-level of q .

If our current object q is a core object, some of the objects $o_i \in N_\varepsilon^{filter}(q)$ are inserted into the $Xseedlist$ (cf. Figure 2). All objects for which we have already computed $d_o(o_i, q)$, and for which $d_o(o_i, q) \leq \varepsilon$ holds, certainly belong to the same cluster as the core-object q . At the beginning of the *object list*, we add the entry (o_i, NIL) , where $PL(o_i) = NIL$ indicates that o_i certainly belongs to the same cluster as q . Objects o_i for which $d_o(o_i, q) > \varepsilon$ holds are discarded. All objects $o \in N_\varepsilon^{filter}(q)$ for which we did not yet compute $d_o(o_i, q)$ are handled as follows:

- If there exists no entry in the *object list* belonging to o_i , $(o_i, \langle q, Filter, d_f(o_i, q) \rangle)$ is inserted into the *object list* in such a way that the ordering conditions of Figure 2 still hold.
- If there already exists an entry in the *object list* for o_i and, furthermore, $PL(o_i) = NIL$ holds, nothing is done.
- If there already exists an entry in the *object list* for o_i and, furthermore, $PL(o_i) \neq NIL$ holds, $(q, Filter, d_f(o_i, q))$ is inserted into $PL(o_i)$ in such a way that the ordering conditions of Figure 2 still hold.

DBSCAN expands a cluster C as follows. We take the first element o_1 from the object list and, if $PL(o_1) = NIL$ holds, we add o_1 to the current cluster, delete o_1 from the *object list*, carry out a range query around o_1 , and try to expand the cluster C . If $PL(o_1) \neq NIL$ holds, we compute $d_o(o_1, o_{1,1})$. If $d_o(o_1, o_{1,1}) \leq \varepsilon$, we process similar to the case where $PL(o_1) = NIL$ holds. If $d_o(o_1, o_{1,1}) > \varepsilon$ holds and length of $PL(o_1) > 1$, we delete $(o_{1,1}, F_{1,1}, PreDist(o_1, o_{1,1}))$ from $PL(o_1)$. If $d_o(o_1, o_{1,1}) > \varepsilon$ holds

and length of $PL(o_1) = 1$, we delete o_1 from the *object list*. Iteratively, we try to expand the current cluster by examining the first entry of $PL(o_1)$ until the current *object list* is empty.

2.3.4. Length-Limitation of the Predecessor Lists. In this section, we introduce an approach for limiting the size of the predecessor lists to a constant l_{max} trying to keep the main memory footprint as small as possible.

OPTICS. For each object o_i in the *object list*, we store all potential predecessor objects $o_{i,p}$ along with $PreDist(o_i, o_{i,p})$ in $PL(o_i)$. Due to the lower-bounding property of d_p , we can delete all entries in $PL(o_i)$ which are located at positions $l' > l$, if we have already computed the exact distance between o_i and the predecessor object $o_{i,l}$ located at position l . So each exact distance computation might possibly lead to several delete operations in the corresponding predecessor list. In order to limit the main memory footprint, we introduce a parameter l_{max} which restricts the allowed number of elements stored in a predecessor list. If more than l_{max} elements are contained in the list, we compute the exact distance for the predecessor $o_{i,1}$ located at the first position. Such an exact distance computation between o_i and $o_{i,1}$ usually causes $o_{i,1}$ to be moved upward in the list. All elements located behind its new position l are deleted. So if $l \leq l_{max}$ holds, the *predecessor list* is limited to at most l_{max} entries. Otherwise, we repeat the above procedure.

DBSCAN. If the predecessor list of o_i is not NIL, we can limit its length by starting to compute $d_o(o_i, o_{i,1})$, i.e. the exact distance between o_i and the first element of $PL(o_i)$. If $d_o(o_i, o_{i,1}) \leq \epsilon$ holds, we set $PL(o_i) = \text{NIL}$ indicating that o_i certainly belongs to the current cluster. Otherwise, we delete $(o_{i,1}, F_{i,1}, PreDist(o_i, o_{i,1}))$ and if the length of $PL(o_i)$ is still larger than l_{max} , we iteratively repeat this limitation procedure.

3. Evaluation

In this section, we present a detailed experimental evaluation which demonstrates the characteristics and benefits of our new approach.

3.1. Settings

Test Data Sets. As test data, we used real-world CAD data represented by 81-dimensional feature vectors [13] and vector sets consisting of 7 6D points [12]. Furthermore, we used graphs [14] to represent real-world image data. If not otherwise stated, we used 1,000 complex objects from each data set. The used filter and exact object distance functions can be characterized as follows:

- The exact distance computations on the graphs are very expensive. On the other hand, the used filter is rather selective and can efficiently be computed [14].
- The exact distance computations on the feature vectors and vector sets are also very expensive as normalization aspects for the CAD objects are taken into account. We compute 48 times the distance between two 81-dimensional feature vectors, and between two vector sets, in order to determine a normalized distance between two CAD objects [12, 13]. The filter used for the feature vectors is not very selective, but can be computed very efficiently as we only have to compute once the distance between two numerical values. The filter used for the vector sets is more selective than the filter for the feature vectors but also computationally more expensive.

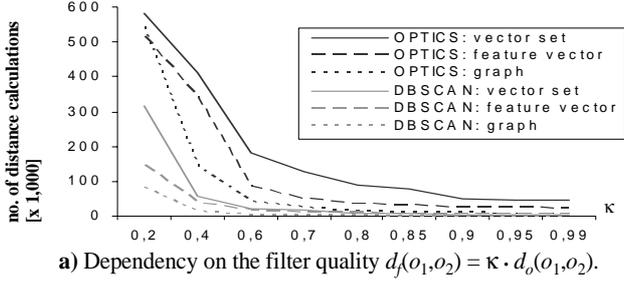
Implementation. The original OPTICS and DBSCAN algorithms, along with their extensions introduced in this paper and the used filter and exact object distances were implemented in Java 1.4. The experiments were run on a workstation with a Xeon 2.4 GHz processor and 2 GB main memory under Linux.

Parameter Setting. As suggested in [1], we used for an OPTICS run a maximum ϵ -parameter in order to create reachability plots containing the complete hierarchical clustering information. For DBSCAN, we chose an ϵ -parameter yielding as many flat clusters as possible. Furthermore, if not otherwise stated, the *MinPts*-parameter is set to 5, the length of the predecessor lists is not limited, and the used filters are the ones sketched above.

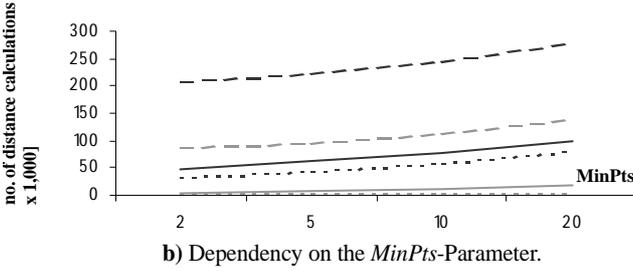
Comparison Partners. As a comparison partner for extended OPTICS, we chose the full table scan based on the exact distances, because any other approach would include an unnecessary overhead and is not able to reduce the number of the required $|DB|^2$ exact distance computations. Furthermore, we compared our extended DBSCAN algorithm to the original DBSCAN algorithm based on a full table scan on the exact object distances, and we compared it to a version of DBSCAN which is based on ϵ -range queries efficiently carried out according to the multi-step query processing paradigm [2]. According to all our tests, this second comparison partner outperforms a DBSCAN algorithm using ϵ -range queries based on an M-tree [6] and the DBSCAN algorithm according to [4].

3.2. Experiments

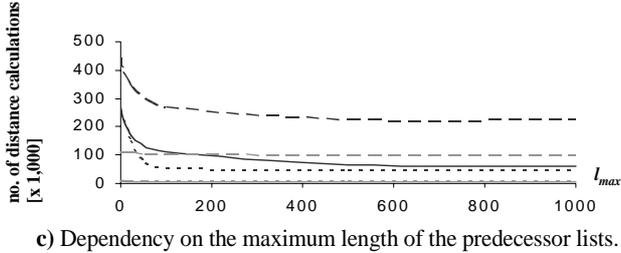
In this section, we first investigate the dependency of our approach on the filter quality, the *MinPts*-parameter, and the maximum allowed length of the predecessor lists. For these tests, we concentrate on the discussion of the overall number of distance computations. Furthermore, we investigate the influence of the ϵ -value in the case of DBSCAN, and, finally, we present the absolute runtimes, in order to show that the required overhead of our approach is negligible compared to the saved exact distance computations.



a) Dependency on the filter quality $d_f(o_1, o_2) = \kappa \cdot d_o(o_1, o_2)$.



b) Dependency on the *MinPts*-Parameter.



c) Dependency on the maximum length of the predecessor lists.

Figure 4. Dependency on various parameters.

Dependency on the Filter Quality. In order to demonstrate the dependency of our approach on the quality of the filters, we utilized in a first experiment artificial filter distances d_f lower bounding the exact object distances d_o , i.e. $d_f(o_1, o_2) = \kappa \cdot d_o(o_1, o_2)$ where κ is between 0 and 1. Figure 4a depicts the number of distance computations n_{dist} w.r.t. κ . In the case of DBSCAN, even rather bad filters, i.e. small values of κ , help to reduce the number of required distance computations considerably, indicating a possible high speed up compared to both comparison partners of DBSCAN. For good filters, i.e. values of κ close to 1, n_{dist} is very small for DBSCAN and OPTICS indicating a possible high speed up compared to a full table scan based on the exact distances d_o .

Dependency on the *MinPts*-Parameter. Figure 4b demonstrates the dependency of our approach for a varying *MinPts*-parameter while using the filters introduced in [12, 13, 14]. As our approach is based on *MinPts*-nearest neighbor queries, obviously, the efficiency of our approach is the better the smaller the *MinPts*-parameter. Note that even for rather high *MinPts*-values around $10 = 1\% \cdot |DB|$, our approach saves up to one order of magnitude of exact

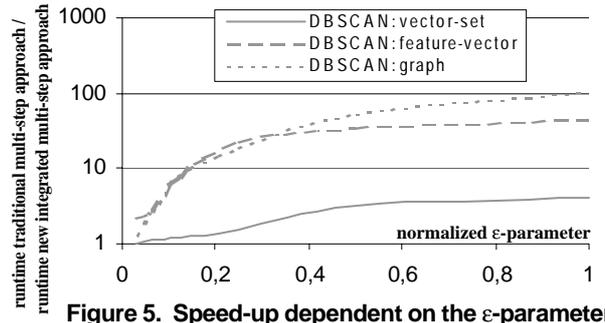


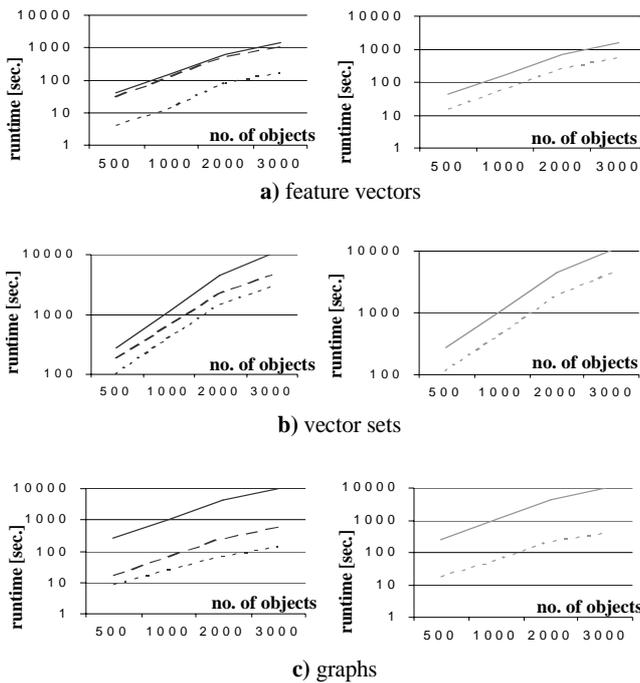
Figure 5. Speed-up dependent on the ϵ -parameter.

distance computations compared to a full table scan based on d_o , if selective filters are used, e.g. the filters for the vector sets and the graphs. Furthermore, even for the filter of rather low selectivity used by the feature vectors, our approach needs only 1/9 of the maximum number of distance computations in the case of DBSCAN and about 1/4 in the case of OPTICS.

Dependency on the Maximum Allowed Length of the Predecessor Lists. Figure 4c depicts how the number of distance computations n_{dist} depends on the available main memory, i.e. the maximum allowed length l_{max} of the predecessor lists. Obviously, the higher the value for l_{max} , the less exact distance computations are required. The figure shows that for OPTICS we have an exponential decrease of n_{dist} w.r.t. l_{max} and for DBSCAN n_{dist} is almost constant w.r.t. changing l_{max} parameters indicating that small values of l_{max} are sufficient to reach the best possible runtimes.

Dependency on the ϵ -parameter. Figure 5 shows how the speed-up for DBSCAN between our integrated multi-step query processing approach and the traditional multi-step query processing approach depends on the chosen ϵ -parameter. The higher the chosen ϵ -parameter, the more our new approach outperforms the traditional one which has to compute the exact distances between o and q for all $o \in N_{\epsilon}^{filter}(q)$. In contrast, our approach confines itself to *MinPts*-nearest neighbor queries on the exact distances and computes further distances only if compulsory to compute the exact clustering result.

Absolute Runtimes. Figure 6 presents the absolute runtimes of the new extended DBSCAN and OPTICS algorithms which integrate the multi-step query processing paradigm compared to the full-table scan on the exact object representations. Furthermore, we compare our extended DBSCAN also to a DBSCAN variant using ϵ -range queries based on the traditional multi-step query processing paradigm. Note, that this comparison partner would induce an unnecessary overhead in the case of OPTICS where we have to use very high ϵ -parameters in order to detect the complete hierarchical clustering order. In all experiments, our approach was always the most efficient one. For instance, for DBSCAN on the feature vectors, our approach



DBSCAN	OPTICS	
————	————	full table scan
- - - -	- - - -	traditional multi-step approach
· · · ·	· · · ·	integrated multi-step approach

Figure 6. Absolute runtimes w.r.t. varying database sizes. (left: DBSCAN, right: OPTICS)

outperforms both comparison partners by an order of magnitude indicating that already rather bad filters are useful for our new extended DBSCAN algorithm. Note that the traditional multi-step query processing approach does not benefit much from non-selective filters even when small ϵ -values are used. In the case of OPTICS, the performance of our approach improves with increasing filter quality. For instance, for the graphs we achieve a speed-up of more than 30 indicating as well the suitability of our extended OPTICS algorithm.

4. Conclusion

In many different application areas, density-based clustering is an effective approach for mining complex data. Unfortunately, the runtime of these data-mining algorithms is very high, as the distance functions between complex object representations are often very expensive. In this paper, we showed how to integrate the well-known multi-step query processing paradigm directly into the two density-based clustering algorithms DBSCAN and OPTICS. We replaced the expensive exact ϵ -range queries by *MinPts*-nearest neighbor queries which themselves are based on ϵ -range queries on the

lower-bounding filter distances. Further exact complex distance computations are only carried out at that stage of the algorithms where they are compulsory to compute the correct clustering result.

In a broad experimental evaluation based on real-world test data sets we demonstrated that our new approach leads to a significant speed-up compared to a full-table scan on the exact object representations as well as compared to an approach, where the ϵ -range queries are accelerated by means of the traditional multi-step query processing concept.

In our future work, we will demonstrate that other data mining algorithms dealing with complex object representations also benefit from a direct integration of the multi-step query processing paradigm.

References

- [1] Ankerst M., Breunig M., Kriegel H.-P., Sander J.: *OPTICS: Ordering Points To Identify the Clustering Structure*. SIGMOD'99, pp. 49-60.
- [2] Agrawal R., Faloutsos C., Swami A. *Efficient Similarity Search in Sequence Databases*. FODO'93, pp. 69-84.
- [3] Böhm C., Braunmüller B., Breunig M., Kriegel H.-P.: *High Performance Clustering Based on the Similarity Join*. CIKM'00, pp. 298-313.
- [4] Braunmüller B., Ester M., Kriegel H.-P., Sander J.: *Efficiently Supporting Multiple Similarity Queries for Mining in Metric Databases*. ICDE'00, pp. 256-267.
- [5] Chávez E., Navarro G., Baeza-Yates R., Marroquín J.: *Searching in Metric Spaces*. ACM Computing Surveys 33(3): pp. 273-321, 2001.
- [6] Ciaccia P., Patella M., Zezula P.: *M-tree: An Efficient Access Method for Similarity Search in Metric Spaces*. VLDB'97, pp. 426-435.
- [7] Ester M., Kriegel H.-P., Sander J., Xu X.: *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*. KDD'96, pp. 226-231.
- [8] Fonseca M. J., Jorge J. A.: *Indexing High-Dimensional Data for Content-Based Retrieval in Large Databases*. DASFAA'03, pp. 267-274.
- [9] Gaede V., Günther O.: *Multidimensional Access Methods*. ACM Computing Surveys 30(2), pp. 170-231, 1998.
- [10] Guttman A.: *R-trees: A Dynamic Index Structure for Spatial Searching*. SIGMOD'84, pp. 47-57.
- [11] Jain A. K., Dubes R. C.: *Algorithms for Clustering Data*, Prentice-Hall, 1988.
- [12] Kriegel H.-P., Brecheisen S., Kröger P., Pfeifle M., Schubert M.: *Using Sets of Feature Vectors for Similarity Search on Voxalized CAD Objects*. SIGMOD'03, pp. 587-598.
- [13] Kriegel H.-P., Kröger P., Mashaël Z., Pfeifle M., Pötke M., Seidl T.: *Effective Similarity Search on Voxalized CAD Objects*. DASFAA'03, pp. 27-36.
- [14] Kriegel H.-P., S. Schönauer S.: *Similarity search in structured data*. DAWAK'03, pp. 309-319.