Knowl Inf Syst (2006) 10(2): 211–227
DOI 10.1007/s10115-006-0018-6

Knowledge and Information Systems

SHORT PAPER

Karin Kailing · Hans-Peter Kriegel · Martin Pfeifle · Stefan Schönauer

Extending metric index structures for efficient range query processing

Received: 22 August 2004 / Revised: 24 January 2005 / Accepted: 26 March 2005 /
Published online: 15 March 2006

Abstract Databases are getting more and more important for storing complex objects from scientific, engineering, or multimedia applications. Examples for such data are chemical compounds, CAD drawings, or XML data. The efficient search for similar objects in such databases is a key feature. However, the general problem of many similarity measures for complex objects is their computational complexity, which makes them unusable for large databases. In this paper, we combine and extend the two techniques of metric index structures and multi-step query processing to improve the performance of range query processing. The efficiency of our methods is demonstrated in extensive experiments on real-world data including graphs, trees, and vector sets.

Keywords Complex objects · Metric indexing · Multi-step query processing · Density-based clustering

1 Introduction

Databases are getting more and more important for storing complex objects from scientific, engineering, or multimedia applications. Examples for such data are chemical compounds, CAD drawings, XML data, web sites, or color images. The efficient search for similar objects in such databases, for example to classify new objects or to cluster database objects, is a key feature in those application domains. Often a feature transformation is not possible, therefore a simple distance function like the Euclidean distance cannot be used. In this case, the use of more complex distance functions like the edit distance for graphs or trees is necessary. However,

K. Kailing (✉) · H.-P. Kriegel · M. Pfeifle · S. Schönauer
Institute for Computer Science, University of Munich, Oettingenstr. 67, 80538 Munich, Germany
E-mail: {kailing, kriegel, pfeifle, schoenauer}@dbs.informatik.uni-muenchen.de

a general problem of all such measures is their computational complexity, which disqualifies their use for large databases.

An area where this complexity problem is a strong handicap is that of clustering, one of the primary data mining tasks. Density-based clustering has proved to be successful for clustering complex objects [8, 10]. Density-based clustering algorithms like DBSCAN [6] or OPTICS [2] are based on range queries for each database object. 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. For complex objects, distance calculations are often significantly more expensive than disk accesses.

One approach to improve the performance of range queries is to use a filter-refinement architecture. The core idea is to apply a filter criterion to the database objects in order to obtain a small set of candidate answers to a query. The final result is then retrieved from this candidate set through the use of the complex similarity measure. This reduces the number of expensive object distance calculations and speeds up the search process.

Another possibility is the use of a metric index structure. In [4], several efficient access methods for similarity search in metric spaces are presented. In most real-world applications, a static index structure is not acceptable, so dynamic index structures like the M-tree [5] are applied.

So far, both aforementioned concepts, multi-step query processing and metric index structures, have only been used separately. We claim that these concepts can beneficially be combined and that through the combination a significant speed-up compared to both separate approaches can be achieved. In this paper, we discuss how the two approaches can be combined and present some other techniques to improve the efficiency of range query processing. Filters can easily be used to speed-up the creation and the traversing of a metric index structure like the M-tree. Additionally, caching can be used to prevent the same distance calculations to be performed more than once.

The remainder of the paper is organized as follows. In Sect. 2, we present some recent work in the field of indexing and clustering complex objects. Section 3 presents our techniques used to save costly distance calculations while performing range queries. The performance gain of our new techniques is presented in Sect. 4, while Sect. 5 concludes the paper and gives some hints at future work.

2 Motivation and related work

In the next section, we present three promising and approved modeling approaches and distance measures for complex objects (see Fig. 1 for an illustration). The evaluation part will show that in all those cases we achieve a performance gain using our new techniques. Afterwards, we present some recent approaches for clustering and query processing on complex objects.

As this is an extremely broad field, we do not make any claim on completeness. The main purpose of this section is to motivate the necessity of new techniques which allow efficient similarity range queries on complex objects.

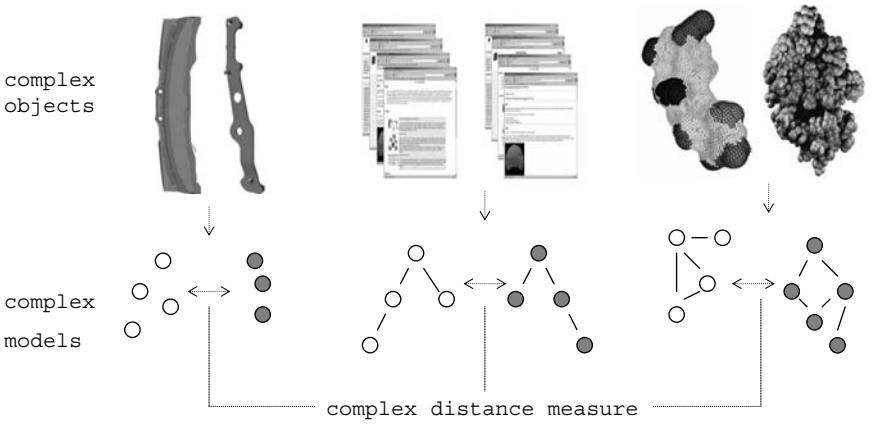

Fig. 1 Examples of complex objects

2.1 Data types of complex objects

In [10], an effective and flexible similarity model for complex 3D CAD data is introduced, which helps to find and group similar parts. It is not based on the traditional approach of describing one object by a single feature vector, but instead an object is mapped onto a set of feature vectors, i.e. an object is described by a vector set. The similarity measure for comparing two such vector sets has cubic time complexity, which makes calculating even a single similarity distance an expensive operation.

In addition to a variety of content-based attributes, complex objects typically carry some kind of internal structure, which often forms a hierarchy. A successful approach is to use degree-2 edit distance [18], which has been applied to trees for web site analysis [17], structural similarity of XML documents [14], shape recognition [15], or chemical substructure search [17].

Attributed graphs are another natural way to model structured data. Most of the known similarity measures for attributed graphs are either limited to a special type of graph or are computationally extremely complex, i.e. NP complete. A new measure for attributed graphs with cubic time complexity has recently been presented in [11].

2.2 Clustering complex objects

In recent years, the research community spent a lot of attention to clustering resulting in a large variety of different clustering algorithms. However, most of those algorithms were designed for vector data, so there is still a need for research on clustering complex objects.

In this paper, we focus on the acceleration of density-based clustering algorithms like DBSCAN [6] and OPTICS [2], which are based on ε-range queries. Density-based clustering algorithms provide the following advantages:

1. They can be used for all kinds of metric data spaces and are not confined to vector spaces.

2. They are robust concerning outliers.
3. They have proved to be very efficient and effective in clustering all sorts of data.
4. OPTICS is – in contrast to most other algorithms – relatively insensitive to its two input parameters, ε and *MinPts*. The authors in [2] state that the input parameters just have to be large enough to produce good results.
5. Traditional clustering algorithms are based on one representation space, usually a vector space. However, for complex objects, often multiple representations exist for each object. Proteins for example are characterized by an amino acid sequence, a secondary structure and a 3D representation. In [8], an efficient density-based approach to cluster such multi-represented data, taking all available representations into account is presented.
6. In [3] the authors show how visualizing the hierarchical clustering structure of a database of objects can aid the user in his time-consuming task to find similar objects. Based on reachability plots produced by the density-based clustering algorithm OPTICS [2], approaches which automatically extract the significant clusters in a hierarchical cluster representation along with suitable cluster representatives are provided.

2.3 Query processing on complex objects

2.3.1 Multi-step query processing

The main goal of a filter-refinement architecture, as depicted in Fig. 2, is to reduce the number of complex and, therefore, time consuming object distance calculations in the query process. To achieve this goal, query processing is performed in two or more steps. The first step is a filter step, which returns a number of candidate objects from the database. For these candidate objects, the exact object distance is then determined in the refinement step and the objects fulfilling the query predicate are reported. To reduce the overall search time, it is essential that the filter predicate is considerably easier to evaluate than the exact similarity measure and a substantial part of the database objects must be filtered out. Additionally, the completeness of the filter step is essential, i.e. all database objects satisfying the query condition are included in the candidate set. Available similarity search algorithms guarantee completeness if the distance function in the filter step fulfills the lower-bounding property. Using a multi-step query architecture requires efficient algorithms, which actually make use of the filter step. Agrawal et al. [1] proposed such an algorithm for range search.

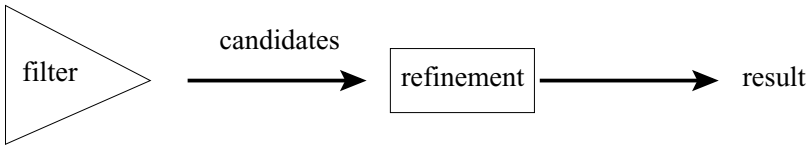

Fig. 2 A multi-step query processing architecture

2.3.2 Metric index structures

In some applications, objects cannot be mapped into feature vectors. However, there still exists some notion of similarity between objects, which can be expressed as a metric distance between the objects, i.e. the objects are embedded in a metric space. Several index structures for pure metric spaces have been proposed in the literature (see [4] for an overview). A well-known dynamic index structure for metric spaces is the M-tree [5]. The M-tree, which is explained in detail in Sect. 3.1, aims at providing good I/O performance as well as reducing the number of distance computations.

3 Efficient range queries on complex objects

So far, the concepts of multi-step query processing and metric index structures have only been used separately. We claim that these concepts can beneficially be combined and that, through the combination, a significant speed-up compared to both separate approaches can be achieved. In the following, we will demonstrate the ideas for range queries with the M-tree as index structure and arbitrary filters fulfilling the lower-bounding criterion. It is worth noting that the techniques can also be applied to similar metric index structures like the Slim-tree [16].

This section is organized as follows. After introducing the necessary concepts for similarity range queries using the M-tree, we present the concept of "positive pruning" in Sect. 3.2. In Sect. 3.3, we combine the two worlds of direct metric index structures and multi-step query processing based on filtering. Finally, we show in Sect. 3.4 how caching can be applied to accelerate the processing of similarity range queries.

3.1 Similarity range queries using the M-tree

The M-tree (*metric tree*) [5] is a balanced, paged and dynamic index structure that partitions data objects not by means of their absolute positions in the multidimensional feature space but on the basis of their relative distances in this feature space. The only prerequisite is that the distance function between the indexed objects is metric. Thus, the M-tree's domain of applicability is quite general, and all sorts of complex data objects can be organized with this index structure.

The maximum size of all nodes of the M-tree is fixed. All database objects O_d or references to them are stored in the leaf nodes of an M-tree, along with their feature values and the distance $d(O_d, P(O_d))$ to their parent object $P(O_d)$. Inner nodes contain so-called *routing objects*, which correspond to database objects to whom a *routing role* was assigned by a promoting algorithm that is executed whenever a node has to be split. In addition to the object description and the distance to the parent object, routing objects O_r also store their *covering radius* $r(O_r)$ and a pointer $\text{ptr}(T(O_r))$ to the root node of their subtree, the so-called *covering tree* of O_r. For all objects O_d in this covering tree, the condition holds that the distance $d(O_r, O_d)$ is smaller or equal to the covering radius $r(O_r)$. This property induces a hierarchical structure of an M-tree, with the covering radius of a parent

```
1   simRange(queryObject O_q, range ε) → ResultSet
2     result = NIL;
3     rangeSearch(root, O_q, ε);
4     return result;
```

```
1   rangeSearch(Node N, queryObject O_q, range ε)
2     O_p := parent object of node N;
3     if N is not a leaf then
4       for each O_r in N do
5         if |d(O_p, O_q) − d(O_r, O_p)| ≤ r(O_r) + ε
6         then
7           compute d(O_r, O_q);
8           if d(O_r, O_q) <= r(O_r) + ε then
9             rangeSearch(ptr(T(O_r), O_q, ε);
10          end if
11        end if
12      end for
13    else
14      for each O_d in N do
15        if |d(O_p, O_q) − d(O_d, O_p)| ≤ ε then
16          compute d(O_d, O_q);
17          if d(O_d, O_q) ≤ ε then
18            add O_d to result;
19          end if
20        end if
21      end for
22    end if
```

Fig. 3 Pseudo-code description of similarity range search on M-trees

object always being greater or equal than all covering radii of their children and the root object of an M-tree storing the maximum of all covering radii.

Range queries are specified by a query object O_q and a range value ε by which the answer set is defined to contain all the objects O_d from the database that have a distance to the query object O_q of less than or equal to ε.

Definition 1 (similarity range query) Let $\mathcal{O}$ be a domain of objects and $DB \subseteq \mathcal{O}$ be a database. For a query object $O_q \in \mathcal{O}$ and a query range $\varepsilon \in I\!R_0^+$, the similarity range query $simRange : \mathcal{O} \times I\!R_0^+ \mapsto 2^{DB}$ returns the set

$$simRange(O_q, \varepsilon) = \{O_d \in DB | dist(O_d, O_q) \leq \varepsilon\}.$$

Given a query object O_q and a similarity range parameter ε, a similarity range query $simRange(O_q, \varepsilon)$ starts at the root node of an M-tree and recursively traverses the whole tree down to the leaf level, thereby pruning all subtrees which certainly contain no result objects.

A description of *simRange* in pseudo-code and the recursive procedure *rangeSearch* used to traverse the M-tree is given in Fig. 3.

The subtree of a routing object O_r can be pruned, if the absolute value of the distance of the routing object's parent object O_p to the query object O_q, $d(O_p, O_q)$, minus the distance between O_r and O_p is greater than the covering radius of O_r plus ε:

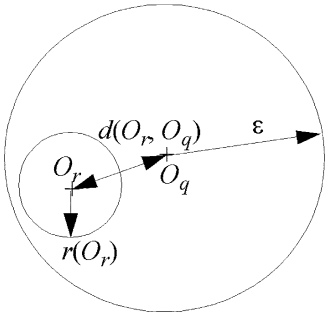

Fig. 4 Positive pruning for the M-tree

$$|d(O_\mathrm{p}, O_\mathrm{q}) - d(O_\mathrm{p}, O_\mathrm{r})| > r(O_\mathrm{r}) + \varepsilon$$

A proof for this is given in [5]. Thus, as the distance between O_p and O_q has already been computed when accessing a node N, subtrees can be pruned without further distance computations (see line 5 of the algorithm in Fig. 3).

3.2 Positive pruning

A hierarchical index structure, like the M-tree, is composed of directory nodes with routing objects O_r which represent all objects in their respective subtree $T(O_\mathrm{r})$. For all objects $O \in T(O_\mathrm{r}), d(O_\mathrm{q}, O_\mathrm{r}) \leq r(O_\mathrm{r})$ holds. Efficient processing of range queries on the original M-tree is based on the concept of "negative pruning". During the query processing, certain subtrees are excluded from the search based on the following formula: $d(O_\mathrm{q}, O_\mathrm{r}) > r(O_\mathrm{r}) + \varepsilon$ (see line 7 of the algorithm in Fig. 3).

In this section, we introduce the concept of "positive pruning". If a directory node is completely covered by the query range, we can report all objects on the leaf level of the M-tree without performing any cost intensive distance computations (cf. Fig. 4).

Lemma 1 *Let $O_\mathrm{q} \in \mathcal{O}$ be a query object and $\varepsilon \in \mathbb{R}_0^+$ a query range. Furthermore, let O_r be a routing object in an M-tree with a covering radius $r(O_\mathrm{r})$ and a subtree $T(O_\mathrm{r})$. Then the following statement holds:*

$$d(O_\mathrm{r}, O_\mathrm{q}) + r(O_\mathrm{r}) \leq \varepsilon \Rightarrow \quad \forall O \in T(O_\mathrm{r}) : d(O, O_\mathrm{q}) \leq \varepsilon$$

Proof The following inequalities hold for all $O \in T(O_\mathrm{r})$ due to the triangle inequality and due to $d(O_\mathrm{r}, O_\mathrm{q}) + r(O_\mathrm{r}) \leq \varepsilon$:

$$d(O, O_\mathrm{q}) \leq d(O, O_\mathrm{r}) + d(O_\mathrm{r}, O_\mathrm{q}) \leq r(O_\mathrm{r}) + d(O_\mathrm{r}, O_\mathrm{q}) \leq \varepsilon \qquad \square$$

```
1     rangeSearch(Node N, queryObject O_q, range ε)
                              ⋮
7     compute d(O_r, O_q);
7a    if d(O_r, O_q) + r(O_r) ≤ ε then
7b      report all objects in T(O_r);
8     else if d(O_r, O_q) <= r(O_r) + ε then
                              ⋮
```

Fig. 5 Adaptation of similarity range search on M-trees for positive pruning

In the case of negative pruning, we skip the recursive tree traversal of a subtree $T(O_\mathrm{r})$, if the query range does not intersect the covering radius $r(O_\mathrm{r})$. In the case of positive pruning, we skip all the distance calculations involved in the recursive tree traversal if the query range completely covers the covering radius $r(O_\mathrm{r})$. In this case, we can report all objects stored in the corresponding leaf nodes of this subtree without performing any further distance computations. Figure 5 shows how this concept can be integrated into the original method *rangeSearch* depicted in Fig. 3.

This approach is very beneficial for accelerating density-based clustering on complex objects. DBSCAN, for instance, only needs the information whether an object is contained in $simRange(O_\mathrm{q}, \varepsilon) = \{O \in DB | d(O, O_\mathrm{q}) \leq \varepsilon\}$ but not the actual distance of this object to the query object O_q.

3.3 Combination of filtering and indexing

The M-tree reduces the number of distance calculations by partitioning the data space even if no filters are available. Unfortunately, the M-tree may suffer from the navigational cost related to the distance computations during the recursive tree traversal. On the other hand, the filtering approach heavily depends on the quality of the filters.

When combining both approaches, these two drawbacks are reduced. We use the filter distances to optimize the required number of exact object distance calculations needed to traverse the M-tree. Thereby, we do not save any I/O cost compared to the original M-tree, as the same nodes are traversed, but we save a lot of costly distance calculations necessary for the traversal. The filtering M-tree stores the objects along with their corresponding filter values within the M-tree. A similarity query based on the filtering M-tree always computes the filter distance values prior to the exact distance computations. If a filter distance value is already a sufficient criterion to prune branches of the M-tree, we can avoid the exact distance computation. If we have several filters, the filter distance computation always returns the maximum value of all filters.

The pruning quality of the filtering M-tree benefits from both the quality of the filters and the clustering properties of the index structure. In the following, we will show that the number of distance calculations used for range queries as well as for the creation and update of an M-tree can be optimized by using lower-bounding filters.

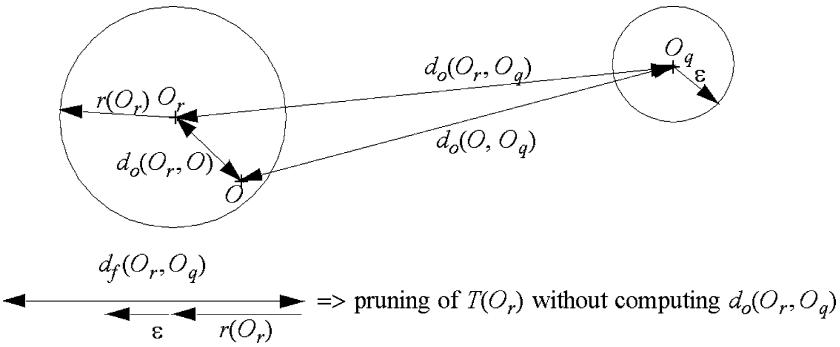

Fig. 6 Similarity range query based on the filtering M-tree

3.3.1 Range queries

Similarity range queries are used to retrieve all objects from a database which are within a certain similarity range from the query object (cf. Definition 1). By computing the filter distance prior to the exact distance, we can save on many distance computations. Based on the following lemma, we can prune many subtrees without computing the exact distances between a query object O_q and a routing object O_r (cf. Fig. 6).

Lemma 2 *Let $\mathcal{O}$ be a set of objects and $DB \subseteq \mathcal{O}$ a database. Furthermore, let $d_o, d_f : \mathcal{O} \times \mathcal{O} \mapsto \mathbb{R}_0^+$ be two distance functions, for which d_f lower bounds d_o, i.e. $\forall O_1, O_2 \in \mathcal{O} : d_f(O_1, O_2) \leq d_o(O_1, O_2)$ holds. Let $O_q \in \mathcal{O}$, $\varepsilon \in \mathbb{R}_0^+$. For each routing object $O_r \in DB$ with covering radius $r(O_r) \in \mathbb{R}_0^+$ and subtree $T(O_r)$ the following statement holds:*

$$\forall O \in T(O_r) : (d_f(O_q, O_r) > r(O_r) + \varepsilon) \Rightarrow d_o(O_q, O) > \varepsilon$$

Proof As $\forall O_1, O_2 \in \mathcal{O} : \lceil_{\{}(\mathcal{O}_\infty, \mathcal{O}_\in) \leq \lceil_{\wr}(\mathcal{O}_\infty, \mathcal{O}_\in)$ holds, the following statement is true:

$$d_f(O_q, O_r) > r(O_r) + \varepsilon \Rightarrow d_o(O_q, O_r) > r(O_r) + \varepsilon$$

Based on the triangle inequality and our assumption that $d_o(O, O_r) \leq r(O_r)$, we can proof the aforementioned lemma as follows:

$$\begin{aligned} & d_f(O_q, O_r) > r(O_r) + \varepsilon \\ \Rightarrow\ & d_o(O_q, O_r) > r(O_r) + \varepsilon \\ \Rightarrow\ & d_o(O_q, O_r) - r(O_r) > \varepsilon \\ \Rightarrow\ & d_o(O_q, O_r) - d_o(O, O_r) > \varepsilon \\ \Rightarrow\ & d_o(O_q, O) > \end{aligned}$$

□

Let us note that a similar optimization can be applied to the objects stored on the leaf level with the assumption that their 'covering radius' is 0. Figure 7 shows how this concept can be integrated into the original method *rangeSearch* of Fig. 3.

```
1    rangeSearch(Node N, queryObject O_q, range ε)
                              ⋮
5        if |d(O_p, O_q) − d(O_r, O_p)| ≤ r(O_r) + ε
6        then
6a           compute d_f(O_r, O_q)
6b           if d_f(O_r, O_q) <= r(O_r) + ε then
7                compute d(O_r, O_q);
8                if d(O_r, O_q) <= r(O_r) + ε then
                              ⋮
15       if |d(O_p, O_q) − d(O_d, O_p)| ≤ ε then
15a          compute d_f(O_d, O_q)
15b          if d_f(O_d, O_q) ≤ ε then
16               compute d(O_d, O_q);
17               if d(O_d, O_q) ≤ ε then
                              ⋮
```

Fig. 7 Adaptation of similarity range search on M-trees for filtering

3.3.2 Construction and update

Filters can also be used for accelerating the creation and update of an M-tree. For the necessary adaptations of the insertion and split algorithms, we refer the reader to [7].

3.4 Caching distance calculations

In this section, we present a further technique which helps to avoid costly distance computations for index construction and query processing.

Efficient query processing of range queries also benefits from the idea of caching distance calculations. During the navigation through the M-tree directory, the same distance computations may have to be carried out several times. Although each object O is stored only once on the leaf level of the M-tree, it might be used several times as routing object. Furthermore, we often have the situation that distance calculations carried out on the directory level have to be repeated at the leaf level.

As shown in Fig. 3, a natural way to implement range queries is by means of recursion resulting in a depth-first search. We suggest to keep all distance computations in main memory which have been carried out on the way from the root to the actual node. After leaving the node, i.e. when exiting the recursive function, we delete all distance computations carried out at this node. This limits the actual main memory footprint to $O(h \cdot b)$, where h denotes the maximum height of a tree and b denotes the maximum number of stored elements in a node. Even in multi-user environments, this rather small worst-case main memory footprint is tolerable. The necessary adaptations of the rangeSearch algorithm are drafted in Fig. 8.

```
1     rangeSearch(Node N, queryObject O_q, range ε)
                    ⋮
6       distCache(N, O_r, O_q);
                    ⋮
16      distCache(N, O_d, O_q);
                    ⋮
22    end if
22a   deleteCache(N);
```

```
distCache(Node N, Object O_1, Object O_2) → float
  result = hashtable.lookup(O_1, O_2);
  if result = null then
    result = compute d(O_1, O_2);
    hashtable.add(N, O_1, O_2, result);
  end if
  return result;
```

```
deleteCache(Node N)
  hashtable.delete(N);
```

Fig. 8 Adaptation of similarity range search on M-trees for caching

4 Evaluation

To show the efficiency of our approach, we chose the applications and data types described in Sect. 2 and performed extensive experiments. All algorithms were implemented in Java 1.4 and the experiments were run on a workstation with a Xeon 1.7 GHz processor and 2 GB main memory under Linux. We implemented the M-tree as described in [5]. As in all cases, the time for distance calculations was dominating the runtime of a range query, we only show the number of distance calculations and not the runtime.

4.1 CAD vector set data

For the experiments with this data type, we used the similarity model presented in [10], where CAD objects were represented by a vector set consisting of seven vectors in 6D. All experiments were carried out on a data set containing 5000 CAD objects from an American aircraft producer. As distance measure between sets of feature vectors we used the minimal matching distances which can be computed in $O(k^3)$, where k denotes the cardinality of the point set, by means of the Kuhn–Munkres algorithm [12, 13]. As filter, we used the centroid filter introduced in [10].

Figures 9 and 10 show in what way the different approaches for range query processing depend on the chosen ε value. Figure 9 shows that for the investigated data set, the original M-tree is the worst access method for all ε values. On the other hand, the pure filter performs very well. For this data set, reasonable ε values for density-based clustering would be about 1 for DBSCAN and about 2 for

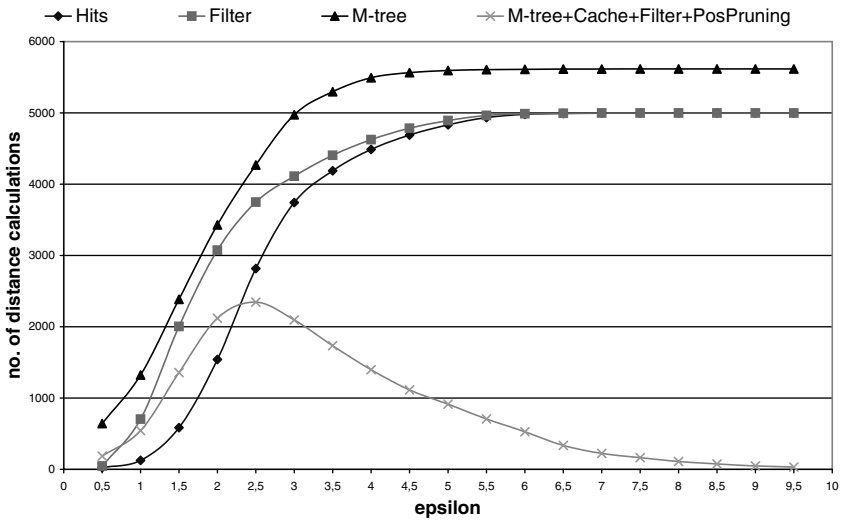

Fig. 9 Comparison of our best technique to M-tree and filtering for vector set data

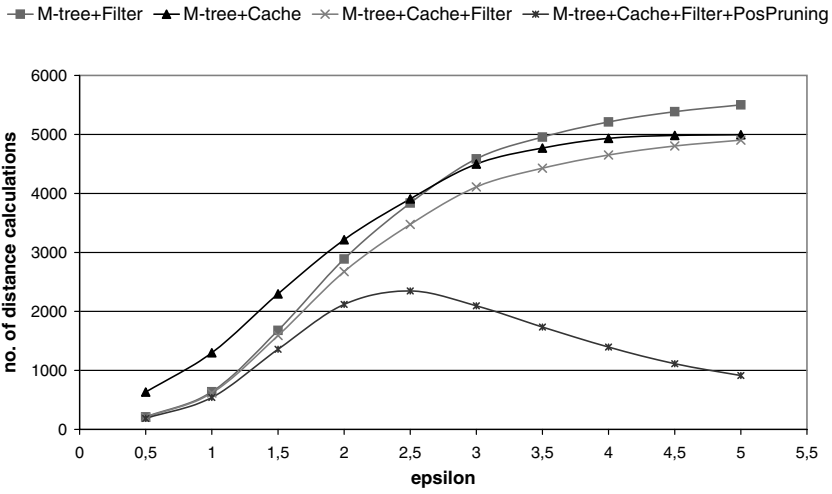

Fig. 10 Comparison of our techniques for vector set data

OPTICS. In this parameter range, our approach clearly outperforms both the filter and especially the original M-tree.

In Fig. 10, one can see that for small ε values, we benefit from the filtering M-tree, whereas for higher values, we benefit from caching and positive pruning.

Furthermore, we clustered the data set using OPTICS [2]. which forms the basis for the visual data mining tool presented in Sect. 2.2. With a suitable parameter setting for OPTICS we achieved a speed-up of 16% compared to the centroid filter, 33% compared to the original M-tree, and 104% compared to the sequential

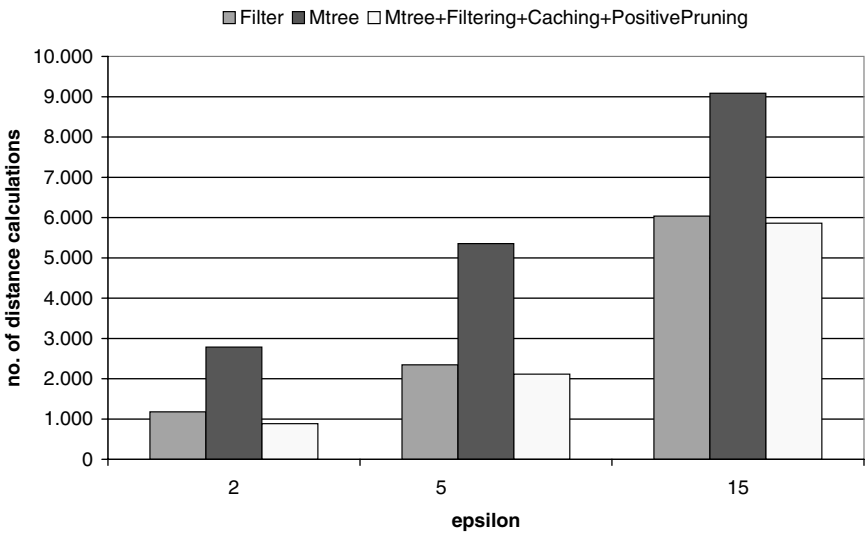

Fig. 11 Comparison of our best technique to M-tree and filtering for tree structured data

scan. Let us note that the average cardinality of the result set of each range query was almost 2000, which limits the best achievable speed-up to 150%.

4.2 Image data

Image data are a good example for multi-represented complex objects. A lot of different similarity models exist for image data, each having its own advantages and disadvantages. Using for example text descriptions of images, one is able to cluster all images related to a certain topic, but these images need not look alike. Using color histograms instead, the images are clustered according to the distribution of color in the image. The approach for clustering multi-represented objects presented in [8] is able to get the best out of all these different types of representations. We present some experiments for image data represented as trees or graphs, where the efficiency of range query processing is especially important.

4.2.1 Tree structured image data

Images can be described as segmentation trees. Thereby, an image is first divided into segments of similar color, then a tree is created from these segments by iteratively applying a region growing algorithm, which merges neighboring segments if their colors are sufficiently alike. As similarity measure for the resulting trees, we used the degree-2 edit distance and implemented the filter-refinement architecture as described in [9]. We used a sample set of 10,000 color TV images. For the experiments, we chose reasonable epsilon values for the multi-represented clustering algorithm.

Figure 11 shows that we achieve a significant speed-up compared to the original M-tree. As can be seen, we also outperform the pure filtering approach.

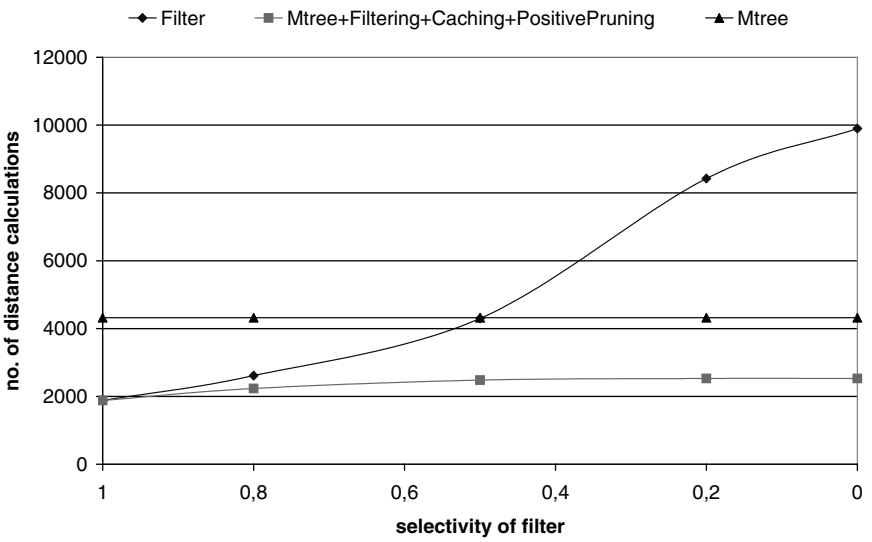

Fig. 12 Comparison of our techniques for graph data

4.2.2 Graph structured image data

To extract graphs from the images, they were segmented with a region growing technique and neighboring segments were connected by edges to represent the neighboring relationship. We used the edge matching distance and the image data set as described in [11]. The filter presented in this paper is almost optimal, i.e. the number of unnecessary distance calculations during query processing is very low. Even in this case, our techniques is as good as the filter.

To show the robustness of our approach against the filter selectivity, we reduced it in a stepwise process. We weighted the original filter distances with constant factors to decrease the filter selectivity. Figure 12 shows that independent of the filter selectivity, our approach outperforms the original M-Tree by a factor of almost 2 and is at least as good as the pure filtering approach.

5 Conclusions

The similarity measures used for complex objects are often computationally very complex, which makes them unusable for large databases. To overcome the efficiency problems, metric index structures or multi-step query processing are applied. We combined and extended these approaches to achieve the best from two worlds. More precisely, we presented three improvements for metric index structures, i.e. positive pruning, the combination of filtering and indexing, and caching. In a broad experimental evaluation based on real-world data sets, we showed that a significant speed-up for similarity range queries is achieved with our approach. By means of our new techniques, application areas like visually mining through cluster hierarchies of complex objects or clustering of complex multi-represented objects can be extended to larger databases.

References

1. Agrawal R, Faloutsos C, Swami AN (1993) Efficient similarity search in sequence databases. In: Proceedings of the 4th international conference of foundations of data organization and algorithms (FODO), pp 69–84
2. Ankerst M, Breunig MM, Kriegel H-P, Sander J (1999) OPTICS: ordering points to identify the clustering structure. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD'99), Philadelphia, PA, pp 49–60
3. Brecheisen S, Kriegel H-P, Kröger P, Pfeifle M (2004) Visually mining through cluster hierarchies. In: Proceedings of the SIAM international conference on data mining (SDM'04), Orlando, FL
4. Chavez E, Navarro G, Baeza-Yates R, Marroquin JL (2001) Searching in metric spaces. ACM Comput Surv 33(3):273–321
5. Ciaccia P, Patella M, Zezula P (1997) M-tree: An efficient access method for similarity search in metric spaces. In: VLDB'97, Proceedings of the 23rd international conference on very large databases, August 25–29, 1997, Athens, Greece, pp 426–435
6. Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the 2nd international conference on knowledge discovery and data mining (KDD'96), Portland, OR, pp 291–316
7. Kailing K, Kriegel H-P, Pfeifle M, Schönauer S (2004) Efficient indexing of complex objects for density-based clustering. In: Proceedings of the 5th international workshop on multimedia data mining (MDM/KDD), Seattle, WA, pp 28–37
8. Kailing K, Kriegel H-P, Pryakhin A, Schubert M (2004) Clustering multi-represented objects with noise. In: Proceedings of the 8th Pacific-Asia conference on knowledge discovery and data mining (PAKDD'04), Sydney, Australia, pp 394–403
9. Kailing K, Kriegel, H-P, Schönauer S, Thomas S (2004) Efficient similarity search for hierachical data in large databases. In: Proceedings of the 9th international conference on extending database technology (EDBT 2004), pp 676–693
10. Kriegel H-P, Brecheisen S, Krger P, Pfeifle M, Schubert M (2003) Using sets of feature vectors for similarity search on voxelized cad objects. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD'03), San Diego, CA, pp 587–598
11. Kriegel H-P, Schönauer S (2003) Similarity search in structured data. In: Proceedings of the 5th international conference, DaWaK 2003, Prague, Czech Republic, pp 309–319
12. Kuhn H (1955) The Hungarian method for the assignment problem. Naval Res Logist Quart 2:83–97
13. Munkres J (1957) Algorithms for the assignment and transportation problems. J SIAM 6:32–38
14. Nierman A, Jagadish HV (2002) Evaluating structural similarity in XML documents. In: Proceedings of the 5th international workshop on the web and databases (WebDB 2002), Madison, Wisconsin, USA, pp 61–66
15. Sebastian TB, Klein PN, Kimia BB (2001) Recognition of shapes by editing shock graphs. In: Proceedings of the 8th international conference on computer vision (ICCV'01), Vancouver, BC, Canada, vol 1, pp 755–762
16. Traina C Jr., Traina A, Seeger B, Faloutsos C (2000) Slim-trees: high performance metric trees minimizing overlap between nodes. In: Proceedings of the 7th international conference on extending database technology, Konstanz, Germany, March 27–31, 2000, pp 51–65
17. Wang JTL, Zhang K, Chang G, Shasha D (2002) Finding approximate patterns in undirected acyclic graphs. Pattern Recog 35(2):473–483
18. Zhang K, Wang J, Shasha D (1996) On the editing distance between undirected acyclic graphs. Int J Found Comput Sci 7(1):43–57

Author Biographies

Karin Kailing received her PhD from the University of Munich where she is working as a research and teaching assistant. She is currently on a leave of absence to the IBM Almaden Research Center. Her research interests are in query processing and knowledge discovery in databases. One of her focus areas is the development of new techniques for mining complex objects.

Hans-Peter Kriegel is a full professor for database systems in the Department of Computer Science at the University of Munich and the department head since 2003. His research interests are in spatial and multimedia database systems, particularly in query processing, performance issues, similarity search, high-dimensional indexing as well as in knowledge discovery and data mining. He received his MS and Ph.D. in 1973 and 1976, respectively, from the University of Karlsruhe, Germany. Hans-Peter Kriegel has been chairman and program committee member in many international database conferences. He has published over 200 refereed conference and journal papers. In 1997 Hans-Peter Kriegel received the "SIGMOD Best Paper Award" for the publication and prototype implementation "Fast Parallel Similarity Search in Multimedia Databases" together with four members of his research team.

Dr. Martin Pfeifle works as a research and teaching assistant in the group of Prof. Dr. Hans-Peter Kriegel. The research interests of Martin Pfeifle include database support for virtual engineering, with a strong emphasis on spatial index structures and similarity search in spatial databases. Furthermore, he is interested in the area of knowledge discovery in databases, especially in density-based clustering.

Stefan Schönauer currently is a Post-Doc at IBM Almaden Research Center in the reasearch group of Rakesh Agrawal. He received his MS and Ph.D. in 1999 and 2004, respectively, from the University of Munich, Germany. His research interests are in similarity search and data mining in complex objects, content-based image retrieval and bioinformatics. He is a member of ACM SIGMOD.