Clustering Multidimensional Extended Objects to Speed Up Execution of Spatial Queries

Cristian-Augustin Saita, François Llirbat

Clustering Multidimensional Extended Objects to Speed Up Execution of Spatial Queries

Cristian-Augustin Saita and François Llirbat

INRIA-Rocquencourt
Domaine de Voluceau, B.P. 105
78153 Le Chesnay Cedex, France
firstname.lastname@inria.fr

Abstract. We present a cost-based adaptive clustering method to improve average performance of spatial queries (intersection, containment, enclosure queries) over large collections of multidimensional extended objects (hyper-intervals or hyper-rectangles). Our object clustering strategy is based on a cost model taking into account the spatial object distribution, the query distribution, and a set of database and system parameters affecting the query performance: object size, access time, transfer and verification costs. We also employ a new grouping criterion to group objects in clusters, more efficient than traditional approaches based on minimum bounding in all dimensions. Our cost model is flexible and can accommodate different storage scenarios: in-memory or disk-based. Experimental evaluations show that our approach is efficient in a number of situations involving large spatial databases with many dimensions.

1 Introduction

In this paper we present a cost-based adaptive clustering solution to faster answer *intersection, containment, or enclosure-based spatial queries* over large collections of *multidimensional extended objects*. While a multidimensional point defines single values in all dimensions, a *multidimensional extended object* defines range intervals in its dimensions. A multidimensional extended object is also called *hyper-interval* or *hyper-rectangle*. Although a point can be also represented as an extended object (with zero-length extensions over dimensions), we are interested in collections of objects with real (not null) extensions.

Motivation. Our work is motivated by the development of new dissemination-based applications (SDI or Selective Dissemination of Information) [1][8][12]. Such applications involve timely delivery of information to large sets of subscribers and include stock exchange, auctions, small ads, and service delivery. Let us consider a publish-subscribe notification system dealing with small ads. An example of subscription is "Notify me of all new apartments within 30 miles from Newark, with a rent price between 400$ and 700$, having between 3 and 5 rooms, and 2 baths". In this example, most subscription attributes specify range intervals instead of single values. Range subscriptions are more suitable for notification systems. Indeed, subscribers often wish to consult the set of alternative

offers that are close to their wishes. Range intervals allow them to express more flexible matching criteria. High rates of new offers (events), emitted by publishers, need to be verified against the subscription database. The role of the system is to quickly retrieve and notify all the subscribers matching the incoming events. The events can define either single values or range intervals for their attributes. An example of range event is "Apartments for rent in Newark: 3 to 5 rooms, 1 or 2 baths, 600-900". In such context, subscriptions and events can be represented as *multidimensional extended objects*. The matching subscriptions are retrieved based on *intersection, containment, or enclosure queries* (spatial range or window queries). In some applications the reactivity of the notification system is crucial (stock exchange, auctions). As the number of subscriptions can be large (millions) and the number of possible attributes significant (tens of dimensions), an indexing method is required to ensure a good response time. Such method should cope with large collections of multidimensional extended objects, with many dimensions, and with high rates of events (actually spatial queries).

Problem Statement. Most multidimensional indexing methods supporting spatial queries over collections of multidimensional extended objects descend from the R-tree approach[11]. R-tree employs minimum bounding boxes (MBBs) to hierarchically organize the spatial objects in a height-balanced tree. Construction constraints like preserving tree height balance or ensuring minimal page utilization, corroborated with the multidimensional aspect, lead to significant overlap between MBBs at node level. During searches, this overlap determines the exploration of multiple tree branches generating serious performance degradation (notably for range queries). Because the probability of overlap increases with the number of dimensions [2] [4], many techniques have been proposed to alleviate the "dimensional curse". Despite this effort, experiments show that, for more than 5-10 dimensions, a simple database Sequential Scan outperforms complex R-tree implementations like X-tree[4], or Hilbert R-tree[9]. This was reported for range queries over collections of hyper-space points [3]. When dealing with spatially-extended data, the objects themselves may overlap each others, further increasing the general overlap and quickly leading to poor performance. For these reasons, R-tree-based methods can not be used in practice on collections of multidimensional extended objects with more than a few dimensions.

Contributions. We propose a new approach to cluster multidimensional extended objects, to faster answer spatial range queries (intersection, containment, enclosure), and to satisfy the requirements outlined in our motivation section: large collections of objects, many dimensions, high rates of spatial queries, and frequent updates. Our main contributions are:

1. An adaptive cost-based clustering strategy: Our cost-based clustering strategy takes into account the spatial data distribution and the spatial query distribution. It is also parameterized by a set of database and system parameters affecting the query performance: object size, disk access time, disk transfer rate, and object verification cost. The cost model is flexible and can easily adapt different storage scenarios: in-memory or disk-based. Using the cost-based clustering we always guarantee better average performance than Sequential Scan.

2. An original criterion for clustering objects: Our adaptive clustering strategy is enabled by a new approach to cluster objects, which consists in clustering together objects with "similar" intervals on a restrained number of dimensions. The grouping intervals/dimensions are selected based on local statistics concerning cluster-level data distribution and cluster access probability, aiming to minimize the cluster exploration cost. Our object clustering proves to be more efficient than classical methods employing minimum bounding in all dimensions.

Paper Organization. The rest of the paper is organized as follows: Section 2 reviews the related work. Section 3 presents our adaptive cost-based clustering solution and provides algorithms for cluster reorganization, object insertion, and spatial query execution. Section 4 presents our grouping criterion. Section 5 provides details on the cost model supporting the clustering strategy. Section 6 considers implementation-related aspects like model parameters and storage utilization. Section 7 experimentally evaluates the efficiency of our technique, comparing it to alternative solutions. Finally, conclusions are presented in Section 8.

2 Related Work

During last two decades numerous indexing techniques have been proposed to improve the search performance over collections of multidimensional objects. Recent surveys review and compare many of existing multidimensional access methods [10][6][17]. From these surveys, two families of solutions can be distinguished. First family descends from the K-D-tree method and is based on recursive space partitioning in disjoint regions using one or several, alternating or not, split dimensions: quad-tree, grid file, K-D-B-tree, hB-tree, but also Pyramid-tree and VA-file [1]. These indexing methods work only for multidimensional points.Second family is based on the R-tree technique introduced in [11] as a multidimensional generalization of B-tree. R-tree-based methods evolved in two directions. One aimed to improve performance of nearest neighbor queries over collection of multidimensional points: SS-tree, SR-tree, A-tree [2]. Since we deal with extended objects, these extensions do not apply in our case. The other direction consisted in general improvements of the original R-tree approach, still supporting spatial queries over multidimensional extended objects: R^+-tree[14], R^*-tree[2]. Although efficient in low-dimensional spaces (under 5-10 dimensions), these techniques fail to beat Sequential Scan in high dimensions due to the large number of nodes/pages that need to be accessed and read in a random manner. Random disk page read is much more expensive than sequential disk page read. In general, to outperform Sequential Scan, no more than 10% of tree nodes should be (randomly) accessed, which is not possible for spatial range queries over extended objects with many dimensions. To transform the random access into sequential scan, the concept of *supernode* was introduced in X-tree[4]. Multiple pages are assigned to directory nodes for which the split would generate too much overlap. The size of a supernode is determined based

[1] See surveys [10][6] for further references.
[2] See surveys [6][17] for further references.

on a cost model taking into account the actual data distribution, but not considering the query distribution. A cost-based approach is also employed in [7] to dynamically compute page sizes based on data distribution. In [15] the authors propose a general framework for converting traditional indexing structures to adaptive versions exploiting both data and query distributions. Statistics are maintained in a global histogram dividing the data space into bins/cells of equal extent/volume. Although efficient for B-trees, this technique is impractical for high-dimensional R-trees: First, because the number of histogram bins grows exponentially with the number of dimensions. Second, the deployed histogram is suitable for hyper-space points (which necessarily fit the bins), but is inappropriate for hyper-rectangles that could expand over numerous bins. Except for the node size constraint, which is relaxed, both X-tree and Adaptive R-tree preserve all defining properties of the R-tree structure: height balance, minimum space bounding, balanced split, and storage utilization. In contrast, we propose an indexing solution which drops these constraints in favor of a cost-based object clustering. In high-dimensional spaces, VA-File method[16] is a good alternative to tree-based indexing approaches. It uses approximated (compressed) data representation and takes advantage of Sequential Scan to faster perform searches over collections of multidimensional points. However, this technique can manage only point data. An interesting study regarding the optimal clustering of a static collection of spatial objects is presented in [13]. The static clustering problem is solved as a classical optimization problem, but data and query distributions need to be known in advance.

3 Cost-based Database Clustering

Our database clustering takes into consideration both data and query distributions, and allows clusters to have different sizes. Statistical information is associated to each cluster and employed in the cost model, together with other database and system parameters affecting the query performance: object size, disk access cost, disk transfer rate, and object check rate. The cost model supports the creation of new clusters and the detection and removal of older inefficient clusters.

3.1 Cluster Description

A *cluster* represents a group of objects accessed and checked together during spatial selections. The grouping characteristics are represented by the *cluster signature*. The cluster signature is used to verify (a) if an object can become a member of the cluster: only objects matching the cluster signature can become cluster members; (b) if the cluster needs to be explored during a spatial selection: only clusters whose signatures are matched by the spatial query are explored. To evaluate the average query cost, we also associate each cluster signature with two performance indicators:

1. The number of queries matching the cluster signature over a period of time: This statistics represents a good indicator for the *cluster access probability*. Indeed, the access probability can be estimated as the ratio between the number of queries exploring the cluster and the total number of queries addressed to the system over a period of time.

2. The number of objects matching the cluster signature: When combined with specific database and system parameters (object size, object access, transfer and verification costs), this statistics allows to estimate the *cluster exploration cost*. Indeed, cluster exploration implies individual checking of each of its member objects.

3.2 Clustering Strategy

The clustering process is recursive in spirit and is accomplished by relocating objects from existing clusters in new (sub)clusters when such operation is expected to be profitable. Initially, the collection of spatial objects is stored in a single cluster, *root cluster*, whose general signature accepts any spatial object. The access probability of the root cluster is always 1 because all the spatial queries are exploring it. At root cluster creation we invoke the *clustering function* to establish the signatures of the potential subclusters of the root cluster. The potential subclusters of an existing cluster are further referred as *candidate (sub)clusters*. Performance indicators are maintained for the root cluster and all its candidate subclusters. Decision of cluster reorganization is made periodically after a number of queries are executed and after performance indicators are updated accordingly. Clustering decision is based on the *materialization benefit function* which applies to each candidate subcluster and evaluates the potential profit of its materialization. Candidate subclusters with the best expected profits are selected and materialized. A subcluster materialization consists in two actions: First, a new cluster with the signature of the corresponding candidate subcluster is created: all objects matching its signature are moved from the parent cluster. Second, the clustering function is applied on the signature of the new cluster to determine its corresponding candidate subclusters. Performance indicators are attached to each of the new candidate subclusters in order to gather statistics for future re-clustering. Periodically, clustering decision is re-considered for all materialized clusters. As a result, we obtain a tree of clusters, where each cluster is associated with a signature and a set of candidate subclusters with the corresponding performance indicators. Sometimes, the separate management of an existing cluster can become inefficient. When such situation occurs, the given cluster is removed from the database and its objects transferred back to the parent cluster (direct ancestor in the clustering hierarchy). This action is called *merging operation* and permits the clustering to adapt changes in object and query distributions. A merging operation is decided using the *merging benefit function* which evaluates its impact on the average spatial query performance. To facilitate merging operations, each database cluster maintains a reference to the direct parent, and a list of references to the child clusters. The root cluster has no parent and can not be removed from the spatial database.

3.3 Functions Supporting the Clustering Strategy

Clustering Function. Based on the signature σ_c of the database cluster c, the *clustering function* γ produces the set of signatures $\{\sigma_s\}$ associated to the candidate subclusters $\{s\}$ of c. Formally, $\gamma(\sigma_c) \rightarrow \{\sigma_s\}$; $\sigma_s \in \gamma(\sigma_c)$ is generated such that any spatial object qualifying for the subcluster s also qualifies for the cluster c. It is possible for a spatial object from the cluster c to satisfy the signatures of several subclusters of c. The clustering function ensures a backward object compatibility in the clustering hierarchy. This property enables merging operations between child and parent clusters.

Materialization Benefit Function. Each database cluster is associated with a set of candidate subclusters potentially qualifying for materialization. The role of the *materialization benefit function* β is to estimate for a candidate subcluster the impact on the query performance of its possible materialization. For this purpose, β takes into consideration the performance indicators of the candidate subcluster, the performance indicators of the original cluster, and the set of database and system parameters affecting the query response time. Formally, if $\sigma_s \in \gamma(\sigma_c)$ (s is a candidate subcluster of the cluster c) then

$$\beta(s, c) \rightarrow \begin{cases} > 0 \text{ if materialization of } s \text{ is profitable;} \\ \leq 0 \text{ otherwise.} \end{cases}$$

Merging Benefit Function. The role of the *merging benefit function* μ is to evaluate the convenience of the merging operation. For this purpose, μ takes into consideration the performance indicators of the considered cluster, of the parent cluster, and the set of system parameters affecting the query response time. Formally, if $\sigma_c \in \gamma(\sigma_a)$ (a is the parent cluster of the cluster c) then

$$\mu(c, a) \rightarrow \begin{cases} > 0 \text{ if merging } c \text{ to } a \text{ is profitable;} \\ \leq 0 \text{ otherwise.} \end{cases}$$

Table 1. Notations

$\mathcal{C}$	set of materialized clusters	$parent(\mathsf{c})$	parent cluster of cluster c
$\sigma(\mathsf{c})$	signature of cluster c	$children(\mathsf{c})$	set of child clusters of cluster c
$n(\mathsf{c})$	nb. of objects in cluster c	$candidates(\mathsf{c})$	set of candidate subclusters of c
$q(\mathsf{c})$	nb. of exploring queries of c	$objects(\mathsf{c})$	set of objects from cluster c
$p(\mathsf{c})$	access probability of c	$\beta(), \mu()$	benefit functions

3.4 Cluster Reorganization

Regarding an existent cluster, two actions might improve the average query cost: the cluster could be split by materializing some of its candidate subclusters, or the cluster could be merged with its parent cluster. Figures 1, 2, and 3 depict the procedures invoked during the cluster reorganization process. Table 1 summarizes the notations used throughout the presented algorithms.

The main cluster reorganization schema is sketched in Fig. 1. First, the merging benefit function μ is invoked to estimate the profit of the merging operation. If a positive profit is expected the merging procedure is executed. Otherwise, a cluster split is attempted. When none of the two actions is beneficial for the query performance, the database cluster remains unchanged.

```
ReorganizeCluster ( cluster c )
  1.  if  μ(c, parent(c)) > 0  then MergeCluster(c);
  2.  else TryClusterSplit(c);
End.
```

Fig. 1. Cluster Reorganization Algorithm

The merging procedure is detailed in Fig. 2. The objects from the input cluster are transferred to the parent cluster (step 2). This yields the actualization

```
MergeCluster ( cluster c )
  1.  let a ← parent(c);
  2.  Move all objects from c to a;
  3.  let n(a) ← n(a) + n(c);
  4.  for each s in candidates(a) do
  5.    let M(s, c) ← {o ∈ objects(c) | o matches σ(s)};
  6.    let n(s) ← n(s) + card(M(s, c));
  7.  for each s in children(c) do
  8.    let parent(s) ← a;
  9.  Remove c from database;
End.
```

Fig. 2. Cluster Merge Algorithm

of the performance indicators associated to the clusters involved: the number of objects in the parent cluster, as well as the number of objects for each candidate subcluster of the parent cluster (steps 4-6). To preserve the clustering hierarchy, the parent cluster becomes the parent of the children of the input cluster (steps 7-8). Finally, the input cluster is removed from database (step 9).

The cluster split procedure is depicted in Fig. 3. The candidate subclusters promising the best materialization profits are selected in step 1: $\mathcal{B}$ is the set of the best candidate subclusters exhibiting positive profits. If $\mathcal{B}$ is not empty

```
TryClusterSplit ( cluster c )
  1.  let B ← {b ∈ candidates(c) | β(b, c) > 0 ∧
                β(b, c) ≥ β(d, c), ∀d ≠ b ∈ candidates(c)};
  2.  if (B ≠ ∅) then
  3.    let b ∈ B;
  4.    Create new database cluster d;
  5.    let M(b, c) = {o ∈ objects(c) | o matches σ(b)};
  6.    Move M(b, c) objects from c to d;
  7.    let σ(d) ← σ(b); let n(d) ← n(b); let parent(d) ← c;
  8.    let n(c) ← n(c) − n(d);
  9.    for each s in candidates(c) do
  10.     let M(s, d) ← {o ∈ objects(d) | o matches σ(s)};
  11.     let n(s) ← n(s) − card(M(s, d));
  12.   go to 1.
End.
```

Fig. 3. Cluster Split Algorithm

(step 2), one of its members becomes subject to materialization (step 3): a new database cluster is created (step 4), the objects qualifying for the selected candidate are identified (step 5) and moved from the input cluster to the new cluster

(step 6), the configuration of the new cluster is set in step 7 (signature, number of objects, parent cluster), the number of remaining objects in the input cluster is updated (step 10), as well as the number of objects in the candidate subclusters of the input cluster (steps 11-13). Steps 11-13 are necessary because objects from the input cluster might qualify for several candidate subclusters. This is possible because the candidate subclusters are virtual clusters. However, once removed from the input cluster, an object can no more qualify for the candidate subclusters. The split procedure continues with the selection of the next best candidate subclusters. The materialization process repeats from step 1 until no profitable candidate is found. The selection is performed in a greedy manner and the most profitable candidates are materialized first. To take into consideration the changes from the input cluster and from the candidate subclusters, induced by subsequent materializations, the set of best candidates $\mathcal{B}$ needs to be re-computed each time (step 1). At the end, the input cluster will host the objects not qualifying for any of the new materialized subclusters.

3.5 Object Insertion

When inserting a new object in the spatial database, beside the root cluster whose general signature accepts any object, other database clusters might also accommodate the corresponding object. These candidate clusters are identified based on their signatures. Among them, we choose to place the object in the one with the lowest access probability. Fig. 4 depicts our simple insertion strategy

ObjectInsertion (object o)
1. **let** $\mathcal{B} \leftarrow \{\mathsf{b} \in \mathcal{C} \mid \mathsf{o}$ matches $\sigma(\mathsf{b}) \ \wedge p(\mathsf{b}) \leq p(\mathsf{c}), \forall \mathsf{c} \neq \mathsf{b} \in \mathcal{C}\}$;
2. **let** $\mathsf{b} \in \mathcal{B}$;
3. Insert object o in selected cluster b;
4. **let** $n(\mathsf{b}) \leftarrow n(\mathsf{b}) + 1$;
5. **let** $\mathcal{S} \leftarrow \{\mathsf{s} \in \mathit{candidates}(\mathsf{b}) \mid \mathsf{o}$ matches $\sigma(\mathsf{s})\}$;
6. **for each** s **in** $\mathcal{S}$ **do**
7. **let** $n(\mathsf{s}) \leftarrow n(\mathsf{s}) + 1$;

End.

Fig. 4. Object Insertion Algorithm

(steps 1-3). Object insertion needs to update statistics n of the selected cluster, and of the candidate subclusters of the selected cluster (steps 4-6).

3.6 Spatial Query Execution

A *spatial query* (or *spatial selection*) specifies the *query object* and the *spatial relation* (*intersection, containment, or enclosure*) requested between the query object and the database objects from the answer set.

Answering a spatial query implies the exploration of the materialized clusters whose signatures satisfy the spatial relation with respect to the query object. The objects from the explored clusters are individually checked against the spatial selection criterion. The spatial query execution algorithm is straightforward and is depicted in Fig. 5. The number of exploring queries is incremented for each explored cluster, as well as for the corresponding candidate subclusters virtually explored (steps 7-10).

```
SpatialQuery ( query object ρ ) : object set
  1.  R ← ∅; // query answer set
  2.  let X ← {c ∈ C | ρ matches σ(c)};
  3.  for each cluster c ∈ X do
  4.    for each object o in objects(c) do
  5.      if (ρ matches o) then
  6.        let R ← R ∪ {o};
  7.    q(c) ← q(c) + 1;
  8.    let S ← {s ∈ candidates(c) | ρ matches σ(s)};
  9.    for each s in S do
  10.     q(s) ← q(s) + 1;
  11. return R;
End.
```

Fig. 5. Spatial Query Execution Algorithm

4 Clustering Criterion

This section presents our approach to group objects, more flexible than traditional methods which are based on minimum bounding in all dimensions. Basically, it consists in clustering together objects with "similar" intervals on a restrained number of dimensions. We first define the cluster signatures used to implement our clustering criterion. Then we present an instantiation of the clustering function which applies on such cluster signatures.

4.1 Cluster Signatures

Let N_d be the data space dimensionality. We consider each dimension taking values in the domain $[0,1]$. A spatial object specifies an interval for each dimension: $o = \{d_1[a_1,b_1], d_2[a_2,b_2], \ldots, d_{N_d}[a_{N_d},b_{N_d}]\}$ where $[a_i,b_i]$ represents the interval defined by the spatial object o in dimension d_i ($0 \leq a_i \leq b_i \leq 1, \forall i \in \{1,2,\ldots,N_d\}$).

A *cluster* represents a group of spatial objects. To form a cluster we put together objects defining *similar intervals* for the same dimensions. By *similar intervals* we understand intervals located in the same domain regions (for instance in the first quart of the domain). The grouping intervals/dimensions are represented in the cluster signature. The cluster signature is defined as

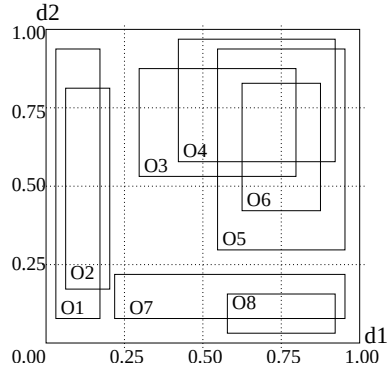
$$\sigma = \{d_1 \ [a_1^{min}, a_1^{max}] : [b_1^{min}, b_1^{max}], \quad d_2 \ [a_2^{min}, a_2^{max}] : [b_2^{min}, b_2^{max}], \\ \ldots, \quad d_{N_d} \ [a_{N_d}^{min}, a_{N_d}^{max}] : [b_{N_d}^{min}, b_{N_d}^{max}] \ \}$$

where σ regroups spatial objects whose intervals in dimensions d_i start between a_i^{min} and a_i^{max}, and end between b_i^{min} and b_i^{max}, $\forall i \in \{1,2,\ldots,N_d\}$. The *intervals of variation* $[a^{min}, a^{max}]$ and $[b^{min}, b^{max}]$ are used to precisely define the notion of interval similarity: All the intervals starting in $[a^{min}, a^{max}]$ and ending in $[b^{min}, b^{max}]$ are considered similar with respect to $a^{min}, a^{max}, b^{min}$ and b^{max}.

Example 1. The signature of the *root cluster* must accept any spatial object. For this reason, the intervals of variation corresponding to the signature of the root cluster are represented by complete domains in all dimensions:

$\sigma_r = \{d_1[0,1] : [0,1], \ldots, d_{N_d}[0,1] : [0,1]\}$.

Example 2. Considering the objects $O_1, O_2, O_3, \ldots, O_8$ from the 2-dimensional space depicted in Fig. 6, we can form 3 sample clusters as follows:



O_1 and O_2 in a cluster represented by σ_1:
$\sigma_1 = \{d_1[0.00, 0.25) : [0.00, 0.25),$
$d_2[0.00, 1.00] : [0.00, 1.00]\};$

O_3 and O_4 in a cluster represented by σ_2:
$\sigma_2 = \{d_1[0.25, 0.50) : [0.75, 1.00],$
$d_2[0.50, 0.75) : [0.75, 1.00]\};$

O_5 and O_6 and O_8 in a cluster repres. by σ_3:
$\sigma_3 = \{d_1[0.50, 0.75) : [0.75, 1.00],$
$d_2[0.00, 1.00] : [0.00, 1.00]\}.$

Fig. 6. Example 2

4.2 Clustering Function

The role of the clustering function is to compute the signatures of the candidate subclusters of a given cluster. Many possible signatures can be used to group objects. A good clustering function should solve the following trade-off: On one hand, the number of candidate subclusters should be sufficiently large to ensure good opportunities of clustering. On the other hand, if this number is too large, it will increase the cost of maintaining statistics (recall that performance indicators are maintained for each candidate subcluster). Our clustering function works as follows: Given a cluster signature we iteratively consider each dimension. For each dimension we divide both intervals of variation in a fixed number of subintervals. We call *division factor* and note f the number of subintervals. We then replace the pair of intervals of variation by each possible combination of subintervals. We have f^2 combinations of subintervals for each dimension and thus f^2 subsignatures[3]. Since we apply this transformation on each dimension we obtain $N_d \cdot f^2$ subsignatures. As a result, the number of candidate subclusters keeps linear with the number of dimensions.

Example 3. We consider σ_1 from the preceding example and apply the clustering function on dimension d_1 using a division factor $f = 4$. The signatures of the corresponding candidate subclusters are:

$\sigma_1^1 = \{d_1[0.0000, 0.0625) : [0.0000, 0.0625), d_2[0, 1] : [0, 1]\};$
$\sigma_1^2 = \{d_1[0.0000, 0.0625) : [0.0625, 0.1250), d_2[0, 1] : [0, 1]\};$
$\sigma_1^3 = \{d_1[0.0000, 0.0625) : [0.1250, 0.1875), d_2[0, 1] : [0, 1]\};$
$\sigma_1^4 = \{d_1[0.0000, 0.0625) : [0.1875, 0.2500], d_2[0, 1] : [0, 1]\};$
$\sigma_1^5 = \{d_1[0.0625, 0.1250) : [0.0625, 0.1250), d_2[0, 1] : [0, 1]\};$
$\sigma_1^6 = \{d_1[0.0625, 0.1250) : [0.1250, 0.1875), d_2[0, 1] : [0, 1]\};$
$\sigma_1^7 = \{d_1[0.0625, 0.1250) : [0.1875, 0.2500], d_2[0, 1] : [0, 1]\};$

[3] When the intervals of variation of the selected dimension are identical, only $N_f = \frac{f \cdot (f+1)}{2}$ subintervals combinations are distinct because of the symmetry.

$\sigma_1^8 = \{d_1[0.1250, 0.1875) : [0.1250, 0.1875), d_2[0,1] : [0,1]\}$;
$\sigma_1^9 = \{d_1[0.1250, 0.1875) : [0.1875, 0.2500], d_2[0,1] : [0,1]\}$;
$\sigma_1^{10} = \{d_1[0.1875, 0.2500] : [0.1875, 0.2500], d_2[0,1] : [0,1]\}$.

There are 16 possible subintervals combinations, but only 10 are valid because of the symmetry. Similarly, applying the clustering function on d_2 we use the subintervals $[0.00, 0.25)$, $[0.25, 0.50)$, $[0.50, 0.75)$ and $[0.75, 1.00]$ and obtain 10 more candidate subclusters.

5 Cost Model and Benefit Functions

Our database clustering is based on a cost model evaluating the average query performance in terms of execution time. The expected query execution time associated to a database cluster c can be generally expressed as:

$$T_c = A + p_c \cdot (B + n_c \cdot C) \quad (1)$$

where p_c represents the access probability associated to the cluster c, n_c the number of objects hosted by c, and A, B and C three parameters depending on the database and system characteristics. The access probability and the number of objects are performance indicators we maintain for each database cluster. Regarding parameters A, B and C, we consider the following scenarios:

i. Memory Storage Scenario. The spatial database fits the main memory, the objects from the same clusters are sequentially stored in memory in order to maximize the data locality and benefit from the memory cache line and read ahead capabilities of the nowadays processors. In this case:

A represents the time spent to check the cluster signature in order to decide or not the cluster exploration (signature verification time);

B includes the time required to prepare the cluster exploration (call of the corresponding function, initialization of the object scan) and the time spent to update the query statistics for the current cluster and for the candidate subclusters of the current cluster;

C represents the time required to check one object against the selection criterion (object verification time).

ii. Disk Storage Scenario. The signatures of the database clusters, as well as the associated statistics and parameters, are managed in memory, while the cluster members are stored on external support. The objects from the same clusters are sequentially stored on disk in order to minimize the disk head repositioning and benefit from the the better performance of the sequential data transfer between disk and memory. In this case:

$A' = A$ the same as in the first scenario;

$B' = B$ plus the time required to position the disk head at the beginning of the cluster in order to prepare the object read (disk access time), because the cluster is stored on external support.

$C' = C$ plus the time required to transfer one object from disk to memory (object read time).

Materialization Benefit Function. The materialization benefit function β takes a database cluster c and one of its candidate subclusters s, and evaluates

the impact on the query performance, of the potential materialization of s. To obtain the expression of β, we consider the corresponding query execution times before and after the materialization of the candidate subcluster: $T_{bef} = T_c$ and $T_{aft} = T_{c'} + T_s$. T_{bef} represents the execution time associated to the original database cluster c, and T_{aft} represents the joint execution time associated to the clusters c' and s resulted after the materialization of the candidate s of c. The materialization benefit function is defined as

$$\beta(s, c) = T_{bef} - T_{aft} = T_c - (T_{c'} + T_s) \tag{2}$$

and represents the profit in terms of execution time, expected from the materialization of the candidate s of c. Using (1) to expand $T_c = A + p_c \cdot (B + n_c \cdot C)$, $T_{c'} = A + p_{c'} \cdot (B + n_{c'} \cdot C)$, and $T_s = A + p_s \cdot (B + n_s \cdot C)$, and assuming i. $p_{c'} = p_c$ and ii. $n_{c'} = n_c - n_s$, (2) becomes:

$$\beta(s, c) = ((p_c - p_s) \cdot n_s \cdot C) - (p_s \cdot B) - A \tag{3}$$

The interest of the materialization grows when the candidate subcluster has a lower access probability, and when enough objects from the original cluster qualify for the considered candidate subcluster.

Merging Benefit Function. The merging benefit function μ takes a cluster c and its parent cluster a, and evaluates the impact on the query performance of the possible merging of the two clusters. To obtain the expression of μ, we consider the corresponding query execution times before and after the merging operation: $T_{bef} = T_c + T_a$ and $T_{aft} = T_{a'}$. T_{bef} represents the joint execution time associated to the original database cluster c and to the parent cluster a, and T_{aft} represents the execution time associated to the cluster a' resulted from merging c to a. The materialization benefit function is defined as

$$\mu(c, a) = T_{bef} - T_{aft} = (T_c + T_a) - T_{a'} \tag{4}$$

and represents the profit in terms of execution time, expected from the merging operation between clusters c and a. Using (1) to expand $T_c = A + p_c \cdot (B + n_c \cdot C)$, $T_a = A + p_a \cdot (B + n_a \cdot C)$, and $T_{a'} = A + p_{a'} \cdot (B + n_{a'} \cdot C)$, and assuming i. $p_{a'} = p_a$ and ii. $n_{a'} = n_a + n_c$, (4) becomes:

$$\mu(c, a) = A + (p_c \cdot B) - ((p_a - p_c) \cdot n_c \cdot C) \tag{5}$$

The interest in a merging operation grows when the access probability of the child cluster approaches the one of the parent cluster (for instance due to changes in query patterns), or when the number of objects in the child cluster decreases too much (due to object removals).

6 Implementation Considerations

Cost Model Parameters. Parameters A, B, and C are part of the cost model supporting the clustering strategy and depend on the system performance with respect to the adopted storage scenario. They can be either experimentally measured and hard-coded in the cost model, or dynamically evaluated for each

Table 2. I/O and CPU Operations Costs

I/O	Cost	CPU	Cost
Disk Access Time	15ms	Cluster Signature Check	$5 \cdot 10^{-7}$ ms
Disk Transfer Rate	20MBytes/sec	Object Verification Rate	300Mbytes/sec
Transfer Time per Byte	$4.77 \cdot 10^{-5}$ ms	Verification Time per Byte	$3.18 \cdot 10^{-6}$ ms

database cluster and integrated as model variables to locally support the clustering decision. Cost values for I/O and CPU operations corresponding to our system are presented as reference in Table 2.

Clustering Function. For the clustering function we used a domain division factor $f = 4$. According to Section 4, the number of candidate subclusters associated to a database cluster is between $10 * N_d$ and $16 * N_d$ where N_d represents the space dimensionality. For instance, considering a 16-dimensional space, we have between 160 and 256 candidate subclusters for each database cluster. Because the candidate subclusters are virtual, only their performance indicators have to be managed.

Storage Utilization. As part of our clustering strategy, each cluster is sequentially stored in memory or on external support. This placement constraint can trigger expensive cluster moving operations during object insertions. To avoid frequent cluster moves, we reserve a number of places at the end of each cluster created or relocated. For the number of reserved places, we consider between 20% and 30% of the cluster size, thus taking into account the data distribution. Indeed, larger clusters will have more free places than smaller clusters. In all cases, a storage utilization factor of at least 70% is ensured.

Fail Recovery. In the disk-based storage case, maintaining the search structure across system crashes can be an important consideration. For recovery reasons, we can store the cluster signatures together with the member objects and use an one-block disk directory to simply indicate the position of each cluster on disk. Performance indicators associated to clusters might be also saved, on a regular basis, but this is optional since new statistics can be eventually gathered.

7 Performance Evaluation

To evaluate our adaptive cost-based clustering solution, we performed extensive experiments executing intersection-based and point-enclosing queries over large collections of spatial objects (hyper-rectangles with many dimensions and following uniform and skewed spatial distributions). We compare our technique to Sequential Scan and to R*-tree evaluating the query execution time, the number of cluster/node accesses, and the size of verified data.

7.1 Experimental Setup

Competitive Techniques. R*-tree is the most successful R-tree variant still supporting multidimensional extended objects. It has been widely accepted in literature and often used as reference for performance comparison. Sequential Scan is a simple technique: it scans the database and checks all the objects

against the selection criterion. Although quantitatively expensive, Sequential Scan benefits of good data locality, and of sustained data transfer rate between disk and memory. Sequential Scan is considered a reference in high-dimensional spaces because it often outperforms complex indexing solutions [3] [5].

Experimental Platform. All experiments are executed on a Pentium III workstation with i686 CPU at 650MHz, 768MBytes RAM, several GBytes of secondary storage, and operating under Red Hat Linux 8.0. The system has a SCSI disk with the following characteristics: disk access time = 15ms, sustained transfer rate = 20MBps. To test the disk-based storage scenario, we limited the main memory capacity to 64MBytes and used experimental databases of multidimensional extended objects whose sizes were at least twice larger than the available memory. This way we forced data transfer between disk and memory.

Data Representation. A spatial object consists of an object identifier and of N_d pairs of real values representing the intervals in the N_d dimensions. The interval limits and the object identifier are each represented on 4 bytes. The R*-tree implementation follows [2]. In our tests we used a node page size of 16KBytes. Considering a storage utilization of 70%, a tree node accommodates 35 objects with 40 dimensions, and 86 objects with 16 dimensions. Using smaller page sizes would trigger the creation of too many tree nodes resulting in high overheads due to numerous node accesses, both in memory and on disk.

Execution Parameters and Performance Indicators. The following parameters are varied in our tests: number of database objects (up to 2,000,000), number of dimensions (from 16 to 40), and query selectivity (between 0.00005% and 50%). In each experiment, a large number of spatial queries is addressed to the indexing structure and average values are raised for the following performance indicators: query execution time (combining all costs), number of accessed clusters/nodes (relevant for the cost due to disk access operations), size of verified data (relevant for data transfer and check costs).

Experimental Process. For Sequential Scan, the database objects are loaded and stored in a single cluster. Queries are launched, and performance indicators are raised. For R*-tree, the objects are first inserted in the indexing structure, then query performance is evaluated. For Adaptive Clustering, the database objects are inserted in the root cluster, then a number of queries are launched to trigger the object organization in clusters. A database reorganization is triggered every 100 spatial queries. If the query distribution does not change, the clustering process reaches a stable state (in less than 10 reorganization steps). We then evaluate the average query response time. The reported time also includes the time spent to update query statistics associated to accessed clusters.

7.2 Experiments

Uniform Workload and Varying Query Selectivity (2,000,000 objects). In the first experiment we examine the impact of the query selectivity on the query performance. We consider 2,000,000 database objects uniformly-distributed in a 16-dimensional data space (251MBytes of data) and evaluate the query response time for intersection queries with selectivities varying from 0.00005% to

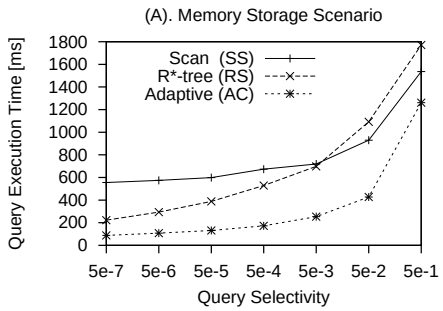


Table 1. Memory - Data Access						
Query	Clusters		Expl. %		Objs. %	
Select.	AC	RS	AC	RS	AC	RS
5e-7	25561	24337	6	28	9	29
5e-6	24624	24337	7	35	11	36
5e-5	21168	24337	9	46	14	47
5e-4	16150	24337	12	58	19	59
5e-3	10483	24337	18	72	26	73
5e-2	7760	24337	22	93	35	94
5e-1	3940	24337	54	100	71	100

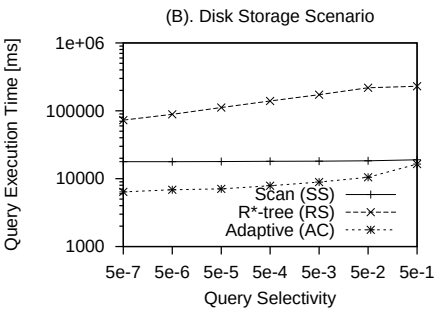


Table 2. Disk - Data Access						
Query	Clusters		Expl. %		Objs. %	
Select.	AC	RS	AC	RS	AC	RS
5e-7	1360	24337	12	28	19	29
5e-6	1115	24337	14	35	22	36
5e-5	826	24337	16	46	25	47
5e-4	539	24337	22	58	32	59
5e-3	464	24337	28	72	39	73
5e-2	276	24337	35	93	49	94
5e-1	157	24337	60	100	84	100

Fig. 7. Query Performance when Varying Query Selectivity (Uniform Workload)

50%. Each database object defines intervals whose sizes and positions are randomly distributed in each dimension. The intervals of the query objects are also uniformly generated in each dimension, but minimal/maximal interval sizes are enforced in order to control the query selectivity. Performance results are presented in Fig. 7 for both storage scenarios: in-memory and disk-based. Charts A and B illustrate average query execution times for the three considered methods: Sequential Scan (SS), Adaptive Clustering (AC), and R*-tree (RS). Tables 1 and 2 compare AC and RS in terms of total number of clusters/nodes, average ratio of explored clusters/nodes, and average ratio of verified objects. Unlike RS for which the number of nodes is constant, AC adapts the object clustering to the actual data and query distribution. When the queries are very selective many clusters are formed because few of them are expected to be explored. In contrast, when the queries are not selective fewer clusters are created. Indeed, their frequent exploration would otherwise trigger significant cost overhead. The cost model supporting the adaptive clustering always ensures better performance for AC compared to SS[4]. RS is much more expensive than SS on disk, but also in memory for queries with selectivities over 0.5%. The bad performance of RS confirms our expectations: RS can not deal with high dimensionality (16 in this case)

[4] The cost of SS in memory increases significantly (up to 3x) for lower query selectivities. This happens because an object is rejected as soon as one of its dimensions does not satisfy the intersection condition. When the query selectivity is low, more attributes have to be verified on average.

because the MBBs overlap within nodes determines the exploration of many tree nodes[5]. AC systematically outperforms RS, exploring fewer clusters and verifying fewer objects both in memory and on disk. Our object grouping is clearly more efficient. In memory, for instance, we verify three times fewer objects than RS in most cases. Even for queries with selectivities as low as 50%, when RS practically checks the entire database, only 71% of objects are verified by AC. The difference in number of verified objects is not so substantial on disk, but the cost overhead due to expensive random I/O accesses is remarkably inferior[6]. This happens because the number of AC clusters is much smaller than the number of RS nodes. Compared to the memory storage scenario, the small number of clusters formed on disk is due to the cost model that takes into consideration the negative impact of expensive random I/O accesses. This demonstrates the flexibility of our adaptive cost-based clustering strategy. Thanks to it AC succeeds to outperform SS on disk in all cases.

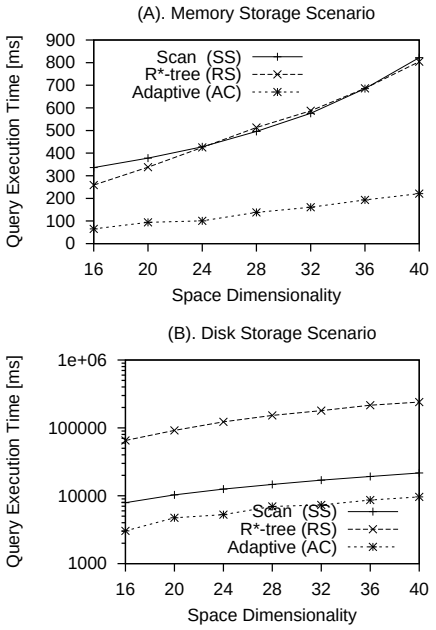


Table 1. Memory - Data Access

Nb.of	Clusters		Expl. %		Objs. %	
Dims.	AC	RS	AC	RS	AC	RS
16	7202	12092	10	58	14	58
20	8520	15229	12	66	17	67
24	9214	18542	12	72	18	72
28	10419	21975	13	74	18	75
32	12397	24922	12	76	17	77
36	14281	28420	13	77	17	78
40	16001	31766	11	78	17	78

Table 2. Disk - Data Access

Nb.of	Clusters		Expl. %		Objs. %	
Dims.	AC	RS	AC	RS	AC	RS
16	226	12092	21	58	31	58
20	272	15229	27	66	34	67
24	311	18542	23	72	32	72
28	371	21975	27	74	35	75
32	423	24922	24	76	32	77
36	480	28420	24	77	32	78
40	520	31766	24	78	32	78

Fig. 8. Query Performance when Varying Space Dimensionality (Skewed Data)

Skewed Workload and Varying Space Dimensionality (1,000,000 objects). With this experiment we intend to demonstrate both the good behavior with increasing dimensionality and the good performance under skewed data. Skewed data is closer to reality where different dimensions exhibit different characteristics. For this test, we adopted the following skewed scenario: we generate uniformly-distributed query objects with no interval constraints, but consider

[5] See ratio of explored nodes in Tables 1 and 2.
[6] Note the logarithmic time scale from Chart 7-B.

1,000,000 database objects with different size constraints over dimensions. We vary the number of dimensions between 16 and 40. For each database object, we randomly choose a quart of dimensions that are two times more selective than the rest of dimensions. We still control the global query selectivity because the query objects are uniformly distributed. For this experiment we ensure an average query selectivity of 0.05%. Performance results are illustrated in Fig. 8. We first notice that the query time increases with the dimensionality. This is normal because the size of the dataset increases too from 126MBytes (16d) to 309MBytes (40d). Compared to SS, AC again exhibits good performance, scaling well with the number of dimensions, both in memory and on disk. AC resists to increasing dimensionality better than RS. RS fails to outperform SS due to the large number of accessed nodes ($> 72\%$). AC takes better advantage of the skewed data distribution, and groups objects in clusters whose signatures are based on the most selective similar intervals and dimensions of the objects regrouped. In contrast, RS does not benefit from the skewed data distribution, probably due to the minimum bounding constraint, which increases the general overlap. In memory, for instance, RS verifies four times more objects than AC.

Point-Enclosing Queries. Because queries like "find the database objects containing a given point" can also occur in practice (for instance, in a publish-subscribe application where subscriptions define interval ranges as attributes, and events can be points in these ranges), we also evaluated point-enclosing queries considering different workloads and storage scenarios. We do not show here experimental details, but we report very good performance: up to 16 times faster than SS in memory, and up to 4 times on disk, mostly due to the good selectivity. Compared to spatial range queries (i.e. intersections with spatial extended objects), point-enclosing queries are best cases for our indexing method thanks to their good selectivity.

Conclusion on Experiments. While R*-tree fails to outperform Sequential Scan in many cases, our cost-based clustering follows the data and the query distribution and always exhibits better performance in both storage scenarios: in-memory and disk-based. Experimental results show that our method is scalable with the number of objects and has good behavior with increasing dimensionality (16 to 40 in our tests), especially when dealing with skewed data or skewed queries. For intersection queries, performance is up to 7 times better in memory, and up to 4 times better on disk. Better gains are obtained when the query selectivity is high. For point-enclosing queries on skewed data, gain can reach a factor of 16 in memory.

8 Conclusions

The emergence of new applications (such as SDI applications) brings out new challenging performance requirements for multidimensional indexing schemes. An advanced subscription system should support spatial range queries over large collections of multidimensional extended objects with many dimensions (millions of subscriptions and tens to hundreds of attributes). Moreover, such

system should cope with workloads that are skewed and varying in time. Existing structures are not well suited for these new requirements. In this paper we presented a simple clustering solution suitable for such application contexts. Our clustering method uses an original grouping criterion, more efficient than traditional approaches. The cost-based clustering allows us to scale with large number of dimensions and to take advantage of skewed data distribution. Our method exhibits better performance than competitive solutions like Sequential Scan or R*-tree both in memory and on disk.

References

1. M. Altimel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proc. 26th VLDB Conf., Cairo, Egypt*, 2000.
2. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Conf., Atlantic City, NJ*, 1990.
3. S. Berchtold, C. Böhm, and H.-P. Kriegel. The Pyramid-technique: Towards breaking the curse of dimensionality. In *Proc. ACM SIGMOD Conf., Seattle, Washington*, 1998.
4. S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proc. 22nd VLDB Conf., Bombay, India*, 1996.
5. K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? *Lecture Notes in Computer Science*, 1540:217–235, 1999.
6. C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
7. C. Böhm and H.-P. Kriegel. Dynamically optimizing high-dimensional index structures. In *Proc. 7th EDBT Conf., Konstanz, Germany*, 2000.
8. F. Fabret, H. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha. Filtering algorithm and implementation for very fast publish/subscribe systems. In *Proc. ACM SIGMOD Conf., Santa Barbara, California, USA*, 2001.
9. C. Faloutsos and P. Bhagwat. Declustering using fractals. *PDIS Journal of Parallel and Distributed Information Systems*, pages 18–25, 1993.
10. V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.
11. A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Conf., 47-57*, 1984.
12. H. Liu and H. A. Jacobsen. Modelling uncertainties in publish/subscribe systems. In *Proc. 20th ICDE Conf., Boston, USA*, 2004.
13. B.-U. Pagel, H.-W. Six, and M. Winter. Window query-optimal clustering of spatial objects. In *Proc. ACM PODS Conf., San Jose*, 1995.
14. T. Sellis, N. Roussopoulos, and C. Faloustos. The R+-tree: A dynamic index for multi-dimensional objects. In *Proc. VLDB Conf., Brighton, England*, 1987.
15. Y. Tao and D. Papadias. Adaptive index structures. In *Proc. 28th VLDB Conf., Hong Kong, China*, 2002.
16. R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th VLDB Conf., New York, USA*, 1998.
17. C. Yu. *High-dimensional indexing. Transformational approaches to high-dimensional range and similarity searches.* LNCS 2341, Springer-Verlag, 2002.