

Scalable QSF-Trees: Retrieving Regional Objects in High-Dimensional Spaces

Ratko Orlandic
Illinois Institute of Technology
Department of Computer Science
10 West 31st Street, Chicago, IL 60616
phone: 312-567-5343, fax: 312-567-5067
email: ratko@cs.iit.edu

Byunggu Yu
University of Wyoming
Department of Computer Science
PO Box 3315, Laramie, WY 82071-3315
phone: 307-766-2440, fax: 307-766-4036
email: yu@uwyo.edu

ABSTRACT

Many database applications require effective representation of regional objects in high-dimensional spaces. By applying an original query transformation, a recently proposed access method for regional data, called the simple QSF-tree (sQSF-tree), effectively attacks the limitations of traditional spatial access methods in spaces with many dimensions. Nevertheless, sQSF-trees are not immune to all problems associated with high data dimensionality. Based on the analysis of sQSF-trees, this paper presents a new variant of sQSF-trees, called the scalable QSF-tree (cQSF-tree), which relies on a heuristic optimization to reduce the number of false drops into pages that contain no object satisfying the query. By increasing the selectivity of search predicates, cQSF-trees improve the performance of multi-dimensional selections. Experimental evidence shows that cQSF-trees are more scalable than sQSF-trees to the growing data dimensionality. The performance improvements also increase with more skewed data distribution.

Keywords: database management, spatial access methods, data dimensionality.

INTRODUCTION

There is a large body of literature on the problem of accessing data in high-dimensional spaces (Berchtold et al., 1996 and 1998; Lin et al., 1995; Orlandic and Yu, 2002; Sakurai et al., 2000; Weber et al., 1998; White and Jain, 1996). However, the proposed techniques almost always assume data sets representing points in space. In many applications, effective representation of extended (regional) data is also important. Regional data are usually associated with low-dimensional spaces of geographic applications. However, through aggregation or clustering, such data may naturally appear in high-dimensional spaces as well.

For example, when the massive high-dimensional data of advanced scientific applications are clustered in files on tertiary storage, storage considerations often prevent the corresponding access structure from keeping the descriptors of all items in the repository. Instead, the content of each file can be approximated in the access structure by the minimal bounding rectangle (MBR) enclosing all data points of the given file (Orlandic, 2003). Similarly, in order to reduce the cost of dynamic updates, the multi-dimensional

databases of location-based services frequently approximate the position of a moving object by the bounded rectangle of a larger area in which the object currently resides. Since the position is usually only one of many relevant parameters describing a moving object, the index (access) structure appropriate for these environments must deal with regional data in spaces with possibly many dimensions.

Other applications in which regional objects naturally appear in higher dimensional spaces include on-line analytical processing as well as multimedia and image-recognition systems. In the latter systems, objects are usually mapped to long d -dimensional feature vectors. However, for the purposes of recognition, only a subset of features, called principal components, is actually used (Swets and Weng, 1996). After populating the projected space, images are grouped in classes, each of which can be represented by its approximate region in the space and stored in a spatial access method. In order to identify the most likely class for the given object, the process of image recognition must employ a form of spatial retrieval with the probabilistic ranking of retrieved objects.

Unlike point access methods (PAMs), spatial access methods (SAMs) are designed to support different search operators (e.g., overlap, containment, and enclosure) over both points and regional objects in multi-dimensional spaces (Gaede and Gunther, 1998). To reduce the storage overhead of the index structure, extended regional objects are typically approximated by their MBRs. There are many MBR-based SAMs, which are usually classified into: region-overlapping (Guttman, 1984; Beckmann et al., 1990), object-clipping (Sellis et al., 1987) and object-transformation (Pagel et al., 1993) schemes.

Unfortunately, each group of traditional SAMs suffers from major conceptual problems that have a tendency to grow with data dimensionality (Orlandic and Yu, 2000). We call these problems *conceptual* because they tend to be associated with the very idea underlying a group of SAMs. For example, the region overlap in R-trees (Guttman, 1984) and R*-trees (Beckmann et al., 1990) translates into a necessity to traverse many index paths, which increases the number of accessed *nodes* (*index pages*). However, the amount of overlap in these structures rapidly grows with data dimensionality (Berchtold et al., 1996). Object clipping (Sellis et al., 1987) creates multiple clips of a single regional object, which increases the size of the structure and degrades retrieval performance. Because the probability of clipping an object grows with dimensionality, these negative effects of clipping are more pronounced in higher dimensional spaces. A major drawback of object-transformation schemes (Pagel et al., 1993) is that a relatively small query window in the original space may map into a large search region in the transformed space. The magnitude of this problem increases rapidly as the number of dimensions grows (Orlandic and Yu, 2000).

Few access methods for high-dimensional data can accommodate extended regional objects. *X-trees* (Berchtold et al., 1996) and *simple QSF-trees*, or just *sQSF-trees* (Orlandic and Yu, 2000; Yu et al., 1999), are the exceptions. X-trees are designed to address the problem of region overlap in R*-trees. Instead of allowing splits that introduce high overlap, they extend index pages over the usual size. These clusters of pages, called super-nodes, are searched sequentially. Therefore, the advantages of the reduced overlap come at the expense of scanning the super-nodes and a higher complexity of dynamic updates.

By attacking the conceptual problems of traditional SAMs, sQSF-trees improve the performance of multi-dimensional queries in high-dimensional spaces. Unlike R-trees and R*-trees, which maintain hierarchies of possibly overlapping MBRs, they employ a simple modification of a PAM to avoid any region overlap. In contrast to traditional object transformations (Pagel et al., 1993), MBRs are not mapped to points in higher dimensional space. Instead, sQSF-trees apply an original query transformation that calculates search and filtering regions from the given query and uses these regions to search the index tree and to filter the result set. This is where the name "Query-to-Search-and-Filter"-trees (QSF-trees) comes from. Prior experiments (Orlandic and Yu, 2000) have shown that sQSF-trees outperform an improved variant of R*-trees (Papadias et al., 1995) and an object-transformation scheme (Seeger and Kriegel, 1988) by as much as an order of magnitude.

However, since the structure of sQSF-trees indexes only the low endpoints of the rectangular regions bounding the actual objects (*object MBRs*), it incurs a potentially large number of false drops into pages containing no objects that can satisfy the query. This paper proposes a variant of sQSF-trees, called the *scalable QSF-tree (cQSF-tree)*, which is designed to reduce the number of false drops in sQSF-trees. By propagating the information about the high endpoints of the object MBRs and increasing the selectivity of the search predicates, cQSF-trees improve the performance of multi-dimensional selections. The experimental evidence shows that cQSF-trees are much more scalable than sQSF-trees with respect to the increasing data dimensionality and more skewed data distributions.

In the rest of the paper, we review the design of sQSF-trees. Then we give an accurate analytical estimate of the expected sQSF-tree performance. Analyzing this estimate, we present both the idea and the design of cQSF-trees. Following that, we give the results of an experimental study comparing the performance of sQSF-trees and cQSF-trees over a range of spaces with different dimensionalities. Finally, we summarize the paper and discuss the practical appeal of the proposed indexing technique.

SIMPLE QSF-TREES

Like an object-transformation scheme, the sQSF-tree performs an explicit query transformation. However, while traditional object-transformation schemes (Seeger and Kriegel, 1988) map d -dimensional objects and queries onto their equivalents in a $2d$ -dimensional space, this query transformation takes place in the original d -dimensional space. To describe the query transformation, we consider four *topological relations (query predicates)* between two MBRs r' and q' , where \subseteq and \supseteq represent the *subset* and *superset* relations, respectively (e.g., $r' \supseteq q'$ means that every point of q' is also in the interior or on the boundary of r'):

equal (r', q') $\rightarrow r' = q'$;
covers (r', q') $\rightarrow r' \supseteq q'$;
covered_by (r', q') $\rightarrow r' \subseteq q'$; and
not_disjoint (r', q') $\rightarrow (r' \cap q' \neq \emptyset)$.

We assume a square d -dimensional universe U and universal sets R and P of all rectangles and points in U , respectively. Next, we define two functions $l, h: R \rightarrow P$. For each d -dimensional rectangle $r' \in R$, these functions give its low endpoint $l(r')$ and high endpoint $h(r')$, respectively. Due to the geometry of rectangles, the low and high endpoints are the vertices of the given rectangle (the d -dimensional vectors) with the lowest (highest) coordinates along each dimension $i = 1, \dots, d$. The coordinates of the low and the high endpoint of r' along each axis i are denoted by $l_i(r')$ and $h_i(r')$, respectively.

The sQSF-tree represents each object MBR r' by a pair $\langle l(r'), h(r') \rangle$ of its endpoints in the original d -dimensional space. For each dimension i , the implementation dynamically keeps track of two values, m_i and M_i . Given an MBR r' , let $r'_i = h_i(r') - l_i(r')$ be the *length* of its side along the axis i . Then, m_i and M_i are respectively the minimum and maximum r'_i among all object MBRs r' in the data set.

The basic question behind the query transformation of sQSF-trees can be formulated as follows: where could the low and high endpoints of the object MBRs that satisfy the query predicate possibly lie in the space? To answer this, the transformation uses the notions of the *L-region* and *H-region*, which are defined as the portions of space containing the low (high) endpoints of all possible object MBRs that could satisfy the given query predicate (*equal*, *covers*, *covered_by*, or *not_disjoint*). The precise coordinates of the corresponding L- and H-regions for each type of query are defined in (Yu et al., 1999).

Figure 1 illustrates the L- and H-regions generated for different topological relations with a query window q' . As in the rest of the paper, the origin of the universe is assumed to be in the low left corner of each figure. In the figure, v_{dim} is a d -dimensional vector whose component along each dimension i has magnitude $M_i - q'_i$, where q'_i is the length of the query window along this dimension. Similarly, v_{min} and v_{max} are d -dimensional vectors whose lengths along each dimension i are m_i and M_i , respectively.

For the relation *equal*, the regions L_e and H_e are just the low and high endpoints of the query window. For queries with the relation *covers*, the lengths of the regions L_c and H_c along the axis i are $M_i - q'_i$, unless they are truncated. For queries with the relation *covered_by*, the extents of the regions L_{cb} and H_{cb} along the same axis are $q'_i - m_i$. Finally, for the relation *not_disjoint*, the lengths of the i -th sides of the regions L_{nd} and H_{nd} , if they are not truncated, are $q'_i + M_i$.

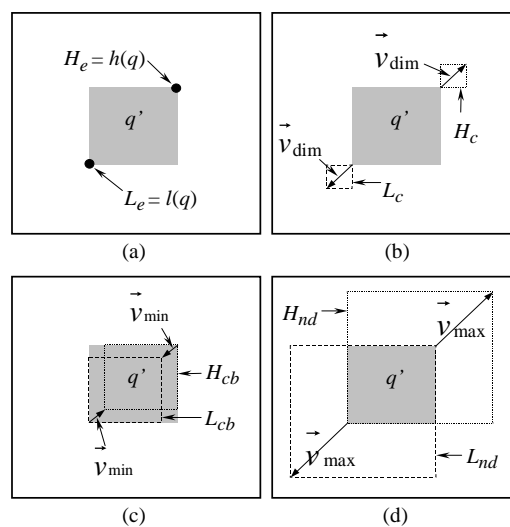


Figure 1: Query transformation of sQSF-trees.

The structure of sQSF-trees is a slightly modified PAM (Yu et al., 1999). When KDB-trees (Robinson, 1981) underlie the implementation of sQSF-trees, the insertion algorithm requires a simple modification to accommodate both low and high endpoints (l and h , respectively) of object MBRs at the leaf level of the tree structure. However, the leaf-level entries are indexed solely by the low endpoints. Since the space-partitioning strategy is that of KDB-trees when taking into account only the low endpoints of object MBRs, the format of interior entries is the same as in KDB-trees. As in (Orlandic and Yu, 2000; Yu et al., 1999), we assume here the splitting policy for interior pages based on the first-division plane (Freeston, 1995; Orlandic and Yu, 2001), which avoids downward propagation of splits associated with forced splitting of the original KDB-trees (Robinson, 1981). This enables greater storage utilization than in the original KDB-trees and an improved retrieval performance (Orlandic and Yu, 2001). As a forward reference, the space partition and the structure of sQSF-trees are illustrated in Figure 3a.

Table 1: Search predicates of sQSF-trees.

Query Predicate	Search Predicate of sQSF-trees
<i>equal</i> (r', q')	<i>covers</i> (R', L_e)
<i>covers</i> (r', q')	<i>not_disjoint</i> (R', L_c)
<i>covered_by</i> (r', q')	<i>not_disjoint</i> (R', L_{cb})
<i>not_disjoint</i> (r', q')	<i>not_disjoint</i> (R', L_{nd})

With the L- and H-regions, the original query is translated into the problem of finding MBRs whose low endpoints lie in the L-region and whose high endpoints lie in the H-region. While traversing the interior nodes (pages) of the index tree, the search operations rely solely on the L-region. Table 1 shows the search predicates applied to the interior entries. In the table, R' denotes the rectangular region representing an index page of the sQSF-tree at one level below. When searching a leaf page, each object MBR whose low endpoint lies in the L-region is further checked to see whether its high endpoint falls in the H-region.

The query transformation translates the original query involving extended objects into a problem of finding relevant points in the space. Since the later problem is resolved by employing a KDB-tree index structure that partitions the space into non-overlapping regions, sQSF-trees automatically eliminate the possibility of region overlap. In the process, they avoid object clipping and the transformation of object MBRs into points of a dual ($2d$ -dimensional) space. Furthermore, because the L- and H-regions are tuned to the semantics of individual queries, they achieve a differentiation of search operations with different query predicates that can further reduce the number of accessed pages per average query.

The experimental studies show that sQSF-trees outperform traditional spatial access methods, typically by a significant margin (Orlandic and Yu, 2000; Yu et al., 1999). As the number of dimensions increases, the improvements grow quickly, eventually stabilizing at a high level. For example, using the number of page accesses (which, ignoring the effects of buffering, translate into disk accesses) as the measure of retrieval performance, sQSF-trees outperformed both an optimized version of R*-trees (Papadias et al., 1995) and an object-transformation scheme (Seeger and Kriegel, 1988) by as much as an order of magnitude (Orlandic and Yu, 2000). Since sQSF-trees, R*-trees and the object-transformation scheme are structurally similar, no significant difference was observed with respect to their storage requirements.

ANALYSIS OF SIMPLE QSF-TREES

Let us assume an implementation of sQSF-trees based on KDB-trees. Considering an sQSF-tree that contains a large (but fixed) number n of object MBRs uniformly distributed in the d -dimensional universe, the structure has some $h > 1$ levels, where the leaf pages are at level $k = 1$, while the root is at level $k = h$. The length of an object MBR along each dimension is randomly chosen between m_i and M_i . Thus, the sides of each object MBR (as well as each query window) are independent of each other. We also assume that the minimum object size m_i along each dimension i is relatively small.

Every node (page) of an sQSF-tree is represented by the *node's bounding rectangle (node BR)*, stored in an index entry of the parent page. The node BRs are induced by the KDB-tree partition of the space that takes into account only the low endpoints of object MBRs. If the given index page I is at level $k = 1$ (leaf level), its node BR covers the low endpoints of object MBRs stored in I . On the other hand, if the index page I is at level $k > 1$, its corresponding BR covers the low endpoints of all object MBRs stored in the subtree rooted at I . Due to the policy of first-division splitting (Orlandic and Yu, 2001), the node BRs at any level do not overlap and completely cover the universe. The BR of the root page is the universe itself.

The performance of spatial selections, measured in terms of page accesses, is governed by the search predicates of Table 1 applied to the interior entries of the sQSF-tree. At each level of the tree, the search procedure visits all index pages whose node BRs satisfy the search predicate. Ignoring the queries that generate no page access, the root page is always accessed. Therefore, given a query with a topological relation t , the number $Q_t(d)$ of page accesses in an sQSF-tree is:

$$Q_t(d) = 1 + \sum_{k=1}^{h-1} Q_t(d, k),$$

where $Q_t(d, k)$ is the number of pages accessed at a level $k < h$ of the sQSF-tree. $Q_t(d, k)$ is determined by the number $N(d, k)$ of pages at the level k and the probability that a page at this level is accessed (i.e., the probability that the node's BR satisfies the search predicate).

Consider the search predicates of Table 1. Given a topological relation t , a node BR R' , and an L-region L_t of the given query, the search predicates of Table 1 test whether: $covers(R', L_t)$ or $not_disjoint(R', L_t)$. For the relation *equal*, since the L-region L_e is just a point, the corresponding search predicate $covers(R', L_e)$ can be represented as $not_disjoint(R', L_e)$. Consequently, for any type of query predicate t , the probability of accessing a page with the BR R' is the probability that $not_disjoint(R', L_t)$ is true.

Given an L-region L_t , the probability that a page is accessed depends on the size of its corresponding node BR. Thus, to derive an accurate estimate of the search performance of sQSF-trees, one must consider the volumes of node BRs at any level of the structure. The key observation here is that, if the d -dimensional universe is uniformly filled with rectangles and the minimum object size is relatively small, the distribution of the low endpoints of object MBRs tends to be uniform.

Under the assumption that points are uniformly distributed, our analysis of KDB-trees (Orlandic and Yu, 2002) has shown that, at any level $k < h$ with $N(d, k)$ pages, the KDB-tree division of the space produces only two types of node BRs. Let $j = \lfloor \log_2(N(d, k)) / d \rfloor$, i.e. $j \geq 0$ is the maximum integer for which $2^{dj} \leq N(d, k)$, and $j' = \lfloor \log_2(N(d, k)) - d \cdot j \rfloor$, i.e. $j' \geq 0$ is the maximum integer for which $2^{d \cdot j'} \leq N(d, k)$. Then, at each level $k < h$ there are:

- 1) approximately $2(N(d, k) - 2^{d \cdot j + j'})$ node BRs whose average extent is $1/2^{j+1}$ along the first $j'+1$ dimensions and $1/2^j$ along the remaining dimensions; and
- 2) about $2^{d \cdot j + j'} - N(d, k)$ node BRs whose average extent is $1/2^{j+1}$ along the first j' dimensions and $1/2^j$ along the remaining dimensions.

A formal proof of this fact can be found in (Orlandic and Yu, 2002).

Now, let $P(R'_i, L_{t,i})$ denote the probability that, along dimension i , a node BR R' , whose extent along axis i is R'_i , is not disjoint with the L-region L_t , whose extent along i is $L_{t,i}$. Then, the number $Q_t(d, k)$ of page accesses at each level $k < h$ of an sQSF-tree can be estimated as follows:

$$Q_t(d, k) \approx (2N(d, k) - 2^{d \cdot j + j'}) \prod_{i=1}^{j'+1} P(1/2^{j+1}, L_{t,i}) \prod_{i=j'+2}^d P(1/2^j, L_{t,i}) + (2^{d \cdot j + j'} - N(d, k)) \prod_{i=1}^{j'} P(1/2^{j+1}, L_{t,i}) \prod_{i=j'+1}^d P(1/2^j, L_{t,i}).$$

Substituting the above estimate of $Q_t(d, k)$ in the earlier equation, one arrives at the following theorem:

Theorem 1. Given a multi-dimensional query with a topological relation t and the corresponding L-region L_t , the expected number of page accesses in an sQSF-tree is:

$$Q_t(d) \approx 1 + \sum_{k=1}^{h-1} \left((2N(d, k) - 2^{d \cdot j + j'}) \prod_{i=1}^{j'+1} P(1/2^{j+1}, L_{t,i}) \prod_{i=j'+2}^d P(1/2^j, L_{t,i}) + (2^{d \cdot j + j'} - N(d, k)) \prod_{i=1}^{j'} P(1/2^{j+1}, L_{t,i}) \prod_{i=j'+1}^d P(1/2^j, L_{t,i}) \right),$$

where $j = \lfloor \log_2(N(d, k)) / d \rfloor$ and $j' = \lfloor \log_2(N(d, k)) - d \cdot j \rfloor < d$.

Each value $P(1/2^m, L_{t,i})$ in the theorem, where m is either j or $j+1$, can be obtained using the estimate of the probability of overlap derived in (Berchtold et al., 1998):

$$P(1/2^m, L_{t,i}) \approx \frac{\min\{l_i(L_t) + L_{t,i}, 1 - 1/2^m\} - \max\{l_i(L_t) - 1/2^m, 0\}}{1 - 1/2^m},$$

where $l_i(L_t)$ is the i -th coordinate of the low endpoint of L_t . Note, when $m = 0$, $P(1/2^m, L_{t,i})$ evaluates to 1.

The only remaining task is to estimate $N(d, k)$. Let $C(d)$ be the page capacity (the maximum number of entries in a single page) of a d -dimensional sQSF-tree. With a KDB-tree implementation of sQSF-trees, the capacity $C(d)$ of both leaf and interior pages is virtually the same. For a large number of uniformly distributed objects, the average page utilization produced by the KDB-tree splitting algorithm is approximately $\ln 2$ (Orlandic and Yu, 2002). Therefore, for all $k < h$, $N(d, k) \approx n/(\ln 2 \cdot C(d))^k$.

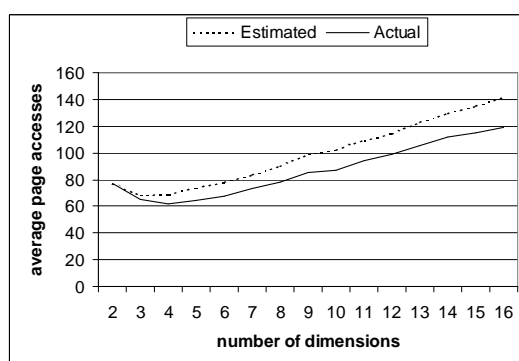


Figure 2: Estimated and observed number of page accesses per average query.

To verify the accuracy of Theorem 1, we performed an experiment with randomly generated d -dimensional object MBRs whose length along each dimension was relatively small (between 0.1% and 0.5% of the maximum extent along the dimension). For each dimensionality d between 2 and 16, we constructed an sQSF-tree with exactly 65,536 objects. The page capacity was 4K bytes. Each type of spatial selection was performed 10,000 times with randomly generated query windows. In the experiment, the L-region of each query was supplied to the actual sQSF-tree implementation as well as a routine that calculates the expected number of page accesses based on Theorem 1. The average page utilization was set to $\ln 2$. As one can see from Figure 2, the estimated performance of spatial selections for an average query was very close to the observed number of page accesses on the actual sQSF-trees.

SCALABLE QSF-TREES

While sQSF-trees eliminate certain conceptual problems of contemporary spatial access methods, they are not immune to all problems associated with high data dimensionality. Since the structure of sQSF-trees is a simple modification of a point access method, it inherits all problems of the underlying PAM. For example, since we have chosen KDB-trees as the underlying PAM structure, the size of index entries increases proportionally to the dimensionality d . This increases the storage overhead and decreases retrieval performance. To address some of these problems, we have proposed a variant of KDB-trees optimized for high-dimensional spaces (Orlandic and Yu, 2002).

Another way to improve the performance of sQSF-trees can become obvious after carefully considering Theorem 1. As one can see from the theorem, the expected number of page accesses per query (generally used as a measure of retrieval performance) is determined by the probability that any given node BR and the given L-region overlap (are not disjoint). Thus, even though sQSF-trees calculate both L- and H-regions, only the L-region figures into the expected retrieval performance. This is due to the fact that sQSF-trees index only the low endpoints of object MBRs. As a result, they can incur many false drops, especially in high-dimensional situations where the regions enclosing the high-endpoints of object MBRs in leaf pages tend to be relatively small.

By propagating the information about not only low but also high endpoints of object MBRs to the interior levels of the index tree, many of these false drops could be eliminated. One way to propagate this information is to extend each interior entry e_i of the index tree to include the MBR enclosing the high endpoints of all subordinate object MBRs in the tree (object MBRs appearing in the subtree rooted at e_i). Unfortunately, this would almost double the size of interior entries, which would reduce the capacity of interior pages approximately in half. Several pilot experiments, conducted to observe the effects of this optimization on the retrieval performance, revealed that the improved selectivity of the search predicates can hardly compensate for the reduced capacity of interior pages.

However, a different heuristic optimization can lead to significant performance improvements. Instead of keeping the information about high endpoints of MBRs in every interior entry, one can assign to each interior page only one entry that would keep the information about the high endpoints of MBRs located in all subtrees of the given page. In other words, for each interior page, the additional entry of the form $\langle l(E'), h(E') \rangle$, called the *H-entry*, would represent the minimum bounding rectangle E' enclosing the high endpoints of object MBRs that appear in any branch spawning from the given interior page.

The resulting structure is called the *scalable QSF-tree*, or just *cQSF-tree*. The addition of the H-entry can decrease the capacity of interior pages by at most one entry. (If the unused space that typically appears in index pages is sufficiently large, the page capacity need not be reduced at all.) On the other hand, since leaf pages do not have H-entries, their capacity remains the same as in the equivalent sQSF-tree.

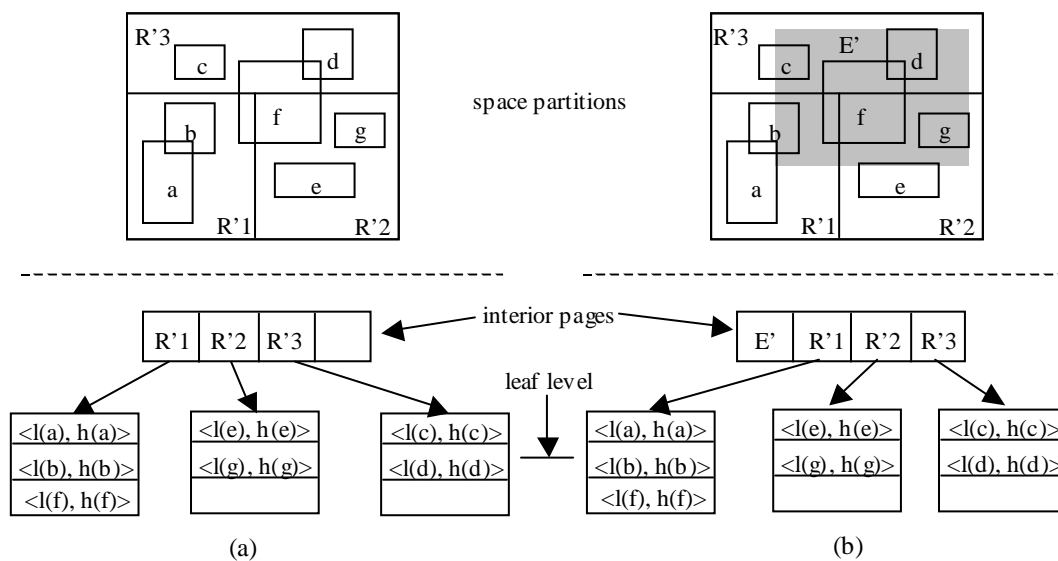


Figure 3: The space partition and structure of (a) sQSF-trees and (b) cQSF-trees.

Assuming that the QSF-tree variants are implemented using KDB-trees, Figure 3 illustrates the structures of an sQSF-tree and a cQSF-tree built on the same set of object MBRs and their corresponding space partitions. Contrasting Figure 3a with Figure 3b, one can see that the only structural difference is the appearance of the H-entries in the interior pages of cQSF-trees (in Figure 3, each structure has only one interior page). The H-entry in an interior page maintains the MBR E' enclosing the high endpoints of all object MBRs stored in the leaf pages of the subtree rooted at the given interior page. The shaded window in Figure 3b represents the region E' associated with the only H-entry in the given cQSF-tree structure.

To develop a cQSF-tree, the maintenance algorithms of sQSF-trees must be modified so that each H-entry is up-to-date. In particular, the *insertion* of an object MBR $r' = \langle l(r'), h(r') \rangle$ proceeds as follows:

- 1) *Search*. Starting from the root page and using only $l(r')$, search the underlying KDB-tree with the point-search algorithm of (Robinson, 1981) in order to locate the leaf page where r' belongs.
- 2) *Insertion*. Insert the object MBR r' into the leaf page. For each dimension i , if necessary, update the minimum m_i and/or maximum M_i extension along the axis i among all object MBRs in the data set.
- 3) *Updating H-entries*. If the new entry enlarges the MBR E' corresponding to the H-entry of the parent page, the H-entry needs to be updated. These updates may propagate upwards, up to the root of the cQSF-tree. (Note that, if node splitting occurs, the updates of H-entries can be propagated upwards along with the splitting of node BRs.)
- 4) *Splitting leaf pages*. If the leaf page overfills, perform the split operation according to the splitting algorithm for leaf pages given in (Robinson, 1981), taking into account only the low endpoints of object MBRs. Split the node BR of the old leaf in its parent page.
- 5) *Splitting interior pages*. If an interior page overfills, perform the split operation according to the rules of first-division splitting (Orlandic and Yu, 2001). Split the node BR of the old interior page in its parent page, if any. The splitting may propagate up to the root, in which case a new root page is created and the number of levels in the index tree is incremented by one.

The approximate information about the high endpoints of object MBRs maintained in the H-entries can be used effectively to improve the performance of search operations. Table 2 shows the search predicates applied to the interior entries of the cQSF-tree. As before, R' denotes a node BR stored in the given interior entry and E' represents the MBR corresponding to the H-entry of the given interior page. In contrast to the search predicates of sQSF-trees (Table 1), the predicates of Table 2 test whether the MBR E' overlaps the given H-region, which can be performed once for each interior page.

Table 2: Search predicates of cQSF-trees.

Query Predicate	Search Predicate
$equal(r', q')$	$covers(R', L_e) \wedge covers(E', H_e)$
$covers(r', q')$	$not_disjoint(R', L_c) \wedge not_disjoint(E', H_c)$
$covered_by(r', q')$	$not_disjoint(R', L_{cb}) \wedge not_disjoint(E', H_{cb})$
$not_disjoint(r', q')$	$not_disjoint(R', L_{nd}) \wedge not_disjoint(E', H_{nd})$

In summary, given a query of the form “find all object MBRs r' that satisfy topological relation t with respect to the given query window q' ,” cQSF-trees perform the *search* operation as follows:

- 1) *Initialization*. For the given query window q' and the relation t of the query predicate, calculate the corresponding L-region L_t and H-region H_t (Yu et al., 1999), as illustrated in Figure 1.
- 2) *Search*. Starting from the root page, perform the breadth-first search of the underlying tree structure by applying the corresponding search predicate of the second column of Table 2 to the index entries and the H-entry of every accessed interior page in the cQSF-tree.
- 3) *Selection*. When a leaf page is accessed, include in the result set each object MBR r' that satisfies the topological relation t with respect to q' .

EXPERIMENTAL RESULTS

As noted in the previous section, the motivation behind cQSF-trees is to reduce the false drops in sQSF-trees. Since the number of false drops tends to grow with dimensionality, the effects of this measure are likely to be more pronounced in high-dimensional spaces. More precisely, as the number of dimensions grows, the size of index entries increases, thus decreasing the effective capacity of the index pages. When the number of objects in the data set is constant and page capacity reduces, the MBRs E' of the H-entries enclose fewer points and they become progressively smaller than the universe. Due to the smaller regions E' , fewer pages must be visited while traversing the tree. With more skewed data distributions, the MBRs E' are likely to be even smaller. As a result, the probability of accessing an index page is likely to be reduced, which would lead to even greater performance improvements over equivalent sQSF-trees.

However, the expected improvements of cQSF-trees over sQSF-trees are by no means guaranteed. They may depend on a number of different factors, including the dimensionality of data as well as their size and distribution in space. To investigate the performance of the two variants of QSF-trees in various scenarios, we conducted several experiments with different test cases. Both versions of QSF-trees were implemented by modifying KDB-trees to accommodate extended entries at the leaf level of the tree and using the first-division splitting of interior pages.

In each experiment, the number of dimensions was varied between 2 and 15. The page size of every structure was 2K bytes. The retrieval performance was measured in terms of the average number of page accesses over 2,000 randomly generated queries (500 for each type of query). Every side of a query window was obtained as a pair of random numbers between 0 and 1. The experiments were performed for both uniform and highly skewed data distributions.

The first set of experiments involved uniformly distributed data. For each d -dimensional space, we constructed three data files with small, large or widely varying objects. Each file contained 65,536 (2^{16}) random d -dimensional rectangles whose lengths along each axis, relative to the side of the universe, were between: 0.1% and 0.5% (small objects), 15% and 30% (large objects), or 0.5% and 30% (widely varying objects). Objects of each file were inserted into an sQSF-tree and the equivalent cQSF-tree.

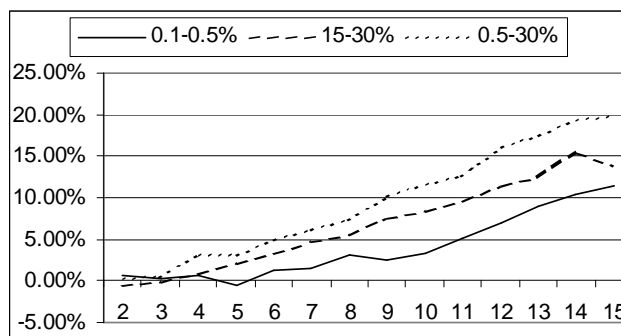


Figure 4: Percentage improvements of cQSF-trees over sQSF-trees for uniform data distribution as dimensionality grows.

Figure 4 shows the percentage improvements of cQSF-trees over sQSF-trees for an average query with varying data dimensionality. For each input file in every d -dimensional space, the improvement was measured using the following formula: $100 \cdot (T_s(d) - T_c(d)) / T_s(d)$, where $T_s(d)$ and $T_c(d)$ are the total

page accesses generated by all queries performed on sQSF-trees and the corresponding cQSF-trees, respectively. As expected, the improvements had a tendency to grow with data dimensionality.

Obviously, the highest improvements were obtained for objects of widely varying size. To see the reason for this, observe first that, for both widely varying and large objects, the volumes of the L-regions L_{nd} , L_c , and L_e are the same (recall Figure 1). However, since the size of the L-regions L_{cb} for *covered_by* queries is greater for widely varying than for large objects, sQSF-trees generate more false drops for widely varying objects. As a result, the more restrictive search predicates of cQSF-trees have greater impact for widely varying than for large objects. The lowest percentage improvements were obtained for small objects. This can be explained by considering the *not_disjoint* queries, which generally dominate the average performance. For small objects, the enlargement of the search regions (the L-regions L_{nd}) is relatively small, and so few false drops are generated by the sQSF-tree. When this is the case, the more restrictive search predicates of cQSF-trees have relatively smaller impact on the retrieval performance.

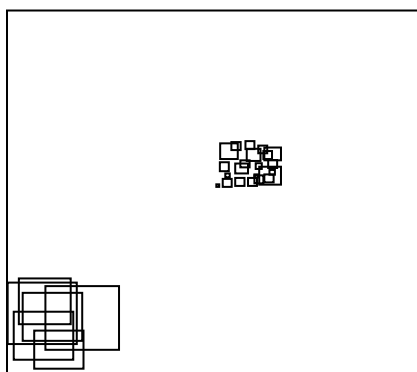


Figure 5: A skewed distribution of data in a 2-dimensional space.

To observe the performance of cQSF-trees and sQSF-trees for non-uniform data distributions, for each d -dimensional space, we constructed an input file with exactly 32,768 (2^{15}) synthesized objects of varying size, which were concentrated in two different clusters. While one of the clusters appeared close to the origin of the space, the other was placed near the center of the universe. Figure 5 illustrates the distribution in the 2-dimensional space. For each d -dimensional space, the objects of the corresponding data file were inserted into both sQSF-trees and cQSF-trees.

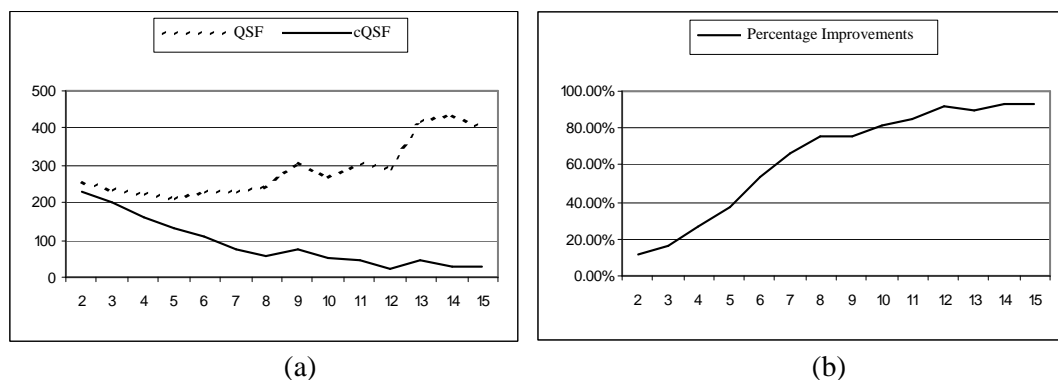


Figure 6: Relative performance of cQSF-trees and sQSF-trees for the skewed distribution of Figure 5 as data dimensionality grows.

Figure 6 shows the difference in the retrieval performance as data dimensionality grows, which is expressed in terms of the average page accesses per query (Figure 6a) and the percentage improvement (Figure 6b). Obviously, for skewed data distribution, the performance improvements of cQSF-trees over the sQSF-trees can be rather remarkable. In the 15-dimensional space, cQSF-trees generated about 13.5 times fewer page accesses than the corresponding sQSF-trees. This also confirms the anticipated impact, stated earlier in this section, of the optimization applied by cQSF-trees.

SUMMARY AND DISCUSSION

Numerous database applications must deal with regional data in high-dimensional spaces. Unfortunately, traditional spatial access methods for regional objects do not scale well to higher dimensionalities. Simple QSF-trees (sQSF-trees) were designed to attack the conceptual problems that traditional spatial access methods experience in spaces with many dimensions. They eliminate certain conceptual problems of region-overlapping schemes, while avoiding the conceptual problems of both object clipping and object transformation. Using an original query transformation that results in two regions in the original space as well as a space-partitioning strategy of a point access method that incurs no region overlap, sQSF-trees adapt more gracefully to the growing dimensionality of data.

In this paper, we have proposed a variant of sQSF-trees, called the cQSF-tree, which reduces the number of false drops into index pages containing no objects that can satisfy the query. These false drops are due to the fact that sQSF-trees index only the low endpoints of object MBRs. By indexing not only low endpoints of the object MBRs but also some approximate information about their high endpoints, cQSF-trees increase the selectivity of search predicates, improving the performance of multi-dimensional selections. The experimental evidence shows that cQSF-trees are more scalable than sQSF-trees with respect to the increasing data dimensionality. For highly skewed data, cQSF-trees reduce the number of page accesses generated by the corresponding sQSF-trees by as much as an order of magnitude.

The proposed indexing technique is an attractive alternative to the existing spatial access methods for low-dimensional data of geographic applications. But more importantly, its ability to scale well to spaces with many dimensions makes it highly appropriate for situations when the aggregation or clustering of high-dimensional data forces the need for efficient handling of not only points but also regional objects. As noted in the introduction, these situations regularly arise in advanced scientific applications, location-based services with moving objects, multimedia systems, and on-line analytical processing.

Other than the higher performance and scalability of multi-dimensional selections, an advantage of cQSF-trees over the most popular spatial access methods is the lower cost of dynamic updates. For example, since page splitting in R^* -trees employs several complex optimizations (Beckmann et al., 1990), the construction of R^* -trees tends to be extremely slow (Gaede and Gunther, 1998). Even with the upward propagation of H-entry updates, the dynamic updates in cQSF-trees are much faster. As a result, cQSF-trees are more appropriate for environments where the cost of updates is an important factor.

In this paper, we have presented the design of cQSF-trees assuming a variant of KDB-trees as its underlying point access method. However, with few simple modifications, any PAM structure can be used to implement cQSF-trees. The only required modifications of the given PAM include a simple change in the structure of the leaf entries to differentiate between the low and high endpoints of object MBRs, the maintenance of H-entries in the interior nodes, and the application of new search predicates in the selection process. Other than that, the cQSF-tree is just a simple layer of software implemented on top of any existing PAM structure that supports a diverse set of search operations over both points and regional objects.

This flexibility and simplicity of cQSF-tree implementation is desirable in many practical environments. In particular, the provision for reuse of indexing techniques already deployed in typical database management systems can enable rapid integration of advanced multi-dimensional capabilities into the existing database management systems. With the query transformation of sQSF- and cQSF-trees, these systems could support highly dimensional regional data they have not been able to support so far.

REFERENCES

- Beckmann, N., Kriegel, H., Schneider, R., & Seeger, B. (1990). The R^{*}-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 322-331.
- Berchtold, S., Bohm, C., & Kriegel, H.P. (1998). The Pyramid-Technique: Towards breaking the curse of dimensionality. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 142-153.
- Berchtold, S., Keim, D.A., & Kriegel, H. (1996) The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB)*, 28-39.
- Freeston, M. (1995). A general solution of the N-dimensional B-tree problem. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 80-91.
- Gaede, V. & Gunther, O. (1998). Multidimensional access methods. *ACM Computing Surveys*, 30(2), 170-23.
- Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 47-54.
- Lin, K., Jagadish, H., & Faloutsos, C. (1995). The TV-tree: An index structure for high-dimensional data. *VLDB Journal*, 3, 517-542.
- Orlandic, R. (2003). Effective management of hierarchical storage using two levels of data clustering. In *Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies*. (in press)
- Orlandic, R. & Yu, B. (2000). A study of MBR-based spatial access methods: How well they perform in high-dimensional spaces. In *Proceedings of the International Database Engineering and Applications Symposium (IDEAS)*, 306-315.
- Orlandic, R. & Yu, B. (2001). Implementing KDB-trees to support high-dimensional data. In *Proceedings of the International Database Engineering and Applications Symposium (IDEAS)*, 58-67.
- Orlandic, R. & Yu, B. (2002). A retrieval technique for high-dimensional data and partially specified queries. *Data and Knowledge Engineering*, 42(1), 1-21.
- Pagel, B.-U., Six, H.-W., & Toben, H. (1993). The transformation technique for spatial objects revisited. In: D. Abel & B.C. Ooi (Eds.), *Advances in Spatial Databases, Lecture Notes in Computer Science 692* (pp. 73-88). Berlin: Springer-Verlag.

Papadias, D., Theodoridis, Y., Sellis, T., & Egenhofer, M.J. (1995). Topological relations in the world of minimum bounding rectangles: A study with R-trees. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 92-103.

Robinson, J.T. (1981). The K-D-B tree: A search structure for large multidimensional dynamic indexes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 10-18.

Sakurai, Y., Yoshikawa, M., Uemura, S., & Kojima, H. (2000). The A-tree: An index structure for high-dimensional spaces using relative approximation. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB)*, 516-526.

Seeger, B. & Kriegel, H.P. (1988). Techniques for design and implementation of efficient spatial access methods. In *Proceedings of the 14th International Conference on Very Large Data Bases (VLDB)*, 360-371.

Sellis, T., Roussopoulos, N., & Faloutsos, C. (1987). The R+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB)*, 507-518.

Swets, D.L. & Weng, J. (1996). Using discriminant eigenfeatures for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8), 831-836.

Weber, R., Schek, H.-J., & Blott, S. A (1998). A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB)*, 194-205.

White, D.A. & Jain, R. (1996). Similarity indexing with the SS-tree. In *Proceedings of the 12th IEEE International Conference on Data Engineering (ICDE)*, 516-523.

Yu, B., Orlandic, R., & Evens, M. (1999). Simple QSF-trees: An efficient and scalable spatial access method. In *Proceedings of the 8th International Conference on Information and Knowledge Management (CIKM)*, 5-14.

Bio Sketch of Ratko Orlandic:

Ratko Orlandic received his Ph.D. degree in Computer Science at the University of Virginia in 1989. He is currently an Assistant Professor of Computer Science at the Illinois Institute of Technology. He has published over 30 refereed journal and conference papers and served as a reviewer for a number of journals and international conferences. His major research interests include access methods, content-based retrieval, transaction management and software architecture.

Bio Sketch of Byunggu Yu:

Byunggu Yu received his Ph.D. in Computer Science from the Illinois Institute of Technology in 2000. As a student, he won the best student paper award of the Computer Science Department at the Illinois Institute of Technology and the Gold-Prize of the First Human-Tech Research Paper Competition of Samsung Electronics. He has published over 20 refereed journal and conference papers. He is currently an Assistant Professor of Computer Science at the University of Wyoming. His major research interests include access methods, content-based retrieval, spatial query processing, and spatial databases.