Efficient Dynamic Range Searching using Data Replication *

K. V. Ravi Kanth and Ambuj K. Singh
Computer Science Department
University of California at Santa Barbara
Santa Barbara, CA 93106

Abstract

Given the lower bound of $\Omega(n^{(d-1)/d})$ for range query time complexity on n d-dimensional point data, we investigate whether little replication can improve the query and update times significantly. We propose linear-space index structures that minimize the query and update times; the query time we achieve is $O(n^\epsilon)$ for any $\epsilon > 0$, and, the update time is $O(\log n)$.

1 Introduction

Given a set S of d-dimensional points, a range query q is specified by d 1-dimensional intervals $[a_i, b_i]$ one for each dimension i, and retrieves all points $p = (p_1, p_2, \ldots p_d)$ in S such that $\langle \forall i \in \{1, \ldots, d\} : a_i \leq p_i \leq b_i \rangle$. This type of searching in multiple dimensions is fundamental to geographic information systems, image databases and computer graphics. Several efficient data structures using linear and nonlinear space (measured in terms of the number of data points) have been proposed in the literature [Bentley 1977, Bentley 1980, Guttman 1984]. For large databases, structures that have nonlinear storage requirements may not acceptable. Consequently, several linear-space structures such as the R-trees [Beckman et al. 1990, Guttman 1984], Bang files [Freeston 1987] have been proposed in the database literature. These structures achieve low storage costs by storing each data point exactly once. However, such simple structures have high query time complexity, which is bounded below by $\Omega(n^{(d-1)/d})$ for n d-dimensional data [Ravi and Ambuj 1997]. This complexity can, however, be improved by allowing for replication of data. Using

* Work supported in part by a research grant from NSF/ARPA/NASA IRI–9411330 and NSF instrumentation grant CDA-9421978.

such replication (but still using linear storage), Bentley's nonoverlapping k-level d-range structure [Bentley 1980] achieves a query time complexity of $O(n^{(d-1)/k} \log n)$ [Bentley 1980]. By choosing k as a function of d and ϵ, we can achieve a query time of $O(n^{\epsilon})$ for any $\epsilon > 0$. Note that the storage requirements increase by a constant factor only. However, this structure needs preprocessing of data before being constructed. Consequently, on-line insertion/deletion of data may require the reconstruction of the entire structure, which takes $O(n \log n)$ time. This insertion/deletion time complexity can be reduced to $O(\log^2 n)$ using the static-to-dynamic transformations of Bentley and Saxe [Bentley 1980], or Willard [Willard 1986]. In this paper, we show that much lower insertion/deletion times can be achieved. We first show that insertions and deletions can be accomplished in $O(\log n)$ amortized time. We then transform these amortized times to worst-case times using the *global rebuilding* technique of Overmars and van Leeuwen [Overmars and Leeuwen 1981]. These transformations do not alter the query time complexity and only increase the storage requirements by a constant factor. Consequently, our linear-space structure has worst-case times of $O(n^{\epsilon})$ for queries (for any $\epsilon > 0$) and $O(\log n)$ for insertions and deletions.

In the next section, we define some preliminary concepts. The ideas developed in that section avoid the preprocessing stages of Bentley's static structure. Using these ideas, we introduce a variant of Bentley's k-level d-range structure in Section 3 and analyze its range query time complexity. In Section 4, we describe insertions and deletions in this structure. We summarize these results in the concluding section.

2 Preliminaries

Consider a set S of 2-dimensional points. These points can be ordered using their coordinates in the x-dimension. Points with same x-coordinates can be ordered using their y-coordinates. Alternately, the points can be ordered using their y-coordinates first, and then using their x-coordinates. The former order is referred to as the $\langle x, y \rangle$-order to denote the priority of x-coordinates over the y-coordinates. The latter order is referred to as the $\langle y, x \rangle$-order for similar reasons. Let σ denote any such order specifying a permutation of the dimensions. We denote that point p_1 precedes p_2 in σ-order by $p_1 \prec_{\sigma} p_2$. Let S_{σ} denote the sequence of points in S sorted using σ-order. The extent of the set S in σ-order can be represented by the first and the last points in the sequence S^{σ}. If these points are p_1 and p_2, we denote the extent as $\langle p_1, p_2 \rangle$. Let us illustrate these concepts with an example. Let $S' = \{(1,1), (2,6), (2,4), (3,5), (5,1), (6,6), (6,4), (7,5)\}$. The $\langle x, y \rangle$-order orders the points in S' using the x-coordinates. Points that have same x-coordinates, for instance $(2,4)$ and $(2,6)$ (or $(6,4)$ and $(6,6)$), are ordered using the y-coordinates. The sorted sequence S' is then given by $S'_{\langle x,y \rangle} = \langle (1,1), (2,4), (2,6), (3,5), (5,1), (6,4), (6,6), (7,5) \rangle$. The extent of the set S' is the first and the last points in this sequence, which are $(1,1)$ and $(7,5)$ respectively.

Total orders, such as the ones above, can be used to partition a set S into smaller subsets, which are disjoint and are of approximately same size. We refer to such subsets as *partitions* of S in the specific order. For example, the set S'

can be divided into 4 partitions, $P_1, \ldots, P_4$ using x-order as:

$$P_1 = \{(1,1),(2,4)\}, P_2 = \{(2,6),(3,5)\};$$

$$P_3 = \{(5,1),(6,4)\}, P_4 = \{(6,6),(7,5)\}.$$

Next, we extend the total order on points to partitions, by defining an order on their extents. Partition extent $\langle a, b\rangle$ precedes another partition extent $\langle c, d\rangle$ in σ-order if b precedes c in σ-order. Since partition extents can be totally ordered, they can be used to construct a B$^+$-tree over a set of partitions P. We refer to this tree as a B-tree for P using σ-order, and denote it by $B(|P|, \sigma)$. Figure 1 shows the B-tree for the partitions of S'. Such a tree facilitates efficient searching in the partitions. For instance, identifying a partition in which a new point s is to be stored can be accomplished as follows. We start at the root of the tree and traverse down the tree by choosing the i^{th} entry (corresponding subtree) such that $p_1^i \prec_\sigma s \prec_\sigma p_1^{i+1}$, where $\langle p_1^i, p_2^i\rangle$ and $\langle p_1^{i+1}, p_2^{i+1}\rangle$ denote the extents for the i^{th} and $(i+1)^{st}$ entries respectively. If no such entry i can be found, we choose the first entry. Consider the example tree of Figure 1. To insert a new point $(4,1)$ in this tree, we choose the right subtree at the root and then the partition P_2 at vertex v. In general, this procedure takes $\log k$ time if the tree has k partitions. Similarly a range query q can be processed by traversing down all subtrees, whose extents *intersect* the query. Next we define this intersection.

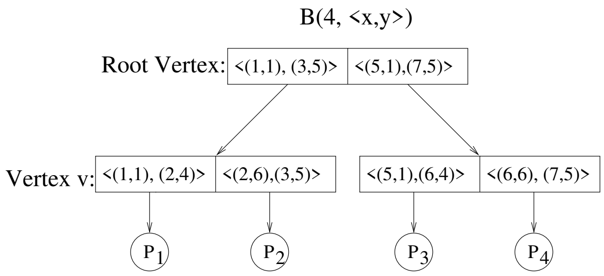

Figure 1. A B-tree for 4 partitions of S' using $\langle x, y\rangle$-order: Facilitates efficient search of the partitions.

Let point $a \preceq_\sigma$ point b if a and b are identical, or, $a \prec_\sigma b$. Consider a query $q = \langle [a,b],[c,d]\rangle$. Then, extent $e_\sigma = \langle p_1, p_2\rangle$ *intersects* the query q if there is a point $p = (s,t) : p_1 \preceq_\sigma p \preceq_\sigma p_2$ such that p is in the query range of q (i.e., $(a \leq s \leq b) \wedge (c \leq t \leq d)$ holds). This intersection is *partial* if either $p_1 \prec_\sigma (a,c)$ or $(b,d) \prec_\sigma p_2$. Otherwise, the extent is *contained* in the query. It is easy to show that in any B-tree for a set of partitions using an arbitrary order $\sigma = \langle \sigma_1, \sigma_2\rangle$, at most two partitions partially intersect an arbitrary query. All other partitions are contained in the query. We can easily show that the σ-coordinates of the points in these partitions are contained in the interval of the query in dimension σ_1 (see Appendix for more details). Using these observations, we propose an efficient structure for answering range queries.

3 An efficient structure for range queries

We now investigate how best to organize a set of multi-dimensional points to answer range queries efficiently. First, we describe an efficient structure for 2-dimensions, and then, generalize it to an arbitrary number of dimensions. Insertions and deletions in this structure are described in the next section.

3.1 2-dimensions

Consider a set S of 2-dimensional data. Let S be partitioned into k subsets using the order $\sigma = \langle x, y \rangle$ as shown in Figure 2. Consider a query q specified by the interval $[a, b]$ in the x-dimension and $[c, d]$ in the y-dimension. This query is shown as a dashed box in Figure 2 and intersects partitions $S_i, \ldots, S_j$. Since these partitions are in $\langle x, y \rangle$-order, they can be categorized into the following two groups based on their intersection with the query – those that partially intersect the query and those that are contained in the query. The *extreme* two partitions, S_i and S_j in Figure 2, are of the former category, whereas all the *interior* partitions, $S_{i+1}, \ldots, S_{j-1}$, fall into the latter category. In the interior partitions, the data points satisfy the query interval $[a, b]$ in the x-dimension. Therefore, the query has to retrieve all points whose y-coordinates are in the interval $[c, d]$. This can be accomplished by organizing each interior partition as a B-tree along the y-dimension. For the extreme partitions, no such organization is helpful. The procedure for partitioning S is applied to these partitions, yielding a recursive structure. Hence, each partition P of S is organized as two structures – a B-tree along the y-dimension and a recursive structure. If the depth of this recursion is l, then we refer to the structure as an l-level structure. Since the structure is designed to answer range queries efficiently, we refer to it as an l-level range structure.

Let $T^k(n, l, \langle x, y \rangle)$ denote an l-level range structure for n 2-dimensional points. The first parameter specifies the number of points, the second the depth of the recursion in the structure, and the last the ordering. The superscript k indicates the number of partitions in the B-tree at each level of this structure. The optimal value for k (as a function of the parameters of the structure) will be determined shortly. Based on the above discussion, we have the following organization for an l-level range structure $T^k(n, l, \langle x, y \rangle)$.

1. A B-tree of k partitions for the set of n points using $\langle x, y \rangle$-order (i.e., $B(k, \langle x, y \rangle)$).
2. Each partition has $O(n/k)$ points. If this is not the last level in the recursion (i.e., if $l > 1$), then it is further organized as:
   (a) A B-tree of $O(n/k)$ partitions using the $\langle y \rangle$-order (i.e., a 1-dimensional B-tree using just the y-coordinates of the points)
   (b) An $(l - 1)$-level range structure using $\langle x, y \rangle$-order (i.e., as $T^k(n/k, l - 1, \langle x, y \rangle)$)

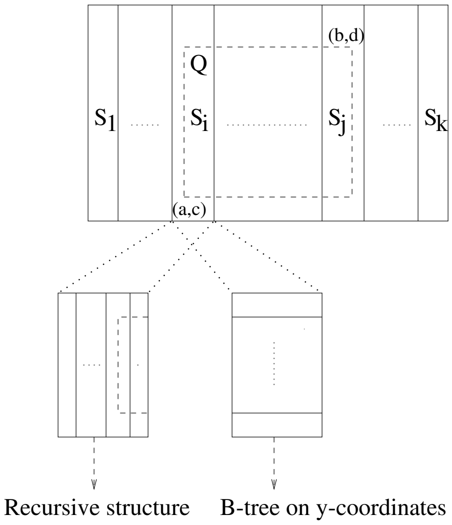

Figure 2. Organizing k partitions along x-dimension: Each partition is organized both as a recursive structure and a B-tree on y-coordinates.

Figure 3 describes this organization. The topmost B-tree in the l-level range structure (i.e., the B-tree in Step 1 of the preceding specification) is referred to as the level-l B-tree. Each partition ($S_1, \ldots, S_k$ in the figure) of this B-tree is organized as an $(l-1)$-level range structure (in addition to a B-tree on y-coordinates); such a partition is referred to as a level-$(l-1)$ partition. Note that a level-0 partition stores data points and is not further organized.

A range query $q = \langle q_x, q_y \rangle$ on the l-level structure is processed as follows.

1. Locate all partitions intersecting the query in the level-l B-tree. These partitions can be categorized into interior partitions and extreme partitions based on their intersection with the query.

2. For each interior partition P, search the B-tree that uses the $\langle y \rangle$-order. Only the query interval for the y-dimension is used in this search.

3. For each extreme partition, search in the corresponding recursive structure $T(|P|, l-1, \langle x, y \rangle)$.

Next, let us analyze the worst-case query time complexity in the above l-level range structure for n points. We can then determine the value of k so that the query time complexity is minimized. Let $QT^k(n, l, |\sigma|)$ denote the query time complexity in a $T^k(n, l, \sigma)$ structure. Since we are dealing with 2-dimensional data, $|\sigma| = 2$. For convenience, we consider only the query overheads and ignore the time for retrieving the data points that satisfy the query. The query overheads are composed of:

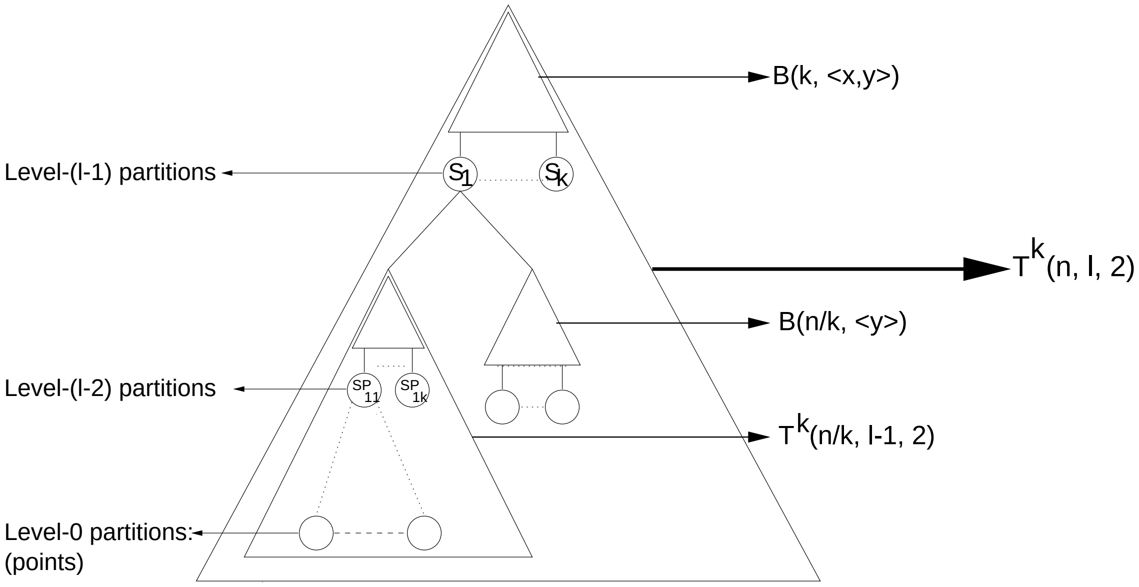

Figure 3. An l-level range structure for n 2-dimensional data: Organized as a B-tree of k partitions. Each partition of the B-tree is organized both as an $(l-1)$-level range structure and as a B-tree.

1. Overheads of locating all partitions that intersect a query in the level-l B-tree

2. Overheads of searching in the interior partitions

3. Overheads of searching in the extreme partitions

The overhead of searching in the level-l B-tree of k partitions is $O(k)$. Next, let us examine the overheads in the interior and extreme partitions. Each interior partition has $O(n/k)$ points and is searched using the B-tree on y-coordinates. Consequently, the overheads are $O(\log(n/k)) = O(\log n)$. The overheads in the extreme partitions, which are searched using the recursive structure, are expressed recursively. Noting that there are at most 2 extreme partitions for any query, the worst-case query overheads are given by:

$$QT^k(n,l,2) = k + (k-2)\log n + 2QT^k(n/k, l-1, 2).$$

$$QT^k(n,1,2) = n.$$

This equation for $QT^k(n,l,2)$ has minimum value of $O(n^{1/l}\log^{(l-1)/l} n)$ when $k = O((n/\log n)^{1/l})$. For simplicity, Bentley chooses $k = n^{1/l}$ and obtains a complexity of $O(n^{1/l}\log n)$. We also choose $k = O(n^{1/l})$ in this paper, the exact value being specified when we generalize this structure to arbitrary dimensions. One may, however, note that the insertion/deletion techniques proposed in this paper also extend to the optimal value of $k = O((n/\log n)^{1/l})$. Since k is no longer an independent parameter, we drop the superscript k in $T^k(n, l, \sigma)$ and in $QT^k(n, l, 2)$.

3.2 Range structure for arbitrary number of dimensions

Next, we extend the 2-dimensional l-level range structure to higher dimensions in a straightforward manner. In this description, we omit the details of the extension of σ-order to higher dimensions. The d-dimensional structure differs from the 2-dimensional one in the following manner.

1. The level-l B-tree is formed using σ-order instead of the $\langle x, y\rangle$-order for 2-dimensions.
2. Each partition of this level-l B-tree is organized as:
   (a) An $(l-1)$-level range structure using all but the first dimension of σ (this replaces the B-tree on $\langle y\rangle$-order of the 2-dimensional structure)
   (b) An $(l-1)$-level range structure using σ-order instead of the $\langle x, y\rangle$-order

The complete specification of an l-level range structure for n d-dimensional points, which is denoted as $T(n, l, \sigma)$, is as follows.

1. It has a B-tree of at most $4n^{1/l}$ partitions in σ-order.
2. If $l = 1$, then the partitions correspond to points. Otherwise, each partition P is organized as:
   (a) a $T(|P|, l-1, \langle\sigma_2, \sigma_3, \ldots, \sigma_d\rangle)$, referred to as the *interior structure* for partition P (If $|\sigma| = 2$, then this structure is a B-tree of at most $4n^{(l-1)/l}$ points)
   (b) a $T(|P|, l-1, \sigma)$, referred to as the *extreme structure* for partition P

Note that the above specification gives a bound on the number of partitions (or points) in each B-tree. The factor of 4 in the number of partitions allows us to support insertions and deletions efficiently as explained in the next section. In addition, every level-$(l-1)$ partition P is organized as two $(l-1)$-level range structures. This is in contrast to Bentley's original proposal which has one l-level range structure and one $(l-1)$-level range structure. This modification simplifies insertions and deletions. Of the two structures for a partition, the interior structure (a B-tree on y-coordinates in the 2-dimensional case) is searched for an interior partition of a query, and, the extreme structure (the recursive structure in the 2-dimensional case) for an extreme partition of a query. It is important to note that both the interior and extreme structures of a partition are never searched in the same query. The query algorithm, which is also a direct generalization from the 2-dimensional case, is summarized as follows.

1. Locate all partitions whose extents intersect the query using the level-l B-tree.
2. For each interior partition P in PS_i, search in the corresponding interior structure $T(|P|, l-1, \langle\sigma_2, \ldots, \sigma_d\rangle)$ using the query intervals in dimensions $\sigma_2, \ldots, \sigma_d$.
3. For each extreme partition, search in the corresponding extreme structure $T(|P|, l-1, \sigma)$.

Let us estimate the query overheads $QT(n, l, |\sigma|)$ for the above algorithm. As in the 2-dimensional case, the overheads of searching in a B-tree of k partitions is $O(k)$. There are at most 2 extreme partitions, the rest being interior partitions. If the interior partition is a B-tree on one of the dimensions, then the overheads are logarithmic in the number of points (i.e., $QT(n, l, 1) = \log n$). Otherwise, the query overhead $QT(n, l, d)$, $d = |\sigma|$, is obtained using the following recurrence relation.

$$QT(n, l, d) = O(k) + O(k)QT(n/k, l-1, d-1) + 2QT(n/k, l-1, d).$$

Since all points are searched in a 1-level range structure, the complexity $QT(n, 1, d) = n$. Solving the above with this information, we obtain $QT(n, l, d) = O(n^{(d-1)/l} \log n)$. This results in the following lemma.

Lemma 1 *The query time complexity of an l-level range structure for n d-dimensional points is $O(n^{(d-1)/l} \log n)$.*

3.3 Space Complexity

Before we describe the insertion/deletion process in an l-level range structure, we analyze its space complexity and prove that it is linear in the number of data points. Let $T(n, l, |\sigma|)$ denote the space complexity of a $T(n, l, \sigma)$ structure. When $|\sigma| = 1$, the structure is a 1-dimensional B-tree. The space requirements of a B-tree are linear in the number of its leaves. This yields $ST(n, l, 1) = O(n)$. When $|\sigma| > 1$, the structure has a level-l B-tree of $4n^{1/l}$ partitions. This B-tree takes $O(n^{1/l})$ space. Each partition of this B-tree is organized as two $(l-1)$-level range structures. Combining these, the storage costs for d-dimensional data (i.e., $\sigma| = d$) are expressed recursively as:

$$ST(n, l, d) = O(n^{1/l}) + O(n^{1/l})[ST(n^{(l-1)/l}, l-1, d-1) + ST(n^{(l-1)/l}, l-1, d)].$$

$$ST(n, l, 1) = O(n).$$

Solving these two equations, we obtain $ST(n, l, d) = O(l^{d-1}n)$. This corresponds to a replication of $O(l^{d-1})$ for each data point. For large databases, almost all index structures including R*-trees [Beckman et al. 1990, Guttman 1984] and Bang files [Freeston 1987] are maintained in secondary memory even when there is no such replication. Since the worst-case query time in such structures is at least $\Omega(n^{(d-1)/d})$, the l-level range structure provides a feasible alternative for improving query performance with a constant increase in storage space.

4 Insertions and deletions

In this section, we describe our strategy to support insertions and deletions in an l-level range structure for data of arbitrary dimensionality. First, we describe how such a structure can be constructed for a given set of points, and then we describe on-line insertions and deletions.

4.1 Construction of an l-level range structure for n d-dimensional points

Construction of an l-level range structure for a set S of n d-dimensional points is accomplished by partitioning the data into $n^{1/l}$ subsets using σ-order, where σ denotes an arbitrary permutation of the d-dimensions. For each subset, two $(l-1)$-level range structures are constructed – one using the σ-order and other using $\langle \sigma_2, \sigma_3, \ldots, \sigma_d \rangle$-order. This algorithm is specified as follows.

Procedure *ConstructRangeStructure(n, l, σ, d)*
begin

1. If $|\sigma| = 1$, organize it as a B-tree of n points using the dimension σ_1.
2. Otherwise,
   (a) Sort S in σ-order and divide the resulting sequence into $n^{1/l}$ partitions.
   (b) Construct a B-tree for these partitions.
   (c) If $l > 1$, construct the following two structures for each partition P (of $n^{(l-1)/l}$ points).
      - An $(l-1)$ range structure using the sequence $\langle \sigma_2, \ldots, \sigma_d \rangle$
      - An $(l-1)$ range structure using σ

end

Let $CT(n, l, |\sigma|)$ denote the time complexity of constructing an l-level range structure for n points. The first step in the preceding algorithm takes $O(n \log n)$ time. The sorting of points (Step 2(a)) and the construction of a B-tree (Step 2(c)) can also be accomplished in $O(n \log n)$ time. The time for recursive construction of the two alternate structures for each partition in Step 2(c) can be estimated accordingly. Thus, the time complexity for d-dimensional data (i.e., when $\sigma| = d$) is recursively expressed as:

$$CT(n,l,d) = O(n \log n) + O(n^{1/l})[CT(n^{(l-1)/l}, l-1, d-1) + CT(n^{(l-1)/l}, l-1, d)].$$

Since $CT(n,l,1) = O(n \log n)$ and $CT(n,1,d) = O(n \log n)$, the above equation yields $CT(n,l,d) = O(n \log n)$. This property of an l-level range structure is summarized in the following lemma.

Lemma 2 *An l-level range structure for n d-dimensional points can be constructed in $O(n \log n)$ time.*

4.2 Insertions and deletions

Our strategy for supporting insertions and deletions in the l-level range structure is the following:

- Start with a newly constructed structure of n_0 points.
- Reconstruct it after $\alpha_l n_0$ insertions or deletions.

From Lemma 2, the cost of reconstruction for $(1+\alpha_l)n_0$ points, which is the maximum number after $\alpha_l n_0$ insertions or deletions, is $O(n_0 \log n_0)$. This reconstruction cost is amortized among the $\alpha_l n_0$ insertions/deletions that occurred. Assuming logarithmic overheads for insertions/deletions when no reconstructions occur, the total amortized cost for these operations is $O(\log n)$. This technique is applied to each k-level range structure in the l-level range structure, for all $k \leq l$. Specifically, let $s_k(n_0)$ denote the number of points in a k-level range structure. From the construction algorithm, this number is $s_k(n_0) = n_0^{k/l}$. This structure of $s_k(n_0)$ points is reconstructed after $\alpha_k s_k(n_0)$ insertions or deletions occur. Combining these costs, the amortized cost for insertion/deletion in a k-level range structure is $O(\log s_k(n_0)) = O(\log n)$. Since an insertion/deletion can affect a constant number, l, of these structures, the total amortized time is $O(\log n)$. This technique for supporting insertions and deletions is called *partial rebuilding* [Overmars 1983] and is used in several dynamic structures including the divided k-d tree [van Kreveld and Overmars 1991], and the pseudo quad and k-d tree[Overmars and Leeuwen 1982]. n what follows, we describe this insertion/deletion strategy for the l-level range structure in more detail.

Insertion or deletion of a point p is accomplished in an l-level range structure $T(n,l,\sigma)$ as follows. If $\alpha_l n_0$ insertions/deletions have already occurred in this structure, then reconstruct the entire structure taking into account the current insertion/deletion of p. Otherwise insert/delete p in the structure $T(n,l,\sigma)$ as follows.

Procedure *Insert-Delete($T(n,l,\sigma), p$)*
begin

1. If $|\sigma| = 1$, then insert/delete the point p as in a 1-dimensional B-tree.

2. Otherwise,

   (a) Identify partition P of the level-l B-tree of T in which p is to be inserted or deleted.

   (b) If $\alpha_{l-1}s_{l-1}(n_0)$ insertions or deletions have occurred in P (i.e., in the interior and extreme structures for P), then reorganize it as follows:

      i. Read all points in the partition P.

      ii. Partition them into two equal subsets P_1 and P_2 using σ-order.

      iii. If $l > 1$, then for each partition P_1 and P_2 construct two $(l-1)$-level structures – one as an interior structure and the other as an extreme structure.

      iv. Propagate the split of partition P into P_1 and P_2 in the level-l B-tree.

   (c) Otherwise,

      i. if $l = 1$, then update the B-tree to include/exclude p for insertion/deletion.

      ii. Otherwise, insert p in the two $(l-1)$-level structures for partition P as follows.

         - *Insert-Delete*($T(|P|, l-1, \sigma), p$)
         - *Insert-Delete*($T(|P|, l-1, \langle\sigma_2, \ldots, \sigma_d\rangle), p$)

end

Before analyzing the complexity of the preceding insertion/deletion algorithm, we first need to show that after each insertion (or deletion) into an l-level range structure of n points, the resulting structure is an l-level range structure of $n+1$ points (or $n-1$ points for deletion). We ensure that this happens by choosing the fractions $\alpha_1, \ldots, \alpha_l$ so that they satisfy the following inequalities.

$$\alpha_l < 1 - \frac{1}{\sqrt{2}}.$$

$$0 < \alpha_l < \alpha_{l-1} < \ldots < \alpha_1 < 1.$$

Note that, unlike the divided k-d tree [Overmars 1983] which has specific values for these fractions, we obtain a range of values for each of them. It can be shown that these values can also be applied to the divided k-d tree. Next, we prove that these fractions ensure that the specifications of an l-level range structure are never violated due to insertions and deletions.

Lemma 3 *In an l-level range structure for n points, every $T(x, k, \sigma)$ structure*

- *is a B-tree of at most $4n^{k/l}$ points if $|\sigma| = 1$, or*

- *has a B-tree of at most* $4n^{1/l}$ *partitions*

Proof: Assume the whole structure was last constructed for n_0 points. Since the number of insertions/deletions in such a structure can be at most $\alpha_l n_0$, the current size (in terms of the number of points) $n \geq n_0/(1-\alpha_l)$ (i.e., $n_0 \leq n/(1-\alpha_l)$). Using this observation, we prove the lemma.

Consider the structure $T(x, k, 1)$. This is a B-tree using the coordinates in one of the dimensions. The leaves correspond to data points. The number of such points just after construction of the l-level range structure is $n_0^{k/l}$. Since it corresponds to a level-k partition, it is reorganized after $\alpha_k s_k(n_0) = \alpha_k n_0^{k/l}$ insertions or deletions occur in it. Consequently, the total number of points before such a reorganization is:

$$n_0^{k/l}(1+\alpha_k) < n_0^{k/l} * 2 < 2 * n^{k/l}/(1-\alpha_l)^{k/l}.$$

Since $\alpha_l < 1 - 1/\sqrt{2}$, we have $1/(1-\alpha_l) < \sqrt{2}$. Consequently, the number of points is at most

$$2 * n^{k/l}/(1-\alpha_l)^{k/l} < 2 * 2^{k/2l} n^{k/l} < 4n^{k/l}.$$

Next, we prove the lemma when $|\sigma| \neq 1$. The proof is using induction on k, (i.e., by proving the lemma for each k-level range structure of the l-level range structure).

Basis: $k = 1$. The proof is similar to the one for $T(x, k, 1)$. This structure corresponds to a level-1 B-tree with data points as its partitions. The number of such points just after construction of the entire l-level range structure is $n_0^{1/l}$. This structure is reorganized after every $\alpha_1 n_0^{1/l}$ new insertions/deletions occur in it. Therefore, the maximum number of points (partitions) is at most $(1+\alpha_1)n_0^{1/l}$. This number, following the above reasoning for $|\sigma| = 1$, is at most $4n^{1/l}$.

Hypothesis: Assume it is true for all k-level range structures, $k \leq r$. Consider the structure $T(x, r+1, \sigma)$. It consists of a level-$(r+1)$ B-tree on a set of partitions and two r-level range structures for each partition of the B-tree. By induction hypothesis, each r-level structure satisfies the lemma. The $(r+1)$-level structure also satisfies the lemma provided the level-$(r+1)$ B-tree has at most $4n^{1/l}$ partitions. Let us estimate the maximum number of partitions in this B-tree. The number of insertions/deletions that can occur in $T(x, r+1, \sigma)$ before it is reconstructed is $\alpha_{r+1} n_0^{(r+1)/l}$. Each partition (the corresponding structures) of the B-tree is reconstructed after $\alpha_r n_0^{r/l}$ new insertions or deletions. Each such reconstruction creates a new partition in the B-tree. Consequently, the total number of partitions in the

B-tree of $T(x, r+1, \sigma)$ due to $\alpha_{r+1} n_0^{(r+1)/k}$ insertions/deletions is:

$$n_0^{1/l} + \frac{\alpha_{r+1} n_0^{(r+1)/l}}{\alpha_r n_0^{r/l}} = n_0^{1/l}(1 + \frac{\alpha_{r+1}}{\alpha_r}) < 2 * n_0^{1/l} < 4 * n^{1/l}.$$

This completes the lemma. □

Now, we analyze the time complexity of the insertion/deletion algorithm for an l-level range structure of n points.

Lemma 4 *Insertions and deletions in an l-level range structure of n points take $O(\log n)$ amortized time.*

Proof: Consider the case when no reorganizations occur. Let $ID(n, l, |\sigma|)$ denote the time complexity of inserting/deleting a point in an l-level range structure $T(n, l, \sigma)$ of n points. When $|\sigma| = 1$, the structure is a B-tree on one of the dimensions, which supports insertions/deletions in $O(\log n)$ time. Therefore, $ID(n, l, 1) = \log n$. When $|\sigma| > 1$, the structure has a B-tree on a set of partitions using a specific order. Since this B-tree has at most $k = 4n^{1/l}$ partitions, identification of a partition in which the point p is to be inserted/deleted can be accomplished in $O(\log k) = O(\log n)$ time (as described in the discussion for a B-tree). The point is then recursively inserted in the two structures corresponding to the partition. Note that the recursion bottoms out when $l = 1$. The partitions in such a 1-level structure correspond to points. The complexity of insertion/deletion in such a structure is just that of searching in the associated B-tree and updating it which takes $O(\log n)$ time. Therefore, the insertion/deletion complexity $QT(n, l, d), |\sigma| = d$, when no reorganizations occur is given by:

$$ID(n, l, d) = \log n + ID(n^{(l-1)/l}, l-1, d) + ID(n^{(l-1)/l}, l-1, d-1).$$

$$ID(n, l, 1) = \log n.$$

$$ID(n, 1, d) = \log n.$$

Solving these equations, we obtain $ID(n, l, d) = \log n$.

Next, consider the case when a level-k partition P of m points, $k \leq l$, is to be reconstructed. First, note that the number of points in this partition, m, is at most $(1 + \alpha_k)s_k(n_0)$. Therefore, splitting the data in partition P into two subsets takes $O(m \log m) = O(s_k(n_0) \log(s_k(n_0)))$ time. From Lemma 2, the corresponding structures for the two subsets can be constructed in $O(s_k(n_0) \log(s_k(n_0)))$ time. Consequently, the total reorganization cost is $O(s_k(n_0) \log(s_k(n_0)))$. This cost is charged to the $\alpha_k s_k(n_0)$ insertions/deletions that triggered it. Consequently, each insertion/deletion accrues a cost of $O(\log s_k(n_0))$ due to reorganization of a level-k partition. Since there are at most l such partitions, one at each level, that may add to the cost of an insertion/deletion, the total amortized cost due to reconstructions is:

$$\sum_{1\le k\le l} \log(s_k(n_0)) = \sum_{1\le k\le l} \log(n_0^{k/l}) = \log n_0 \sum_{1\le k\le l} k/l = O(l \log n_0) = O(l \log(n/(1-\alpha_l))) = O(\log n).$$

Combining this with the non-reorganization cost of $O(\log n)$, the total amortized cost for insertion/deletion is $O(\log n)$ □.

Combining lemmas 1 to 4 and choosing l as a function of d and ϵ, we obtain a dynamic structure that supports range queries in $O(n^\epsilon)$ for any $\epsilon > 0$.

Theorem 1 *There exists a linear-space data structure that supports range searching in multiple dimensions in $O(n^\epsilon)$ time for any $\epsilon > 0$. Insertions and deletions have an amortized time complexity of $O(\log n)$.*

Worst-case bounds The amortized insertion and deletion time bounds of Theorem 1 can be changed into worst-case bounds with the general technique of global rebuilding, described in [Overmars 1983]. Next, we describe the central idea of of this technique. More details can be found in [Overmars 1983].

Consider a structure S that supports *weak* insertions and *weak* deletions. Insertions and deletions on a structure of n points are *weak*, if after αn insertions and/or deletions, with $0 < \alpha < 1$, all time and storage bounds of the structure have not increased asymptotically. The global rebuilding technique converts weak updates (insertions and deletions) into normal worst-case updates as follows: Maintain two copies of the data structure – the first one is constructed for n points and the second one is constructed after an additional $\alpha n/2$ updates have occurred. Let the number of points in the structure at this time be n_1. Queries are processed on the first structure until at most αn updates occur on it. They then switch to the second structure, whose construction would have completed by then. This ensures that the query processing time is not affected. Let us now see how the second structure is constructed. Instead of constructing the second structure all at once, the construction work is distributed across $\alpha n_1/6$ updates that follow. Since the construction can be accomplished in $O(n_1 \log n_1)$ time, this strategy adds a logarithmic overhead to each of the αn_1 updates. The updates that occur on the first structure after the construction of the second has started are buffered so that they can be applied on the second structure once its construction is completed. By ensuring that the number of such buffered updates is less than $\alpha n_1/2$, we have at most one structure under construction at all times. This technique is called global rebuilding and is explained in more detail in [Overmars 1983]. The following theorem summarizes this technique.

Theorem 2 *Given a tree T representing n points with rebuilding time bounded by $O(n \log n)$ that allows for weak insertions and weak deletions, we can dynamize it into a structure T' such that*

- *the insertion time on T' is $O(WI_T(n) + \log n)$.*
- *the deletion time on T' is $O(WD_T(n) + \log n)$.*
- *the query time and amount of storage space do not change asymptotically.*

Here, $WI_T(n)$ and $WD_T(n)$ denote the worst-case times for performing weak insertions and weak deletions on a structure T of n points. An insertion (or deletion) on a structure T is called *weak* if there exists an α, $0 < \alpha < 1$, such that after αn insertions (or deletions), all time and storage bounds of the structure have not increased asymptotically.

We apply this theorem on the structures corresponding to each level-i partition of an l-level range structure starting from $i = 1$ to $i = l$. This process converts all amortized bounds on these structures to worst-case bounds. This leads to the following result.

Theorem 3 *There exists a linear-space data structure that supports range searching in multiple dimensions in $O(n^\epsilon)$ time for any $\epsilon > 0$. Insertions and deletions can be performed in $O(\log n)$ time in the worst case.*

Proof: We show that every structure for a level-k partition can support insertions and deletions in $O(\log n)$ worst-case time. (Note that the l-level range structure corresponds to a level-l partition.) The proof is by induction on the level k.

Basis: $k = 1$. The B-tree corresponding to a level-1 partition has at most $4n^{1/l}$ points as its partitions. Therefore, it supports insertions/deletions in $O(\log n)$ worst-case time.

Hypothesis: Assume it holds for any structure T_k corresponding to any level-k partition, $k \leq r$. Consider a structure $T(x, r+1, \sigma)$ corresponding to a level-$(r+1)$ partition. This structure has a B-tree on a set of partitions. Each partition is organized as two r-level range structures. By induction hypothesis, each of these structures supports insertions/deletions in $O(\log n)$ worst-case time. Consequently, the weak insertion/deletion time for the $(r+1)$-level structure is given by

1. The time for identifying the partition in the level-$(r+1)$ B-tree
2. The time for insertion/deletion in the partition
3. The time for incorporation of the split of a partition in the level-$(r+1)$ B-tree

Since the B-tree has $O(n)$ partitions, Steps 1 and 3 take $O(\log n)$ time in the worst-case. By induction hypothesis, Step 2 also takes $O(\log n)$ time. Consequently, the total weak insertion/deletion time in the $(r+1)$-level structure is $O(\log n)$. Applying theorem 2, the weak insertion/deletion times can be transformed into worst-case bounds. □

5 Conclusions

In this paper we considered the problem of dynamic range searching using linear space structures. Since current structures such as the R-trees [Beckman et al. 1990, Guttman 1984] and Bang files [Freeston 1987] have high query time complexity ($O(n)$ for n points [Ravi and Ambuj 1997]), we focussed on alternate means to improve query performance in dynamic environments. We observed that Bentley's non-overlapping range structure [Bentley 1980] achieves $O(n^{\epsilon})$ query time for any $\epsilon > 0$. However, this structure is static in nature. We demonstrated that the same query performance can be achieved in dynamic environments by applying the dynamization techniques of Overmars [Overmars 1983]. The resulting structure has $O(n^{\epsilon})$ query time for any $\epsilon > 0$ and $O(\log n)$ insertion/deletion times.

In the future, we would like to compare the performance of the l-level range structure with the performance of conventional structures such as the R*-tree [Beckman et al. 1990, Berchtold et al. 1996] and the BV-tree [Freeston 1995] by conducting several experiments on real spatial and image data from the Alexandria Digital Library. An important point to note is that the l-level range structure does not form overlapping index regions as the R*-tree. Unlike the BV-tree, the l-level range structure does not require varying length tree nodes (the tree nodes increase by a factor of $\log n$ as we move up a BV-tree) to answer exact match queries in $O(\log n)$ time. The fact that partitions in an l-level range structure are always disjoint and that this information is captured in their extents is sufficient to guarantee logarithmic performance for exact match queries. Moreover, since data objects are clustered in multiple ways to suit to query needs and the best one is chosen at query time, we surmise that the l-level range structure would perform much better than conventional index structures.

References

[Beckman et al. 1990] Beckmann, N., Kreigel, H., Schneider, R., and Seeger. B. "The R*-tree: An efficient and robust access method for points and rectangles", *Porc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pp 322-331, 1990.

[Bentley 1977] Bentley, J. L. "Multidimensional binary search trees used for associative searching", *Communications of the ACM*, 18, pp. 509-517, 1977.

[Bentley 1980] Bentley J. L. and Maurer, H.A. "Efficient worst-case data structures for range searching", *Acta Informatica* 13, pp. 155-168, 1980.

[Bentley 1980] Bentley, J. L., Saxe, J. B. "Decomposable searching problems 1: Static-to-dynamic transformation", *Journal of Algorithms 1*, pp. 301-358, 1980.

[Berchtold et al. 1996] Berchtold, S., Keim, D.A., and Kreigel, H.P. "The X-tree: An index structure for high dimensional data", *Proc. of the Conf. on Very Large Data bases"*, Bombay, India, 1996.

[Freeston 1987] Freeston, M. " The Bang file: A new kind of grid file", *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pp 260-269, 1987.

[Freeston 1995] Freeston, M. "A general solution to the n-dimensional B-tree problem", *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, San Jose, CA, USA, May 1995.

[Guttman 1984] Guttman, A. "R-trees : A dynamic index structure for spatial searching", *Proceedings of the ACM SIGMOD Intl. Conf. on Management of Data*, pp. 47-57, 1984.

[Overmars and Leeuwen 1982] Overmars, M. H. and Leeuwen, J. "Dynamic multidimensional data structures based on quad and k-d trees", *Acta Informatica*, 17, pp. 267-283, 1982.

[Overmars and Leeuwen 1981] Overmars, M. H., and Leeuwen, J. "Worst-case optimal insertion and deletion methods for decomposable searching problems", *Information Processing Letters*, Vol. 12, no. 4, pp. 168-172, August 1981.

[Overmars 1983] Overmars, M. H. "The design of dynamic data structures", *LNC S 156*, Springer-Verlag, 1983.

[Ravi and Ambuj 1997] Ravi Kanth, K. V. and Ambuj K. Singh "Optimal dynamic range searching in non-replicating index structures", *Technical Report TRCS97-13*, Department of Computer Science, University of California, Santa Barbara, July 1997.

[van Kreveld and Overmars 1991] van Kreveld, M. and Overmars, M. H. "The divided k-d tree", *Algorithmica*, No. 6, pp. 840-858, 1991.

[Willard 1986] Willard, D. E. "On the application of sheared retrieval to orthogonal range queries", *ACM Symposium on Computational Geometry*, pp. 80-89, 1986.

Appendix A

Here, we will prove the properties of a B-tree for a set of partitions using a specific order.

Lemma 5 *Let* $q = \langle [a_1, b_1], \ldots, [a_d, b_d] \rangle$ *denote an arbitrary d-dimensional query. In a B-tree of k partitions using* $\sigma = \langle \sigma_1, \ldots, \sigma_d \rangle$*-order, at most 2 partitions partially intersect query q.*

Proof: Here, we prove the lemma for $d = 2$. The same can be generalized to an arbitrary number of dimensions. Assume for contradiction that there are three partitions that partially intersect a query $q = [a_1, b_1], [a_2, b_2]$. Let the extents of these partitions be denoted by $\langle p_1^1, p_2^1 \rangle$, $\langle p_1^2, p_2^2 \rangle$ and $\langle p_1^3, p_2^3 \rangle$, respectively. Since extent $\langle p_1^1, p_2^1 \rangle$ intersects q, we have that $\exists (s^1, t^1) : p_1^1 \preceq_\sigma (s^1, t^1) \preceq_\sigma p_2^1$ such that $(a_1 \leq s^1 \leq b_1) \wedge (a_2 \leq t^1 \leq b_2)$. This implies $(a_1, a_2) \preceq_\sigma (s^1, t^1)$. Since this intersection is partial, we also have that either $p_1^1 \prec_\sigma (a_1, a_2)$, or, $(b_1, b_2) \prec_\sigma p_2^1$ holds. Similar conditions hold for the other two partitions (extents). Of these three, two of them have a precedence relation with the same query endpoint. Let this endpoint be (a_1, a_2) and the extents be $\langle p_1^1, p_1^2 \rangle$ and $\langle p_1^2, p_2^2 \rangle$. This implies that both satisfy the precedences $p_1^1 \prec_\sigma (a_1, a_2)$ and $p_1^2 \prec_\sigma (a_1, a_2)$. By definition of intersection, $(a_1, a_2) \preceq_\sigma (s^1, t^1) \preceq_\sigma p_2^1$ for some (s^1, t^1) and $(a_1, a_2 \preceq_\sigma (s^2, t^2) \preceq_\sigma p_2^2$. Combining these equations with those for partial intersection, we obtain that the extents overlap. This contradicts the fact that the extents are always disjoint (i.e., $p_1^1 \preceq_\sigma p_1^2 \prec_\sigma p_1^2 \preceq_\sigma p_2^2$). □

Lemma 6 *Let* $q = \langle [a_1, b_1], \ldots, [a_d, b_d] \rangle$ *denote an arbitrary d-dimensional query. In a B-tree of k partitions using* $\sigma = \langle \sigma_1, \ldots, \sigma_d \rangle$*-order, any data point* $p = (p_1, \ldots, p_d)$ *in a partition, which is contained in q, satisfies* $a_{\sigma_1} \leq p_{\sigma_1} \leq b_{\sigma_1}$.

Proof: Let P be a partition contained in q (i.e., if $\langle p_1, p_2 \rangle$ is its extent then $(a_1, a_2) \preceq_\sigma p_1 \preceq_\sigma p_2 \preceq_\sigma (b_1, b_2)$). By definition of σ-order, it trivially follows that the σ_1-coordinates of all points in P lie in the required range. □