# Approximate Orthogonal Range Search using Patricia Tries

by

Qingxiu Shi and Bradford G.Nickerson

TR05-172, April 30, 2005

Faculty of Computer Science

University of New Brunswick

Fredericton, N.B. E3B 5A3

Canada

Phone: (506) 453-4566

Fax: (506) 453-3566

Email: fcs@unb.ca

www: http://www.cs.unb.ca

**Abstract**

We use Patricia tries to answer $\epsilon$-approximate orthogonal range search on a set of $n$ random points and rectangles in $k$-dimensional space. Given $n$ $k$-dimensional random points or rectangles and a $k$-dimensional query rectangle, $\epsilon$-approximate orthogonal range query counts (or reports) the points in the query rectangle or the rectangles intersecting the query rectangle, allowing errors near the boundary of the query rectangle. Points within a distance of a function of $\epsilon$ the boundary of the query rectangle might be misclassified. The approximate orthogonal range search time using Patricia tries is determined theoretically to be $O(k \log n/\epsilon^{k-1})$ for cubical range queries. Patricia tries are evaluated experimentally for $\epsilon$-approximate orthogonal range counting and reporting queries (for $2 \leq k \leq 10$ and $n$ up to 1,000,000) using uniformly distributed random points and rectangles, and we compared the performance of the Patricia trie for $k$-d points with the $k$-d tree and the adaptive $k$-d tree. The experimental results show that allowing small errors can significantly improve the query execution time of the approximate range counting. For $epsilon = 0.05$, an average of 50% fewer nodes are visited for the Patricia trie (compared to the exace range search).

# Contents



# List of Tables



# List of Figures

# 1 Introduction

Range search is among the fundamental problems in computational geometry, geographical information systems, computer graphics and applications of databases. Given a collection of keys (each containing multidimensional attributes) and a multidimensional query rectangle, an orthogonal range search asks for all keys in the collection with attribute values each inside the given rectangle. Over the past 30 years, more than 60 data structures for range search have been presented [1][5][11][13][17]. The motivation of our work is to improve the speed of range search while allowing for small counting (or reporting) errors.

The $k$-d tree was proposed by Bentley [3] as a generalization of the binary search tree in $k$-dimensional space. At each intermediate node, the $k$-d tree divides the $k$-dimensional space in two parts by a ($k$-1)-dimensional hyperplane parallel to the coordinate axes. The direction of the hyperplane alternates between the $k$ possibilities from one tree level to the next. Each splitting hyperplane has to contain at least one data point (see Figure 1). There are several disadvantages of the $k$-d tree: deleting a data point from the tree is complicated and may cause a reorganization of the subtree below this point; $k$-d tree is sensitive to the order in which the points are inserted, and data points are scattered all over the tree. An improved version proposed in [10] is the adaptive $k$-d tree. When splitting, the adaptive $k$-d tree chooses a hyperplane orthogonal to one of the coordinate axes that divides the space in two subspaces with equal number of data points. All data points are stored in the leaves. Splitting is continued recursively until each subspace contain only a certain number of points (see Figure 2). However, the adaptive $k$-d tree is a static structure, and it is difficult to keep the tree balanced in the presence of frequent updates.

It is instructive to associate with each node of the $k$-d tree an unique $k$-dimensional rectangle and the subset of data points that lie within this rectangle. The root node is associated with the bounding rectangle, which encloses all the data points. Bentley's range search algorithm simply visits recursively all nodes whose rectangle has a nonempty intersection with the query rectangle. Analysis of $k$-d trees for range searching has been considered by several researchers. The work required to construct a $k$-d tree and its storage requirements are $O(n \log n)$ and $O(kn)$ [4]. The search cost depends on the nature of the query. Lee and Wong [14] have shown that the search takes $O(n^{1-1/k} + F)$ worst-case time , where $F$ is the number of data points in range.

Instead of partitioning the search space using the data, the $k$-d trie [16] splits the search space based on the digits. Each partition splits a region

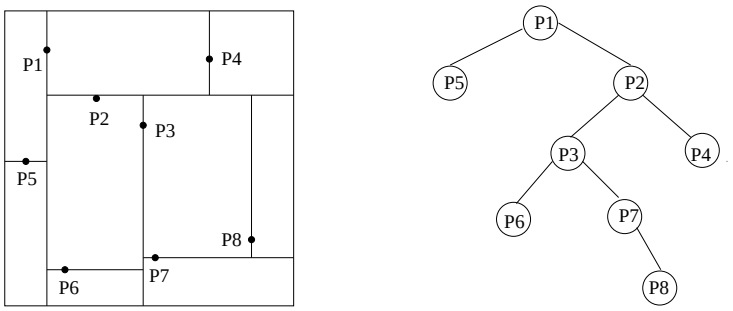



Figure 1: 2-d tree by inserting the points in the order of P1,P2,$\cdots$,P8.

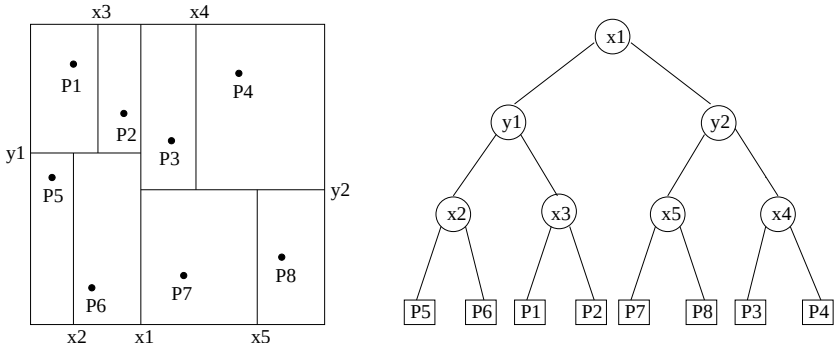



Figure 2: An adaptive 2-d tree for the same set of data points in Figure 1.

of the search space into two sub-regions of equal size. All nodes on level $i$ split the $((i \mod k)+1)$th attribute (we assume the root is at level 0). The branching policy on level $i$ is based on the $\lfloor i/k \rfloor$th bit of the $((i \mod k)+1)$th attribute of the given key represented in binary: go to left if it is 0 and go to right if it is 1. The splitting is not continued further when a sub-region contains one or no data points (see Figure 3). The $k$-d trie has an annoying flaw: there is one-way branching that leads to the creation of extra node in the tree. The Patricia trie was discovered by D.R. Morrison [15] to avoid this problem. A Patricia trie removes all one-child internal nodes from the $k$-d trie and stores the eliminated information in the nodes (see Figure 4). Patricia tries are well-balanced trees [19] in the sense that a random shape of Patricia tries resembles the shape of complete balanced trees. The Patricia trie has many applications, including lexicographical sorting, dynamic hashing algorithms, string matching, file systems, and most recently conflict resolution algorithms for broadcast communications [13]. Patricia tries can be preprocessed in $O(kn \log n)$ time and $O(kn)$ space, and fewer internal nodes are visited for a partial match search of a Patricia trie compared to a $k$-d trie [12].

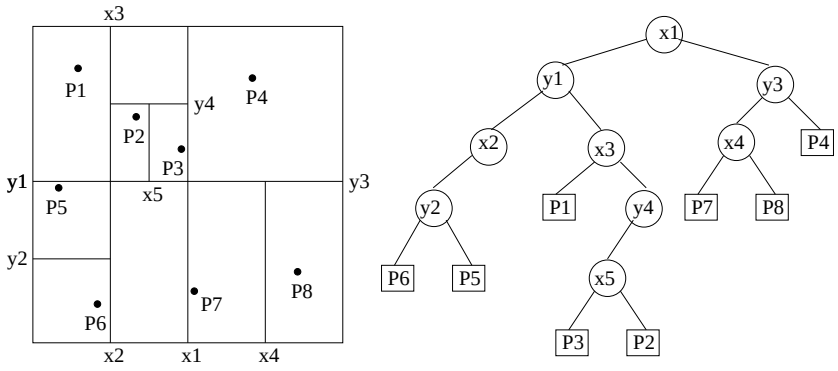



Figure 3: The 2-d trie for the same set of data points in Figure 2.

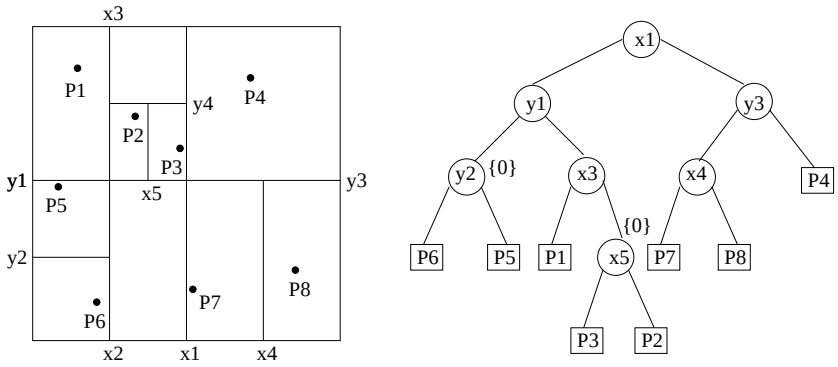



Figure 4: The Patricia trie for the same set of data points in Figure 2.

Lower bounds for range search were studied by Chazelle [9][8], who showed that a sequence of $n$ operations for insertion, deletion, and reporting points in a given range costs $\Omega(n(\log n)^k)$. Chazelle [7] gives a comprehensive overview of data structures for $k$-dimensional searching, including the description of a $k$-dimensional rectangle reporting algorithm (supporting dynamic operations) with $O(F(\log(\frac{2n}{F})^2)+\log^{k-1} n)$ time, which is close to the lower bound. To obtain better performance, several researchers turned to an approximate version of the range searching problem: instead of counting the points in the exact specified ranges, the data point whose distance to the boundary of the range is within $\epsilon$ times the range's diameter may or may not be included in the count. For fixed dimension $k$, the approximate range searching problem was solved optimally by Arya and Mount [2]. With an $O(kn)$-space structure called the balanced box-decomposition tree (BBD-tree) which can be constructed in $O(kn \log n)$ time, $\epsilon$-approximate range queries can be answered in $O(2^k \log n + (3\sqrt{k}/\epsilon)^k)$ time. If the ranges are convex, the approximate range query time can be strengthened to $O(2^k \log n + k^2(3\sqrt{k}/\epsilon)^{k-1})$. They

also presented a lower bound of $\Omega(\log n + 1/\epsilon^{k-1})$, for the complexity of answering $\epsilon$-approximate range queries assuming a partition tree approach for cubical range in fixed dimension.

In [18], we used Patricia tries to represent $k$-dimensional points, rectangles and combined textual and spatial data, and presented a range search algorithm for reporting all $k$-dimensional keys intersecting a query rectangle. We also showed that the Patricia trie can be preprocessed in $O(kn \log n)$ time and $O(kn)$ space. In this paper we study the performance of Patricia trie for $\epsilon$-approximate range searching with a slightly different definition of approximation from that in [2]. In Section 2, we state the problem of $\epsilon$-approximate orthogonal range search, and propose two algorithms for approximate orthogonal range reporting and counting queries, respectively. We theoretically analyze the cost of the approximate orthogonal range search in Section 3. In Section 4 we present an extensive experimental study of the practical performance of the Patricia trie using uniform randomly generated points and rectangles in $k$-dimensional space and compare it against the $k$-d tree and the adaptive $k$-d tree. The experimental results show that allowing small errors can significantly improve the query execution time of the approximate range counting, with a less dramatic effect on the complexity of approximate range reporting.

# 2 Approximate Orthogonal Range Search

Given a $k$-dimensional query rectangle $W = [L_1, H_1] \times [L_2, H_2] \times \cdots \times [L_k, H_k]$, $L_i \leq H_i$ with center $Z = (\frac{L_1+H_1}{2}, \frac{L_2+H_2}{2}, \cdots, \frac{L_k+H_k}{2})$. The edges of $W$ have given lengths $\Delta_1, \Delta_2, \cdots, \Delta_k$, where $\Delta_i = H_i - L_i$, $\forall\, i \in \{1, 2, \cdots, k\}$. We define $[MIN_i, MAX_i]$, $\forall\, i \in \{1, 2, \cdots, k\}$, as the minimum and maximum possible data coordinate values for dimension $i$. Given $0 \leq \epsilon \leq 0.5$, let $W^- = [L_1 + \Delta_1\epsilon, H_1 - \Delta_1\epsilon] \times [L_2 + \Delta_2\epsilon, H_2 - \Delta_2\epsilon] \times \cdots \times [L_k + \Delta_k\epsilon, H_k - \Delta_k\epsilon]$ be the $k$-dimensional inner query rectangle with center at $Z$, and let $W^+ = [L_1 - \Delta_1\epsilon, H_1 + \Delta_1\epsilon] \times [L_2 - \Delta_2\epsilon, H_2 + \Delta_2\epsilon] \times \cdots \times [L_k - \Delta_k\epsilon, H_k + \Delta_k\epsilon]$ be the $k$-dimensional outer query rectangle with center at $Z$. (We assume $MIN_i \leq L_i - \Delta_i\epsilon$ and $H_i + \Delta_i\epsilon \leq MAX_i$, $\forall\, i \in \{1, 2, \cdots, k\}$).

For a set $D$ of $n$ $k$-dimensional data points, define a legal answer to an $\epsilon$-approximate range query for any subset $D'$ such that

$$D \cap W^- \subseteq D' \subseteq D \cap W^+.$$

This definition allows for two-sided errors, by failing to count (or report) points that are barely inside $W$, and counting (or reporting) points barely outside $W$ (see Figure 5).

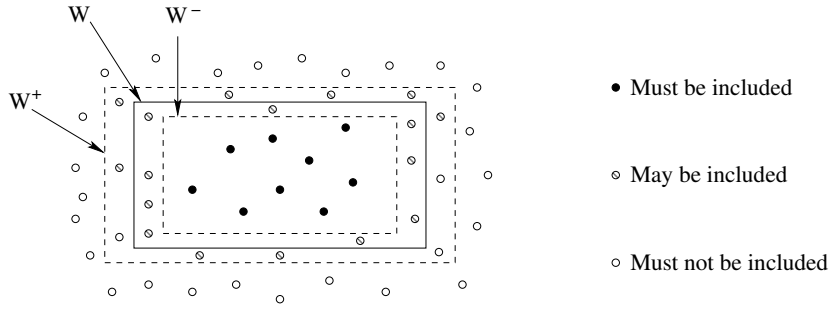
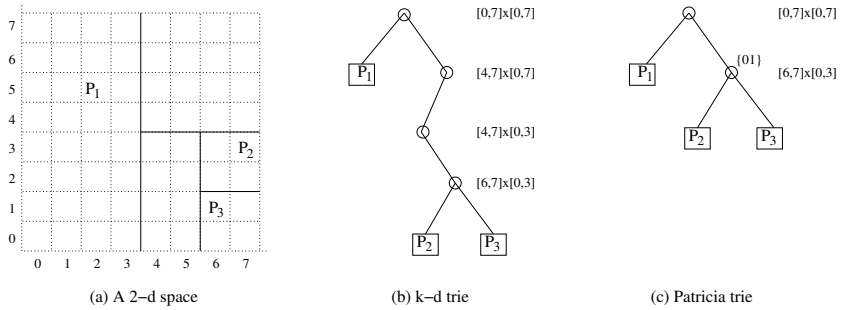



Figure 5: Approximate orthogonal range search queries.

We denote by $T$ the Patricia trie constructed by inserting a set $D$ of $n$ $k$-dimensional data points into an initially empty trie (the insertion algorithm is in [18]). There are altogether $n-1$ internal nodes and $n$ leaves in $T$. The skipped bits are stored in an array SKIPSTR, and every leaf is associated with one point.

Each node in $k$-d tries covers part of the $k$-d space, that is, every node has a cover space defined as $NC = [\mathcal{L}_1, \mathcal{U}_1] \times [\mathcal{L}_2, \mathcal{U}_2] \times \cdots \times [\mathcal{L}_k, \mathcal{U}_k]$. Arrays $\mathcal{L}$ and $\mathcal{U}$ store the lower and upper bounds of a node's cover space. In Figure 6(b) and (c), the list of tuples is the cover space $NC$ of each internal node. The root of $k$-d tries covers the whole space and child nodes cover half of the search space volume of their parent. For the root, $NC$ has $\mathcal{L}_i = MIN_i$ and $\mathcal{U}_i = MAX_i$, $\forall i \in \{1, \cdots, k\}$. The nodes on level $\ell$ split attribute $p = (\ell \mod k) + 1$ (at the root, $\ell = 0$). If a node on level $\ell$ has cover space $[\mathcal{L}_1, \mathcal{U}_1] \times \cdots \times [\mathcal{L}_p, \mathcal{U}_p] \times \cdots \times [\mathcal{L}_k, \mathcal{U}_k]$, then its left child's cover space is $[\mathcal{L}_1, \mathcal{U}_1] \times \cdots \times [\mathcal{L}_p, (\mathcal{L}_p + \mathcal{U}_p)/2] \times \cdots \times [\mathcal{L}_k, \mathcal{U}_k]$, and its right child's cover space is $[\mathcal{L}_1, \mathcal{U}_1] \times \cdots \times ((\mathcal{L}_p + \mathcal{U}_p)/2, \mathcal{U}_p] \times \cdots \times [\mathcal{L}_k, \mathcal{U}_k]$. For the Patricia tries, $\ell$ is not the level of the trie, but the length of the path from root to the node plus the length of the skipped bits in the internal nodes along the path. The node cover space must take the skipped bit string stored in the nodes into consideration. For example, in Figure 6(c), the node cover space of the root of 2-d Patricia trie is $[0, 7] \times [0, 7]$ ($\ell = 0$), the node cover space of its right child (denoted by $NC_r$) is computed as follows: first, $p = \ell \mod 2 + 1 = 1$, $NC_r = [4, 7] \times [0, 7]$ and $\ell = 1$; then, the first bit of the skipped bits string is '0', which means a left child node has been removed from the corresponding $k$-d trie (Figure 6(b)), $p = \ell \mod 2 + 1 = 2$, $NC_r = [4, 7] \times [0, 3]$ and $\ell = 2$; the second and last bit of the skipped bit string is '1', which means a right child node has been removed, $p = \ell \mod 2 + 1 = 1$, $NC_r = [6, 7] \times [0, 3]$ and $\ell = 3$. So the node cover space of the root's right child is $[6, 7] \times [0, 3]$, as shown in Figure 6(c).

The ARR algorithm (see Figure 7) is used to perform an approximate




(a) A 2–d space (b) k–d trie (c) Patricia trie

Figure 6: A 2-d space with three points and their corresponding tries.

range reporting on the Patricia trie $T$. The search proceeds from the root ($\ell = 0$) to the leaves, accounting for possible skipped bits stored at internal nodes ($T.SKIPSTR$). Arrays $\mathcal{L}$ and $\mathcal{U}$ are the lower and upper limits of the node's cover space, and are initialized to be $MIN_i$ and $MAX_i$, respectively, $\forall i \in \{1, \cdots, k\}$. If the node cover space falls within $W^+$ at some node $T$, then all the points in the subtree attached to $T$ are legal answers of the approximate range search and are collected by the COLLECT function into a *List* for reporting. If the node cover space falls outside of $W^-$ at some node $T$, the range search stops. If the node cover space overlaps both $W^-$ and $W^+$, then we recursively visit its children. When we reach a leaf node, we determine whether it is in $W$ using the CHECKNODE function; if so, it is added to *List*. The ARR algorithm can be easily modified for the approximate range counting: we denote by $T.WEIGHT$ the number of data points stored in the subtree attached to $T$. When we arrive at some node $T$ whose node cover space falling within $W^+$, we just add $T.WEIGHT$ to the count instead of using the COLLECT function to traverse all the nodes in the subtree attached to $T$ (see the ARC algorithm in Figure 8).

**Lemma 1** *The List in the* ARR *algorithm includes all the points lying inside the inner range $W^-$ and excludes all the points lying outside the outer range $W^+$.*

**Proof.** Let $p$ be some point lying inside the inner range $W^-$ and let $T$ be the node containing $p$ where we stop traversing down or we start to collect all the points in the subtree attached to $T$ into the *List*. If $T$ is a leaf node, by the CHECKNODE function, $p$ would be added to the *List* since it lies within $W$. Otherwise if $T$ is an internal node, since node $T$ does not result in a recursive ARR call, then the node cover space of $T$ must lie completely inside $W^+$. Therefore the point $p$ would be collected into the LIST by the COLLECT function. In the similar way, let $q$ be some point lying outside the

outer range $W^+$ and let $T$ be the node containing it where the range search stops. If $T$ is a leaf node, $q$ would not be added to the List because it lies outside $W$. If $T$ is an internal node, then, since node $T$ does not result in a recursive ARR call, the node cover space of $T$ must lie completely outside $W^-$, which leads to the stopping of traversing down of $T$'s subtree and adds nothing to the List. □

The correctness of the ARR algorithm follows immediately from Lemma 1 and the fact that no point is added to the *List* twice.

# 3 Approximate Range Searching Cost

Without loss of generality, the following discussions are all based on unit space $[0,1]^k$. We assume the input data and the query rectangle $W$ are drawn from a uniform random distribution.

**Theorem 2** *Given a Patricia trie $T$ built from $n$ random $k$-dimensional data points and a query rectangle $W$ of dimensions $\Delta_1 \times \Delta_2 \times \cdots \times \Delta_k$, and $0 < \epsilon \leq 0.5$, $\epsilon$-approximate range counting queries visit*

$$O(\log n \Sigma_{p=1}^{k}( \prod_{i=1, i\neq p}^{k} (2 + \frac{\Delta_i}{\Delta_p}(\frac{1}{\epsilon} - 2)))$$

*nodes in $T$.*

**Proof.** A node is said to be expanded if the algorithm visits the children of this node. For a node to be expanded, its node cover space must intersect with both the inner query rectangle $W^-$ and the outer query rectangle $W^+$. We call the facet of the query rectangle perpendicular to the $p$th orthogonal axis the $p$-facet. According to the definition of $W^-$ and $W^+$, the $p$-facets of $W^-$ and $W^+$ are separated from each other at least by a distance of $2\Delta_p\epsilon$, $\forall p \in \{1, \cdots, k\}$. So a node in $T$ with $|NC(p)| < 2\Delta_p\epsilon$, $\forall p \in \{1, \cdots, k\}$ is never expanded in the algorithm ARC, where $|NC(p)|$ is the length of the $p$-th side of node cover space $NC$.

Each partition of $T$ splits a region of the search space into two equal sub-regions. Each coordinate axis gets cut in turn, in a cyclical fashion of $1, 2, \cdots, k, 1, 2, \cdots$, which results in regions such that the length of the longest side is equal to or twice that of the smallest side, and $|NC(1)| \leq |NC(2)| \leq \cdots \leq |NC(k)|$. Assume a node in $T$ intersects both the $p$-facet of $W^-$ and the $p$-facet of $W^+$, $1 \leq p \leq k$, then $|NC(p)| \geq 2\Delta_p\epsilon$, and

```
ARR(T, ℓ, 𝓛, 𝓤, W⁻, W, W⁺, List)
 1  if T is a leaf node
 2     then CHECKNODE(T, W, List)
 3     else i ← 0
 4          while i < T.SKIPSTR.length()
 5          do p ← (ℓ mod k) + 1
 6             if T.SKIPSTR[i] = 0
 7                then 𝓤[p] ← (𝓛[p] + 𝓤[p])/2
 8                else 𝓛[p] ← (𝓛[p] + 𝓤[p])/2 + ε
 9             i ← i + 1
10             ℓ ← ℓ + 1
11          black = 0
12          for (i = 1; i <= k; i ← i + 1)
13          do if (W⁺.L[i] ≤ 𝓛[i]) and (𝓤[i] ≤ W⁺.H[i])
14                then black ← black + 1
15                else break
16          if black = k
17            then COLLECT(T, List)
18                 return
19          for (i = 1; i <= k; i ← i + 1)
20          do if (𝓛[i] > W⁻.H[i]) or (𝓤[i] < W⁻.L[i])
21                then return
22          p ← (ℓ mod k) + 1
23          if left[T] ≠ NIL
24            then 𝓤[p] ← (𝓛[p] + 𝓤[p])/2
25                 ARR(left[T], ℓ + 1, 𝓛, 𝓤, W⁻, W, W⁺, List)
26          if right[T] ≠ NIL
27            then 𝓛[p] ← (𝓛[p] + 𝓤[p])/2 + ε
28                 ARR(right[T], ℓ + 1, 𝓛, 𝓤, W⁻, W, W⁺, List)
```

Figure 7: Pseudo-code for the approximate range reporting algorithm in the Patricia trie. $T.SKIPSTR$ is the skipped bit string stored in $T$ and $T.SKIPSTR.length()$ is the length of $T.SKIPSTR$. $T.SKIPSTR[i]$ is the $i$th bit of $T.SKIPSTR$. $\varepsilon$ is a small value to guarantee $T$'s left and right children's cover spaces don't share any point.

```
ARC(T, ℓ, 𝓛, 𝓤, W⁻, W, W⁺, Count)
 1  if T is a leaf node
 2     then if T ∈ W
 3             then Count ← Count + 1
 4     else  i ← 0
 5           while i < T.SKIPSTR.length()
 6           do p ← (ℓ mod k) + 1
 7              if T.SKIPSTR[i] = 0
 8                 then 𝓤[p] ← (𝓛[p] + 𝓤[p])/2
 9                 else  𝓛[p] ← (𝓛[p] + 𝓤[p])/2 + ε
10              i ← i + 1
11              ℓ ← ℓ + 1
12           black = 0
13           for (i = 1; i <= k; i ← i + 1)
14           do if (W⁺.L[i] ≤ 𝓛[i]) and (𝓤[i] ≤ W⁺.H[i])
15                 then black ← black + 1
16                 else  break
17           if black = k
18              then Count ← Count + T.WEIGHT
19                   return
20           for (i = 1; i <= k; i ← i + 1)
21           do if (𝓛[i] > W⁻.H[i]) or (𝓤[i] < W⁻.L[i])
22                 then return
23           p ← (ℓ mod k) + 1
24           if left[T] ≠ NIL
25              then 𝓤[p] ← (𝓛[p] + 𝓤[p])/2
26                   ARC(left[T], ℓ + 1, 𝓛, 𝓤, W⁻, W, W⁺, Count)
27           if right[T] ≠ NIL
28              then 𝓛[p] ← (𝓛[p] + 𝓤[p])/2 + ε
29                   ARC(right[T], ℓ + 1, 𝓛, 𝓤, W⁻, W, W⁺, Count)
```

Figure 8: Pseudo-code for the approximate range counting algorithm in the Patricia trie. $T.WEIGHT$ is the number of points in the subtree attached to $T$.

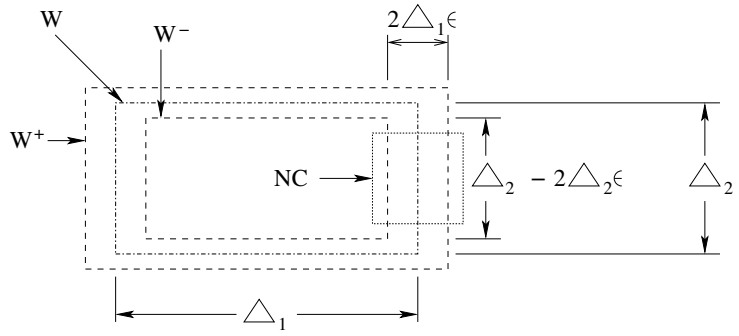



Figure 9: An illustration of a node in $T$ with cover space $NC$ intersecting both the 1-facet of $W^-$ and the 1-facet of $W^+$; $\epsilon = 0.1$.

$|NC(i)| \geq \Delta_p\epsilon$, $\forall i \in \{1, \cdots, p-1\}$, and $|NC(j)| \geq 2\Delta_p\epsilon$, $\forall j \in \{p+1, \cdots, k\}$ (see Figure 9). So there are at most

$$(\prod_{i=1}^{p-1}(1 + \lceil\frac{\Delta_i - 2\Delta_i\epsilon}{\Delta_p\epsilon}\rceil))(\prod_{j=p+1}^{k}(1 + \lceil\frac{\Delta_j - 2\Delta_j\epsilon}{2\Delta_p\epsilon}\rceil))$$

regions intersecting with both the $p$-facet of $W^-$ and the $p$-facet of $W^+$. Each $k$-d rectangle has $2k$ facets, so altogether there are at most

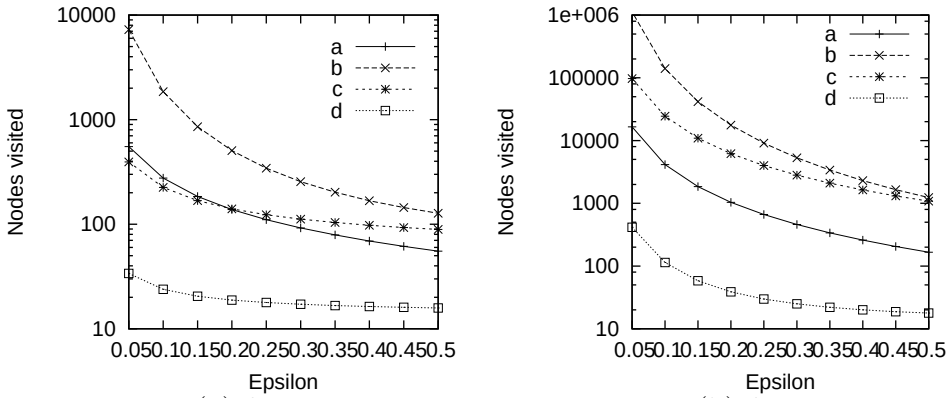
$$\begin{aligned} & 2\Sigma_{p=1}^{k}(\prod_{i=1}^{p-1}(1 + \lceil\frac{\Delta_i - 2\Delta_i\epsilon}{\Delta_p\epsilon}\rceil))(\prod_{j=p+1}^{k}(1 + \lceil\frac{\Delta_j - 2\Delta_j\epsilon}{2\Delta_p\epsilon}\rceil)) \\ \leq\ & 2\Sigma_{p=1}^{k}(\prod_{i=1}^{p-1}(2 + \frac{\Delta_i - 2\Delta_i\epsilon}{\Delta_p\epsilon}))(\prod_{j=p+1}^{k}(2 + \frac{\Delta_j - 2\Delta_j\epsilon}{2\Delta_p\epsilon})) \\ \leq\ & 2\Sigma_{p=1}^{k}(\prod_{i=1, i\neq p}^{k}(2 + \frac{\Delta_i - 2\Delta_i\epsilon}{\Delta_p\epsilon})) \end{aligned}$$

regions overlapping both $W^-$ and $W^+$. The depth of the Patricia trie built from random data points is $O(\log n)$ [13], so we reach the desired result.

□

*The special case.* When the query rectangle $W$ is a square, i.e. $\Delta_1 = \Delta_2 = \cdots = \Delta_k$, then the number of nodes visited in the worst case is $O(k \log n/\epsilon^{k-1})$. Figure 10 shows the comparison between our result and Arya and Mount's results.

Based on the algorithms in Section 2, besides the number of nodes visited for the range counting in the $ARC$ algorithm (Figure 8), up to $2F$ additional




(a) $k = 2$ (b) $k = 3$

Figure 10: Number of nodes visited versus $\epsilon$ using equations (a) $k \log n/\epsilon^{k-1}$, (b) $2^k \log n + (3\sqrt{k}/\epsilon)^k$, (c) $2^k \log n + k^2(3\sqrt{k}/\epsilon)^{k-1}$ and (d) $\log n + 1/\epsilon^{k-1}$ $(n = 1,000,000)$.

nodes are visited in the COLLECT function in *ARR* algorithm (Figure 7) for reporting, where $F$ is the number of points in range, so we have the following conclusion:

**Corollary 3** *Given a Patricia trie $T$ built from $n$ random $k$-dimensional data points and a query rectangle $W$ of dimensions $\Delta_1 \times \Delta_2 \times \cdots \times \Delta_k$, and $0 < \epsilon \leq 0.5$, $\epsilon$-approximate range reporting queries visit*

$$O(\log n \Sigma_{p=1}^{k}( \prod_{i=1, i \neq p}^{k} (2 + \frac{\Delta_i}{\Delta_p}(\frac{1}{\epsilon} - 2)) + F)$$

*nodes in $T$, where $F$ is the number of points reported.*

# 4 Experiments

We have conducted a series of experiments to validate the theoretical analysis of the previous section, and to show the possible savings when we settle for approximate range search instead of exact range search. Our experiments of approximate range counting and reporting were performed using uniform random distributed data from the interval [0,1] for $\epsilon$ ranging from 0 to 0.5, $2 \leq k \leq 10$, $n$ up to 1,000,000, and the query square's volume *vol* ranging from 0.0001 to 0.01, and the query square's side length $w$ ranging from 0.003125 to 0.4. The programs were run on a Sun Microsystems V880 with four 1.2 GHz UltraSPARC III processors, 16 GB of main memory, running Solaris 8.

Each experimental point in the following graphs was done with an average of 300 test cases.

## 4.1 Approximate Range Counting Queries

### 4.1.1 $k$-dimensional points

The $k$-dimensional points were uniformly and randomly generated from the unit space $[0,1]^k$. We tested two different kinds of query squares for the same points set: query squares with fixed volume $vol$ and query squares with fixed side length $w$ for different $k$. The experimental results with $k$-dimensional square window queries with volumes that range from 0.0001 to 0.01 for Patricia tries are shown in Figure 12. The experimental results are found to be consistent with the theoretical analysis in Figure 11. We show the results of experiments with $k$-dimensional query squares with side length that ranges from 0.003125 to 0.4 for different dimension $k$ in Figure 14. We find that there are significant improvements in running time when $\epsilon$ grows from 0 to 0.05. As $\epsilon$ increases, the running times tend to converge, irrespective of $k$ and $w$. For $w < 0.05$, the number of nodes visited is very small for all values of $\epsilon$ tested. For fixed $w$ and $k > 4$, the number of nodes visited is almost the same for all values of $\epsilon$ (see Figure 15).

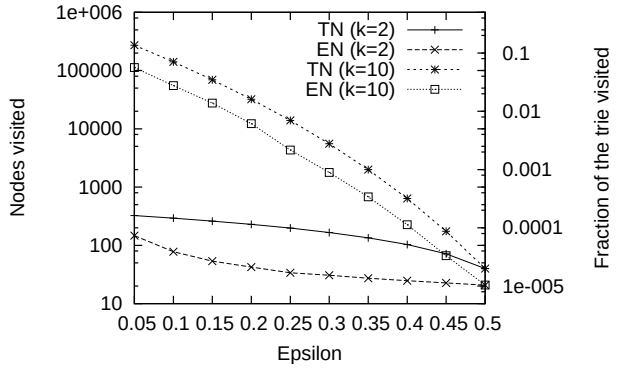



Figure 11: The theoretical (TN) and experimental (EN) number of nodes visited in Patricia trie for $k$-d points with $k$-d query square volume $vol = 0.001$ for different $k$ ($k = 2$, $k = 10$ and $n = 1,000,000$).

We define $F_{\epsilon=x}$ as the number of points counted as in range for an $\epsilon$-approximate range query, and $F_{\epsilon=0}$ as the number of points in the exact range query. For fixed $w$ and $\epsilon = x$, we define the fraction of points miscounted as

$$\delta_{\epsilon=x} = \frac{|F_{\epsilon=x} - F_{\epsilon=0}|}{F_{\epsilon=0}}.$$

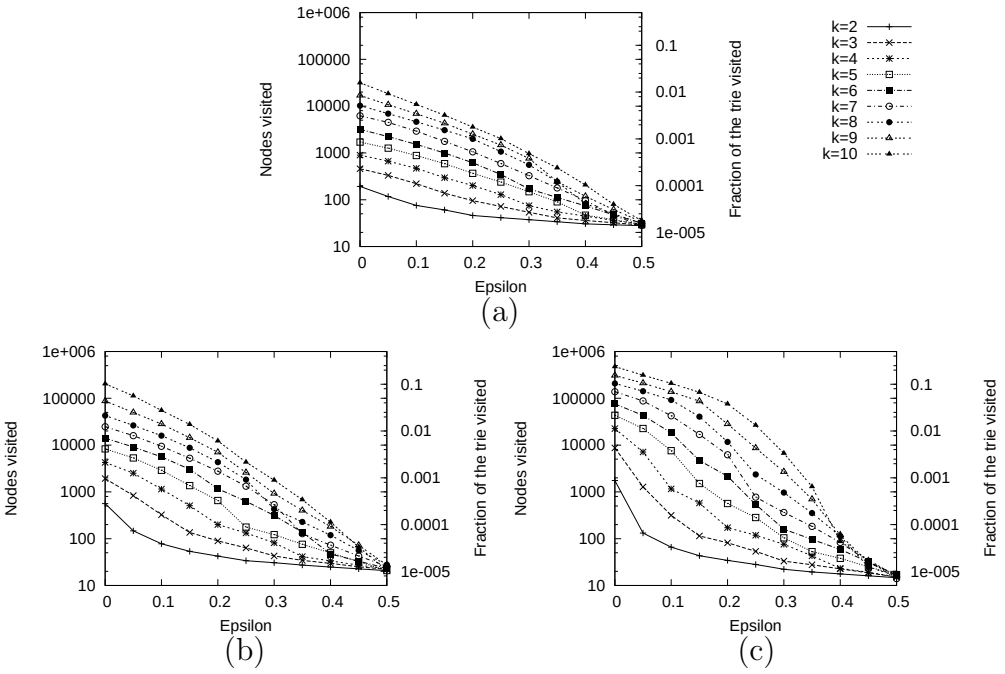



Figure 12: Number of nodes visited versus $\epsilon$ in Patricia trie for $k$-dimensional points with $k$-d query square volume of (a) 0.0001, (b) 0.001 and (c) 0.01 ($n = 1,000,000$).

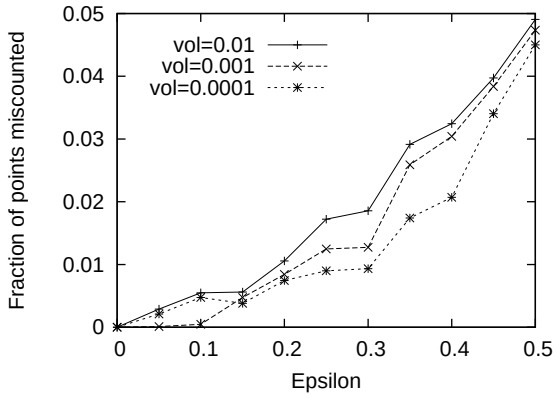
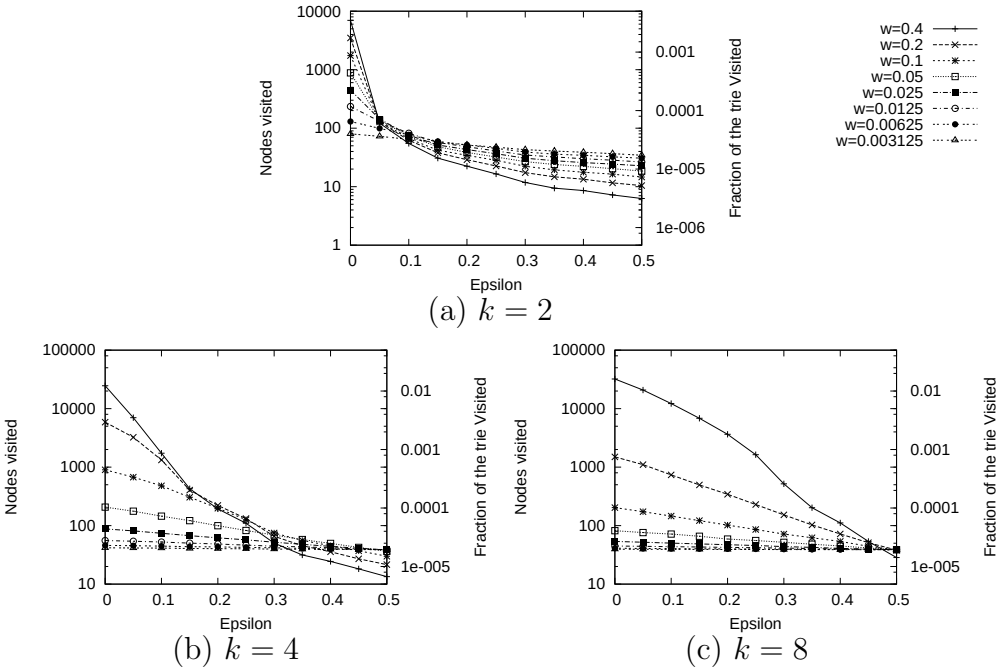



Figure 13: Fraction of points miscounted $\delta_{\epsilon=x}$ versus $\epsilon$ in Patricia trie for 2-dimensional points with 2-d query square's volume ranging from 0.0001 to 0.01 ($n = 1,000,000$).




(a) $k = 2$

(b) $k = 4$

(c) $k = 8$

Figure 14: Number of nodes visited versus $\epsilon$ in Patricia trie for $k$-dimensional points with $k$-d query square's side length from 0.003125 to 0.4 ($n = 1,000,000$).

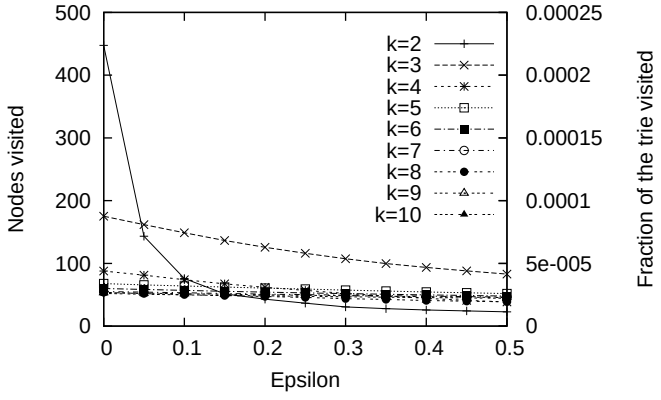



Figure 15: Number of nodes visited versus $\epsilon$ in Patricia trie for $k$-dimensional points with $k$-d query square's side length $w = 0.025$ ($n = 1,000,000$).

In Figures 13 and 16, we show the average error for $k = 2$ and $n = 1,000,000$.

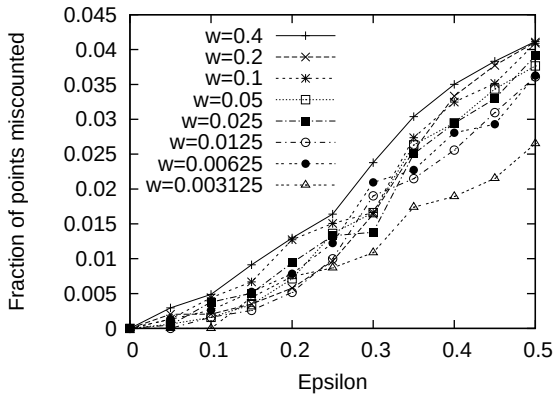



Figure 16: Fraction of points miscounted $\delta_{\epsilon=x}$ versus $\epsilon$ in Patricia trie for 2-dimensional points with 2-d query square's side length from 0.003125 to 0.4 ($n = 1,000,000$).

We compare the approximate range counting performance of the Patricia trie to the $k$-d tree and the adaptive $k$-d tree for different $k$ and query squares. We show the fraction of the tree visited versus $\epsilon$ for $n = 100,000$ in Figures 17 and 18 with fixed query square volume for different $k$, and in Figures 19 and 20 with fixed query square's side size $w$. The performance of the Patricia trie is at least as good as that of the adaptive $k$-d tree in all the cases, and much better than the $k$-d tree when $k = 2$. When $k = 4$, the performance of the $k$-d tree is better than that of both the Patricia trie and the adaptive $k$-d tree when $w = 0.2$ (the query square volume $vol$=0.01) and $\epsilon < 0.2$, however when $w$ (the query square volume) decreases, the Patricia trie and the adaptive $k$-d tree are getting better than the $k$-d tree. Because of the large preprocessing time of the adaptive $k$-d tree, we use the smaller $n = 100,000$ instead of $n = 1,000,000$.

### 4.1.2 $k$-dimensional rectangles

In [6] a binary $2k$-d trie data structure for $k$-d rectangles range search was investigated. Exploiting the same method, we used the Patricia trie for $k$-d rectangles range search in [18]. A $k$-d rectangle $R$ can be represented as a $2k$-d point $(x_1^{min}, x_1^{max}, x_2^{min}, x_2^{max}, \cdots, x_k^{min}, x_k^{max})$. Given a $k$-dimensional query rectangle $W = [L_1, H_1] \times [L_2, H_2] \times \cdots \times [L_k, H_k]$, $R$ intersects $W$ iff $x_i^{min} \in [MIN_i, H_i]$ and $x_i^{max} \in [L_i, MAX_i]$, $\forall i \in \{1, \cdots, k\}$. We denote by $T$ after inserting a set $D$ of $n$ $k$-dimensional rectangles into an initially empty

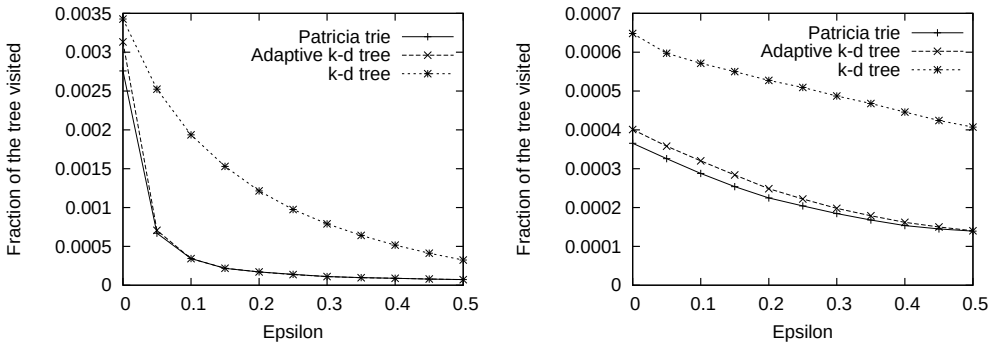



Figure 17: The fraction of the tree visited for the $k$-d tree, Patricia trie, and the adaptive $k$-d tree with the $k$-d query square volume (left)$vol$=0.01 and (right) $vol$=0.0001 ($n = 100,000$ and $k = 2$).

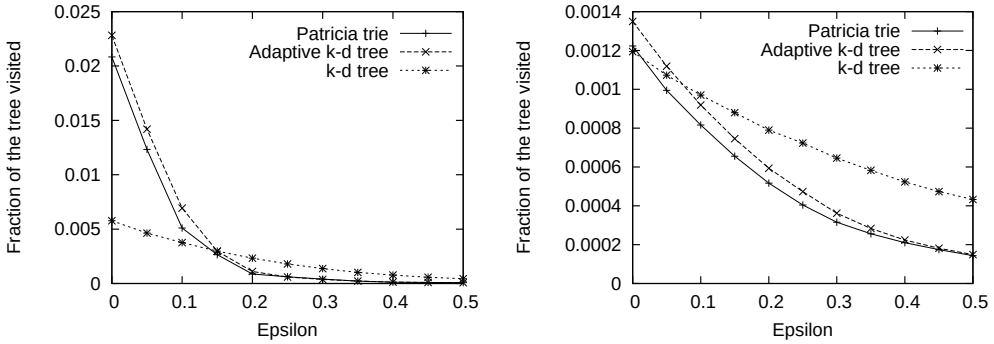



Figure 18: The fraction of the tree visited for the $k$-d tree, Patricia trie, and the adaptive $k$-d tree with the $k$-d query square volume (left) $vol$=0.01 and (right)$vol$=0.0001 ($n = 100,000$ and $k = 4$).

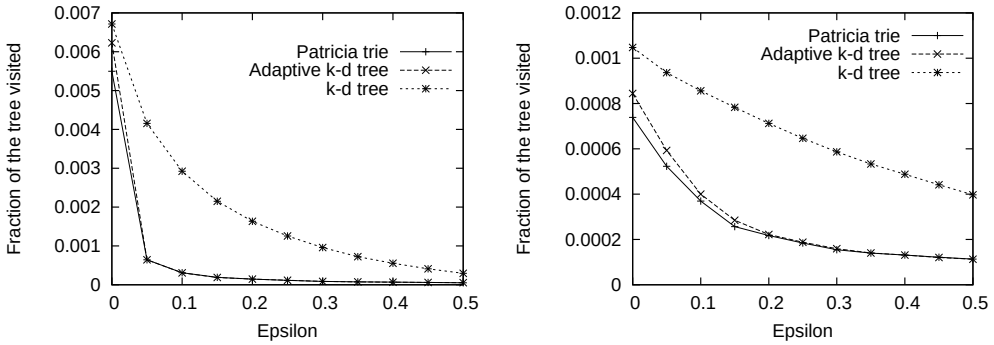



Figure 19: The fraction of the tree visited for the $k$-d tree, Patricia trie, and the adaptive $k$-d tree with the $k$-d query square's side length (left) $w = 0.2$ and (right) $w = 0.025$ ($n = 100,000$ and $k = 2$).

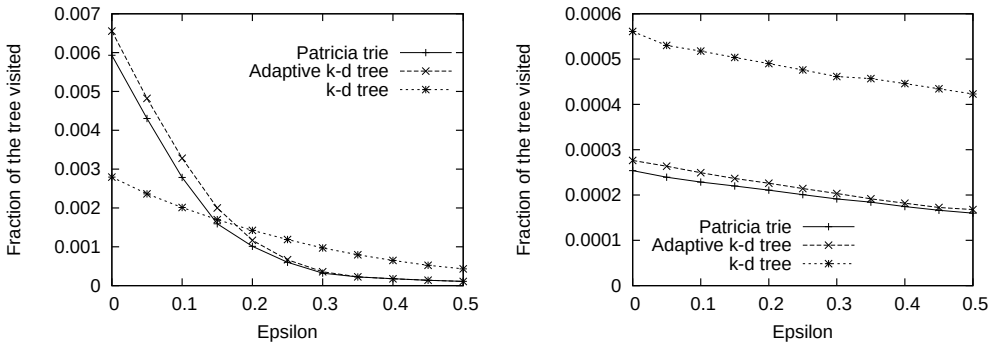


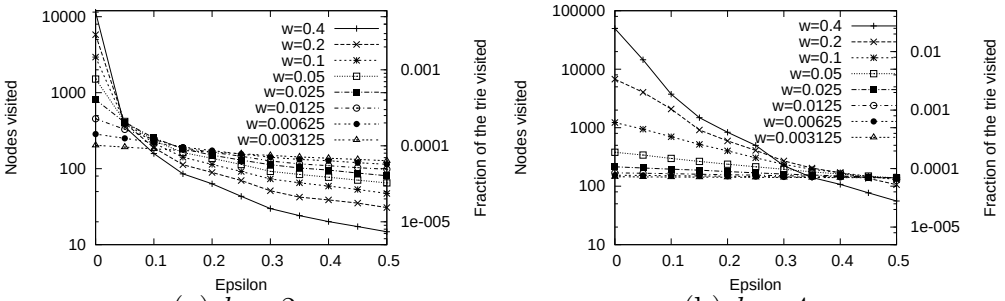

Figure 20: The fraction of the tree visited for the $k$-d tree, Patricia trie, and the adaptive $k$-d tree with the $k$-d query square's side length (left) $w = 0.2$ and (right) $w = 0.025$ ($n = 100,000$ and $k = 4$).

trie. Each node in $T$ covers part of the $2k$-d space, that is, every node has a cover space defined as $NC = [\mathcal{L}_1, \mathcal{U}_1] \times [\mathcal{L}_2, \mathcal{U}_2] \times \cdots \times [\mathcal{L}_{2k}, \mathcal{U}_{2k}]$. We define the query rectangle $W$'s cover space $WC = [MIN_1, H_1] \times [L_1, MAX_1] \times \cdots \times [MIN_k, H_k] \times [L_k, MAX_k]$. In the similar way, we define the inner query rectangle $W^-$'s cover space $WC^- = [MIN_1, H_1 - \Delta_1\epsilon] \times [L_1 + \Delta_1\epsilon, MAX_1] \times \cdots \times [MIN_k, H_k - \Delta_k\epsilon] \times [L_k + \Delta_k\epsilon, MAX_k]$, and the outer query rectangle $W^+$'s cover space $WC^+ = [MIN_1, H_1 + \Delta_1\epsilon] \times [L_1 - \Delta_1\epsilon, MAX_1] \times \cdots \times [MIN_k, H_k + \Delta_k\epsilon] \times [L_k - \Delta_k\epsilon, MAX_k]$. The ARR algorithm (Figure 7) can be used for $k$-d rectangles with some modifications: all $k$s are changed to $2k$s, and $WC^-$, $WC$ and $WC^+$ are used instead of $W^-$, $W$ and $W^+$.

The rectangle centers were uniformly distributed and the lengths of their sides uniformly and independently distributed between 0 and $maxsize$ ($0 \leq maxsize \leq 0.01$). We show the average number of nodes visited versus $\epsilon$ with $k$-dimensional query square's side length $w$ that ranges from 0.003125 to 0.4 for different $maxsize$ in Figures 21 and 22. They have a similar trend as in Figure 14, except that there are more nodes visited for the same $k$ for rectangles. Comparing Figures 21 and 22, the improvements for larger $maxsize$ are more significant. The fraction of points miscounted $\delta_{\epsilon=x}$ for $k = 2$ and $n = 1,000,000$ is shown in Figure 23.

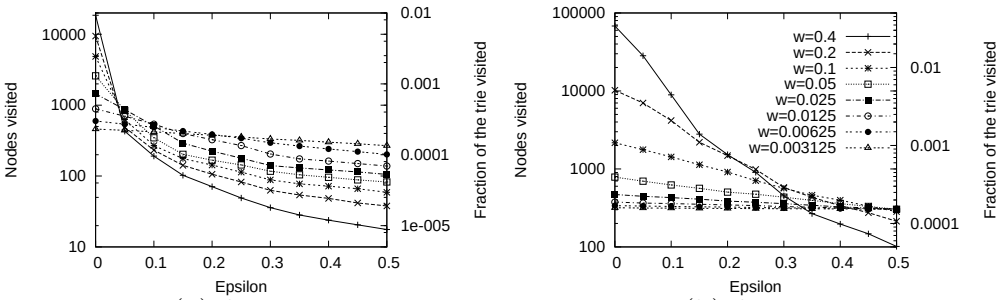
## 4.2 Approximate Range Reporting Queries

From the analysis in Section 3, besides the number of nodes visited for the range counting, additional nodes are visited in the COLLECT function in $ARR$ algorithm (Figure 7) for reporting, that is, $O(F_{\epsilon=0})$. The following experimental results are found to be consistent with the theoretical analysis.




(a) $k = 2$ (b) $k = 4$

Figure 21: Number of nodes visited versus $\epsilon$ in Patricia trie for $k$-dimensional rectangles with 2-d query square's side length from 0.003125 to 0.4 ($n = 1,000,000$ and $maxsize = 0.001$).




(a) $k = 2$ (b) $k = 4$

Figure 22: Number of nodes visited versus $\epsilon$ in Patricia trie for 4-dimensional rectangles with $k$-d query square's side length from 0.003125 to 0.4 ($n = 1,000,000$ and $maxsize = 0.01$).

### 4.2.1 $k$-dimensional points

For the query squares with fixed volume for different $k$, the number of points in range $F = vol \times n$. when $vol = 0.0001$ and $n = 1,000,000$, $F = 100$; in the similar way when $vol = 0.01$, $F = 10,000$. Figure 25 shows the average number of nodes visited during approximate range reporting for query squares with volume ranging from 0.0001 to 0.01. Comparing Figures 25 and 12, we find that there is only a little improvement when $k < 5$ for the different query square volumes, because the average number of nodes visited is dominated by the number of points in range $F$, especially when the query square volume is large.

For the query squares with fixed side size $w$, we show the average number of points in range when $\epsilon = 0$ in Table 1, . We show the average number of nodes visited during approximate range reporting in Figure 26. Comparing

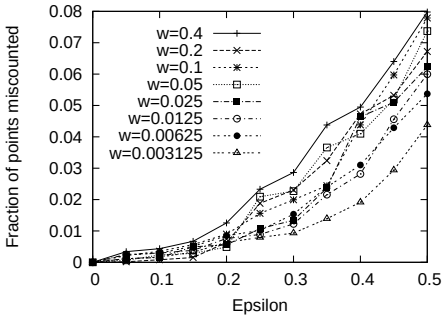



Figure 23: Fraction of points miscounted $\delta_{\epsilon=x}$ versus $\epsilon$ using Patricia trie for 2-dimensional rectangles for 2-d query square's side length from 0.003125 to 0.4 ($n = 1,000,000$ and $maxsize = 0.001$).

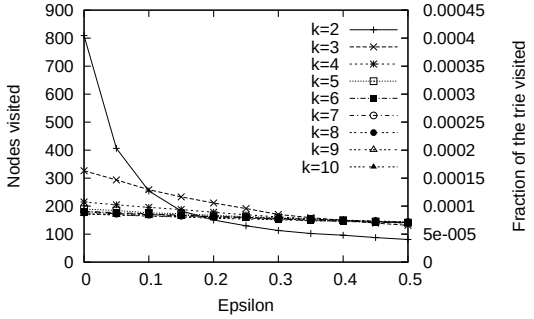



Figure 24: Number of nodes visited versus $\epsilon$ in Patricia trie for $k$-dimensional rectangles for $k$-d query square's side length $w = 0.025$ ($n = 1,000,000$ and $maxsize = 0.001$).

Figure 26(a) with 14(a), we find that there is only a little improvement when $k = 2$ even $\epsilon = 0.5$ , because of the large number of nodes in range according to Table 1 when $w > 0.025$. When $k$ grows, there are less nodes in range. It is easy to see that when $k = 4$ and $w \leq 0.05$, the number of nodes visited in Figure 26(b) is almost the same as that in Figure 14(b), because of zero or a very small number of nodes in range. And we can conclude that as $k$ grows, for the same $w$, the number of nodes visited for the range reporting is close to that for counting, as shown in Figures 14(c) and 26(c). According to the definition of the average error, which only depends on the number of legal answers, $\epsilon$-approximate range reporting and counting have the same average error (see Figure 16).

### 4.2.2 $k$-dimensional rectangles

We show the average number of rectangles in range when $\epsilon = 0$ in Tables 2 and 3 for $maxsize = 0.001$ and $maxsize = 0.01$, respectively. In Figures 27 and 28, we show the average number of nodes visited versus $\epsilon$ during approximate range reporting for different $k$ and $maxsize$. They have the similar trend as points.

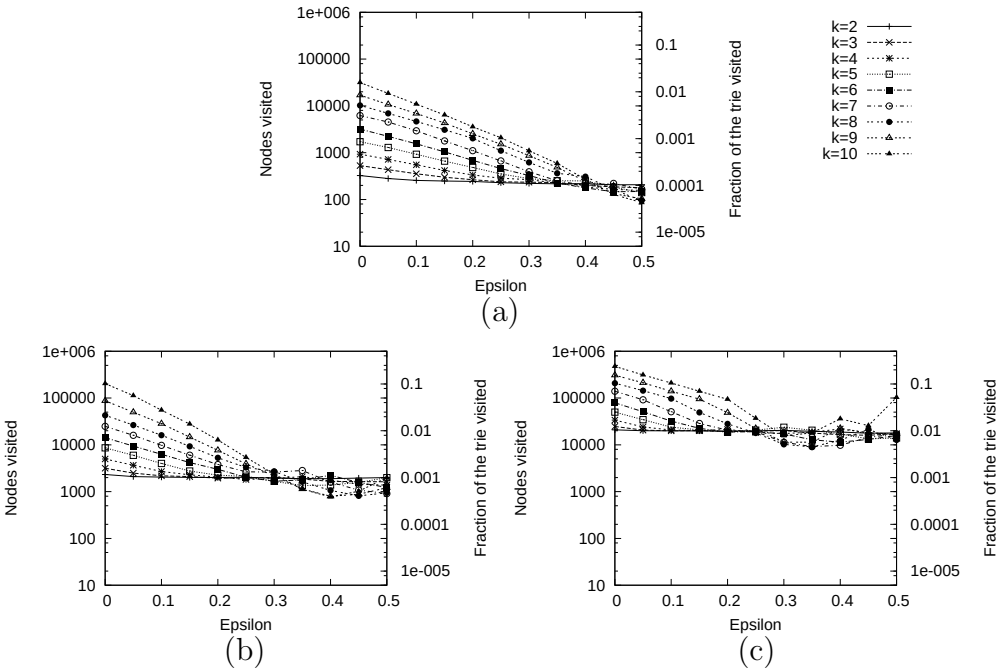
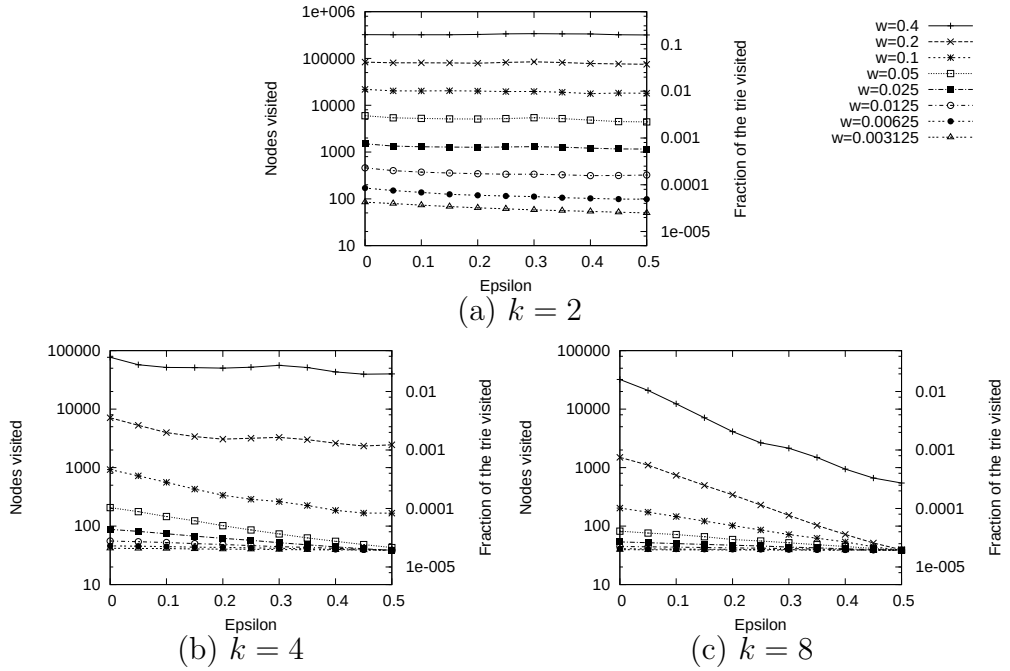



Figure 25: Number of nodes visited versus $\epsilon$ in Patricia trie for $k$-dimensional points with $k$-d query square volume (a) $vol$=0.0001, (b) $vol$=0.001 and (c) $vol$=0.01 ($n = 1,000,000$).

Table 1: The average number of points in range ($n$=1,000,000).

| $F_{\epsilon=0}$ | $w$=0.003125 | 0.00625 | 0.0125 | 0.025 | 0.05 | 0.1 | 0.2 | 0.4 |
|---|---|---|---|---|---|---|---|---|
| $k$=2 | 9.43 | 38.52 | 155.71 | 624.24 | 2500.81 | 10005.71 | 39995.76 | 159984.70 |
| 3 | 0 | 0 | 1.55 | 15.40 | 124.65 | 1000.40 | 8000.50 | 63973.70 |
| 4 | 0 | 0 | 0 | 0 | 5.56 | 99.17 | 1600.56 | 25567.50 |
| 5 | 0 | 0 | 0 | 0 | 0 | 9.57 | 319.76 | 10237.95 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0.52 | 63.57 | 4097.71 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 12.33 | 1636.30 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 2.15 | 653.90 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 260.75 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 103.83 |




(a) $k = 2$

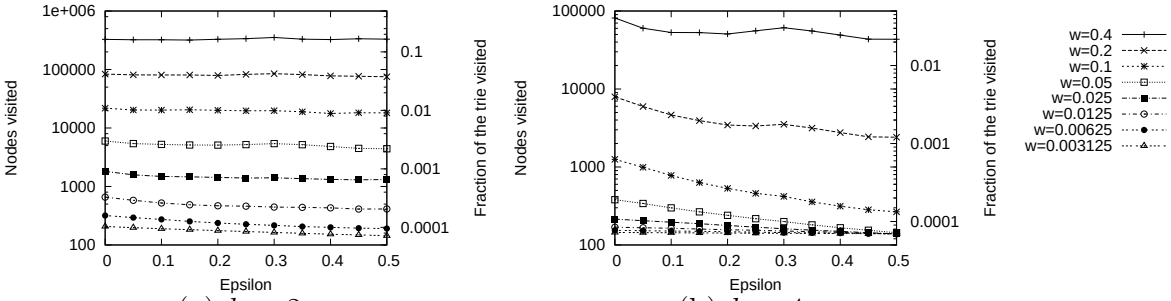
(b) $k = 4$

(c) $k = 8$

Figure 26: Number of nodes visited versus $\epsilon$ in Patricia trie for $k$-dimensional points with $k$-d query square's side length from 0.003125 to 0.4 ($n = 1,000,000$).

Table 2: The average number of rectangles in range ($n = 1,000,000$ and $maxsize = 0.001$).

| $F_{\epsilon=0}$ | $w$=0.003125 | 0.00625 | 0.0125 | 0.025 | 0.05 | 0.1 | 0.2 | 0.4 |
|---|---|---|---|---|---|---|---|---|
| $k$=2 | 12.78 | 45.07 | 169.14 | 651.07 | 2552.36 | 10104.36 | 40247.57 | 160579.43 |
| 3 | 0 | 0 | 1.85 | 16.15 | 128.54 | 1017.23 | 8068.23 | 64338.78 |
| 4 | 0 | 0 | 0 | 0 | 6.07 | 102.21 | 1616.57 | 25751.28 |
| 5 | 0 | 0 | 0 | 0 | 0 | 9.78 | 323.64 | 10327.93 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0.5 | 64.71 | 4134.36 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 12.92 | 1658.69 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 1.92 | 664.58 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 266.42 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 106.25 |

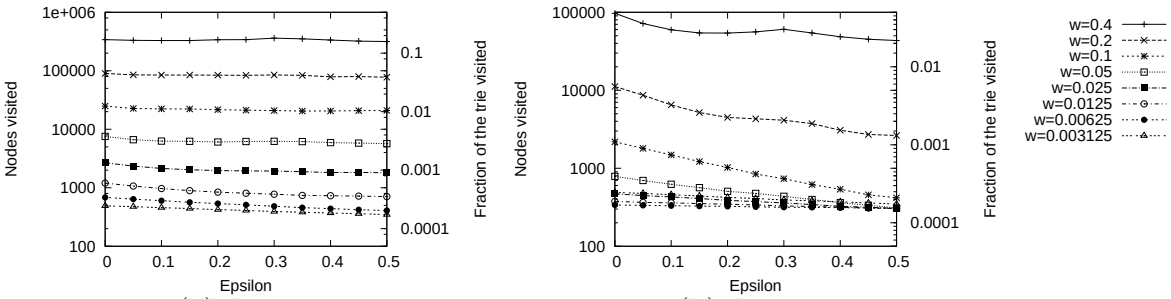



(a) $k = 2$ (b) $k = 4$

Figure 27: Number of nodes visited versus $\epsilon$ in Patricia trie for $k$-dimensional rectangles with $k$-d query square's side length from 0.003125 to 0.4 ($n = 1,000,000$ and $maxsize = 0.001$).

Table 3: The average number of rectangles in range when $\epsilon = 0$ and $maxsize = 0.01$ ($n$=1,000,000).

| $F_{\epsilon=0}$ | $w$=0.003125 | 0.00625 | 0.0125 | 0.025 | 0.05 | 0.1 | 0.2 | 0.4 |
|---|---|---|---|---|---|---|---|---|
| $k$=2 | 65.69 | 126.33 | 305.92 | 904.83 | 3055.08 | 11143.00 | 42443.50 | 165617.69 |
| 3 | 0 | 1.00 | 4.77 | 26.46 | 168.08 | 1171.23 | 8745.67 | 67330.69 |
| 4 | 0 | 0 | 0 | 0.36 | 9.00 | 124.36 | 1802.5 | 27449.33 |
| 5 | 0 | 0 | 0 | 0 | 0.08 | 12.92 | 371.33 | 11164.00 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1.00 | 75.62 | 4536.69 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 15.33 | 1847.78 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 2.91 | 752.00 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 305.20 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 123.90 |

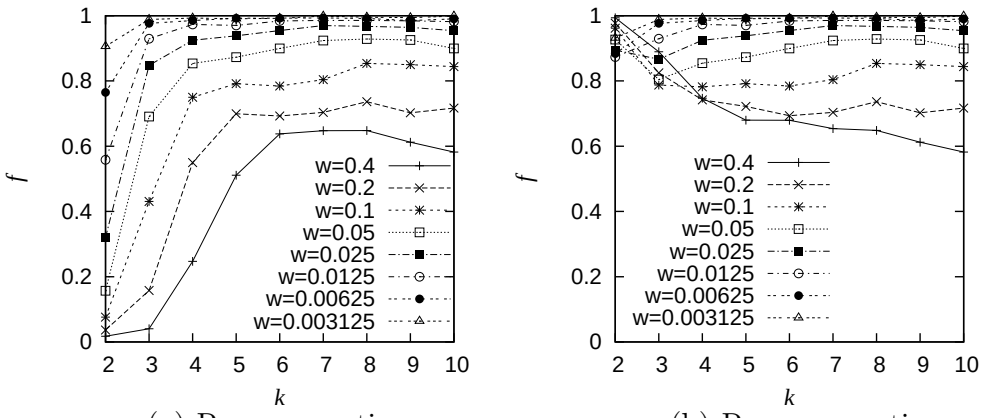



(a) $k = 2$ (b) $k = 4$

Figure 28: Number of nodes visited versus $\epsilon$ in Patricia trie for $k$-dimensional rectangles with $k$-d query square's side length from 0.003125 to 0.4 ($n = 1,000,000$ and $maxsize = 0.01$).

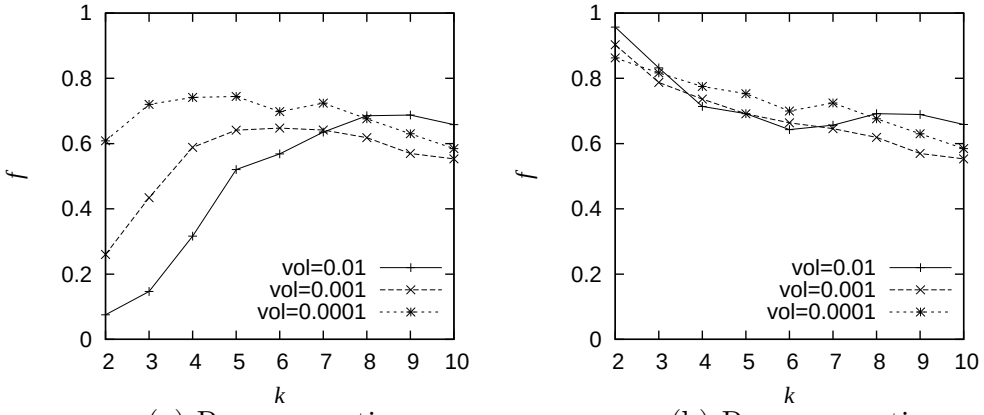
# 5 Conclusions and Future Works

We use Patricia tries to answer $\epsilon$-approximate orthogonal range search queries on a set of $n$ points or rectangles in $k$-dimensional space, and proposed the approximate orthogonal range counting algorithm and range reporting algorithm. Patricia tries are theoretically analyzed and evaluated experimentally for $\epsilon$-approximate orthogonal range counting and reporting queries using uniformly distributed random points and rectangles. The performance of the Patricia trie for $k$-d points was compared with the $k$-d tree and the adaptive $k$-d tree. We show that Patricia tries can be used to answer orthogonal range counting queries visiting $O(k \log n/\epsilon^{k-1})$ for cubical range queries. Can the result be improved closer to the lower bound of $\Omega(\log n + 1/\epsilon^{k-1})$ in fixed dimension? Figures 29 and 30 shows $f = \frac{y_{\epsilon=x}}{y_{\epsilon=0}}$, where $y_{\epsilon=0}$ is the number of nodes visited for an exact range counting query, and $y_{\epsilon=x}$ is the number of nodes visited for an $\epsilon$-approximate range counting query. Experimental results show that if we allow small relative errors, the number of nodes visited for range counting can be reduced at least by 1/4 for query squares with side length $w \geq 0.2$ (see Figure 29(a)) and by more than 1/4 for query squares with volume *vol* ranging from 0.0001 to 0.01 (see Figure 30(a)), $\epsilon = 0.05$ and $2 \leq k \leq 10$. Range reporting queries have a less dramatic improvement when $k \leq 5$, because of additional $O(F)$ nodes visited, which is the dominating term in the number of the nodes visited during range reporting (see Figures 29(b) and 30(b)). Another open question is how to perform combined textual and spatial data approximate range search.




(a) Range counting (b) Range reporting

Figure 29: The fraction $f$ of nodes visited when $\epsilon = 0.05$ for query squares with fixed side length $w$ for different $k$ ($n = 1,000,000$).




(a) Range counting (b) Range reporting

Figure 30: The fraction $f$ of nodes visited when $\epsilon = 0.05$ for query squares with fixed volume *vol* for different $k$ ($n = 1,000,000$).

# References


[1] P. Agarwal. *Handbook of Discrete and Computational Geometry*, chapter Range Searching, pages 575–598. CRC Press LLC, Boca Raton, FL, 1997.

[2] S. Arya and D. Mount. Approximate range searching. *Computational Geometry: Theory and Applications*, 17:135–163, 2000.

[3] J. Bentley. Multidimensional binary search trees for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[4] J. Bentley. Multidimensional binary search trees in database applications. *IEEE Trans. Softw. Eng.*, 5(4):333–340, 1979.

[5] J. Bentley and J. Friedman. Data structures for range searching. *ACM Computing Surveys*, 11(4):397–409, December 1979.

[6] L. Bu and B. Nickerson. Multidimensional orthogonal range search using tries. In *Canadian Conference on Computational Geometry*, pages 161–165, Halifax, N.S., August 2003.

[7] B. Chazelle. A functional approach for data structure and its use in multidimensional searching. *SIAM Journal of Computing*, 17(3):427–462, June 1988.

[8] B. Chazelle. Lower bounds for orthogonal range search: II. the arithmatic model. *Journal of the ACM*, 37(3):39–463, July 1990.

[9] B. Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the ACM*, 37(2):200–212, April 1990.

[10] J. Friedman, J. Bentley, and R. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.

[11] V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30:170–231, 1998.

[12] P. Kirschenhofer and H. Prodinger. Multidimensional digital searching-

alternative data structures. *Random Structures and Algorithms*, 5(1):123–134, 1994.

[13] D. Knuth. *The art of computer programming: sorting and searching*, volume 3, pages 492–512. Addison-Wesley, Reading, Mass., 2 edition, 1998.

[14] D. Lee and C. Wong. Quintary trees: a file structure for multidimensional database systems. *ACM Transaction on Database Systems*, 5:339–353, 1980.

[15] D. Morrison. Patricia - practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM*, 14(4):514–534, October 1968.

[16] J. Orenstein. Multidimensional tries used for associative searching. *Information Processing Letters*, 14(14):150–156, June 1982.

[17] H. Samet. *The design and analysis of spatial data strutures*. Addison-Wesley, Reading, MA, 1990.

[18] Q. Shi and B. G. Nickerson. k-d range search with binary patricia tries. Technical report, TR04-168, Faculty of Computer Science, University of New Brunswick, December 2004, 35 pages.

[19] W. Szpankowski. Patricia tries again revisited. *Journal of the ACM*, 37(4):691–711, 1990.