Tries for combined text and spatial data range search

Bradford G.Nickerson * Qingxiu Shi†

Abstract

We use tries to represent combined text and spatial data, and present a range search algorithm for reporting all 2-d points and rectangles from a set of size n intersecting a query rectangle. Data and queries can include text. Our k-d+t tries are evaluated experimentally (for n up to 300,000) using uniform distributed random spatial data and randomly selected strings from a set of place names. For random queries, we find that k-d+t tries have faster search times compared to naive search. The expected range search time for k-d+t tries was determined theoretically, and found to agree with experimental results for n=100,000 and $2 \leq k \leq 10$.

1 Introduction

Tries were introduced by Rene de la Briandais [8]. E. Fredkin [5] and Donald E. Knuth [7] developed them further. The trie is a simple order preserving data structure supporting fast retrieval of elements and efficient nearest neighbor and range searches. In general, a trie stores a set of n items, which are represented by keys from Σ^∞, the set of all infinite sequences over a finite alphabet Σ. A trie is composed of internal nodes and external nodes that store the keys. The branching policy at level d is based on the d^{th} symbol of a key. Tries are mainly used in text search and have been applied to multidimensional spatial data search using k-d trie [4][11] or quadtries [12].

There are three kinds of trie [13]: full trie, ordinary trie and Patricia trie. The Patricia trie was discovered by D.R. Morrison [10]. All nodes in Patricia trie have degree greater than or equal to two by eliminating all one-child internal nodes. Patricia tries are well-balanced trees [14] 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 [7]. To our knowledge, the Patricia trie hasn't been investigated for combined text and spatial data range search. In this paper, we will discuss tries for 2-dimensional points and strings, and for 2-dimensional rectangles and strings.

*Faculty of Computer Science, University of New Brunswick, Fredericton, N.B., Canada, E3B 5A3, bgn@unb.ca

†Faculty of Computer Science, University of New Brunswick, Fredericton, N.B., Canada, E3B 5A3, l749u@unb.ca

2 Trie for Text and Points

Without loss of generality, the following discussions are all based on unit space $[0,1]^k$ for spatial data. We assume the coordinate value on each spatial data dimension can be represented in B bits, and each character in a string is represented by C bits. Binary tries use a binary representation of a key to store the key as a path in a tree [1], and follow the rule of direction determined by the i^{th} bit information of the key: branch left if bit i=0 and branch right if bit i=1. Binary k-d tries use the principle of bit interleaving.

For a point $P(x,y)$ in two dimensions, each coordinate value has B bits, and the bit interleaved value is $b_0^x\, b_0^y\, b_1^x\, b_1^y \cdots b_{B-1}^x\, b_{B-1}^y$, where b_i^x is the i^{th} bit value for x, b_i^y is the i^{th} bit value for y. Now we add a string field t to the point $P(x,y)$ and treat t as the third dimension, denoted by $P(x,y,t)$. The number of bits of t can vary from 1 to m, $1 \leq m \leq C$. We call the data structure a k-d+t trie.

More precisely, for $P = (x,y,t)$, x and y have B bits, and t has $C \times Length(t)$ bits. If $C \times Length(t) < Bm$, we will add '0' at the end of the bit string of s and extend the length of t to Bm. The bit string after interleaving the binary representations of x, y and t is: $b_0^x\, b_0^y\, b_0^t \cdots b_{m-1}^t\, b_1^x\, b_1^y\, b_m^t \cdots b_{2m-1}^t \cdots b_{B-1}^x\, b_{B-1}^y\, b_{(B-1)m}^t \cdots b_{Bm-1}^t$. The bit interleaving principle can be extended to k-d points in a straightforward manner. Here we consider only $k = 2$ and one string for each point. We denote by T the Patricia trie constructed by inserting n records into an initially empty trie. 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 record [7]. After preprocessing all n records, we obtain the trie T, which allows us to carry out an orthogonal range search.

Given a query rectangle $W = [L_1, H_1] \times [L_2, H_2] \times [L_t, H_t]$, a point $P(x,y,t)$ is in the range iff $x \in [L_1, H_1]$, $y \in [L_2, H_2]$ and $s \in [L_t, H_t]$. For example, if $L_t =$ "a" and $H_s =$ "b", the string of each point in W must begin with "a". Each node in the trie T covers part of the space, denoted by NC (for node cover space), and there are three types of relations of W and NC [2]: BLACK (NC is in W), GREY (NC intersects but not in W) and WHITE (NC doesn't intersect W). The range search algorithm (RANGESEARCH, see Figure 1) traverses from the root of trie T down to its leaves. Arrays $\mathcal{L}$ and $\mathcal{U}$ store the lower and upper bounds of node T's cover space on x and y coordinates, respectively. At the root, the interleaved bit string position $\ell = 0$. For the root, the cover space NC has $\mathcal{L}_p = 0$ and $\mathcal{U}_p = 1$, for $p < k$. The cover space is split on the p^{th} dimension as we

move down, $p = \ell mod (k+m), \forall \ell \in \{0, 1, \cdots,$(k+m)B-$1\}$. On the p^{th} dimension, with $p < k$, if a parent node T has cover space $[\mathcal{L}_p, \mathcal{U}_p]$, then T's left child's cover space is $[\mathcal{L}_p, (\mathcal{L}_p + \mathcal{U}_p)/2]$ and T's right child's cover space is $[(\mathcal{L}_p + \mathcal{U}_p)/2 + \epsilon, \mathcal{U}_p]$. ϵ is a small value to guarantee open intervals.

```
RANGESEARCH(T, ℓ, L, U, RI, W, List, flag)
 1  if T = NIL
 2    then return
 3  if T is a leaf node
 4    then CHECKNODE(T, W, List); return
 5  i ← 0
 6  while i < T.SKIPSTR.length()
 7  do ℓ ← ℓ + 1; p ← (ℓ − 1) mod (k + m)
 8     if p < k
 9       then if T.SKIPSTR[i] = 0
10              then U[p] ← (L[p] + U[p])/2
11              else  L[p] ← (L[p] + U[p])/2 + ε
12            RI[p] ← INRANGE(L[p], U[p], ℓ, W)
13       else  q ← m × ((l − 1)/(m + k)) + p − k
14             STRINGINRANGE(T.SKIPSTR[i],
15             q, RI[k], W, flag)
16     i ← i + 1
17  C ← COLOR(RI)
18  if C is grey
19    then ℓ ← ℓ + 1; p ← (ℓ − 1) mod (k + m)
20         q ← m × ((l − 1)/(m + k)) + p − k
21         if left[T] ≠ NIL
22           then if p < k
23                  then U[p] ← (L[p] + U[p])/2
24                       RI[p] ← INRANGE
25                       (L[p], U[p], ℓ, W)
26                  else  STRINGINRANGE
27                       (0, q, RI[k], W, flag)
28                RANGESEARCH(left[T], ℓ, L, U,
29                RI, W, List, flag)
30         if right[T] ≠ NIL
31           then if p < k
32                  then L[p] ← (L[p] + U[p])/2 + ε
33                       RI[p] ← INRANGE
34                       (L[p], U[p], ℓ, W)
35                  else  STRINGINRANGE
36                       (1, q, RI[k], W, flag)
37                RANGESEARCH(right[T], ℓ, L, U,
38                RI, W, List, flag)
39    else  if C is black
40            then COLLECT(T, List)
```

Figure 1: Range search algorithm for k-d+t trie T. $T.SKIPSTR[i]$ is the $(i+1)^{st}$ bit of bit strings stored at compressed nodes of the Patricia trie T.

The RANGESEARCH algorithm (see Figure 1) is used to perform a range search on a k-d+t trie T. The search proceeds from the root (ℓ = 0) to the leaves, accounting for possible skipped bits stored at internal nodes. If $p < k$, the INRANGE function determines the color; if $p \geq k$, function STRINGINRANGE determines the color. Variable *flag* is used to track the state of text string bits being in range. When $flag = 2$, it indicates that all bits in t = all bits in L_t and H_t up to this point. Figure 2 shows the states 2, 0 and 1 arrived at by comparing string bit t_i with text range bits L_t^i and H_t^i.

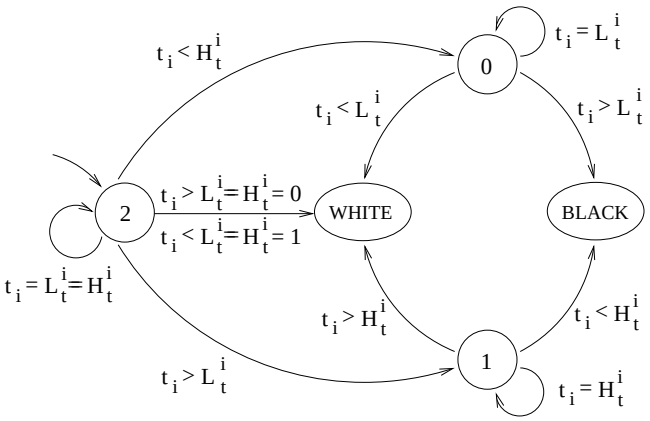


Figure 2: Finite automaton illustrating string range search state within a k-d + t trie. Function STRINGINRANGE implements this automaton.

If all ranges fall within W at some node T_u, then all points in the subtree attached to T_u are in W and are collected by the COLLECT function into a *List* for reporting. Array RI of size $k+1$ (initialized to store all grey values) keeps track of the color of the NC^p to W^p relationship for T and ancestors of T. Function COLOR(RI) determines the color (black, grey or white) of node T. When we reach a leaf node, we determine whether it is in range using the CHECKNODE function; if so, is added to *List*.

3 Trie for Text and Rectangles

In [2] a binary $2k$-d trie data structure for k-d rectangles range search was investigated. A rectangle is represented as four coordinate values $(x^{\min}, x^{\max}, y^{\min}, y^{\max})$, which, after bit interleaving gives the bit string: $b_0^{x^{\min}} b_0^{x^{\max}} b_0^{y^{\min}} b_0^{y^{\max}} b_1^{x^{\min}} b_1^{x^{\max}} \cdots$ $b_{B-1}^{x^{\min}} b_{B-1}^{x^{\max}} b_{B-1}^{y^{\min}} b_{B-1}^{y^{\max}}$. Thus, a rectangle can be represented as a 4-d point in a binary trie. Two rectangles R_1 and R_2 intersect if and only if their sides on every dimension in the data space intersect, i.e. $R_1 \cap R_2$ is true iff $(x^{\min}, x^{\max}) \cap (y^{\min}, y^{\max})$ is true, which happens when $x^{\min} \in [0, y^{\max})$ and $x^{\max} \in (y^{\min}, 1]$. For a given query rectangle $W = [L_1, H_1] \times [L_2, H_2]$, the query rectangle's cover space $WC^4 = [\mathcal{L}_p, \mathcal{U}_p]^4$, $\mathcal{L}_p = 0, \mathcal{U}_p = H_j - \epsilon$, when $p \bmod 2 = 1$; $\mathcal{L}_p = L_j + \epsilon$, $\mathcal{U}_p = 1$, when $p \bmod 2 = 0$, where $1 \leq p \leq 4$, $j = \lceil p/2 \rceil$.

A string t is added to the rectangle, denoted

by $R = (x^{\min}, x^{\max}, y^{\min}, y^{\max}, t)$. Similar to points+text above, we treat t as the fifth dimension. After bit interleaving, the resultant bit string is $b_0^{x^{\min}} b_0^{x^{\max}} b_0^{y^{\min}} b_0^{y^{\max}} b_0^t \cdots b_{m-1}^t b_1^{x^{\min}} b_1^{x^{\max}} \cdots b_{B-1}^{x^{\min}} b_{B-1}^{x^{\max}} b_{B-1}^{y^{\min}} b_{B-1}^{y^{\max}} b_{(B-1)m}^t \cdots b_{Bm-1}^t$. The range search algorithm for rectangles+text is similar to RANGESEARCH for points+text with $k = 4$.

Figures 3 and 4 depict an example of the range search algorithm, illustrated on an integer domain and a full trie for clarity. Figure 4(a) is the full trie for the data (i.e. E=($[1,3] \times [3,5]$, "Fredericton"), F=($[5,6] \times [5,7]$, "Saint-Louis") and G=($[4,7] \times [1,3]$, "Moncton")) shown in Figure 3 and 4(b) is the corresponding Patricia trie. In Figure 4(a), empty circles represent white nodes. A black-filled circle represents a black node, which means all rectangles+text represented by leaves inside the subtree attached to the black node intersect W. A grey-filled circle represents a grey node. For query rectangle $W = [2,5] \times [1,4] \times$ ["E", "G"], the query rectangle's cover space is $WC^4 = [0,5)(2,7][0,4)(1,7]$. The trie is divided into B= 3 bands, each of height $2k + m = 7$ (see Figure 4(a)). For G, traversal of T during the range search stopped at Band 1 when the first white node was encountered. For rectangle F, a white node is found in Band 0, and for rectangle E, the intersection with W is determined in Band 2.

4 Analysis

We adapt the approach used in [3] and [9] to analyze the range search cost of k-d+t tries for text and points (the analysis of tries for text and rectangles is similar), using Theorem P of [6]. We assume the input point data and text and the query hyper-rectangle W are drawn from a uniform random distribution.

Proposition 1 *Given a Patricia trie T containing a set D of n k-d points and text, i.e. $D = \{P_1, \cdots, P_n\}$, assuming D and query rectangle W satisfy the uniform probabilistic model, e.g. $W = ([L_1, H_1], \cdots, [L_k, H_k], [L_t, H_t])$, $S \subset \{1, 2, \cdots, k+1\}$, the expected cost of partial match retrieval $Q_S(n, k+1)$ measured by the number of nodes traversed in trie T is*

$$Q_S(n, k+1) = E\{\Sigma_{t=1}^{2n-1} \prod_{p \in S} |NC_t^p|\}.$$

Proof. The probability that a node in trie T will be visited is determined by the volume of every node's cover space in the space $[0,1]$. □

The time complexity of range search is proportional to the number of grey and black nodes visited in the trie built from input data D. We have the equation: $Q(n, k+1) = \Sigma_{t=1}^{2n-1} 1_{[node_t \in GN \cup BN]}$, where we use $1_{[A]}$ as the characteristic function of the event A.

Lemma 2 $E\{\Sigma_{t=1}^{2n-1} \prod_{p=1}^{k+1} |NC_t^p|\} = O(\log_2 n)$.

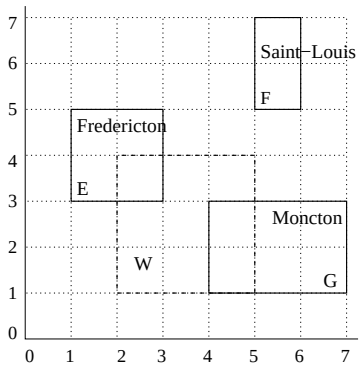
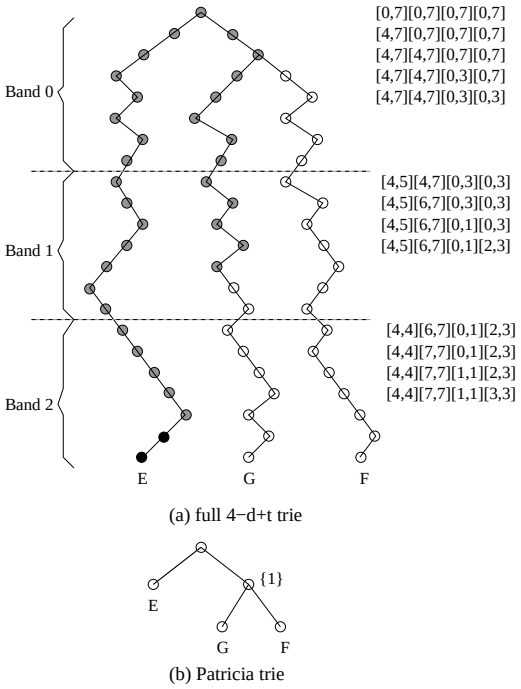


Figure 3: Example of three rectangles E, F, and G with a query rectangle W. The number of data bits B = 3.



(a) full 4−d+t trie

(b) Patricia trie

Figure 4: (a) a binary 7-d full trie for the 2-d data of Figure 3 with $m = 3$. The list of 8-tuples near the right hand side is the cover space NC^4 of each node on the trie path representing rectangle G. (b)The corresponding Patricia trie of the full trie in (a).

Theorem 3 *Given a Patricia trie T containing a set D of n k-d points and text, consider a random orthogonal range search with query rectangle W, assuming D and W satisfy the uniform probabilistic model. The expected orthogonal range search time*

$E\{Q(n, k+1)\} \leq \Sigma_{S \subset \{1, \cdots, k+1\}} (\prod_{p \notin S} |WC^p|)$
$(\gamma(\frac{1}{k+1} \log_2 n) n^{1 - \frac{s}{k+1}} + O(1)) + O(\log_2 n)$,

where $s = |S|$ and γ_u is a periodic function of u with period 1, small amplitude, and mean value

$$\gamma_0 = -\frac{(\frac{s}{k+1}+1)(1-2^{-s/(k+1)})}{(k+1)\log 2} \Gamma(\frac{s}{k+1} - 1)$$

$\Sigma_{\ell=0}^{k+1}(\delta_1\,\delta_2\,\cdots\,\delta_\ell)2^{-\ell(1-s/(k+1))}$ *with* $\delta_\ell = 1$, *if the* ℓ^{th} *attribute of the query is specified, and* $\delta_\ell = 2$ *if it is unspecified.*

Proof. $E\{Q(n,k+1)\} = E\{\Sigma_{t=1}^{2n-1}1_{[node_t \in GN \cup BN]}\}$. The probability that a node is black or grey is given as: $Pr(node_i \in GN \cup BN) \leq \prod_{p=1}^{k+1}(|NC^p| + |WC^p|)$. We have

$$E\{Q(n,k+1)\} \leq E\{\Sigma_{t=1}^{2n-1}\prod_{p=1}^{k+1}(|WC^p| + |NC^p|)\}$$
$$= \Sigma_{S\subseteq\{1,\cdots,k+1\}}(\prod_{p\notin S}|WC_p|)E\{\Sigma_{t=1}^{2n-1}\prod_{p\in S}|NC_t^p|\}$$
$$= \Sigma_{S\subset\{1,\cdots,k+1\}}(\prod_{p\notin S}|WC_p|)E\{\Sigma_{t=1}^{2n-1}\prod_{p\in S}|NC_t^p|\}$$
$$+\Sigma_{S=\{1,\cdots,k+1\}}(\prod_{p\notin S}|WC_p|)E\{\Sigma_{t=1}^{2n-1}\prod_{p=1}^{k+1}|NC_t^p|\}$$
$$= \Sigma_{S\subset\{1,\cdots,k+1\}}(\prod_{p\notin S}|WC_p|)E\{\Sigma_{t=1}^{2n-1}\prod_{p\in S}|NC_t^p|\}$$
$$+E\{\Sigma_{t=1}^{2n-1}\prod_{p=1}^{k+1}|NC_t^p|\}$$

Using Theorem P from [6] for our Patricia trie and Proposition 1 and Lemma 2, we obtain

$$E\{Q(n,k+1)\} \leq \Sigma_{S\subset 1,\cdots,k+1}(\prod_{p\notin S}|WC_p|)$$
$$(\gamma(\tfrac{1}{k+1}\log_2 n)n^{1-\frac{s}{k+1}} + O(1)) + O(\log_2 n). \quad \square$$

A similar approach yields a lower bound of

$$E\{Q(n,k+1)\} \geq \Sigma_{S\subset 1,\cdots,k+1}(\prod_{p\notin S}\tfrac{|WC_p|}{2})$$
$$(\gamma(\tfrac{1}{k+1}\log_2 n)n^{1-\frac{s}{k+1}} + O(1)) - C,$$

where C is constant value determined by k. Based on the lower and upper bound shown above, we can write the expected range search time as

$$E\{Q(n,k+1)\} = c_1\prod_{p=1}^{k+1}|WC^p|n +$$
$$c_2\Sigma_{S\subset 1,\cdots,k+1|,0<S<k+1}(\prod_{p\notin S}|WC^p|)\gamma(\tfrac{\log_2 n}{k+1})n^{1-\frac{s}{k+1}} + O(\log_2 n) \quad (1)$$

where c_1 and c_2 are constant values.

5 Experimental Results

Experimental validation of our approach was performed using uniform random distributed 2-d points and rectangles for n = 1000, 10000, 100000, and 300000. Each string assigned to points and rectangles was randomly selected from a set of more than 300000 place names in the Canadian geographical names database. The string maximum length is 40, B=31, and we tested the range search algorithm for $1 \leq m \leq 8$ for both points+text and rectangles+text. We chose $m = 3$ as it gave the minimum range search time. Experimental results of range search time of points+text and rectangles+text are shown in Tables 1 and 2, respectively. Each entry in tables is the average of approximately 300 random range searches.

The experimental range search time and the actual and theoretical number of nodes visited for percent of data in range=$[0\%, 10\%]$ for k-d points+text is shown in Figure 5, where $2 \leq k \leq 10$ and $n = 100000$. The programs were written in C++ (using SUN Forte Developer 7 C++ 5.4 compiler), and run under Solaris 8 on a SunFire V880 4-processor server. The expected theoretical number of nodes visited given by equation (1) was computed using Maple 9.

Table 1: Experimental results for 2-d points+text (m=3 and Ratio=Naive/Patricia).

% of data in range		*Average range search time(ms)*			
		n=1000	10000	100000	300000
$[0\%, 10\%]$	Patricia	0.021	0.173	0.979	3.105
	Naive	0.122	1.325	28.828	110.293
	Ratio	5.81	7.66	29.45	35.52
$[10\%, 20\%]$	Patricia	0.034	0.291	1.928	6.678
	Naive	0.268	2.710	46.763	165.566
	Ratio	7.88	9.31	24.25	24.79
$[20\%, 30\%]$	Patricia	0.055	0.477	3.024	11.301
	Naive	0.382	3.983	63.942	212.716
	Ratio	6.95	8.25	21.14	18.82

Table 2: Experimental results for 2-d rectangles+text (m=3 and Ratio=Naive/Patricia).

% of data in range		*Average range search time(ms)*			
		n=1000	10000	100000	300000
$[0\%, 10\%]$	Patricia	0.137	1.092	8.093	17.310
	Naive	0.224	2.252	53.544	179.863
	Ratio	1.64	2.06	6.62	10.39
$[10\%, 20\%]$	Patricia	0.213	1.387	11.516	30.406
	Naive	0.387	3.457	65.917	220.444
	Ratio	1.82	2.49	5.724	7.25
$[20\%, 30\%]$	Patricia	0.249	1.741	16.414	45.453
	Naive	0.554	4.846	82.752	271.507
	Ratio	2.23	2.78	5.04	5.97

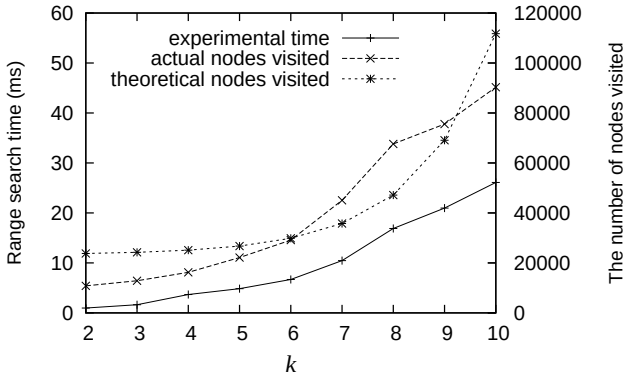


Figure 5: The experimental time (in milliseconds, n = 100000 and $2 \leq k \leq 10$, average of 300 test cases) and the theoretical (based on equation(1)) and actual (based on the average of 300 random range searches) number of nodes visited for percent of data in range = $[0\%, 10\%]$ for k-d points+text (m=3).

6 Conclusions and Future work

A range search algorithm for 2-d points+text and 2-d rectangles+text tries was presented, theoretically analyzed and experimentally compared to the naive method of range search.

Results indicate that as n increases, the range search time of our k-d+t tries improves relative to naive search. Our expected time analysis of range search for k-d points+text compares well our experimental results. In future work, we will consider how to assign multiple text strings to k-d points and rectangles such that efficient range search is supported.

7 Acknowledgement

The Natural Sciences and Engineering Research Council (NSERC) of Canada and the UNB Faculty of Computer Science provided financial support for the research reported here. We are grateful for their support.

References

[1] R. Baeza-Yates and G. Gonnet. Fast text searching for regular expressions or automaton searching on tries. *Jounal of the ACM*, 43(6):915–936, 1996.

[2] 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.

[3] P. Chanzy, L. Devroye, and C.Zamora-Cura. Analysis of range search for random k-d trees. *Acta Informatica*, 37(4/5):355–383, 2001.

[4] P. Flajolet and C. Puech. Partial match retrieval on multidimensional data. *Journal of the Association for Computing Machinery*, 33(2):371–407, April 1986.

[5] E. Fredkin. Trie memory. *Communiations of the ACM*, 3:490–500, 1960.

[6] P. Kirschenhofer and H. Prodinger. Multidimensional digital searching-alternative data structures. *Random Structures and Algorithms*, 5(1):123–134, 1994.

[7] D. Knuth. *The Art of Computer Programming. Sorting and Searching*, volume 3, pages 492–512. Addison-Wesley, Reading, Mass., 2 edition, 1998.

[8] R. la Briandais. File searching using variable length keys. *Proc. Western Joint Computer Conference*, 15:295–298, 1959.

[9] C. Martinez, A. Panholzer, and H. Prodinger. Partial match queries in relaxed multidimensional search trees. *Algorithmica*, 29:181–204, 2001.

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

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

[12] H. Samet. *The Design and Analysis of Spatial Data Strutures*. Addison-Wesley, Reading, MA, 1990.

[13] H. Shang. *Trie Methods for Text and Spatial Data on Secondary Storage*. PhD thesis, McGill University, 1994.

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