On multi-level k-ranges for range search

Sean M. Falconer [*] Bradford G. Nickerson [†]

Abstract

We investigate an implementation of the multi-level k-range data structure. The ℓ-level k-range is compared to naive and R*tree search over N randomly generated k-d points. Results indicate that multi-level k-ranges are not competitive due to their (previously unreported) complexity. We show that storage is $S(N,k,\ell) = O(N^{1+2(k-1)/\ell})$ and when $N \neq a^{\ell}$ where a and ℓ are positive integers, $S(N,k) = \Theta(N^{1+2(k-1)/\log_2 N})$. Our results also indicate that the ℓ-level k-range requires $Q(N,k,\ell) = O((2\ell)^{(k-1)}(\log N + A))$ time for range search.

1 Introduction

Data structures to support efficient searching have been a fundamental research area of computer science for many years. The specific problem of searching we are concerned with is called "orthogonal range searching". We define orthogonal range search as follows:

Definition 1 For a data space R^k, where k = number of dimensions, *orthogonal range search* is defined as finding and reporting the set of points $X \in F$ ($A = |X|$, F = set of k dimensional points, $|F| = N$), such that for all $x_i \in X$, x_i intersects the query rectangle $W = \{[L_1,H_1],[L_2,H_2],\ldots,[L_k,H_k]\}$, where $[L_j,H_j]$ represents a range for dimension j of the query rectangle, and $L_j < H_j$.

We use P, S, and Q to denote the preprocessing time, storage space, and range search time, respectively.

Bentley and Maurer explored the worst-case complexity of range searching by introducing three theoretical data structures; k-ranges, multi-level k-ranges, and non-overlapping k-ranges [4]. Multi-level k-ranges have an intriguing range search time $Q(N) = O(\log N + A)$ with storage $S(N) = O(N^{1+\varepsilon})$, for any fixed value of k and $\varepsilon > 0$. We explored ℓ-level k-ranges experimentally, and were surprised to find the range search performance significantly worse than predicted by Bentley and Maurer's analysis.

[*]Faculty of Computer Science, University of New Brunswick, Fredericton, N.B., Canada, E3B 5A3. Email: g2a71@unb.ca

[†]Faculty of Computer Science, University of New Brunswick, Fredericton, N.B., Canada, E3B 5A3. Email: bgn@unb.ca

2 Data Structures

The following data structures are both static in the sense that they do not support dynamic operations of insertion and deletion.

2.1 One-level k-range

F is normalized by converting each coordinate into the integer space, denoted as $\overline{F}$. This is done by sorting each point $x \in F$ by its corresponding k dimensions. Let $x = (x_1, x_2, \ldots, x_k)$, then $\overline{x} = (\overline{x_1}, \overline{x_2}, \ldots, \overline{x_k})$ where $\overline{x_i}$ corresponds to the rank of x_i with respect to all other points sorted on the i-th coordinate. Therefore, all points are "normalized" by sorting all dimensions in ascending order and removing duplicates, this can be done in $O(kN\log N)$ with $O(N)$ space.

Definition 2 For all i,j,t with $1 \leq i \leq j \leq N$ and $1 \leq t \leq k$ let $F_{i,j}^{(t)} = \{x | x \in F, i \leq \overline{x_t} \leq j\}$.

A 1-range, $F_{i,j}^{(1)}$, is a linear array G, where each element consists of a set of points G_q and a pointer p_q $(1 \leq q \leq N)$. G_q is the set of all points in $\overline{F}$ with first coordinate equal to q, and p_q points to the next nonempty element in G.

For $k = 2$, a 2-range is obtained by storing each of the sets $F_{i,j}^{(2)}$ for $1 \leq i \leq j \leq N$ as a 1-range $F_{i,j}^{(1)}$ and by a 2-d array P of points, with each element $P_{i,j}$ $(1 \leq i \leq j \leq N)$ pointing to $F_{i,j}^{(1)}$. To carry out a range search, we first normalize $[L_1,H_1],[L_2,H_2]$ (by performing a binary search over F to obtain the ranks), and then search the 1-range $F_{\overline{L_2},\overline{H_2}}^{(2)}$ for points between $\overline{L_1}$ and $\overline{H_1}$. In general, we store $F_{i,j}^{(k)}$ as (k-1)-ranges.

2.2 Multi-level k-range

Let us first consider a 2-level 2-range. On the first level, we consider one "block" which contains $N^{1/2}$ "units", each containing $N^{1/2}$ points. Assume N is a perfect square. In the first level we store all $C(N^{1/2}+1, 2) = O(N)$ consecutive intervals of units, that is $O(N)$ 1-ranges [1]. We store each 1-range in an array sorted on the x-value, such that we have storage proportional to the number of points stored in the range. In the second level, we have $N^{1/2}$ blocks, each containing $N^{1/2}$ units. Within each block, we store all possible intervals of units as 1-ranges.

[1]$C(n,p)$ is the number of ways to choose p element subsets from a set of n elements.

To perform a range search on a 2-level 2-range we must choose a covering of the y-range from both the first and second level. This can be done by selecting at most one covering from level 1 and two from level 2 (see Figure 1 for example).

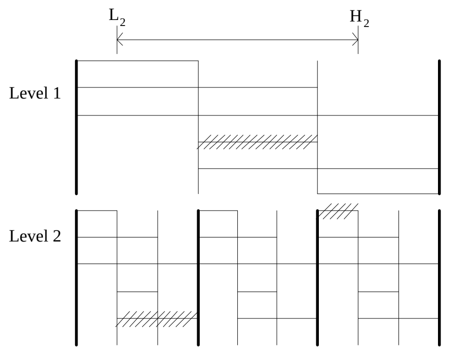

Figure 1: Searching a 2-level 2-range with N = 9 [4]. The bold vertical lines represent block boundaries and the regular vertical lines represent unit boundaries. Each horizontal line represents a 1-range; these 1-ranges do not extend across block boundaries. The crossed-hatched sections represent the ranges being searched; $F^{(2)}_{4,6}$ in level 1, $F^{(2)}_{2,3}$ and $F^{(2)}_{7,7}$ in level 2.

In general, for $\ell > 2$ and $k > 2$, on level i we store $N^{(i-1)/\ell}$ blocks, each containing $C(N^{1/\ell}, 2) = O(N^{2/\ell})$ units representing at most $N^{1-(i-1)/\ell}$ points [4]. Each ℓ-level k-range is inductively constructed out of ℓ-level (k-1)-ranges. To answer a query, we select an appropriate covering of the k-th coordinate for each level, and search the corresponding (k-1)-range (see Figure 2 for pseudocode).

Bentley and Maurer found that by choosing ℓ as a function of k and ε, for any fixed k and $\varepsilon > 0$, the generalized ℓ-level k-range has the following complexities:

$$P(N) = S(N) = O(N^{1+\varepsilon}), \text{ and}$$
$$Q(N) = O(\log N + A).$$

3 Algorithms

There are two main algorithms to consider for the multi-level k-range: construction and searching. The construction algorithm follows the description of the data structure from Bentley and Maurer's paper, except that they assumed $N = a^\ell$, where a and ℓ are both positive integers. In general, this is not true; we use $\lceil N^{1/\ell} \rceil$ and $M = \lceil N^{1/\ell} \rceil^\ell$ to construct the block and unit boundaries. The pseudocode for the construction algorithm is available in [7].

The range search algorithm recursively searches each ℓ-level k-range by finding, for each level, which units overlap the left point value and right point value for coordinate k until $k = 1$. When $k = 1$, the 1-range is searched with a binary search to find the starting element that is greater than or equal to the left point's rank. Then all points are reported between the starting element and the first element greater than the right point's rank. Finding the unit, $F^{(t)}_{i,j}$, that overlaps the query on coordinate t can be done in constant time (see indexing pseudocode in [7]). The algorithm below assumes that RANGESEARCH is a method of an ℓ-level k-range object representing a unit $F^{(t)}_{i,j}$.

```
SEARCHRANGE(L: left point, R: right point, k)
 1  $yi \leftarrow L.Ranks[k],\ yj \leftarrow R.Ranks[k]$
 2  if $k = 1$
 3     then $SearchOneRange(yi, yj)$, return
 4  $numOfBlocks \leftarrow 1,\ blockWidth \leftarrow M$
 5  $unitWidth \leftarrow \lceil blockWidth/levelRootOfN \rceil$
 6  for $i \leftarrow 0$ to $\ell - 1$
 7  do /* Calculate the complete covering for level i*/
 8      $l \leftarrow \lceil yi/unitWidth \rceil * unitWidth$
 9      $r \leftarrow \lfloor (yj+1)/unitWidth \rfloor * unitWidth$
10      if $blocks \leftarrow 1$ OR $rprev < lprev$
11         then if $l < r$
12                then $F^{(k)}_{l,r}$.SEARCHRANGE(L, R, k-1)
13         else  if $l < lprev$
14                then $F^{(k)}_{l,lprev}$.SEARCHRANGE(L, R, k-1)
15               if $rprev < r$
16                then $F^{(k)}_{rprev,r}$.SEARCHRANGE(L, R, k-1)
17      $lprev \leftarrow l,\ rprev \leftarrow r$
18      $numOfBlocks$ *= $levelRootOfN$
19      $blockwidth$ /= $levelRootOfN$
20      $unitWidth$ /= $levelRootOfN$
```

Figure 2: Search algorithm for an ℓ-level k-range.

4 Experimental Results

Experiments were run on a Sun Microsystem V880 with four 1.2 GHz UltraSPARC III processors, 16 GB of main memory, running Solaris 8. Times were obtained using the *timeval* struct, which reports seconds and microseconds.

4.1 Range Search Test

Table 1 lists our results for $k = 2$ to $k = 5$ for naive (linear bruteforce search), multi-level k-range, and R*tree searching [2, 6]. The first column of results represents the average search time, T_{NAIVE}, for the naive approach. The next two columns are T_{Kr}/T_{NAIVE} and T_{R*}/T_{NAIVE} where T_{Kr} and T_{R*} equal the average search time for the multi-level k-range and R*tree, respectively. For each row in Table 1, 300 random queries are processed. The average search time over the set of results is $T_{AVG} = \sum_{i=1}^{q} T_i$, where T_i = time for range search, $30 \leq q \leq 300$ with $A \in [0, \log N^{1/2})$. The same set of random k-d points and queries were used by each data structure. The query window size was kept small in order to avoid the search time being dominated by reporting. For the multi-level k-range, $\ell = \lceil \log_2 N \rceil$ was chosen to minimize $S(N, k)$ (see Lemma 2).

Table 1: Range searching test in milliseconds.

	N	$[0, \log N^{1/2})$		
k = 2	100	0.005	0.83	0.48
	1000	0.072	0.59	0.24
	10000	0.776	0.35	0.20
	100000	15.400	*f*	0.24
k = 3	100	0.005	2.83	0.54
	1000	0.059	1.32	0.42
	10000	0.745	*f*	0.31
	100000	17.035	*f*	0.35
k = 4	100	0.005	6.84	0.52
	1000	0.064	5497.56	0.36
	10000	0.796	*f*	0.35
	100000	17.819	*f*	0.38
k = 5	100	0.005	13.43	0.54
	1000	0.056	*f*	0.46
	10000	0.715	*f*	0.42
	100000	17.475	*f*	0.51

f - failed during construction

We can see that the multi-level k-range could not be constructed for half of the test cases. This is due to the excessive memory requirements of the ℓ-level k-range. We can also see that the R*tree outperformed the multi-level k-range in every test and even the naive search was faster for five cases. Section 5.2 explains why this poor performance occurs. We discovered that naive search always outperforms ℓ-level k-ranges for relatively low values of k [7].

5 Analysis

5.1 Storage Analysis

Bentley and Maurer calculated $S(N) = O(N^{1+\varepsilon})$ for the multi-level k-range for any fixed value of k and ε, but they did not show the exact calculation of ε. We need to know the value of ε in order to understand the relationship between ℓ, k, and N. In this section we further analyze the storage complexity in order to calculate the exact value of ε.

A recursion relation for the storage complexity can be defined as follows:

$$S(N,k,\ell) = \sum_{i=1}^{\ell} N^{(i-1)/\ell} C(N^{1/\ell},2) S(N^{1-(i-1)/\ell}, k-1),$$

where we sum the total storage over each level by multiplying the storage of the blocks by the storage for the units and recursing on the maximum number of points and $k-1$. By solving the recursion, we obtain $S(N,k,\ell) = O(N^{1+2(k-1)/\ell})$. Therefore, $\varepsilon = 2(k-1)/\ell$. We can see that as we increase ℓ, we decrease storage by a factor of $2/\ell$.

5.2 Search Time Analysis

We consider ℓ and k as factors in the query analysis and arrive at the following theorem.

Theorem 1 $Q(N,k,\ell) = O((2\ell)^{(k-1)}(\log N + A))$.

Proof. For each k we recursively search a maximum of 2ℓ ℓ-level (k-1)-ranges until $k = 1$; when $k = 1$, the ℓ-level 1-range is searched in $O(\log N + A)$ time. The following recursion can be defined:

$$Q(N,k,\ell) \leq 2\ell Q(N,k-1,\ell).$$

We solve the recursion as follows:

$$\begin{aligned}
&Q(N,k,\ell) \leq 2\ell Q(N,k-1,\ell)\\
&Q(N,k-1,\ell) \leq 2\ell Q(N,k-2,\ell)\\
&Q(N,k-2,\ell) \leq 2\ell(Q(N,k-3,\ell)\\
&\vdots\\
&Q(N,2,\ell) = O(\log N + A).
\end{aligned}$$

This gives $Q(N,k,\ell) \leq (2\ell)^{(k-1)} O(\log N + A)$, and $Q(N,k,\ell) = O((2\ell)^{(k-1)}(\log N + A))$. □

5.3 Level Analysis

As mentioned in section 5.1 and in [4, 7], increasing ℓ decreases the storage and preprocessing time for the multi-level k-range by a factor of $2/\ell$. Bentley and Maurer always assumed that $N = a^\ell$, where a and ℓ are positive integers, that is, N is a perfect root of ℓ. In general, this will rarely be the case. In order to construct the multi-level k-range when $N \neq a^\ell$, we use $M = \lceil N^{1/\ell} \rceil^\ell$, i.e. $N < a^\ell$. If $N \neq a^\ell$, then M could be very different from N. We must answer two important questions: what value of ℓ will minimize storage based on N? and does the storage complexity change if $N \neq a^\ell$?

Lemma 2 *$S(N,k,\ell)$ is minimal when $\ell = \lceil \log_2 N \rceil$.*

Proof. When $N \neq a^\ell$, where a and ℓ are positive integers, the value of $M = \lceil N^{1/\ell} \rceil^\ell$ is used by the ℓ-level k-range in order construct the block and unit boundaries.

For $\ell \geq \log_2 N$, and $\ell < \infty$, $\lceil N^{1/\ell} \rceil = 2$. This can be rewritten as $N^{1/\ell} \leq 2$, which reduces to

$$\frac{1}{\ell} \log N \leq \log 2,$$

by taking the logarithm of both sides. We see that increasing ℓ past $\log_2 N$ will not decrease storage because $\lceil N^{1/\ell} \rceil^\ell$ becomes equivalent to 2^ℓ and $S(N,k,\ell)$ becomes $O(2^{\ell^{1+2(k-1)/\ell}})$. Therefore, we can only reduce storage by increasing ℓ up to $\lceil \log_2 N \rceil$. □

Theorem 3 *$S(N,k) = \Theta(N^{1+2(k-1)/\log_2 N})$ for fixed $\ell = \lceil \log_2 N \rceil$, where $N \neq a^\ell$ (i.e. $N < a^\ell$), and a and ℓ are positive integers.*

The proof is omitted, as this theorem is a direct consequence of Lemma 2 and the result for $S(N,k,\ell)$.

This result implies that the storage can never be smaller than $\Theta(N^{1+2(k-1)/\log_2 N})$, since ℓ is carefully chosen, based on Lemma 2, to minimize the storage. In general, $N \neq a^\ell$, and the minimal storage is exponential in k. This result helps explain why the multi-level k-range implementation failed in many of the test cases in Table 1.

5.4 Level Experiment

Figure 3 displays a graph representing the theoretical and implementation values for the total number of units and points for two different test cases; $N = 100$ and $N = 1000$ with $k = 3$. We could not verify the storage for large N because of the ℓ-level k-range's large memory requirements. We can see that the value of both the theoretical and implementation storage is minimal when $\ell = \lceil \log_2 N \rceil$ for both cases. Once $\ell \geq \lceil \log_2 N \rceil$, the theoretical and implementation values continually grow, this is due to $\lceil N^{1/\ell} \rceil^\ell$ becoming equivalent to 2^ℓ. Also, the actual storage correlates to the theoretical by a constant factor for most values of ℓ.

The actual storage in bytes for the ℓ-level k-range is not shown in Figure 3, but for optimal ℓ, $N = 1000$, and $k = 3$, the data structure must allocate over 150 million bytes.

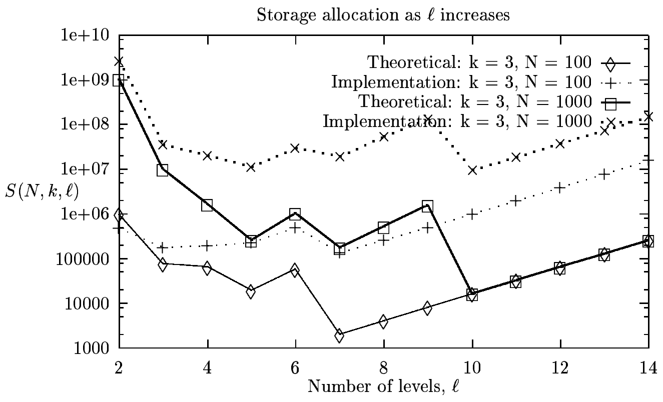

Figure 3: Experimental and theoretical storage allocation illustrating minimum storage for $\ell = \lceil \log_2 N \rceil$. The y-axis is the number of units $F_{i,j}^{(t)}$ for $t > 1$ plus the number of points for all $F_{i,j}^{(1)}$.

6 Conclusion

We fully implemented the multi-level k-range and compared this data structure to naive and R*tree searching. To our knowledge, this data structure has never been implemented. We showed that when ℓ and k are considered as factors for range searching, the ℓ-level k-range has search complexity of $Q(N, k, \ell) = O((2\ell)^{(k-1)}(\log N + A))$. We further analyzed the storage complexity of ℓ-level k-ranges showing that $S(N, k, \ell) = O(N^{1+2(k-1)/\ell})$ and for $N \neq a^\ell$, where a and ℓ are positive integers, $S(N, k) = \Theta(N^{1+2(k-1)/\log_2 N})$. We used these results to demonstrate why multi-level k-ranges are not competitive for range searching.

References

[1] P. K. Agarwar, Range Searching, *Handbook of discrete and computational geometry*, CRC Press Inc., Boca Raton, FL, 1997, pp. 575-581.

[2] N. Beckmann, H. P. Kriegal, R. Schneider, and B. Seeger, The R*Tree: An Efficient and Robust Access Method for Points and Rectangles, *Proc. ACM SIGMOD Intl. Symp. on the Management of Data*, 1990, pp. 322-331.

[3] J. L. Bentley, J. H. Friedman, Data Structures for Range Searching, *Computing Surveys*, Vol. 11, No. 4, 1979.

[4] J. L. Bentley, H. A. Maurer, Efficient Worst-Case Data Structures for Range Searching, *Acta Informatica*, Vol. 13, No. 2, 1980.

[5] B. Chazelle, Filtering Search: A New Approach To Query-Answering, *SIAM J. COMPUT.*, Vol. 15, No. 3, August 1986.

[6] K. K. Chu, Database Research Group: R*tree sourcecode, http://www.cse.cuhk.edu.hk/~kdd/program.html, Last Visit: April, 2004.

[7] S. M. Falconer, B. G. Nickerson, An investigation of multi-level k-ranges, *Technical Report TR04-163*, Faculty of Computer Science, U.N.B., April 2004, 25 pages.

[8] D. E. Knuth, *The Art of Computer Programming*, Vol. 2, *Seminumerical Algorithms*, 2^{nd} edition, Addison Wesley, Mass., USA, 1973.

[9] D. E. Knuth, *The Art of Computer Programming*, Vol. 3, *Sorting and Searching*, 2^{nd} edition, Addison Wesley, Mass., USA, 1973.

[10] F. F. Yao, Computational Geometry, *Handbook of Theoretical Computer Science: Volume A: Algorithms and Complexity*, Elsevier, Amsterdam, 1990, pp. 368-374.