Analyzing the Performance of Spatial Data Structures

Amalia Duch Brown

Universitat Politècnica de Catalunya

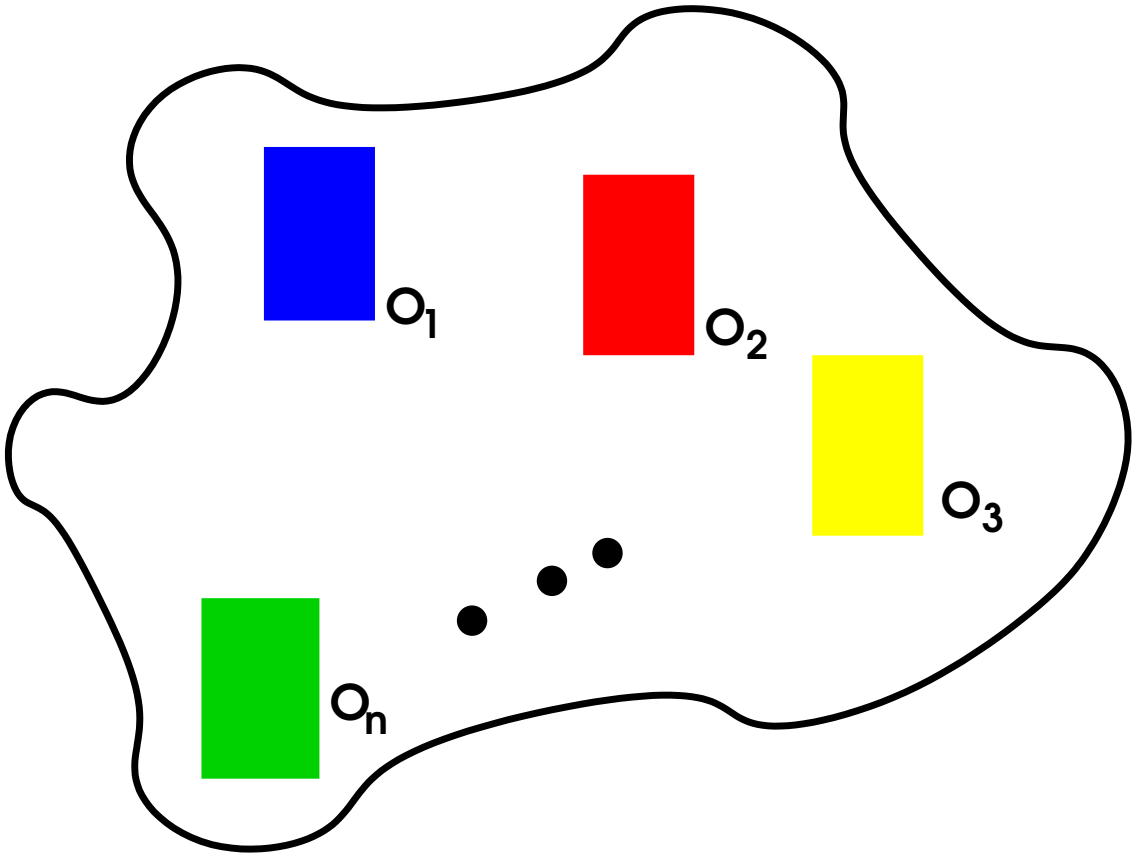
May, 2004

Abstract

In this talk I will give an overview of relaxed K-d trees, as an example of a hierarchical multidimensional data structure that supports a large variety of spatial operations.
In particular, I will introduce the data structure as well as some of the associative queries that it supports (such as partial match, orthogonal range or nearest neighbor queries) and use them to give some examples of the classical ways to address their expected performance analysis.

Motivation: Large Database of Objects

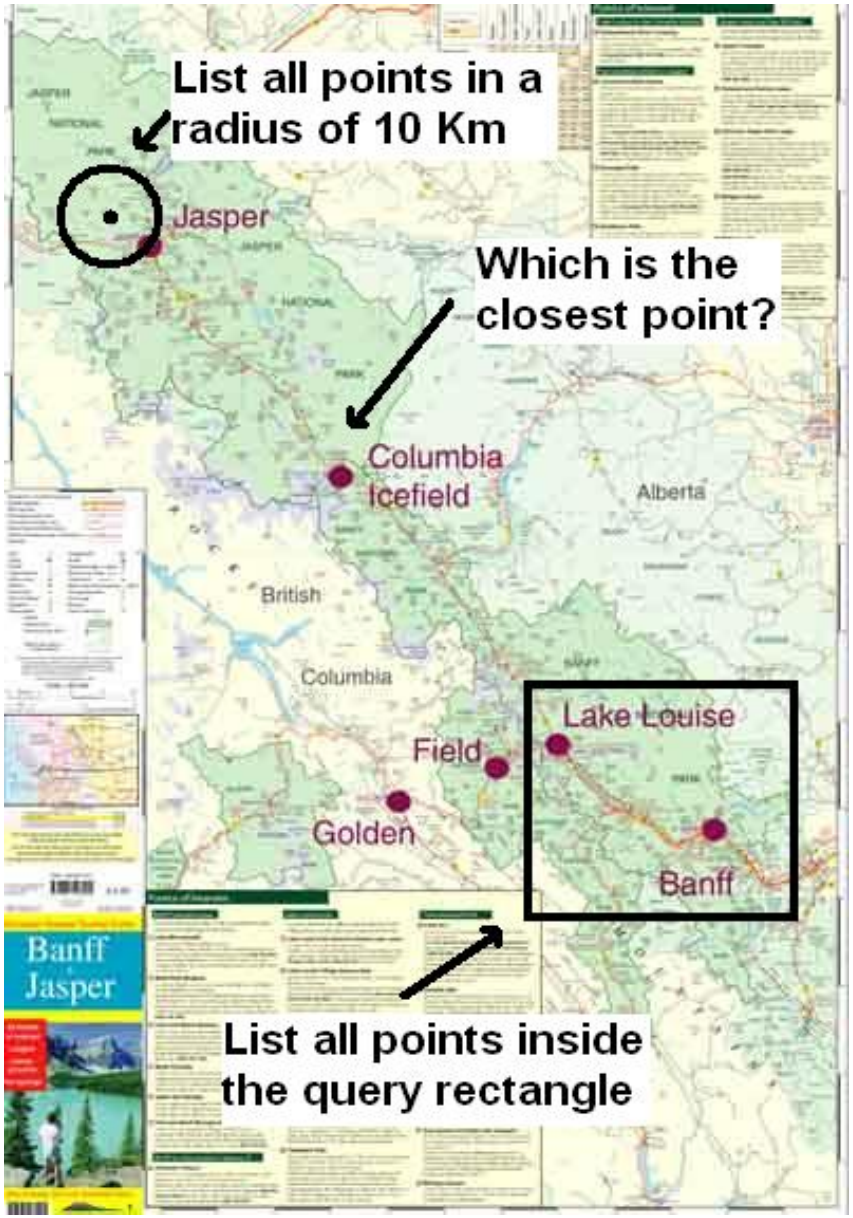
Images, Audio, Mpeg, Text documets, ...



Which object most closely matches Q?

(distance, similarity between two objects)

Motivation: Tourism in Banff



Multidimensional Data Structures

Multidimensional data structures are data management systems that support search and update operations in multidimensional data.

Point Multidimensional Data Structures

- Multidimensional data structures that store multidimensional points.
- Frequently present in applications.
- Used as indexes for accessing general multidimensional data structures.

Some Assumptions and Nomenclature

For simplicity in what follows,

- We will use the term multidimensional data structures to refer to point multidimensional data structures.
- Without loss of generality:
  - We will identify points in a K-dimensional space with their key $x = (x_0, x_1, \ldots, x_{K-1})$.
  - We will assume that each x_i belongs to $D_i = [0, 1]$ and hence the universe D is the hypercube $[0, 1]^K$.

Associative Retrieval

Multidimensional data structures must support:

- Usual insertions, deletions, (exact) queries
- Associative queries such as:

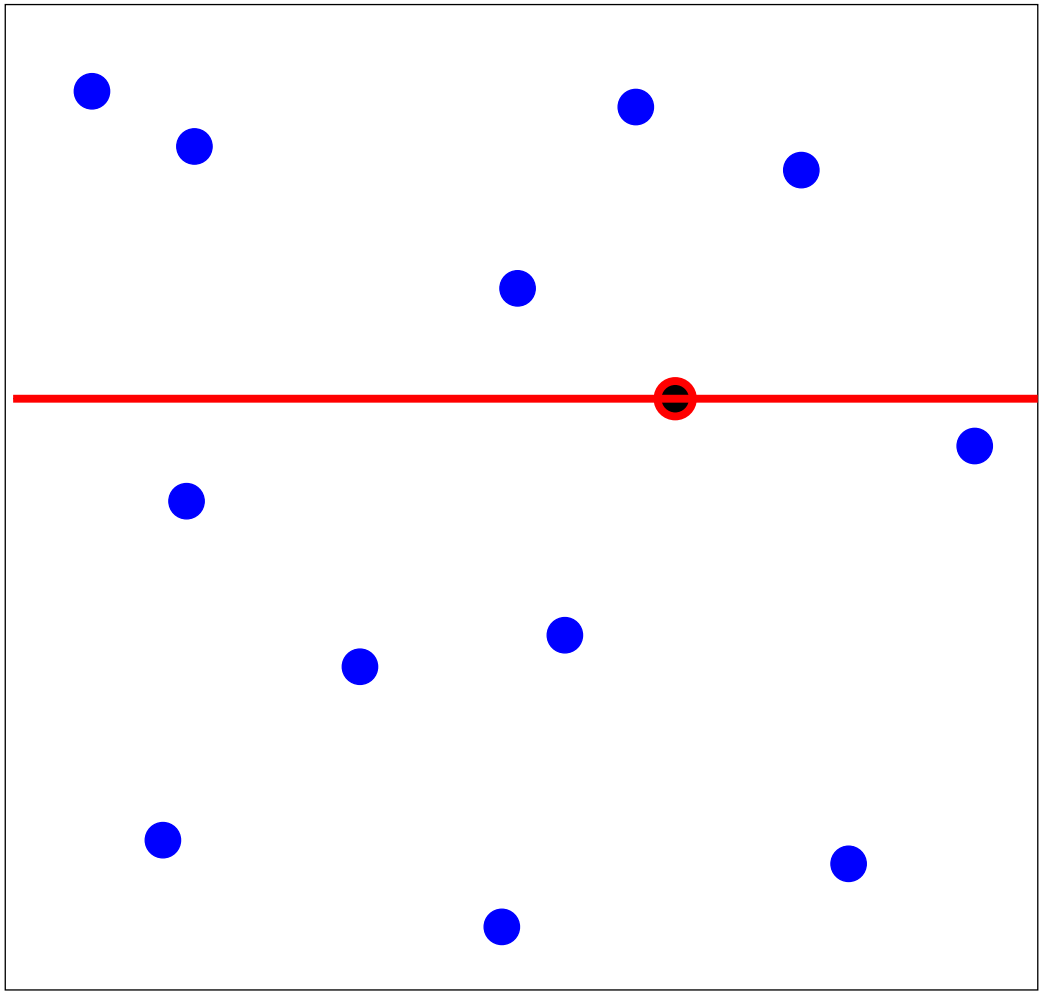
  Partial Match Queries: Find the data points that match some specified coordinates of a given query point q.

  Orthogonal Range Queries: Find the data points that fall within a given hyper rectangle Q (specified by K ranges).

  Nearest Neighbor Queries: Find the closest data point to some given query point q (under a predefined distance).

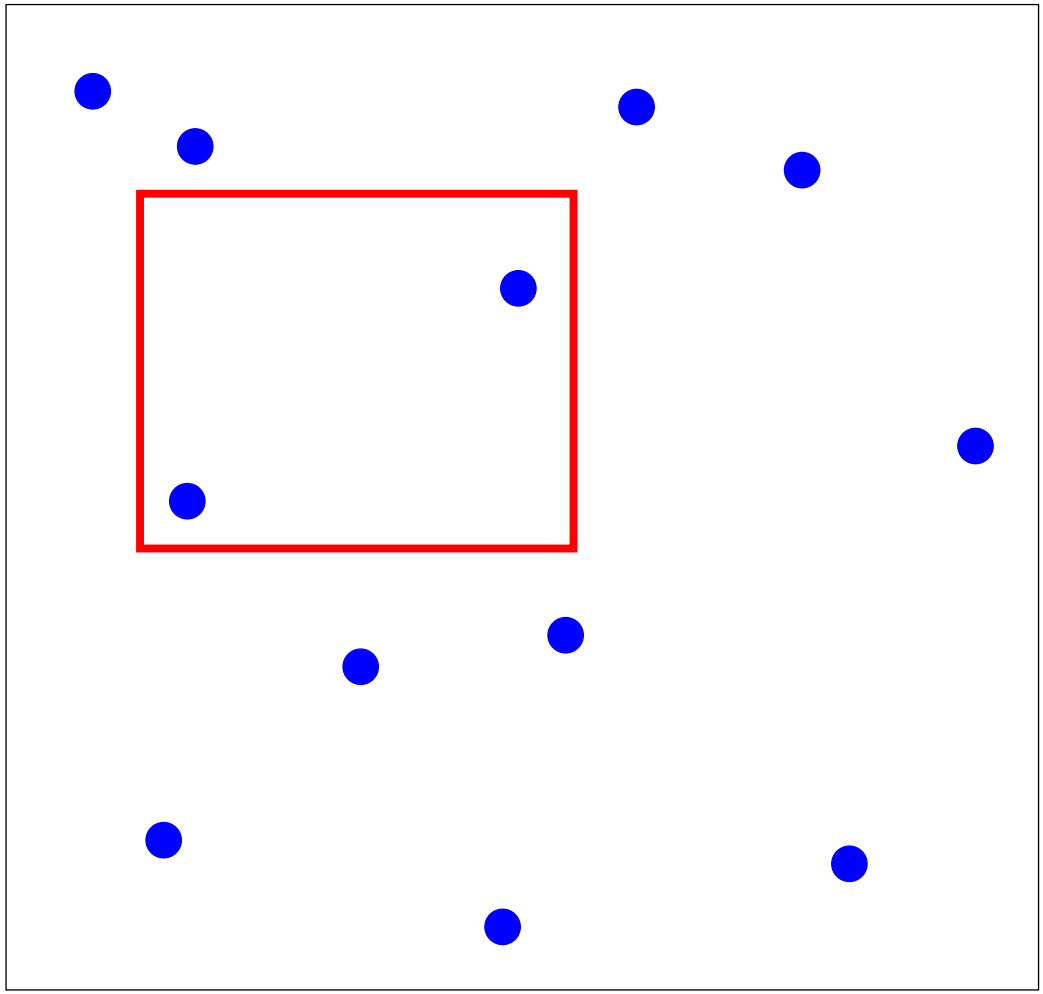
Example of Partial Match Queries

Query: $q = (*, q_2)$ or $q = (q_1, q_2)$ with specification pattern: 01



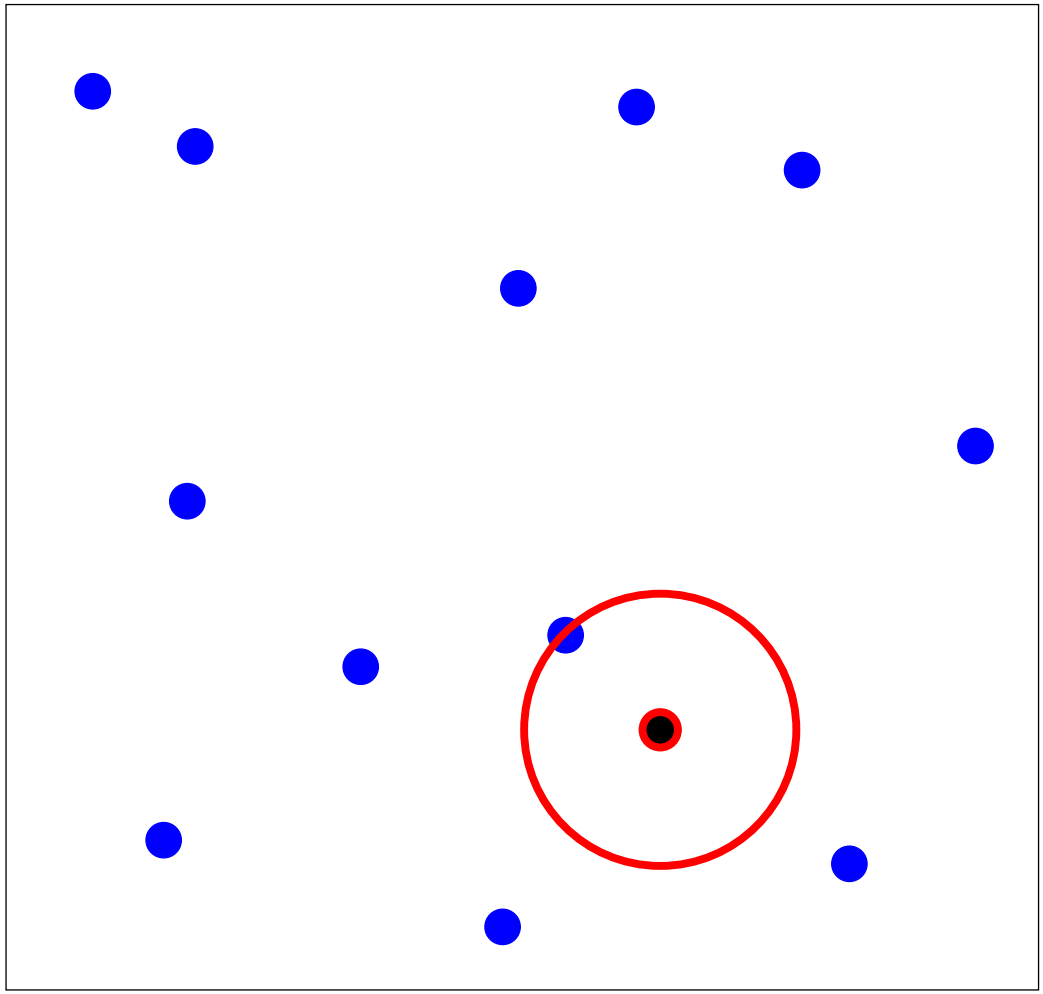
Example of Orthogonal Range Queries

Query: $Q = [\ell_1, u_1] \times [\ell_2, u_2]$



Example of Nearest Neighbor Queries

Query: $q = (q_1, q_2)$



What's the goal?

- Linear scanning of the collection is not efficient: we need to examine all or a substantial fraction of the n points to yield the answer.
- We would like to support insertion of new points and deletion of existing ones from the collection
- There exist specialized solutions for each type of associative query; however we would like to have data structures that support *all* opperations with good expected performance (less than linear and using $\Theta(nK)$ memory space.

Multidimensional Data Structures

Multidimensional data structures can be:

- Hierarchical: the data structure is built using relations among the points to be stored.
- Non-hierarchical: the data structure is built dividing the space in which points lye and associating each point to its corresponding slice.

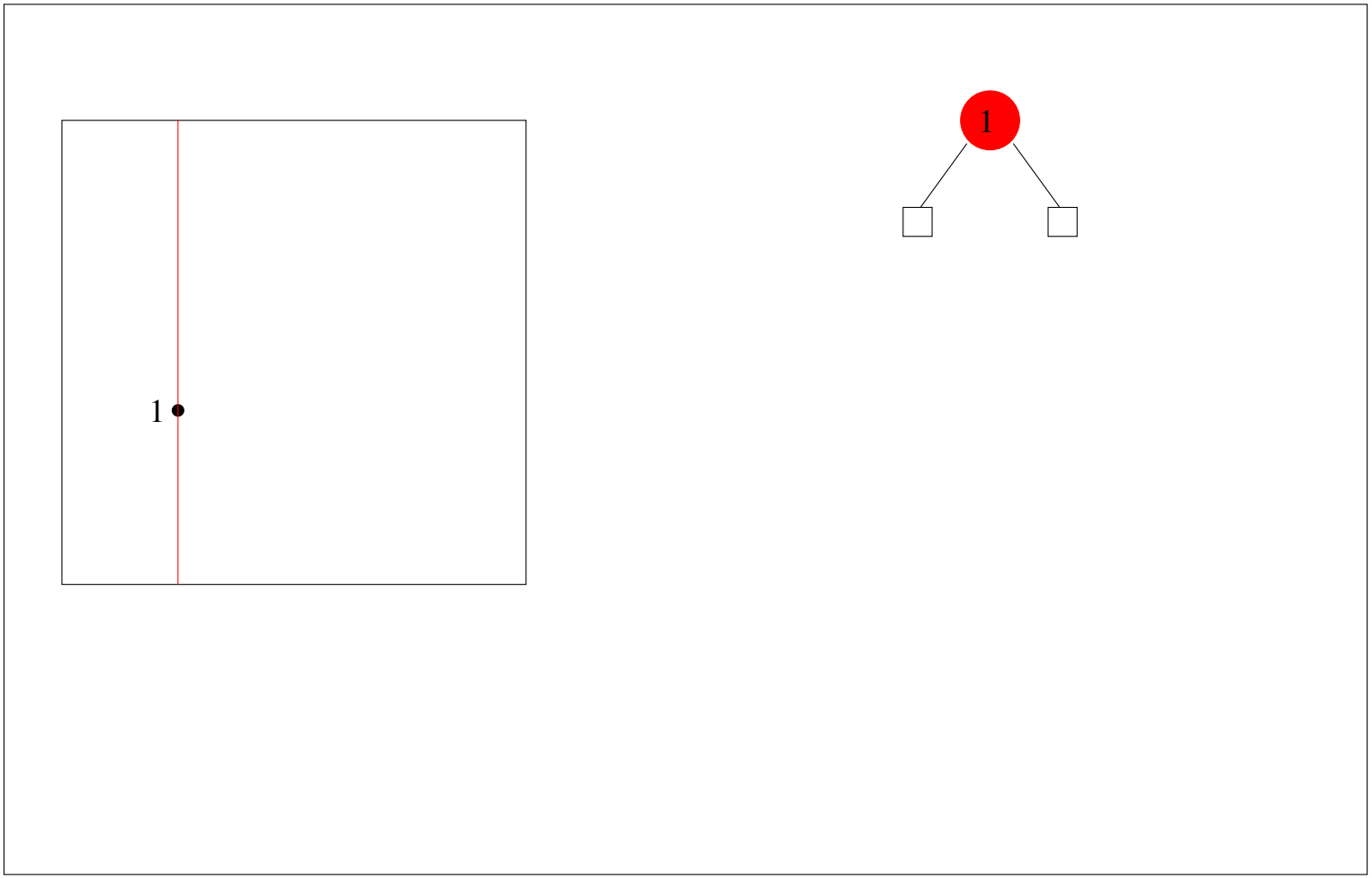
Hierarchical Multidimensional Data Structures

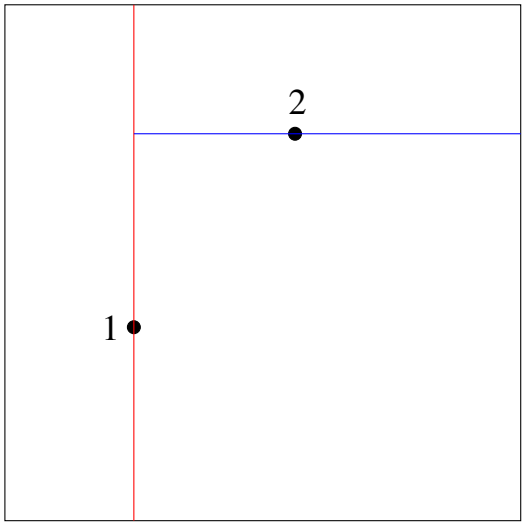
- Quad trees.
- Range trees.
- Standard K-d trees.
- Squarrish K-d trees.
- Relaxed K-d trees.
- ...

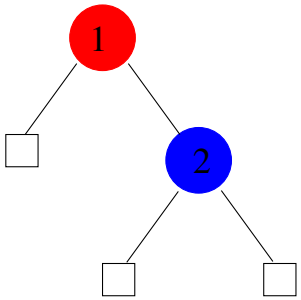
Definition of Relaxed K-d Trees

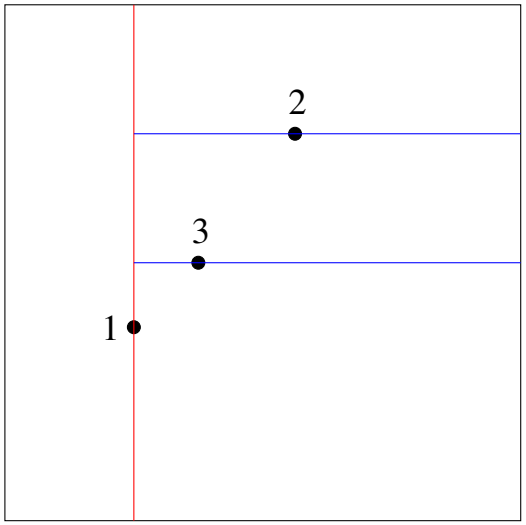
A relaxed K-d tree for a set of K-dimensional keys is a binary tree in which:

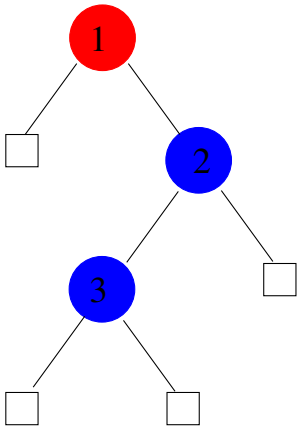
1. Each node contains a K-dimensional record and has associated an arbitrary discriminant $j \in \{0, 1, \ldots, K-1\}$.
2. For every node with key x and discriminant j, the following invariant is true: any record in the right subtree with key y satisfies $y_j < x_j$ and any record in the left subtree with key y satisfies $y_j \geq x_j$.

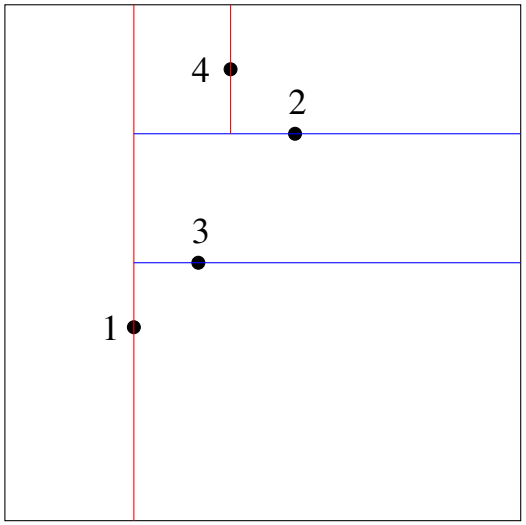

1
1

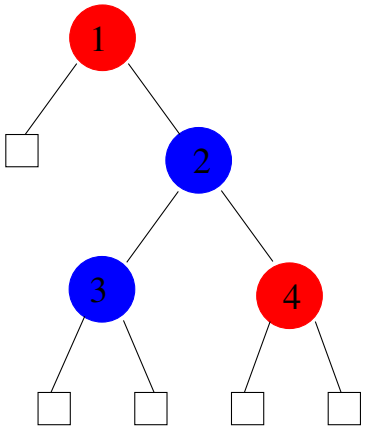

2
1

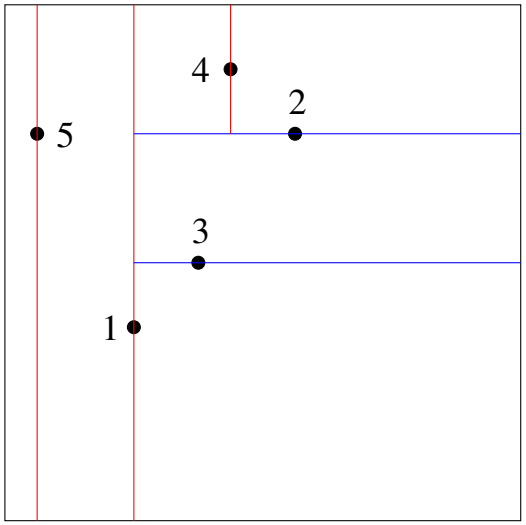

1
2

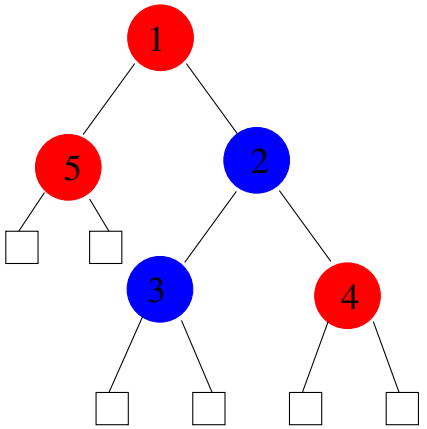

2
3
1


1
2
3


4
2
3
1


1
2
3
4


4
2
5
3
1


1
5
2
3
4

About Relaxed K-d Trees

- Relaxed K-d trees are a variant of K-d trees (Bentley, 1975)
- They were introduced by Duch, Estivill-castro and Martínez (1998) and subsequently analyzed by Martínez, Panholzer and Prodinger (2001), by Duch and Martínez (2002a, 2002b), and by Broutin, Dalal, Devroye and McLeish (2006).

Partial Match Algorithm in Relaxed K-d Trees

Partial match search in relaxed K-d trees works as follows:

- At each node of the tree we verify if it satisfies the query and we examine its discriminant.
- If the discriminant is specified in the query then the algorithm recursively follows in the appropriate subtree depending on the result of the comparison between the key and the query.
- Otherwise the algorithm recursively follows the two subtrees of the node.

Random Relaxed K-d Trees

Definition

We say that a relaxed K-d tree of size n is *random* if the $n!^K \cdot K^n$ possible configurations of input file and discriminant sequence are equiprobable.

The Random Model for the Analysis of Partial Match

The assumptions for the analysis are:

- The relaxed K-d tree is random.
- The query is random: it is a multidimensional point randomly generated from the same distribution as that of the points in the tree, with an arbitrary specification pattern.

The Recurrence of Partial Match Searches

Following the previous random model at each node:

- With probability $\frac{s}{K}$ the discriminant will be specified in the query and the algorithm will follow one of the subtrees.
- With probability $\frac{K-s}{K}$ the algorithm will follow the two subtrees.
- Hence, the cost $M(T)$ of a Partial Match Search in a relaxed K-d tree T of size n with left subtree L of size ℓ and right subtree R is:

$$M(T \mid |L| = \ell) = 1 + \frac{s}{K}\left(\frac{\ell+1}{n+1}M(L) + \frac{n-\ell}{n+1}M(R)\right) + \frac{K-s}{K}(M(L) + M(R)).$$

The Expected Cost of Partial Match

Theorem

The expected cost M_n (measured as the number of comparisons) of a partial match query with s out of K attributes specified in a random relaxed K-d tree of size n is

$$M_n = \beta n^{\alpha} + \mathcal{O}(1), \text{ where}$$

$$\begin{aligned} \alpha &= \alpha(s/K) = 1 - \frac{s}{K} + \phi(s/K) \\ \beta &= \beta(s/K) = \frac{\Gamma(2\alpha+1)}{(1-s/K)(\alpha+1)\Gamma^3(\alpha+1)} \end{aligned}$$

with $\phi(x) = \sqrt{9-8x}/2 + x - 3/2$ and $\Gamma(x)$ the Euler's Gamma function.

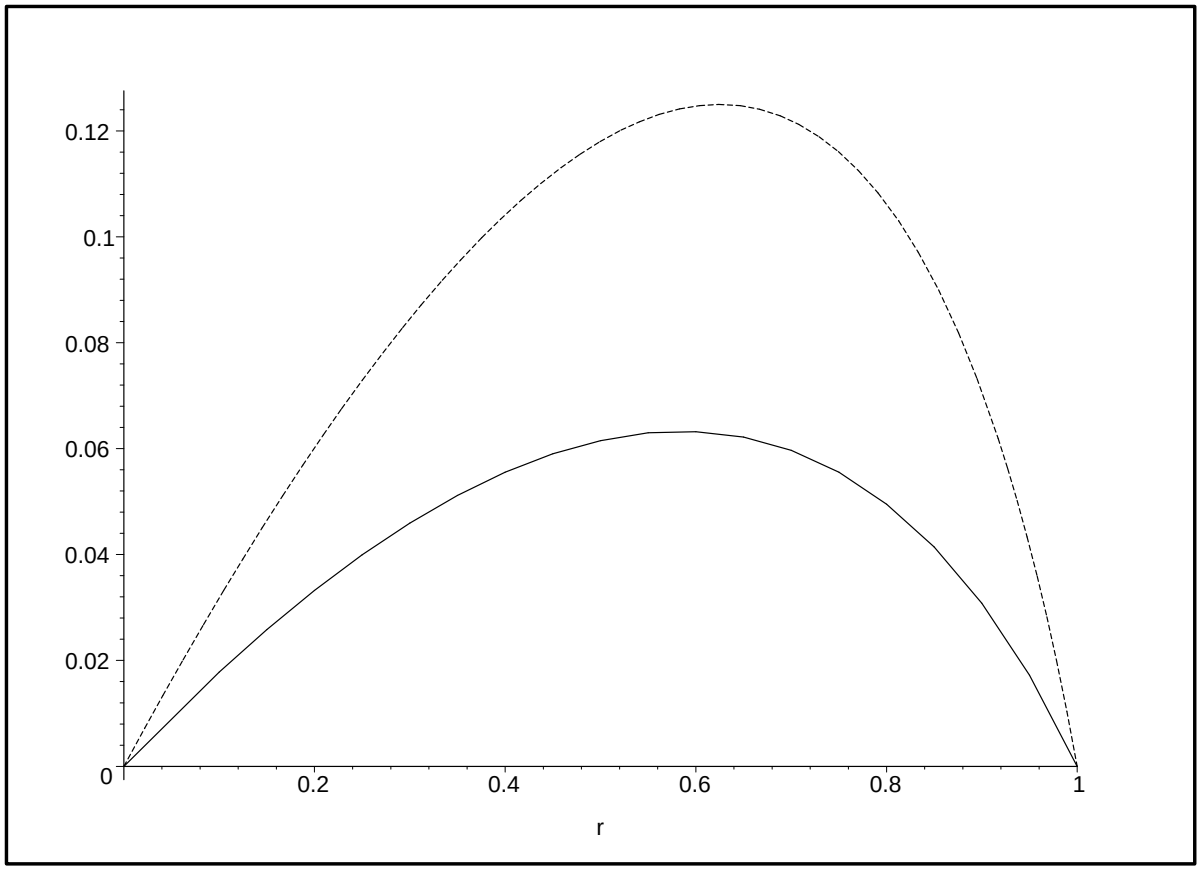
Solving the Recurrence of Partial Match Searches

In order to get the cost of partial match searches we follow the next steps:

- Take averages for all possible values of ℓ in the cost equation.
- Simplify by taking symmetries in the resulting recurrence.
- Translate the recurrence into a hypergeometric differential equation on the corresponding generating function.
- Solve the differential equation and obtain the generating function of the average cost of partial match.
- Use transfer lemmas to extract the coefficients of the average cost of partial match.

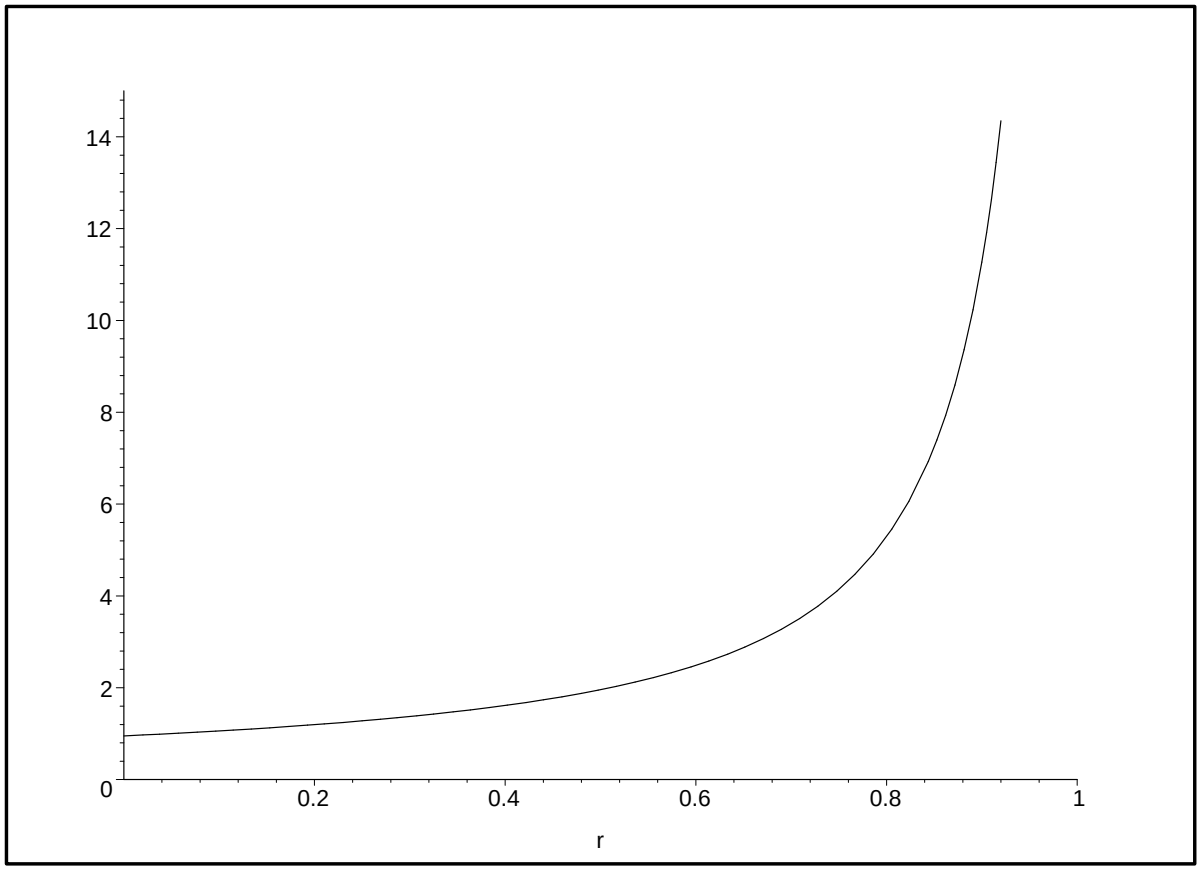
Exponent in the Average Cost of Partial Match Queries

Excess of the exponent α with respect to $1 - s/K$.



Constant in the Average Cost of Partial Match Queries

Plot of β.



Further Analysis

- Variance: C. Martínez, A. Panholzer, H. Prodinger (1998,2001)
- Limiting distribution: R. Neininger (2000)

Comparison with standard K-d trees

- The *alpha* coefficient for standard K-d trees is slightly smaller, but the analysis is more complicated since it involves the solution of a system of differential equations, one for each level of the tree (depending on the discriminant).
- The *beta* coefficient for standard K-d trees is dependent on the query's specification pattern.
- Analysis: Ph. Flajolet and C. Puech (1986), H. Hwang (2003).

Further Results

Analysis of Orthogonal Range Searches

- The expected cost of orthogonal range search in relaxed K-d trees can be obtained from the one of partial match search.
- The argument is valid for other hierarchical multidimensional data structures such as standard K-d trees, quad trees, standard and relaxed K- d tries and quad tries.
- Analysis: Ph. Chanzy, L. Devroye and C. Zamora-Cura (2001), A. Duch and C. Martínez (2002).

But. . .

Is there any problem?

Some Observations of Relaxed K-d trees

Relaxed K-d trees are simple data structures that handle a wide variety of queries. However,

- Their shape is very sensitive to the order in which items are inserted and this affects the performance of their algorithms.
- Their shape can be balanced using optimization techniques, knowing a priori all the records in the file. This process is either useless or expensive in dynamic applications.
- The good average-case performance of their algorithms assumes that file records are independently drawn from a continuous distribution; not always true in applications.
- A long sequence of interleaved insertions and deletions (even if random) may degrade the performance of the data structure.

How to Overcome these Problems?

- A successful approach to overcome these problems preserving the simplicity of K-d trees consists in the use of randomization.
- Randomization guarantees efficient expected performance that no longer depends on assumptions about the order of updates.
- Randomization has been successfully applied to the design of: Treaps by Aragon and Seidel, Skip Lists by Pugh and Randomized Binary Search Trees by Martínez and Roura.

Properties of Randomized K-d Trees

Theorem

If T is a random relaxed K-d tree that contains the set of keys X then insert(T, x) *returns the random relaxed K-d tree containing the set of keys $X \cup \{x\}$.*

Properties of Randomized K-d Trees

Theorem

If T is a random relaxed K-d tree that contains the set of keys X then insert(T, x) *returns the random relaxed K-d tree containing the set of keys $X \cup \{x\}$.*

Theorem

If T is a random relaxed K-d tree that contains the set of keys X, then, delete(T, x) *produces a random relaxed K-d tree T' that contains the set of keys $X \backslash \{x\}$.*

Properties of Randomized K-d Trees

Corollary

The result of any arbitrary sequence of insertions and deletions, starting from an initially empty tree is always a random relaxed K-d tree (quad tree).

Copy-based updates (Broutin, Dalal, Devroye and McLeish 2006)

```
// inserts the tree z in the appropriate leaf of T
rkdt insert_std(rkdt T, rkdt z) {
  if (T == NULL) return z;
  else {
     int i = T -> discr;
     if (z -> key[i] < T -> key[i])
        T -> left = insert(T -> left, z);
     else
        T -> right = insert(T -> right, z);
     return T;
  }
}
```

Copy-based insertion (1)

```
rkdt insert_at_root(rkdt T, const Elem& x) {
  rkdt result = new node(x, random(0, K-1), random(0,1));
  int i = result -> discr;
  priority_queue<rkdt> Q;
  Q.push(T);
  while (!Q.empty()) {
    rkdt z = Q.pop(); if (z == NULL) continue;
    // insert one or both subtrees of z
    // back to Q
    result = insert_std(result, z);
  }
  return result;
}
```

Copy-based insertion (2)

```
...
if (z -> discr != i) {
        Q.push(z -> left);
        Q.push(z -> right);
        z -> left = z -> right = NULL;
} else {
    if (x[i] < z -> key[i]) {
        Q.push(z -> left);
        z -> left = NULL;
     } else {
        Q.push(z -> right);
        z -> right = NULL;
     }
}
...
```

The Expected Cost of Randomized Insertions and Deletions

Theorem

For any fixed dimension $K \geq 2$, the average cost of a randomized insertion or deletion in random relaxed K-d tree of size n using copy-based updates is

$$I_n \sim D_n = 2 \ln n + \Theta(1).$$

The Expected Cost of Randomized Insertions and Deletions

Theorem

For any fixed dimension $K \geq 2$, the average cost of a randomized insertion or deletion in random relaxed K-d tree of size n using copy-based updates is

$$I_n \sim D_n = 2 \ln n + \Theta(1).$$

The "reconstruction" phase has constant cost on the average!

Work in this Area

- Data structures to dessign and analyze.
- Algorithms to dessign and analyze.
- Associative Queries.
- ...