Cost Model of the R-Tree

Yufei Tao

ITEE
University of Queensland

In the previous lecture, we have learned about the R-tree and how it can be used to answer (orthogonal) range queries efficiently. Today, we will introduce a cost model technique that allows us to estimate the I/O cost of range queries.

Cost Model

An (R-tree) cost model is a function $cost(q)$ which, upon given a range query q, returns how many nodes of an R-tree are expected to accessed by q.

There are mainly two reasons why cost models are useful.

1. In a database system, it is what the query optimizer relies on in order to decide whether to use the R-tree to answer a query. As a rule of thumb, the R-tree is used, only if the number of I/Os is at most 1/10 of what is required by sequential scan (i.e., reading the entire dataset in consecutive blocks).
2. Deriving a cost model demands deep understanding of the underlying structure (i.e., the R-tree in our case), and thus, reveals considerable insight into the characteristics of the structure.

Cost Model Derivation: Overview

We will take a probabilistic approach—indeed, as will be clear later, the core of analysis is to resolve several probability questions regarding the intersection of different types of geometric objects.

We will consider that the data space has a unit length on each dimension, namely, each dimension has the range of $[0, 1]$.
Think: Is this a restriction?

Interval-Point Containment Probability (1D)

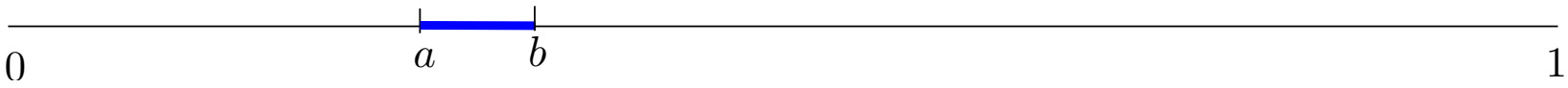
Let us start with a simple probability question.

Suppose that the dimensionality is $d = 1$. Fix an interval $[a, b]$ in the domain $[0, 1]$. Consider a point q that is uniformly distributed in $[0, 1]$. Question: what is the probability that q falls in $[a, b]$?

Interval-Point Containment Probability (1D)

Let us start with a simple probability question.

> Suppose that the dimensionality is $d = 1$. Fix an interval $[a, b]$ in the domain $[0, 1]$. Consider a point q that is uniformly distributed in $[0, 1]$. Question: what is the probability that q falls in $[a, b]$?



Answer: Obviously, $b - a$.

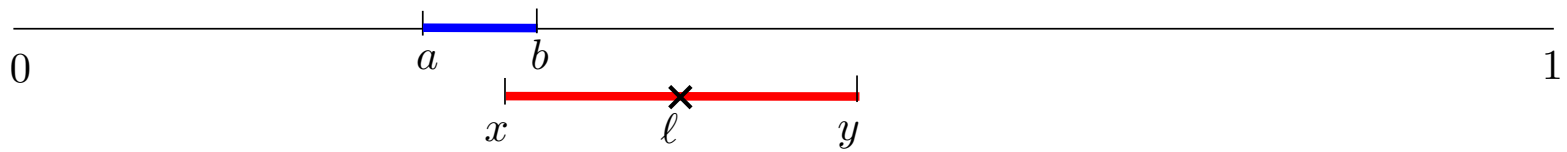
Interval-Interval Intersection Probability (1D)

Now we make the problem a bit more difficult.

Fix a 1D interval $r = [a, b]$ in the domain $[0, 1]$. Consider an interval $q = [x, y]$ that satisfies two conditions:

- The length of the interval $y - x$ equals ℓ.
- The interval is uniformly distributed on condition that it intersects $[0, 1]$. Namely, q can be any interval of length ℓ intersecting $[0, 1]$ with the same probability. Note that part of q is allowed to fall outside $[0, 1]$.

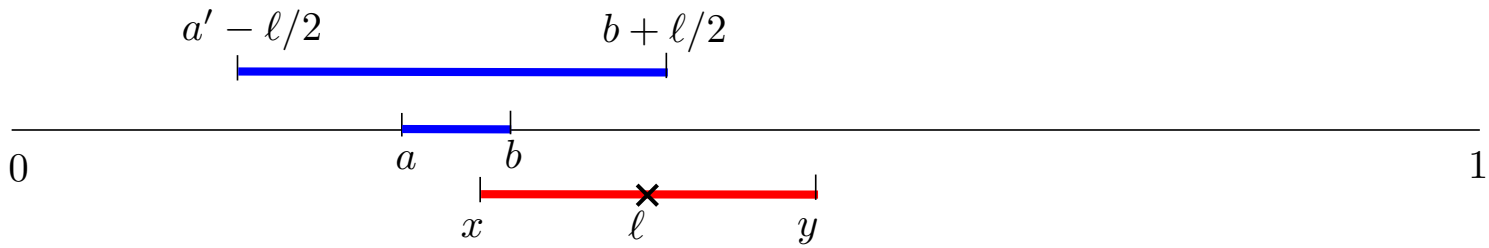
Question: what is the probability that q intersects r?



Interval-Interval Intersection Probability (1D)

To answer this question, we will need an expansion trick.

Observation 1: Define an interval r' that expands r by distance $\ell/2$ at each direction, namely, $r' = [a - \ell/2, b + \ell/2]$. Then, r intersects q if and only if r' contains the center of q.



Interval-Interval Intersection Probability (1D)

Observation 2: The center of q is uniformly distributed in $[-\ell/2, 1+\ell/2]$, namely, a range of length $1+\ell$.

Combining the two observations gives that, q intersects r with probability $\frac{b-a+\ell}{1+\ell}$. Note that when $\ell = 0$, this degenerates into $b - a$.

Rectangle-Rectangle Intersection Probability (2D)

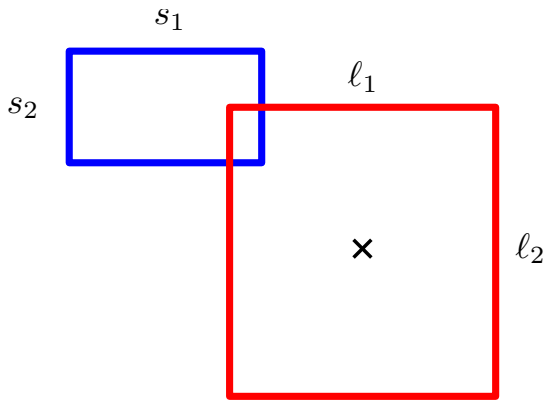
Let us now extend the problem to two dimensions:

Fix a 2D rectangle r whose two sides have lengths s_1 and s_2, respectively (recall that the data space has domain $[0, 1]$ on each dimension). Consider a rectangle q that satisfies two conditions:

- The two sides of q have lengths ℓ_1 and ℓ_2, respectively.
- q is uniformly distributed on condition that it intersects the data space. Namely, it is equally likely that q is any rectangle that (i) has the aforementioned side lengths, and (ii) intersects $[0, 1]^2$. Again, note that part of q is allowed to fall outside $[0, 1]$.

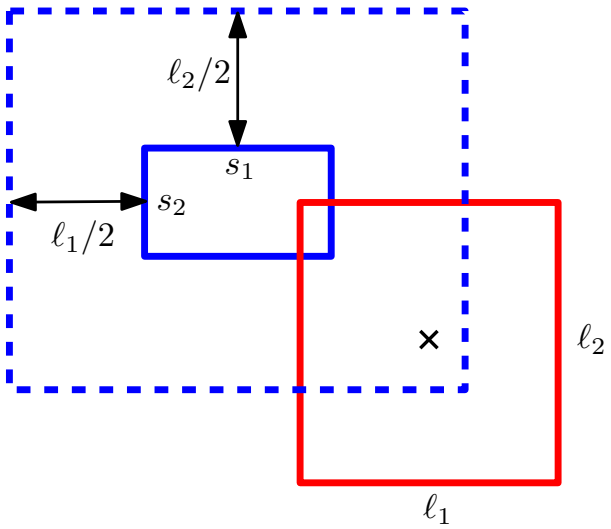
Question: what is the probability that q intersects r?

Rectangle-Rectangle Intersection Probability (2D)



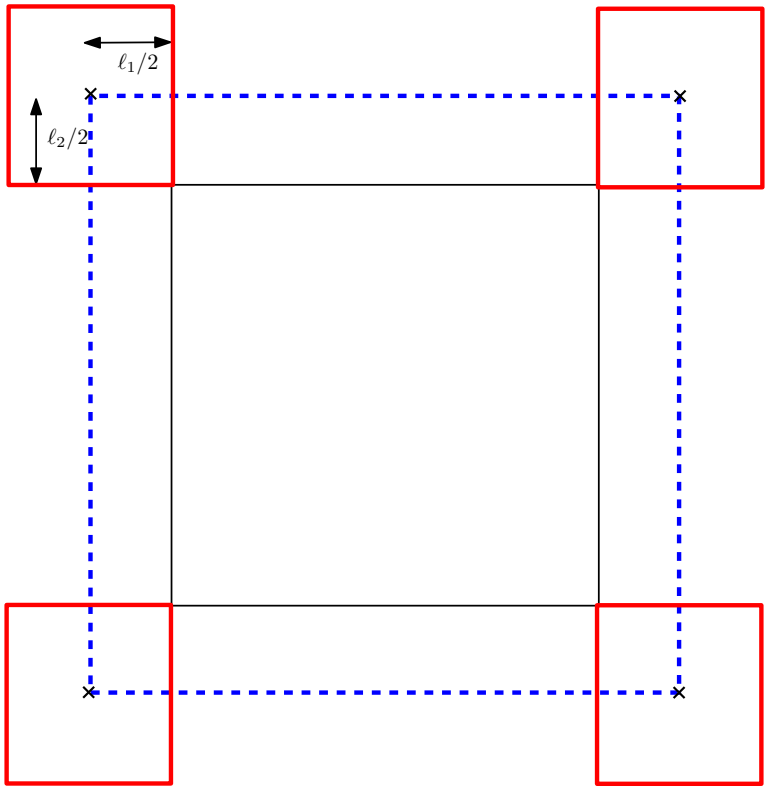
We will solve the problem by generalizing Observations 1 and 2.

Rectangle-Rectangle Intersection Probability (2D)



Observation 3: Define a rectangle r' that expands (i) the left and right sides of r horizontally by distance $\ell_1/2$, and (ii) the top and bottom sides of r vertically by distance $\ell_2/2$. Then, r intersects q if and only if r' contains the center of q.

Rectangle-Rectangle Intersection Probability (2D)



Observation 4: The center of q is uniformly distributed in a rectangle of side lengths $1 + \ell_1$ and $1 + \ell_2$, respectively.

Rectangle-Rectangle Intersection Probability (2D)

Combining Observations 3 and 4 shows that q intersects r with probability

$$\frac{(s_1 + \ell_1)(s_2 + \ell_2)}{(1 + \ell_1)(1 + \ell_2)}.$$

Access Probability of a Node

We are now ready to draw connections between our probability discussion so far and the performance an R-tree.

Focus on an arbitrary node u in a 2D R-tree. Let r be its MBR, with side lengths s_1 and s_2. Suppose that the window query q can be placed at any location with the same probability, on condition that q intersects the data space $[0, 1]^2$. If q has side lengths ℓ_1 and ℓ_2, from the earlier analysis, we know that the probability that u is accessed equals

$$\frac{(s_1 + \ell_1)(s_2 + \ell_2)}{(1 + \ell_1)(1 + \ell_2)}.$$

Why Minimizing the Perimeter?

Recall that a good R-tree should have "small" MBRs—particular, "small' in terms of both area and perimeter. Now we can give a theoretical explanation, using the formula of the previous slide.

An important fact is that, in practice, most queries are squares—namely, $\ell_1 = \ell_2 = \ell$. In this case, the node access probability formula becomes

$$\frac{(s_1 + \ell)(s_2 + \ell)}{(1 + \ell)^2} = \frac{s_1 s_2 + \ell(s_1 + s_2) + \ell^2}{(1 + \ell)^2}$$

Notice that $s_1 s_2$ is the area of the MBR, and $s_1 + s_2$ is half of the perimeter of the MBR. Therefore, lowering the area and perimeter reduces the access probability of the corresponding node!

Expected Query Cost of an R-Tree

Given a node u in a 2D R-tree, let $r(u)$ be its MBR, with side lengths $s_1(u)$ and $s_2(u)$. Suppose that the window query q is uniformly distributed, on condition that q intersects the data space $[0,1]^2$. If q has side lengths ℓ_1 and ℓ_2, the expected cost of the query equals:

$$\sum_u \frac{(s_1(u)+\ell_1)(s_2(u)+\ell_2)}{(1+\ell_1)(1+\ell_2)} = \frac{\sum_u (s_1(u)+\ell_1)(s_2(u)+\ell_2)}{(1+\ell_1)(1+\ell_2)}.$$

Expected Query Cost of an R-Tree

When $\ell_1 = \ell_2 = \ell$, the expected cost of the previous slide becomes

$$\frac{\sum_u (s_1(u) + \ell)(s_2(u) + \ell)}{(1+\ell)^2} = \frac{\sum_u (s_1(u)s_2(u) + \ell(s_1(u) + s_2(u)) + \ell^2)}{(1+\ell)^2}$$
$$= \frac{\mathrm{VolSum} + \ell \cdot \mathrm{PeriSum}/2 + \mathrm{NumNodes} \cdot \ell^2}{(1+\ell)^2}$$

where VolSum is the total volume (i.e., area in 2D) of the MBRs of all the nodes, and $\mathrm{PeriSum}$ is the total perimeter of those MBRs, and $\mathrm{NumNodes}$ is the total number of nodes in the R-tree.

Expected Query Cost of an R-Tree

The cost model on the previous slide reveals a useful observation. For queries with square search regions, namely, $\ell_1 = \ell_2 = \ell$, the query optimizer only needs to remember three values in order to estimate the expected query cost: VolSum, $\mathrm{PeriSum}$, and $\mathrm{NumNodes}$. Note that the value of ℓ is obtained from the query window.

These values can be easily maintained in updating the tree.

- Think: review the insertion algorithm and understand why this is true.

This is a typical cost model adopted for 2D R-trees.

Think:

1. Define the aspect ratio of a rectangle with side lengths ℓ_1, ℓ_2 as ℓ_1/ℓ_2. How to adapt the above analysis so that we can estimate the expected cost of all queries whose search rectangles have a specific aspect ratio ρ, by remembering again only three real values?
2. How to extend the above analysis to arbitrary dimensionality d?

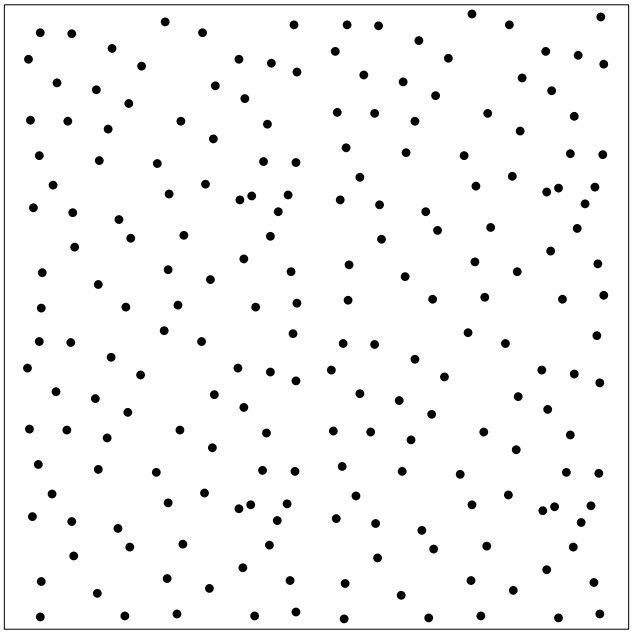
Almost all the multidimensional data structures are known to have worse performance as the dimensionality d increases. This phenomenon is known as the curse of the dimensionality.

Next, we will explain why this is true for the R-tree. For this purpose, we will have to look at "hard" datasets—it turns out that uniform data suffice. In particular, we will show that the leaf MBRs must have larger perimeters as d grows, even in an optimal R-tree.

The rest of the materials in this presentation will not be tested.

Uniform Distribution

Henceforth, we will consider that the dataset S consists of n points following the uniform distribution in the unit data space $[0, 1]^d$. For example, in 2D space ($d = 2$), each point $(x, y) \in S$ is obtained in such a way that x and y are independent, and they are uniformly distributed in the range $[0, 1]$. The value of n is very large.



Leaf Accesses

We will focus on the number of leaf nodes accessed. There are two reasons for this:

- The analysis can be extended to internal levels in a straightforward manner.
- The number of leaf nodes accessed typically dominates the query cost, such that minimizing the cost at the leaf level is the key to performance.

Leaf Accesses

From our earlier analysis, for $d = 2$, we know that the expected number of leaf nodes accessed equals

$$\frac{\mathrm{VolSum}_0 + \ell \cdot \mathrm{PeriSum}_0/2 + \mathrm{NumLeaves} \cdot \ell^2}{(1+\ell)^2}$$

where VolSum_0 is the total volume of the leaf MBRs, and $\mathrm{PeriSum}_0$ is the total perimeter of those MBRs, and $\mathrm{NumLeaves}$ is the number of leaf nodes.

Leaf Accesses

Let us make some crucial observations:

1. For very large n, $\mathrm{VolSum}_0 \geq 0.99$, because the MBRs of all the leaves should cover (nearly) the whole data space.
2. The number of leaves is between n/B and $n/(0.4B)$, where B is the number of points that can fit in a node.
3. The key term is therefore $\mathrm{PeriSum}_0/2$, this is what we will analyze next.

Leaf Accesses

Denote by $t = \mathrm{NumLeaves}$. Let $s_1[i]$ and $s_2[i]$ be the side lengths of the i-th leaf node ($1 \leq i \leq t$). We know:

$$\sum_{i=1}^{t}(s_1[i] \cdot s_2[i]) = \mathrm{VolSum}_0 \geq 0.99 \quad (1)$$

$$\sum_{i=1}^{t}(s_1[i] + s_2[i]) = \mathrm{PeriSum}_0/2 \quad (2)$$

Now we have a purely mathematical question: Given (1), what is the minimum value of (2)?

Leaf Accesses

Standard math techniques reveal that (2) is minimized when $s_j[i]$ is the same for all the $j \in [1, 2]$ and $i \in [1, t]$. In other words:

$$s_1[i] = s_2[i] = \sqrt{\mathrm{VolSum}_0/t}$$

Namely, in the optimal situation, each leaf MBR is a square with side length $\sqrt{\mathrm{VolSum}/t}$. We now have:

$$\begin{aligned} \mathrm{PeriSum}/2 &= 2t \cdot \sqrt{\mathrm{VolSum}_0/t} \\ &= 2\sqrt{t \cdot \mathrm{VolSum}_0} \\ &\geq 2\sqrt{0.99} \cdot \sqrt{t}. \end{aligned}$$

Leaf Accesses

Now we conclude that when n is large, the expected query cost is at least

$$\frac{0.99 + 2\sqrt{0.99} \cdot \sqrt{t} \cdot \ell + t \cdot \ell^2}{(1+\ell)^2}$$

which can be achieved if all the leaf MBRs are squares with side length

$$2\sqrt{0.99/t} \quad = \quad \Omega(\sqrt{B/n})$$

Using the fact that $t = \Theta(n/B)$.

Nodes Must Have Large Perimeters When Dimensionality is Large

Extending the above analysis to a general dimensionality d shows that, the expected query cost is minimized when each leaf MBR is a hyper-square (i.e., a d-dimensional rectangle with equal side length on all dimensions) with side length

$$\Omega\left((B/n)^{1/d}\right)$$

namely, increasing with d rapidly.

In other words, the perimeter sum of the leaf nodes must be large even in an optimal R-tree when d is large.