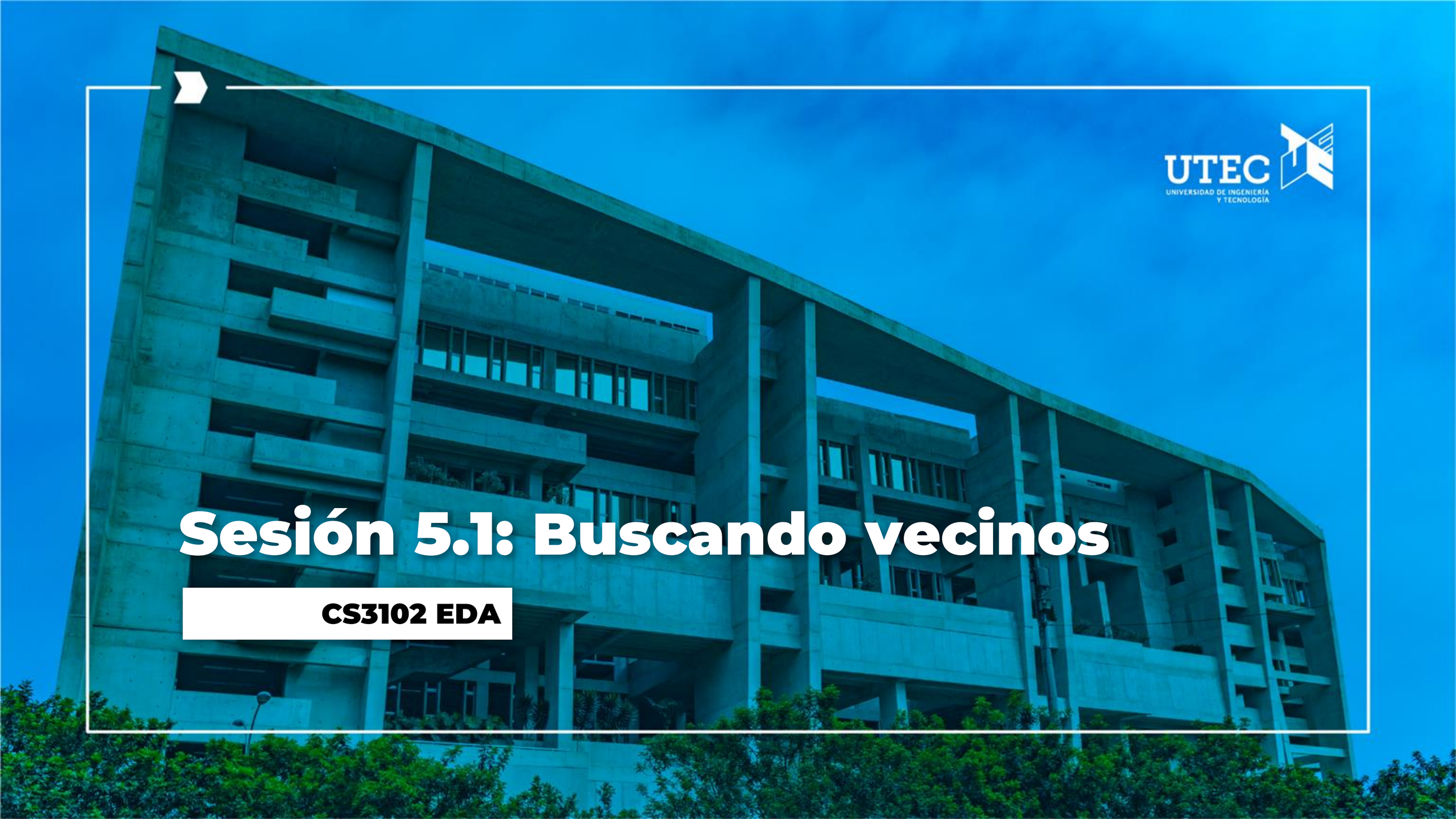


UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Sesión 5.1: Buscando vecinos

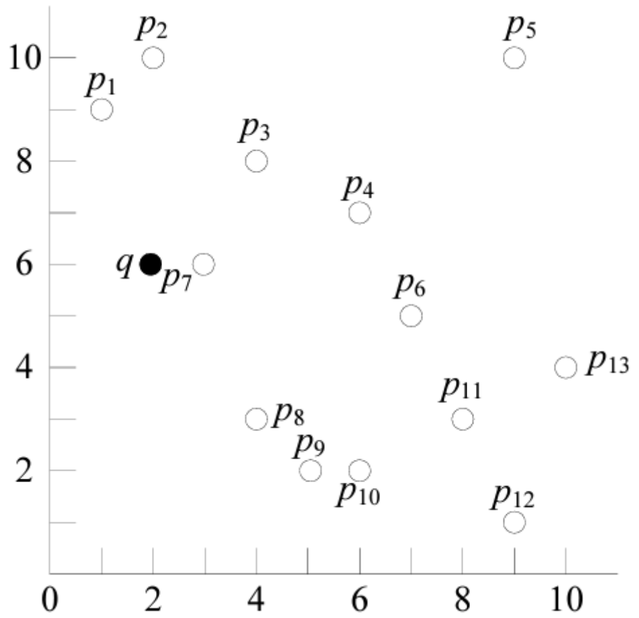
CS3102 EDA

Índice

1. Nearest Neighbor Search
2. R*Tree

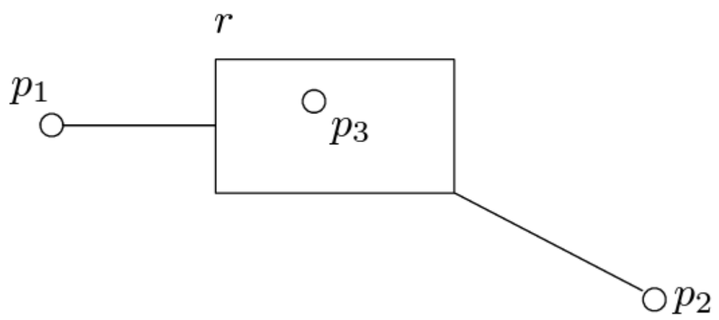
1. Nearest Neighbor Search

Nearest Neighbor *Search*

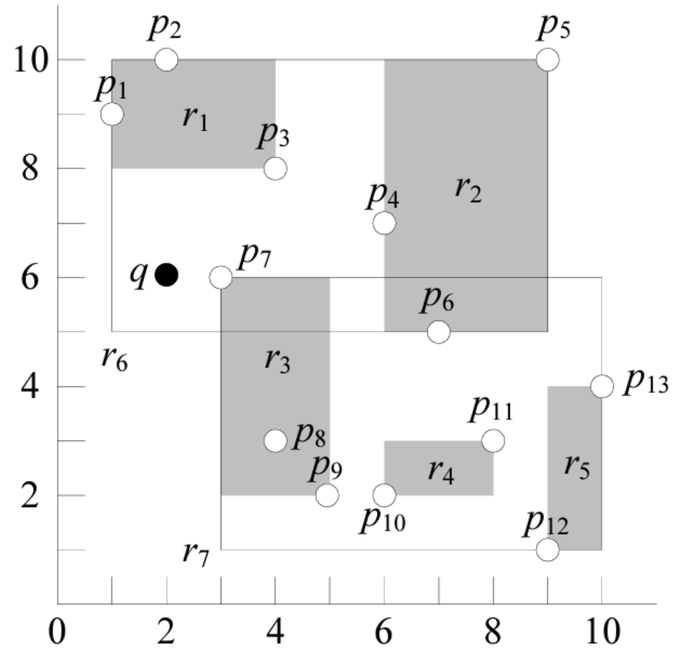


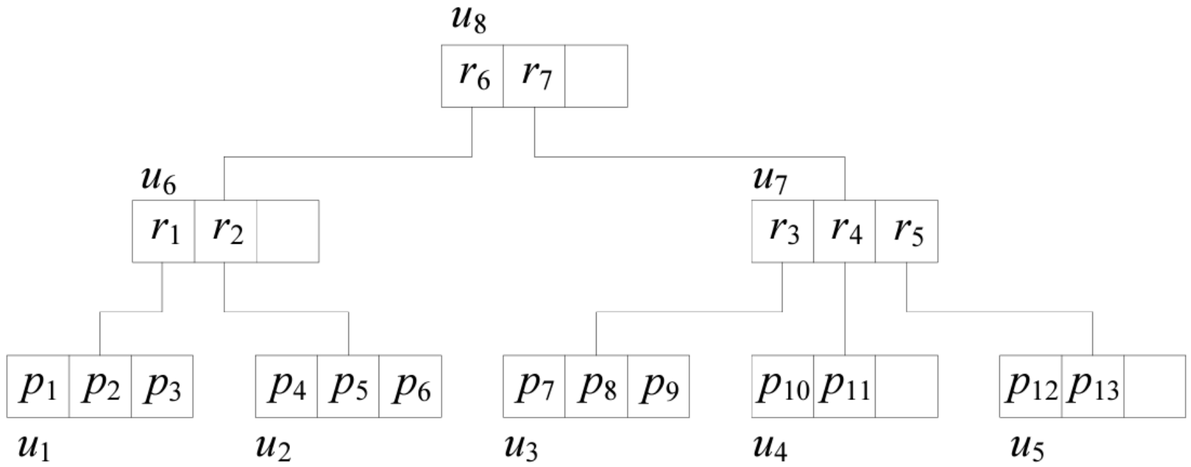
Min*dist*

Dado un punto q y un rectángulo paralelo al eje r, el mindist de q y r, denotado como $\text{mindist}(q, r)$, es igual a $\min_{p \in r} \|q, p\|$.



Branch-and-bound *(BaB)*



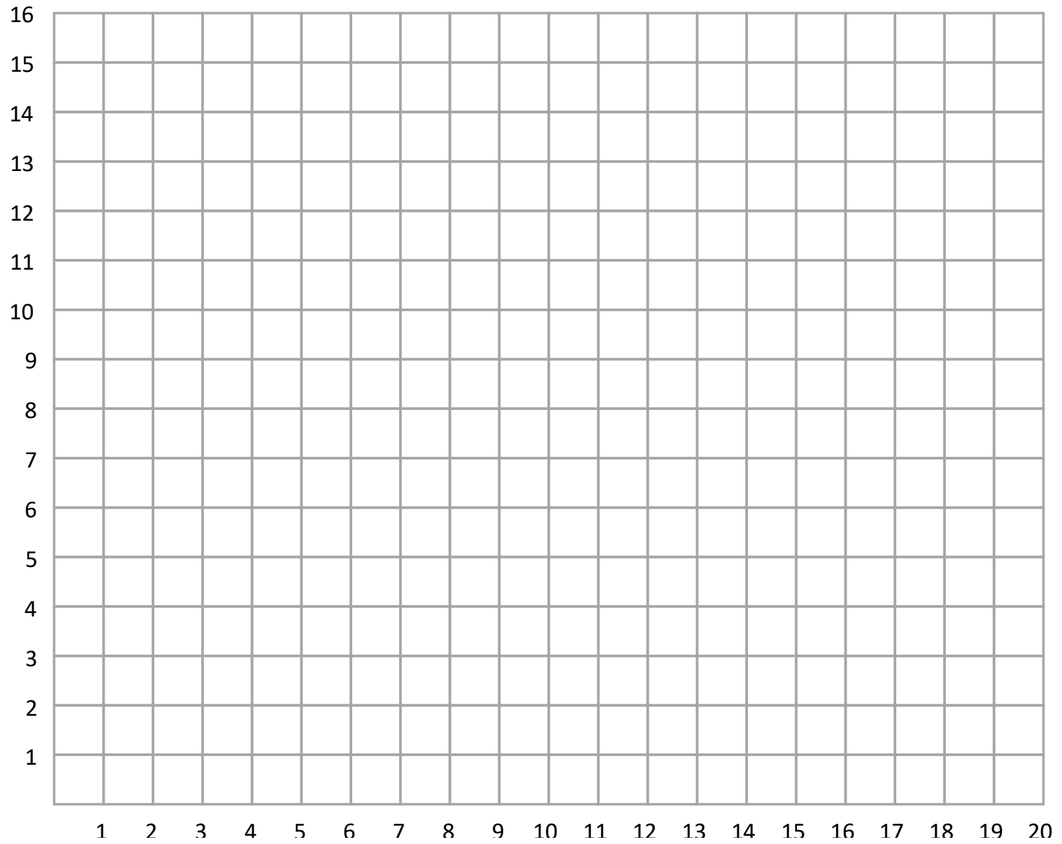


Branch-and-bound *(BaB)*

algorithm BaB(u, q)
/* u is the node being accessed, q is the query point;
p_{best} is a global variable that keeps the NN found so far;
the algorithm should be invoked by setting u to the root */

1. **if** u is a leaf node **then**
2. **if** the NN of q in u is closer to q than p_{best} **then**
3. p_{best} = the NN of q in u
4. **else**
5. sort the MBRs in u in ascending order of their mindists to q
/* let $r_1, ..., r_f$ be the sorted order */
6. **for** $i = 1$ to f
7. **if** $mindist(q, r_i) < \|q, p_{best}\|$ **then**
8. Bab(u_i, q)
/* u_i is child node of r_i */

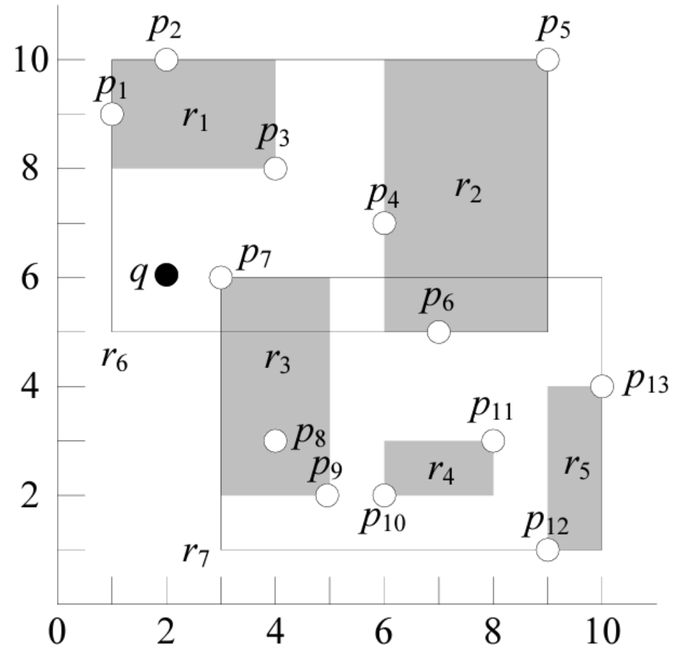
Branch-and-bound *(BaB)*

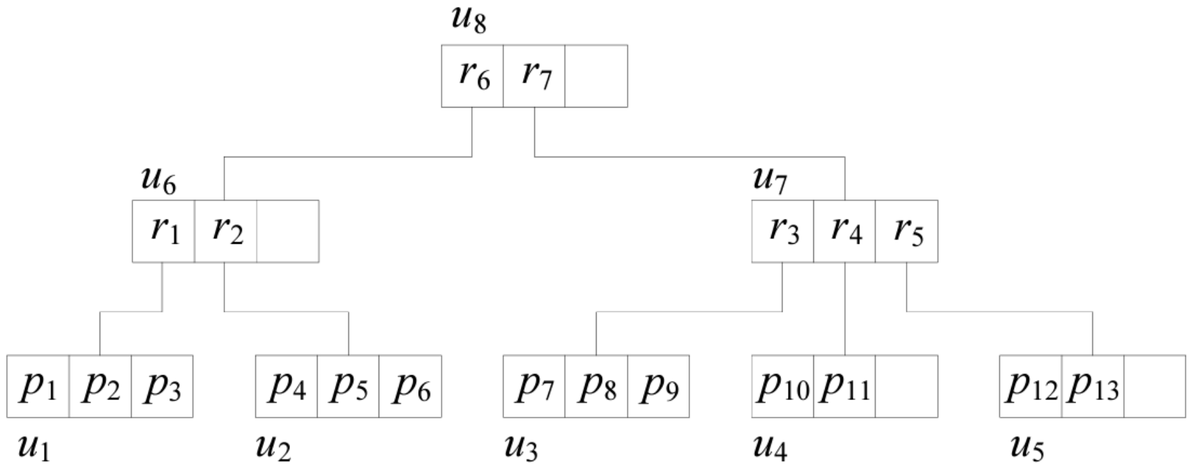


Branch-and-bound *(BaB)*

¿Cómo optimizarían este algoritmo?

Best First *(BF)*

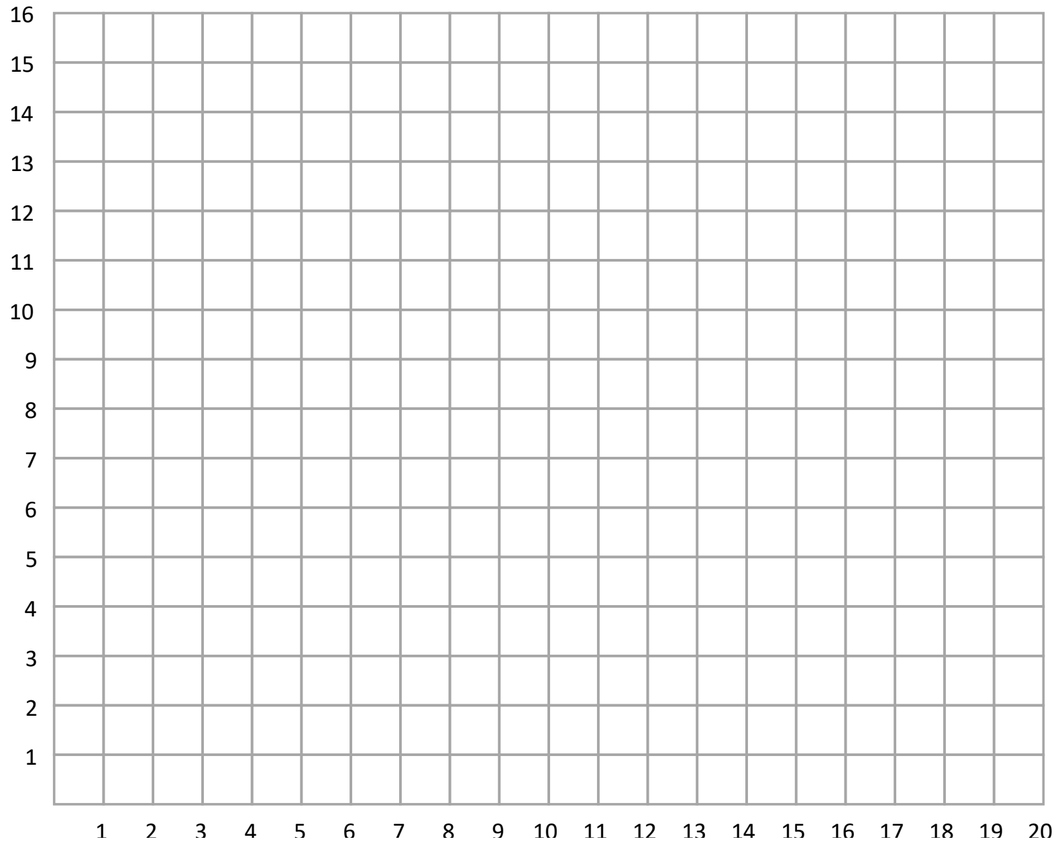




Best First *(BF)*

algorithm BF(q)
/* in the following H is a sorted list where each entry is an MBR
whose sorting key in H is its mindist to q;
p_{best} is a global variable that keeps the NN found so far. */
1. insert the MBR of the root in H
2. **while** $\|q, p_{best}\|$ is greater than the smallest mindist in H
/* if $p_{best} = \emptyset$, $\|q, p_{best}\| = \infty$ */
3. remove from H the MBR r with the smallest mindist
4. access the child node u of r
5. **if** u is an intermediate node **then**
6. insert all the MBRs in u into H
7. **else**
8. **if** the NN of q in u is closer to q than p_{best} **then**
9. p_{best} = the NN of q in u

Best First *(BF)*



Best First *(BF)*

¿Qué estructura de datos usaría para administrar H?

Best First *(BF)*

¿Como podría obtener los k vecinos más cercanos?

Best First *(BF)*

¿Este algoritmo funcionará con datos de más de 2 dimensiones?

2. R*-Tree

R*-Tree

Minimización del solapamiento entre MBB

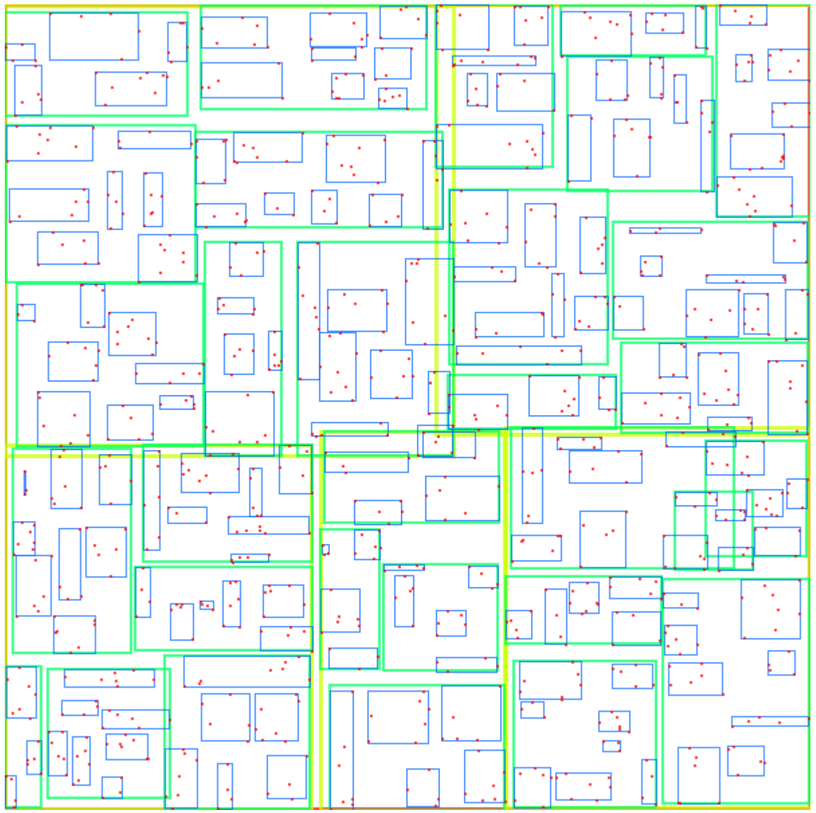
Minimización del área cubierta por cada MBB

Minimización de los márgenes (perímetros) de las MBB

Maximización de la utilización del almacenamiento

Beckmann, Norbert (1990) "The R*-tree: An efficient and robust access method for points and rectangles". Proceedings of the 1990 ACM SIGMOD international conference on Management of data (pp. 322-331).

R*-Tree



Hojas:

Minimizar solapamiento

Si hay empate, minimizar área/perímetro

Nodos internos:

Minimizar área/perímetro

Beckmann, Norbert (1990) "The R*-tree: An efficient and robust access method for points and rectangles". Proceedings of the 1990 ACM SIGMOD international conference on Management of data (pp. 322-331).

R*-Tree: *Inserción*

La inserción es **similar** a R-Tree (diferente regla de inserción, ver diapositiva anterior).
La principal diferencia es en **overflow**.

Overflow No ejecutamos directamente el algoritmo de ***split***. Tenemos dos paso:

Reinserción forzada

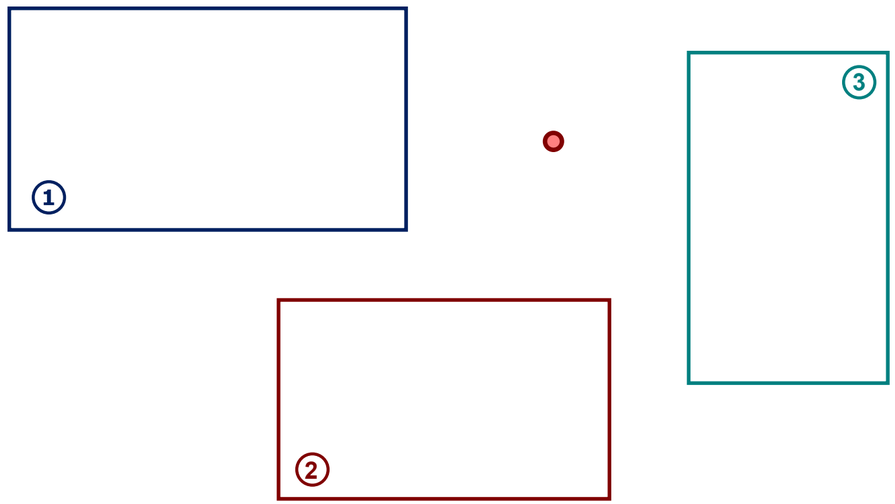
Por inserción, solo se ejecuta una vez por nivel del árbol.
En caso de resolver el *overflow*, pasamos al siguiente paso.

Split Tenemos dos pasos:

- Elegir el eje del *split*
- Elegir la partición (similar al método **brownie** de *split* de R-Tree)

R*-Tree: *Inserción*

Hojas: Minimizar el solapamiento
Si hay empate minimizamos área o perímetro

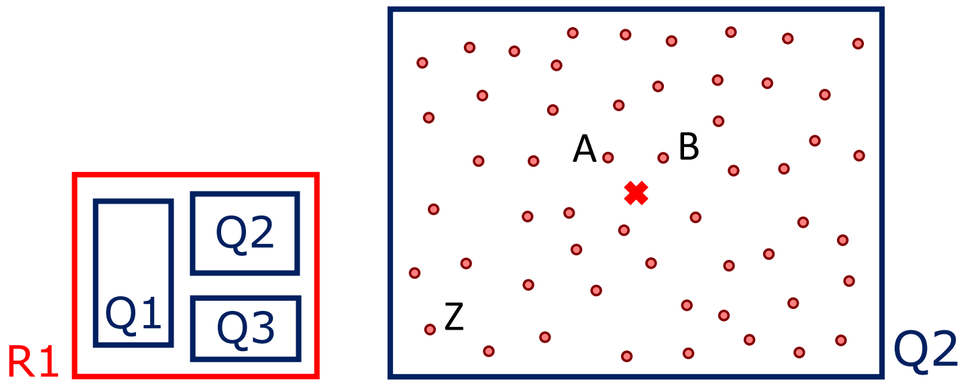


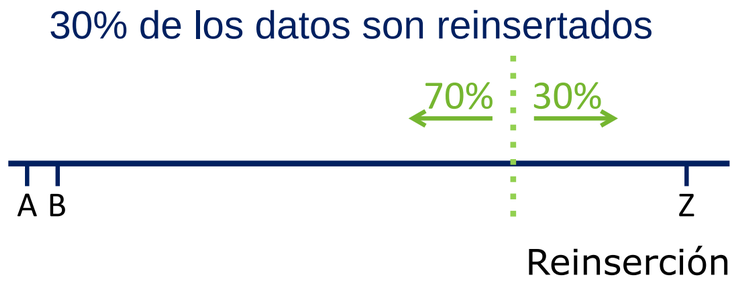
① Sin solapamiento ✓
② Con solapamiento ✗
③ Con solapamiento ✗

Nodos internos: Minimizar área/perímetro

R*-Tree: *Overflow*

Reinserción forzada





Reinsertar primero…

Nodos cercanos al centro **vs** Nodos lejanos al centro

Solo se ejecuta una vez en un nivel por inserción de un objeto

R*-Tree: *Overflow*

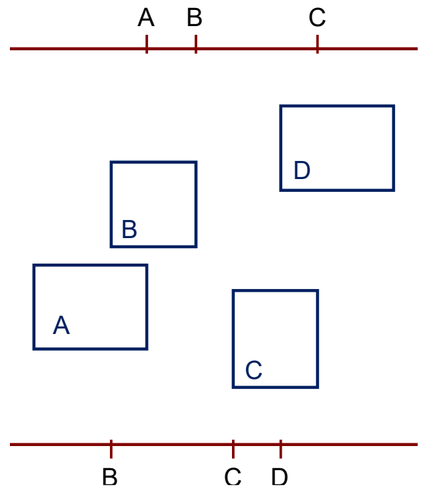
Split

- Seleccionar eje

a) Eje x

- Limite inferior
- Limite superior

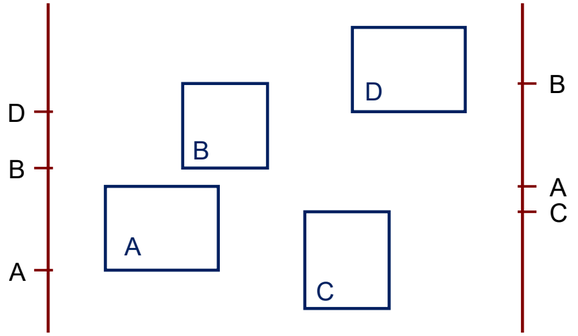
Suma de semiperímetros de las regiones



b) Eje y

- Limite inferior
- Limite superior

Suma de semiperímetros de las regiones



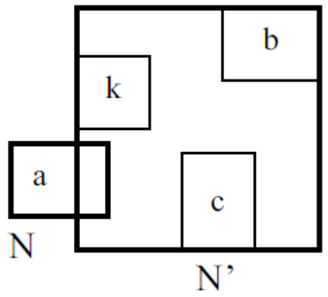
R*-Tree: *Overflow*

Split

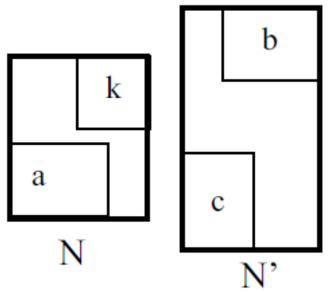
- Posición de la división

  Elegimos la partición con menor semiperímetro

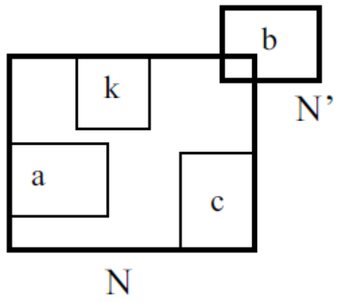
$P = 30$ ✗



$P = 20$ ✓



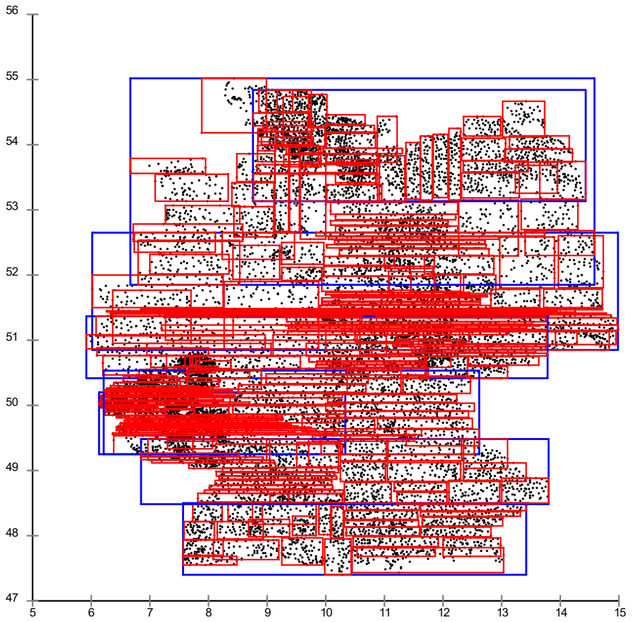
$P = 22$ ✗



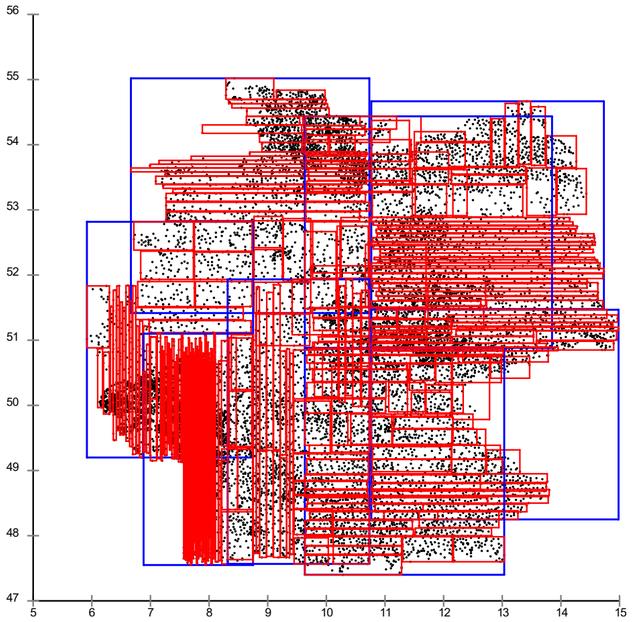
Nota: Una mejor solución es elegir el **menor overlap**, lo cual aumenta el número de operaciones.

En el parcial solo minimizarán el semiperímetro (sino esa pregunta les tomará el doble de tiempo)

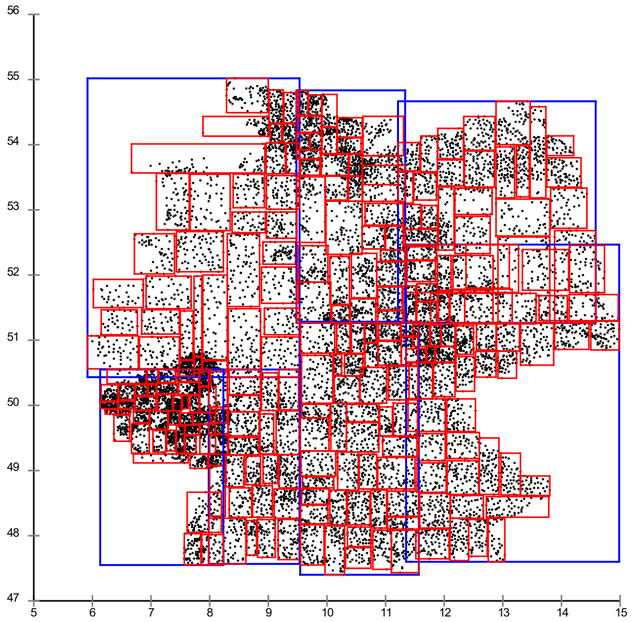
R*-Tree



Quadratic split



Linear split



Topological split

UTEC

UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA