Ciencia de la Computación

Estructuras de Datos Avanzadas

Docente: Erick Mauricio Gomez Nieto

BKD - Tree

Entregado el 15/04/2025

Briceño Quiroz Anthony Angel

Índice

Introducción:

Fig.1. Introducción a los BKD - Tree

Un problema que afecta de manera alarmante es el de la indexación de conjuntos de datos puntuales multidimensionales se ha estudiado cantidad de veces. Lo que ha generado cantidad de estructuras, lo que da a entender la dificultad de optimizar los múltiples requisitos que satisfacen estos índices multidimensionales.

Tradicionalmente, estructuras como los **kd-trees** han sido muy efectivas para consultas en memoria, pero no escalan bien cuando se trabaja con grandes volúmenes de datos en **memoria externa**.

El BKD - nace como una solución a este problema permitiendo mantener buena utilización de disco y garantizar bajo número de accesos a memoria secundaria

KD - Tree y las limitaciones que lleva consigo

Este árbol llega a dividir el espacio en hiperplanos ortogonales, intercalando la cantidad de dimensiones que hay. Sin embargo, es una estructura estática, es decir una vez construido, este sufre una degradación si se realizan inserciones o eliminaciones frecuentes, ya que se necesita reconstruirse para mantener su eficiencia.

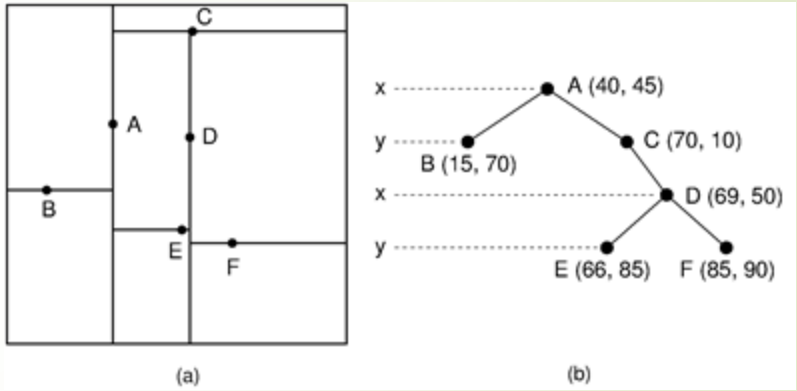

Fig.2. KD - Tree

¿En realidad existe solo el BKD - Tree?

El BKD - Tree se basa en una extensión del KD - Tree. Esta extensión viene a ser el KDB – Tree. Entonces el BKD - Tree se diría que es hijo del KDB – Tree, junto con un modelo logarítmico para dinamizar una estructura estática.

Ahora el BKD logra el casi 100% de utilización del espacio y el rápido procesamiento de consultas de un árbol KDB estático. Sin embargo a diferencia de su padre, estas propiedades se mantienen con actualizaciones masivas.

Consulta de rango ortogonal o de ventana. ¿Que es?

Es un consulta delimitada por un rectángulo alineado a un eje y el objetivo es encontrar todos los puntos de la B.D dentro del rectángulo

Estructuras para consulta de rango ortogonal:

- KDB - Tree
- hB - Tree
- R - Tree

En la práctica estas 3 estructuras suelen responder a las consultas con un número de E/S mucho menor. Sin embargo su rendimiento puede deteriorarse con un gran número de actualizaciones

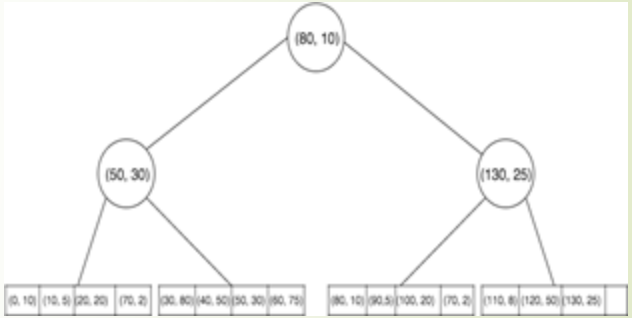

Fig. 3. BKD - Ejemplo

- N: Total de número de puntos
- B: Total de número de puntos que caben en un disco
- K: Número de puntos del rectángulo de consulta

Ω(√(N/B) + (K/B)) es el limite inferior teorico del numero de E/S que necesita un indice de espacio lineal para responder la consulta

Recientemente se han desarrollado estructuras espaciales lineales, con rendimiento de consulta y actualización eficiente garantizado en el peor de los casos

- Cross - Tress
- O - Tress

$$O(\log_B N)$$

Estas estructuras responden a la consulta de ventana en el número óptimo de E/S y pueden actualizarse, Pero...

Tienen un interés práctico limitado, esto significa que su rendimiento promedio de consulta está cerca al rendimiento al peor de los casos

Fig. 4.

Estas estructuras son eficientes para consultas y actualizaciones, pero su rendimiento en la práctica puede ser deficiente si los datos no están bien distribuidos. En bases de datos distribuidas que manejan gran volumen de datos, como en Google o Amazon, estas estructuras pueden ayudar en consultas rápidas. Sin embargo, en escenarios reales, estructuras como B-trees o R-trees son más efectivas, ya que ofrecen un mejor rendimiento con datos no ideales.

Dominando el Espacio de Datos: El Poder de los BKD Trees en Consultas Multidimensionales

¿Pero entonces qué es un BKD - Tree?

Un BKD es un conjunto de árboles KD equilibrados. Cada árbol KD se dispone (o se bloquea) en el disco de forma similar a como se distribuye un KDB. Para almacenar un árbol kd determinado en disco, primero modificamos las hojas para que contengan B puntos, en lugar de sólo uno. De este modo, los puntos se empaquetan en bloques N/B

Fig. 5. Que es un BKD - Tree

¿Y como es que logra su eficiencia?

Lo que sucede o que se hace, es que en vez de reequilibrar dinámicamente tras cada inserción lo que se hace es mantener de $\log_2(N/M)$ árboles KDB estáticos y realizamos actualizaciones reconstruyendo un conjunto elegido cuidadosamente elegido de las estructuras a intervalos regulares, donde M es la capacidad del buffer de la memoria, en número de puntos.

Entonces se comprende que de este modo, mantenemos una utilización del espacio cercana al 100% del árbol KDB estático.

- O'Neill (Log-Structured Merge-Tree LSM - Tree)
- Jagadish (Multiversion Access Structures MVAS)

  Estas dos personas implementaron este tipo de idea y las estructuras de ellos sirven para indexar puntos en un unico atributo y sus tecnicas no pueden extenderse para manejar eficientemente puntos multidimensionales.

Bulk Loading en un BKD - Tree

Es aquel proceso el cual construye de una vez todo el árbol KD a partir de un conjunto grande de puntos, en vez de insertar punto por punto. Eso llega a ser mucho más rápido y eficiente; entonces nos damos cuenta que este método se usa para estructuras indexadas como el BKD - Tree

Binary:

1. Creamos dos listas ordenadas (x,y)
2. Construimos el árbol KD de arriba hacia abajo, empezando por el nodo raíz, y se realiza los sgtes. pasos para cada nodo de manera que la búsqueda sea primero en profundidad.
   a) Encuentre la línea de partición;
   b) Distribuya la entrada en dos conjuntos, basándose en línea de partición.

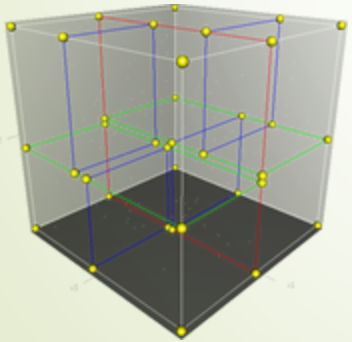

Fig. 6. Espacio tridimensional

Grid:

1. Creamos dos listas ordenadas (x,y)
2. Construir $\log_2 t$ niveles del árbol kd:
   a) Calculamos t líneas de rejilla ortogonales al eje X y t líneas ortogonales al eje Y
   b) Creamos la matriz cuadrícula A que contiene los recuentos de las celdas a la cuadricula. Creamos un subárbol de altura $\log_2 t$ utilizando los recuentos de la matriz de la rejilla.
   c) Distribuimos la entrada en t conjuntos correspondientes a las t hojas.
1. Construir niveles inferiores en la memoria principal o repitiendo el paso (2)

Dynamic Updates:

Un árbol BKD con N puntos en un plano consiste de $\log_2(N/M)$ árboles KD. El ith árbol KD, T_i , esté vacío o contiene exactamente 2^i M puntos. Es así, que T0 almacena como máximo M puntos. Además, en la memoria interna se guarda una estructura T_0^M que contenga a lo sumo M puntos es en la memoria interna.

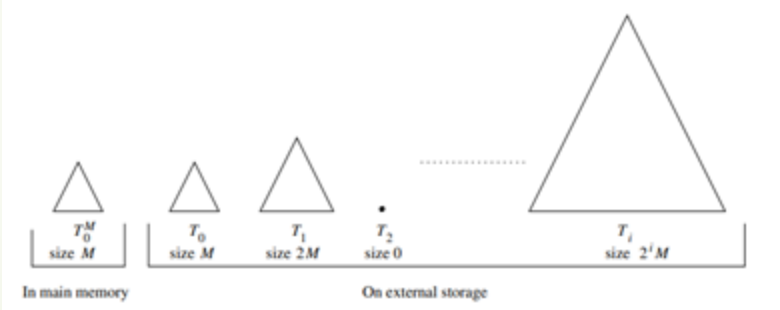

Fig. 8. El bosque de árboles que componen la estructura de datos. En este caso, T2 está vacío.

Algoritmos de Inserción(p) y Eliminación(p)

Insert(p):

1. Insertar p en la memoria intermedia T_0^M;
2. Si T_0^M no está lleno, return; en caso contrario encontrar el primer árbol vacío T_k y extraer los puntos de T_0^M y $T_i \; 0 \leq i < k$ en el fichero F.
3. Hacemos Bulk load T_k a partir de los elementos de F
4. Vaciar T_0^M y T_i , $0 \leq i < k$

Delete(p):

1. Consultamos T_0^M con p, si se encuentra, se elimina p y return.
2. Consultamos en los árboles no vacíos empezando por T_0 y seguimos con los demás árboles, se encuentra, se elimina y return.

¿Consulta de ventana en un BKD?

Para poder responder esta consulta usando un BKD, tenemos que consultar todas las estructuras $\log_2(N/M)$, en lugar de solo una, pero teóricamente mantenemos el límite de consulta óptimo en el peor de los casos $O(\sqrt{N/B} + K/B)$

¿Cómo se logra esto?

- Se usa un algoritmo óptimo de carga masiva de E/S.

  $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$

- La inserción de datos se realiza en $O(\frac{1}{B}(\log_{M/B}\frac{N}{B})(\log_2\frac{N}{M}))$

  E/S amortizadas

- Este límite llega a ser mucho menor que el límite de actualización de árboles B+ de a todos los efectos prácticos

  $O(\log_B N)$

Aunque el BKD tiene buenas propiedades teóricas, se debe demostrar en la práctica, la principal contribución del artículo "Bkd-Tree: A Dynamic Scalable kd-Tree" es una prueba de su viabilidad práctica.

- KDB → 28% de utilización de espacio
- hB (II) → 36% de utilización de espacio
- BKD → superior al 99% de utilización de espacio

Un punto importante en todo esto es la inserción de datos, teniendo en cuenta que un BKD puede ser hasta 100 veces más rápido que un KDB en esta función, en el sentido amortizado

Tablas de Datos TIGER y Datos uniformes

Establecer	1	2	3	4	5	6
Número de puntos	15483533	29703113	39523372	54337289	66562237	77383213
Tamaño (MB)	177.25	340.00	452.38	621.94	761.82	885.69

Tabla 1. Tamaños de los conjuntos TIGER.

Establecer	1	2	3	4	5	6
Número de puntos (millones)	20	40	60	80	100	120
Tamaño (MB)	228.88	457.76	686.65	915.53	1144.41	1373.29

Tabla 2. Tamaños de los conjuntos de datos uniformes

Plataforma Experimental:

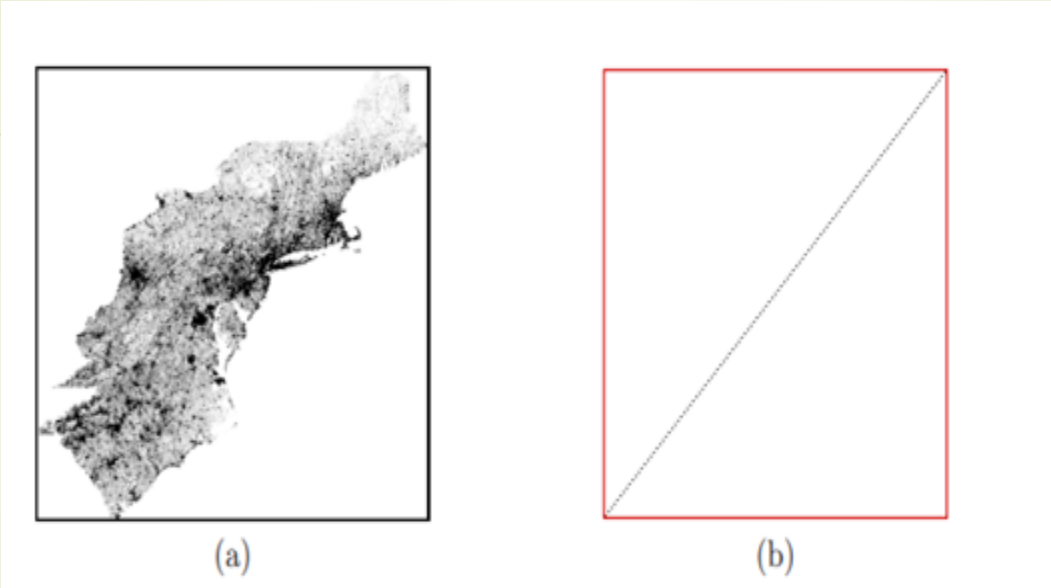

Fig. 8. (a) Una imagen del conjunto TIGER 1 (todos los puntos de las características de las carreteras de 15 estados del este de EE.UU.). La zona blanca no contiene puntos. Las regiones más oscuras tienen la mayor densidad de puntos. (b) Un conjunto de datos diagonal

Rendimiento con la bulk loading con datos uniformes

Establecer	1	2	3	4	5	6
Número de puntos (millones)	20	40	60	80	100	120
Tamaño (MB)	228.88	457.76	686.65	915.53	1144.41	1373.29

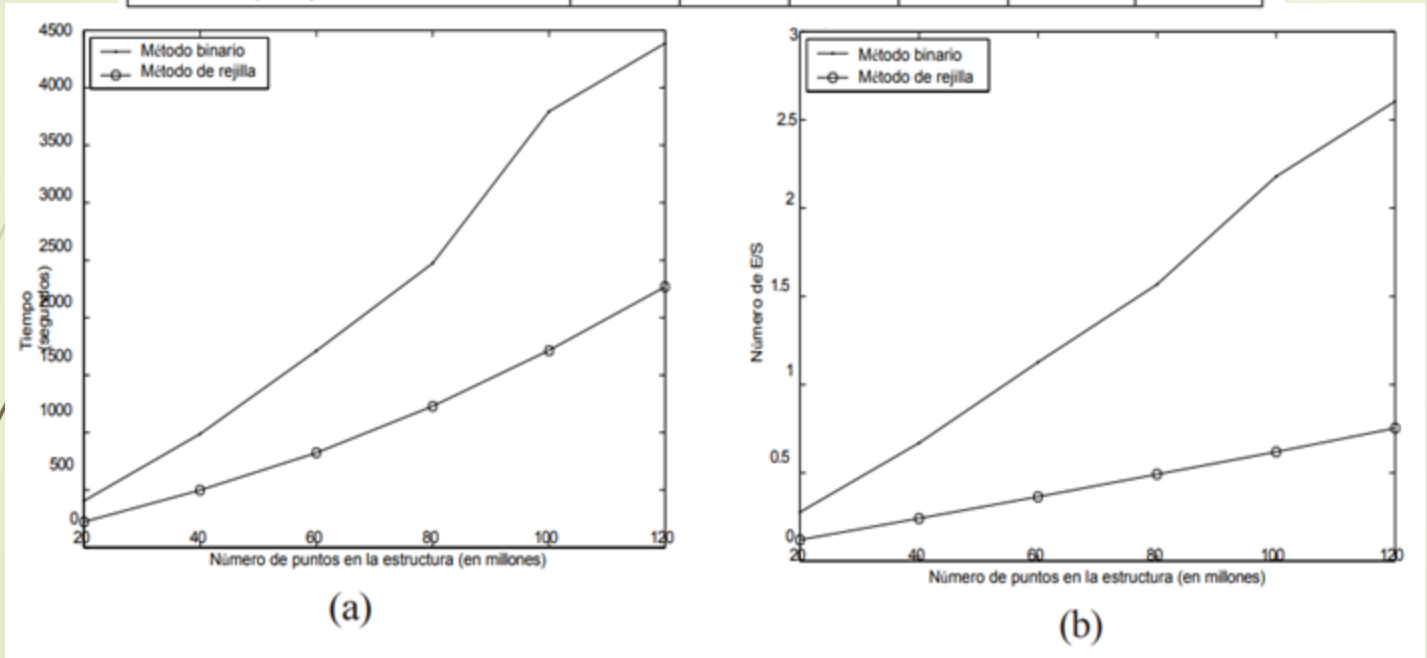

Fig. 9. Rendimiento de la carga masiva con datos uniformes: (a) Tiempo (en segundos), (b) Número de E/S

Rendimiento con bulk loading con datos TIGER

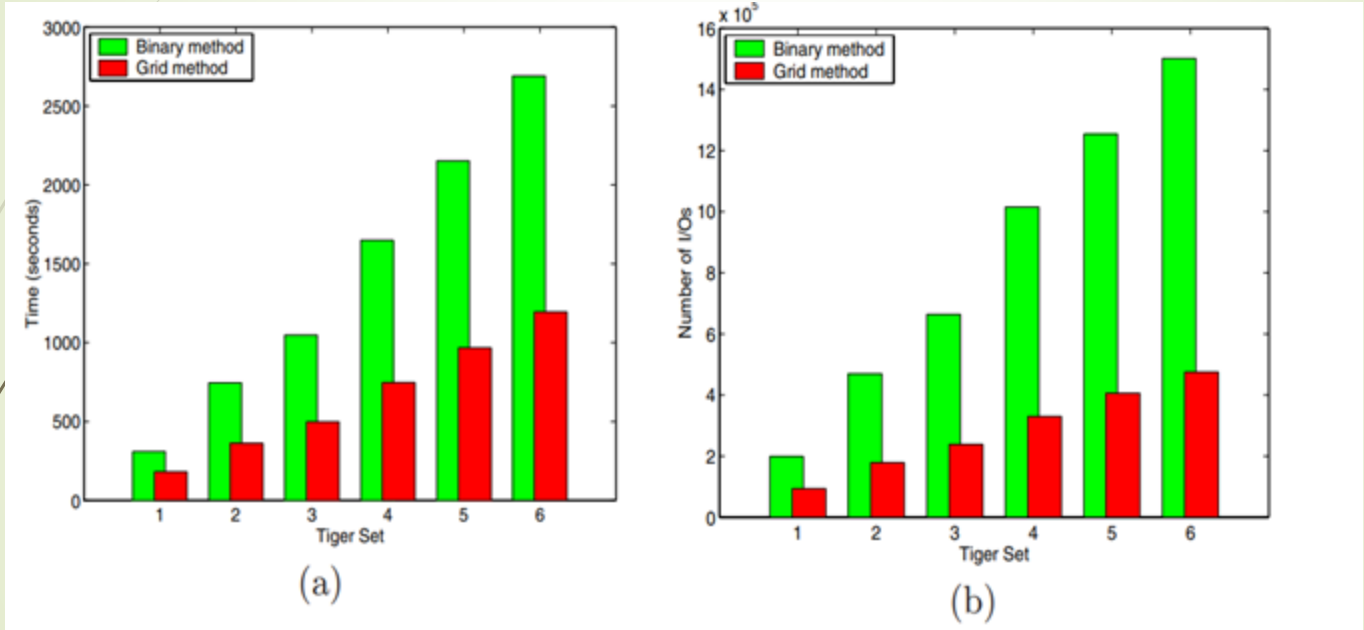

Fig. 10. Rendimiento de la carga masiva en datos TIGER: (a) Tiempo (en segundos), (b) Número de E/S

Rendimiento de Inse

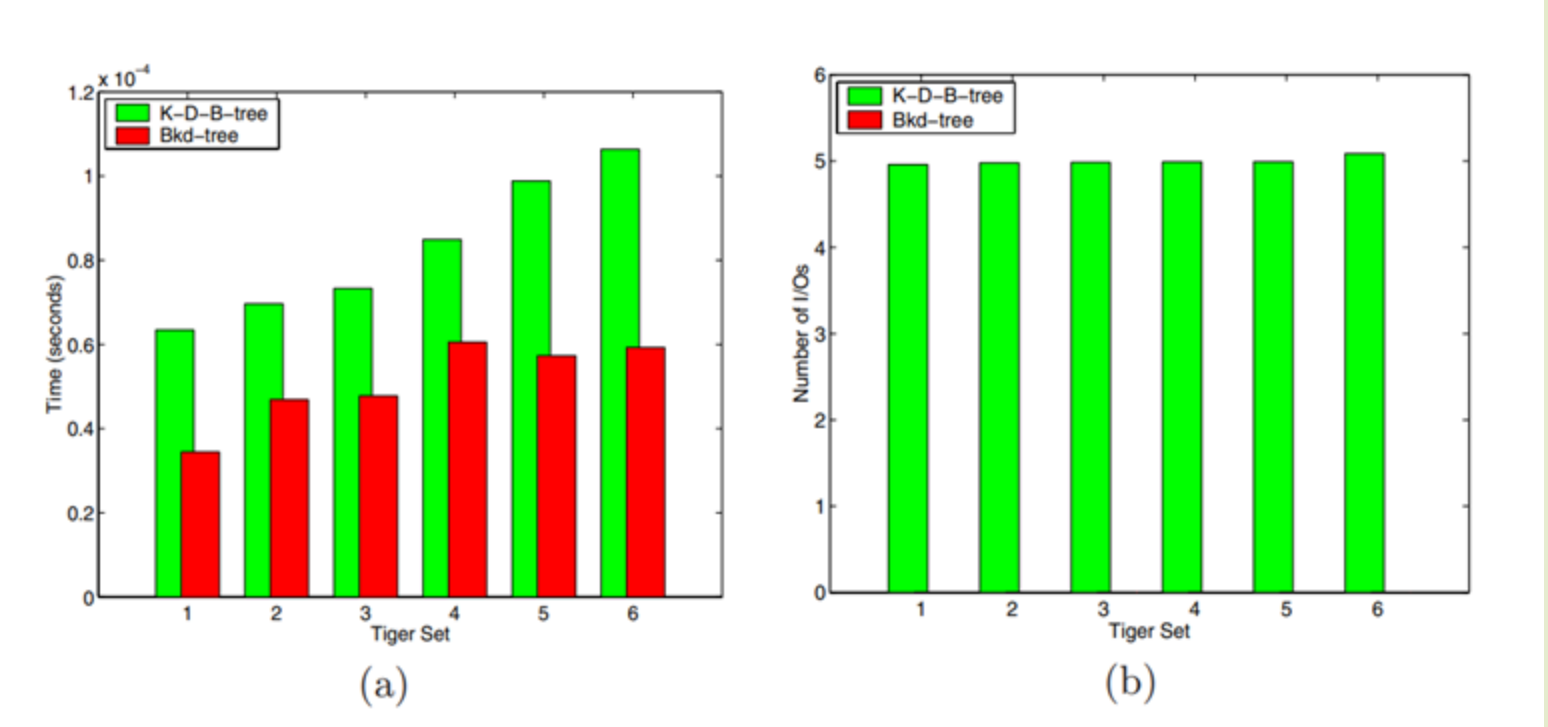

Fig. 11. Rendimiento de inserción en árboles K-D-B y Bkd (datos TIGER): (a) Tiempo (en segundos), (b) Número de E/S

Rendimiento de Consultas:

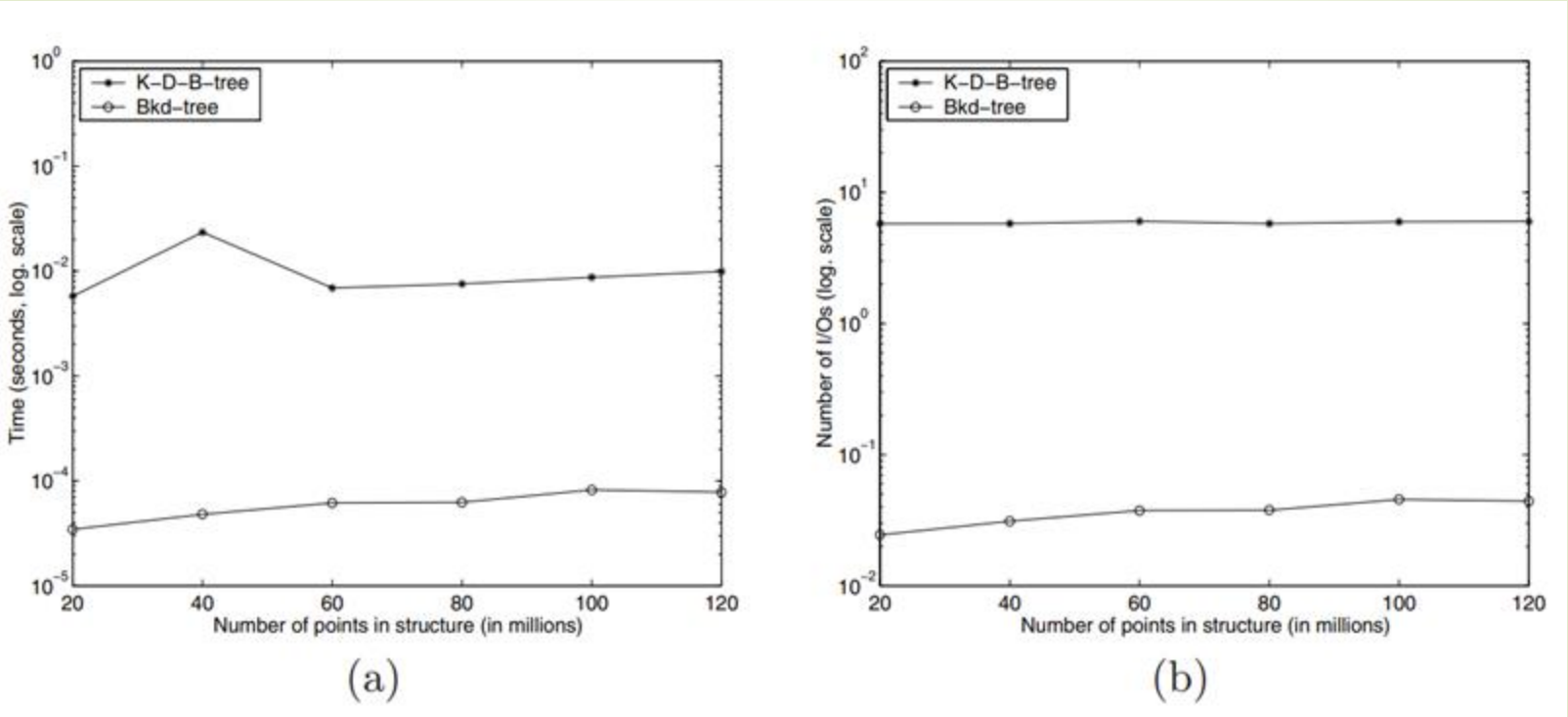

Fig. 12. Rendimiento de inserción en árboles K-D-B y Bkd (datos distribuidos uniformemente): (a) Tiempo (en segundos), (b) Número de E/S

Tablas 3 y 4

Número de puntos (en millones)	20	40	60	80	100	120
Árboles kd no vacíos	3	3	3	4	4	4
Max kd-trees $()\log_2(N/M)e)$	4	5	6	6	7	7

Tabla 3. El número de árboles kd no vacíos y el número máximo de árboles kd, para cada árbol Bkd construido sobre los conjuntos de datos uniformes.

Número de puntos (en millones)	20	40	60	80	100	120
Árbol Bkd	78.4	84.7	88.1	86.5	90.4	90.6
Árbol K-D-B	74.8	83.6	86.2	87.9	90.2	90.6

Tabla 4. El número de puntos devueltos por una consulta de ventana como del número total de puntos recuperados. Para cada conjunto, la ventana cubre el 1% del número total de puntos.

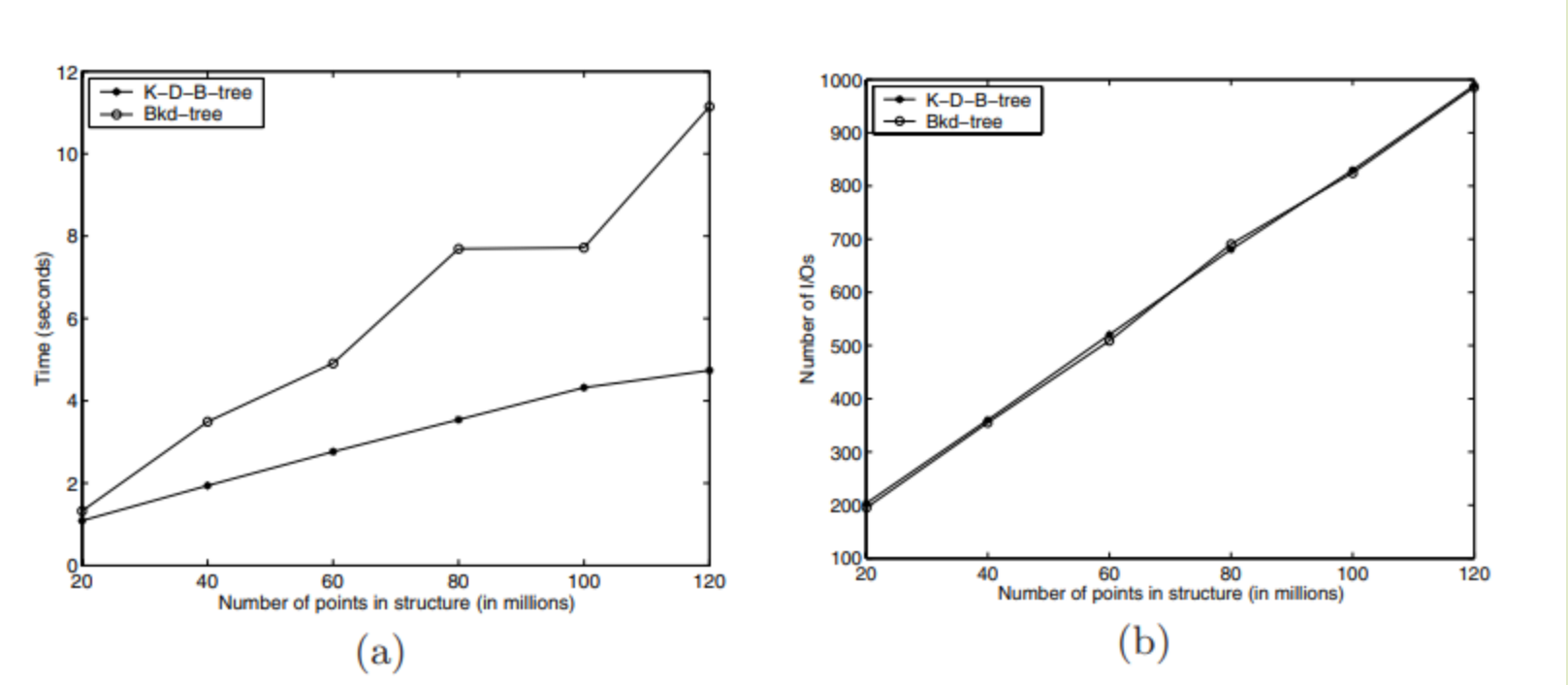

Fig. 12. Rendimiento de la consulta de rango en los datos uniformes (el área de rango es el 1% del área total): (a) Tiempo (en segundos), (b) Número de E/S

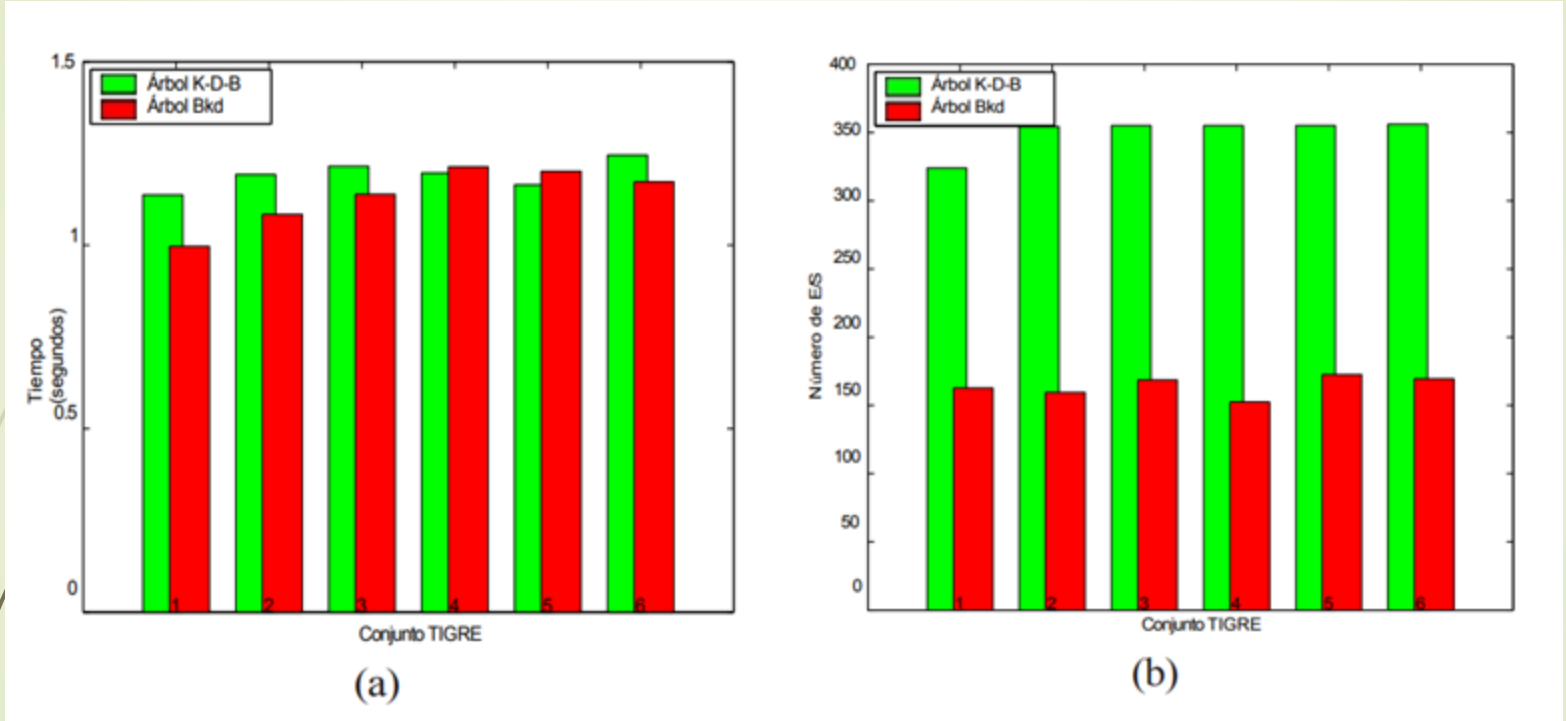

Fig. 13. Rendimiento de la consulta de rango en los datos de TIGER: (a) Tiempo (en segundos), (b) Número de E/S

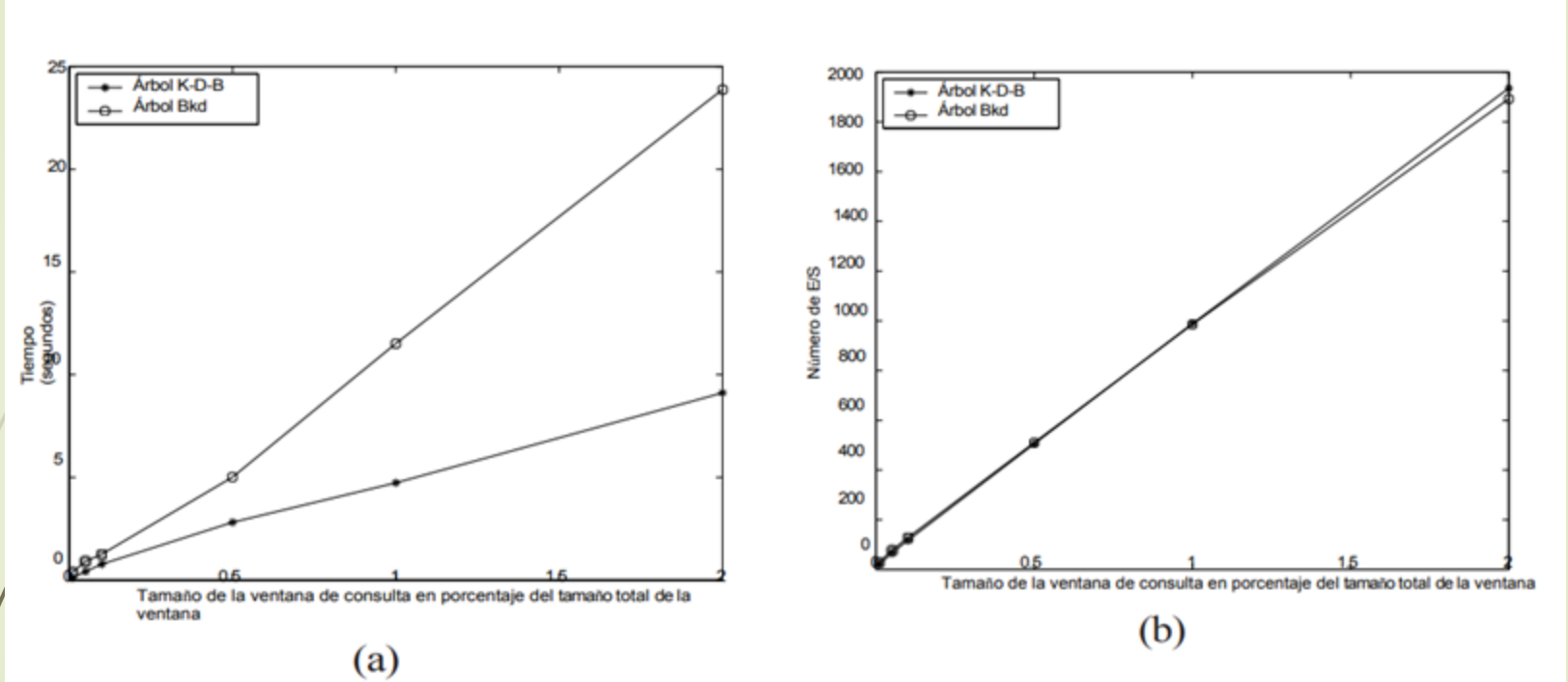

Fig. 14. Rendimiento de las consultas de rango de tamaño creciente (el conjunto de datos consta de 120 millones de puntos distribuidos uniformemente en un cuadrado): (a) Tiempo (en segundos), (b) Número de E/S

Referencias:

[1] O. Procopiuc, P. K. Agarwal, L. Arge, J.S. Vitter. Bkd-Tree: A Dynamic Scalable kd-Tree

Department of Computer Science, Duke University Durham, NC 27708, USA. Department of Computer

Science, Purdue University West Lafayette, IN 47907 USA