Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Maestría en Ciencias
en Ciencias de la Computación

Diseño de un índice para colecciones de nubes de puntos en una rejilla

Tesis
para cubrir parcialmente los requisitos necesarios para obtener el grado de Maestro en Ciencias

Presenta:
Miguel Ramírez Chacón

Ensenada, Baja California, México
2017

Tesis defendida por

Miguel Ramírez Chacón

y aprobada por el siguiente Comité

Dr. Hugo Homero Hidalgo Silva
Codirector de Tesis

Dr. Edgar Leonel Chávez González
Codirector de Tesis

Dr. Carlos Alberto Brizuela Rodríguez

Dr. Heriberto Márquez Becerra

Dr. Jesus Favela Vara
Coordinador del Posgrado en Ciencias de la Computación

Dra. Rufina Hernández Martínez
Directora de Estudios de Posgrado

Resumen de la tesis que presenta Miguel Ramírez Chacón como requisito parcial para la obtención del grado de Maestro en Ciencias en Ciencias de la Computación.

Diseño de un índice para colecciones de nubes de puntos en una rejilla

Resumen aprobado por:

Dr. Hugo Homero Hidalgo Silva Codirector de Tesis	Dr. Edgar Leonel Chávez González Codirector de Tesis

En los últimos años se ha incrementado significativamente el tamaño de las bases de datos de contenido multimedia y existe cada vez más interés en mantener nuevas bases de datos. Ante este incremento, es de suma importancia diseñar estructuras de datos y algoritmos (llamados *índices*) que permitan realizar consultas de proximidad de manera eficiente. Aunque un gran número de objetos multimedia pueden ser representados como nubes de puntos (imágenes, audio, información de sistemas geográficos, puntos y polígonos en sistemas CAD, series de tiempo, etc.), pocos trabajos han abordado el diseño de índices para objetos multimedia que utilizan esta representación. El objetivo de este trabajo es diseñar una estructura de datos que permita realizar búsquedas de proximidad de objetos multimedia representados mediante nubes de puntos, en bases de datos de gran tamaño en tiempo sublineal y que estas estructuras de datos sean robustas ante inserciones y borrados. Para esto, se proponen estructuras de datos basadas en dos esquemas de solución: índices para nubes de puntos basados en índices invertidos y basados en índices métricos. Se realizaron pruebas experimentales en bases de datos de gran tamaño, incluyendo una base de datos sintéticos de 10 millones de nubes de puntos (1000 puntos por cada nube) y la base de datos MIR Flickr-1M, la cual contiene 1 millón de imágenes de alta calidad. El desempeño de los índices propuestos fue evaluado en función de su tiempo promedio de consulta, tiempo de construcción, exhaustividad@k, memoria de almacenamiento utilizada y desempeño ante inserciones/borrados. Los resultados obtenidos indican que los índices para nubes de puntos propuestos basados en índices invertidos, tienen un desempeño superior en tiempo promedio de consulta respecto al índice para nubes de puntos utilizado en la literatura, logrando en algunas pruebas experimentales, una mejora de tres órdenes de magnitud en tiempo promedio de consulta en la base de datos de imágenes MIR Flickr-1M y $exhaustividad@1 \geq 0.989$ bajo el efecto de inserciones y borrados para todos los índices propuestos.

Palabras clave: nubes de puntos, búsqueda de proximidad, índices, búsqueda aproximada de vecinos más cercanos

Abstract of the thesis presented by Miguel Ramírez Chacón as a partial requirement to obtain the Master of Science degree in Computer Science.

Design of an index for collections of point clouds in a mesh.

Abstract approved by:

Dr. Hugo Homero Hidalgo Silva Thesis Co-Director	Dr. Edgar Leonel Chávez González Thesis Co-Director

In recent years there has been a significant increase in the size of multimedia databases, and new databases of large size are created continuously. It is, therefore, of utmost importance the design of indexes and algorithms allowing efficient similarity queries. Although a large number of multimedia objects can be represented as a point cloud (images, audio, information of geographic systems, points and polygons in CAD systems, time series, etc.), just a few works have addressed the design of indexes for multimedia objects using this representation. The objective of this work is to design data structures and algorithms, with sublinear performance, for similarity search of multimedia objects represented by point clouds in large databases, robust to insertions and deletions. To this end, we propose data structures based on two solution schemes: indexes for point clouds based on inverted indexes and based on metric indexes. Experimental tests were performed on large databases, including a synthetic database of 10 million point clouds (1000 points per cloud) and the Flickr-1M MIR database, which contains 1 million high-resolution images. The performance of the proposed indexes was evaluated according to: Average query time, construction time, recall@k, storage memory used and performance under insertions and deletions. The obtained results shows that the indexes proposed for point clouds based on inverted indexes, have a superior performance in average query time compared with the actual index for point clouds used in the literature. In some tests we obtained an improvement of three orders of magnitude on average query time in the image database MIR Flickr-1M and recall@1≥0.989 under the effect of insertions and deletions for all proposed indexes.

Keywords: point clouds, similarity search, index, approximate nearest neighbor search

Dedicatoria

Por apoyarme tanto, por estar siempre conmigo, por animarme en los días difíciles, por creer en mí, por ser mi inspiración y sobre todo, por tu amor. Te amo mi amor.

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por brindarme el apoyo económico mediante la beca no. 587135 para realizar mis estudios de maestría.

Al Centro de Investigación Científica y de Educación Superior de Ensenada, por el excelente trato que recibí de todo su personal durante estos dos años.

En especial a mis directores de tesis, el Dr. Edgar Chávez y el Dr. Hugo Hidalgo, gracias por todo su apoyo, tiempo, consejos y paciencia durante el desarrollo de este trabajo de tesis.

A los miembros de mi comité de tesis, el Dr. Carlos Brizuela y el Dr. Heriberto Márquez, por sus consejos, comentarios y observaciones realizadas durante el desarrollo de esta tesis.

A todos los profesores del departamento de ciencias de la computación, por sus excelentes cursos y por ayudarme a mejorar mi formación académica.

A mi mamá, por todo tu esfuerzo y sacrificio, por entenderme, por tus consejos, por apoyarme y por enseñarme a luchar día a día para alcanzar mis sueños. Por eso y mucho más, gracias mamá.

Tabla de Contenido

Lista de figuras

Lista de tablas

Capítulo 1. Introducción

El problema de búsqueda de proximidad consiste en buscar elementos de una base de datos similares a un elemento de consulta que proporciona un usuario. En este caso, el concepto de similitud se define como una función de distancia (Chávez et al., 2001).

Debido a los recientes avanfces tecnológicos en relación con la capacidad de cómputo y almacenamiento, en los últimos años ha habido un aumento significativo en la cantidad y tamaño de bases de datos de contenido multimedia. Estas bases de datos se pueden encontrar en áreas como: recuperación de imágenes, reconocimiento de patrones, biología computacional, minería de datos, visión por computadora, sistemas de información geográfica, entre otras (Bhima et al., 2012; Chávez et al., 2001; Chen et al., 2016; Du y Li, 2013; Yu et al., 2007). Dado esto, es fundamental el diseño de estructuras de datos que permitan realizar búsquedas de proximidad en bases de datos de contenido multimedia de gran tamaño en tiempo sublineal, ya que una búsqueda secuencial no es factible debido al tamaño de estas.

Figura 1.- Sistema CBIR – En la parte izquierda se muestra la consulta proporcionada por el usuario. En rojo se muestran las imágenes que proporciona el sistema como respuesta. Recuperado el 29/07/17 de: https://www.intechopen.com/source/html/41905/media/image244_w.jpg

Un caso específico de un sistema de búsqueda de proximidad para contenido multimedia es un sistema de recuperación de imágenes basado en contenido (CBIR). Este sistema permite realizar búsquedas de imágenes similares a una imagen de consulta proporcionada por el usuario. En la **Figura 1** se muestra un ejemplo de un sistema CBIR. Debido a que generalmente estas bases de datos contienen millones de imágenes, se deben considerar diversas características en estos sistemas, por ejemplo, tiempo de consulta, exhaustividad[1], tiempo de construcción y cantidad de memoria utilizada para representar una imagen (Jegou et al., 2010). Dado lo anterior, el problema de CBIR ha sido un tema de gran interés para la comunidad de recuperación de información.

La mayoría de los sistemas de recuperación de información para objetos multimedia del estado del arte, representa estos objetos como vectores de dimensión elevada (tuplas de números reales, cadenas binarias, etc.) (Lowe, 2004; Moise et al., 2013; Nguyen Mau Toan y Yasushi, 2016; Sun et al., 2013; Wan et al., 2014; J. Wang et al., 2016; Yandex y Lempitsky, 2016). Es bien conocido que el desempeño de búsquedas de proximidad de objetos representados mediante vectores de dimensión elevada, se aproxima al desempeño de una búsqueda secuencial conforme aumenta la dimensión del vector, debido al problema de la maldición de la dimensión (Chávez et al., 2001). Dado lo anterior, el tamaño de las bases de datos y la dimensión de la representación utilizada para los objetos multimedia, presentan un desafío para los métodos de indexado tradicionales (Ji et al., 2014).

Recientemente, técnicas de *Deep Learning* y aprendizaje de maquina han sido utilizadas en sistemas CBIR, con el objetivo de diseñar una función hash que mapee imágenes con contenidos similares a cadenas binarias similares; este acercamiento es conocido como *Learning to Hash* (Salakhutdinov y Hinton, 2009; J. Wang et al., 2016).

Una representación alternativa de ciertos objetos multimedia es una nube de puntos, donde cada objeto se representa por una colección de puntos de dos dimensiones. Algunos ejemplos de esta representación son: puntos de interés del descriptor SIFT (Lowe, 2004), detalles en huellas digitales (Hong et al., 1998), información de sistemas de información geográfica (Bhima et al., 2012) y aplicaciones de audio como Shazam (A. L. Wang, 2003). Es interesante que pocos trabajos del estado del arte han abordado el diseño de estructuras de datos para objetos multimedia que utilicen nubes de puntos como representación.

[1] Es la proporción de los elementos recuperados y relevantes que entrega un sistema de recuperación de información, entre el número de elementos relevantes de toda la base de datos.

Las nubes de puntos pueden ser sujetas a ruido, inserciones, borrados, transformaciones afines y proyectivas. Es importante señalar que es posible generar nubes de puntos robustas a ruido y a transformaciones de similitud mediante coeficientes de Fourier (Chávez et al., 2016). Sin embargo, la representación anterior no es robusta ante inserciones y borrados. Informalmente el problema de inserciones y borrados consiste en lo siguiente: dada una nube de puntos en una base de datos que representa un objeto multimedia, se desea recuperar el objeto multimedia de la base de datos, dado que se proporciona un subconjunto de puntos de la nube original (la cual pudo ser modificada insertando y/o borrando un subconjunto de puntos de la nube) como consulta.

En este trabajo nos enfocaremos en el diseño de estructuras de datos para nubes de puntos, que sean robustas ante inserciones y borrados, las cuales podrán ser utilizadas en sistemas de recuperación de objetos multimedia basados en contenido. En esta tesis estudiaremos como caso específico la aplicación de estas estructuras de datos en un sistema de recuperación de imágenes basado en contenido (CBIR).

1.1 Justificación

En la actualidad, la mayor parte de los sistemas de recuperación de objetos multimedia, en especial los sistemas de recuperación de imágenes, realizan búsquedas basadas en texto utilizando los metadatos asociados a cada objeto (Alzu'bi et al., 2015). Sin embargo, esto requiere de un proceso de anotación para cada objeto de la base de datos. Un proceso de anotación manual no es preciso y consume mucho tiempo, de igual manera, un proceso automatizado de anotación depende de la precisión del sistema para reconocer características importantes de la imagen como bordes, posición, puntos de interés y relaciones espaciales entre los objetos (Alzu'bi et al., 2015).

De acuerdo con Wang (2016): "El crecimiento explosivo de Big Data ha atraído mucha atención en el diseño de índices y métodos de búsqueda eficientes. En muchas aplicaciones tales como búsqueda a gran escala y reconocimiento de patrones, encontrar el vecino más cercano a una consulta es un problema de investigación fundamental."(p.34).

Existen muchos ejemplos de bases de datos de contenido multimedia de gran tamaño, por ejemplo: Flickr tiene una base de datos de más de 5 mil millones de imágenes, YouTube recibe más de 100 horas de video cada minuto, Twitter procesa más de 100 millones de mensajes diarios, entre otras (J. Wang

et al., 2016). Además, la búsqueda de objetos multimedia basada en contenido presenta diversos desafíos entre los que se encuentran: aumentar la precisión, exhaustividad y reducir el costo computacional de estos sistemas (J. Wang et al., 2016).

Dado lo anterior es fundamental el diseño de estructuras de datos que permitan realizar consultas de proximidad en bases de datos de contenido multimedia de gran tamaño en tiempo sublineal.

1.2 Objetivos

A continuación se muestra el objetivo general, los objetivos específicos y las preguntas de investigación de esta tesis.

1.2.1 Objetivo general

Diseñar una estructura de datos que permita realizar búsquedas de proximidad de objetos multimedia representados mediante nubes de puntos, en bases de datos de gran tamaño en tiempo sublineal.

1.2.2. Objetivos específicos

- Analizar las bases teóricas de índices basados en rejillas.
- Diseñar e implementar índices para nubes de puntos utilizando rejillas.
- Estudiar métodos de indexado basados en estrategias de índices invertidos, bitmaps y árboles.
- Realizar un análisis cuantitativo de los métodos implementados respecto a tiempo de consulta, tiempo de construcción, exhaustividad y memoria utilizada para el almacenamiento del índice.

1.2.3. Preguntas de investigación

- ¿Cómo se pueden adaptar los métodos de indexado utilizados para consultas k-NN Join al problema de recuperación de nubes de puntos?
- ¿Qué función de distancia proporciona un mejor compromiso entre tiempo de consulta y precisión?
- ¿Cuál es la ventaja algorítmica de utilizar rejillas en lugar de puntos en el plano?

1.3 Organización de la Tesis

En esta sección se muestra la organización del trabajo de tesis. En el Capítulo 2, se presentan algunos conceptos básicos relacionados con búsqueda de proximidad. En el Capítulo 3, se estudian los conceptos fundamentales del funcionamiento de los índices, se presentan estrategias de partición, técnicas para reducir el número de evaluaciones de la función de distancia y ejemplos de índices utilizados en la literatura. El Capítulo 4 está dedicado a la reprsesentación de objetos multimedia mediante nubes de puntos, para los casos específicos de archivos de audio e imágenes, así como una descripción de los desafíos que se presentan al utilizar esta representación. En el Capítulo 5 se presenta el planteamiento formal del problema de recuperación de nubes de puntos, así como un breve resumen de los resultados del estado del arte sobre estructuras de datos para este problema y estructuras de datos para búsqueda de proximidad. En el Capítulo 6 se proponen diversas estructuras de datos para el problema de recuperación de nubes de puntos. En los Capítulos 7 y 8 se presenta la metodología utilizada y los resultados de las pruebas experimentales de las estructuras de datos propuestas, enfocados en tiempo de construcción, tiempo de consulta, exhaustividad y comportamiento ante el problema de inserciones y borrados. Finalmente, en el Capítulo 9 se realiza un análisis de los resultados obtenidos y se presentan las conclusiones y algunas ideas para trabajo futuro.

Capítulo 2. Marco Teórico

En este capítulo se presentan conceptos básicos y definiciones del área de recuperación de información, los cuales son fundamentales para el desarrollo de capítulos posteriores.

2.1 Recuperación de información

El proceso de recuperación de información se define como la búsqueda de información dentro de una base de datos, o una colección de objetos organizados, que tiene como objetivo satisfacer un requerimiento de información de un usuario (J. Zhang, 2001). Estos objetos pueden ser documentos de texto, imágenes, video, información de redes sociales, datos de sistemas de información geográfica, etc.

Un caso específico en el que estamos interesados son los sistemas de recuperación de objetos multimedia basados en contenido, donde estos objetos pueden ser imágenes, archivos de audio, etc. Estos sistemas se pueden considerar como una extensión de los sistemas de recuperación de información tradicionales.

Sin embargo, el proceso de recuperación de objetos multimedia basado en contenido es complejo, ya que la mayoría de los objetos multimedia tienen diferentes clases de información asociada. Por ejemplo, en sistemas CBIR, una imagen contiene la siguiente información (Del Bimbo, 1999):

- Metadatos independientes del contenido de la imagen: información que no se relaciona directamente con el contenido del objeto (formato, resolución, fecha, lugar, etc.).

- Metadatos dependientes del contenido de la imagen: características de bajo nivel (color, textura, forma, relaciones espaciales, etc.).

- Metadatos que describen el contenido de la imagen: contenido semántico de la imagen.

2.2 Búsqueda de proximidad

Dada la naturaleza de la mayoría de los objetos multimedia, no tiene sentido realizar una búsqueda exacta del mismo objeto que proporciona el usuario como consulta (a menos que en la base de datos se permitan copias), ya que la probabilidad de que un objeto multimedia sea idéntico a un objeto de consulta proporcionado por el usuario es muy baja, debido a factores externos presentes durante la generación del objeto multimedia. Por ejemplo, factores como iluminación, exposición, ruido y calibración del lente de cámara se pueden encontrar en la generación de imágenes (Chávez et al., 2001). De igual manera, el ruido ambiental, una respuesta en frecuencia diferente de los micrófonos utilizados para generar el archivo de audio que proporciona el usuario, son algunos de los factores externos presentes que evitan que tenga sentido realizar búsquedas exactas. Dado lo anterior, lo que se desea encontrar dado un objeto multimedia de consulta, son objetos de la base de datos parecidos de alguna manera al objeto de consulta. Para esto necesitamos definir formalmente qué significa que un objeto sea similar a otro.

La búsqueda de proximidad consiste en encontrar objetos de una base de datos, similares o parecidos a un objeto proporcionado por el usuario (Chávez et al., 2001). En este caso, la similitud se modela mediante una función de distancia. Objetos similares entre sí, tendrán una distancia pequeña entre ellos, y objetos que no sean similares una distancia más grande.

Un espacio métrico es un par (U, d), donde U es el universo de objetos válidos, $S \subseteq U$ un subconjunto finito de U (base de datos) y $d: U \times U \rightarrow \mathbb{R}^+$ una función de distancia que modela el concepto de similitud y que cumple con las siguientes propiedades:

$$\forall x, y \in U, d(x, y) \geq 0 \tag{1}$$

$$\forall x, y \in U, d(x, y) = d(y, x) \tag{2}$$

$$\forall x \in U, d(x, x) = 0 \tag{3}$$

$$\forall x, y \in U, x \neq y \implies d(x, y) > 0 \tag{4}$$

$$\forall x, y, z \in U, d(x, z) \leq d(x, y) + d(y, z) \tag{5}$$

Un caso particular de un espacio métrico es un espacio vectorial. En un espacio vectorial los elementos de U son tuplas de números reales, es decir, cada elemento de U se representa mediante un vector de dimensión k. Es importante señalar que a diferencia de un espacio métrico, donde solo se tiene como información disponible la distancia entre los objetos, en un espacio vectorial se pueden explotar propiedades geométricas, ya que se tiene como información adicional las coordenadas de cada objeto.

Existe un gran número de funciones de distancia y su elección depende de diversos factores, entre los que se encuentran: el tipo de objeto, el costo computacional de la evaluación de la función de distancia, entre otros.

2.3 Consultas de proximidad

A continuación se presentan algunos tipos fundamentales de consultas.

2.3.1 Consulta de rango

Sea $S \subseteq U$ un subconjunto finito de U, $r \in \mathbb{R}^+$ un umbral y un objeto $q \in U$, una consulta de rango denotada como $R(S, q)$ se define como:

$$R(S, q) = \{o \in S \mid d(q, o) \leq r\} \tag{6}$$

Es decir, son todos los objetos de S que tienen una distancia menor o igual a un umbral r respecto al objeto q.

2.3.2 k - vecinos más cercanos (kNN)

Sea $S \subseteq U$ un subconjunto finito de U, $q \in U$ un objeto y $k \in \mathbb{N}^+$ una constante, la consulta de los k-vecinos más cercanos de q al conjunto S, denotado como $kNN(S, q)$, es un conjunto de k objetos de S tal que:

$$\forall o \in kNN(S,q), \forall s \in S - kNN(S,q),\ \ d(o,q) \leq d(s,q) \tag{7}$$

2.3.3 kNN *join*

Sea $Q \subseteq U$ y $S \subseteq U$ dos subconjuntos finitos de U y $k \in \mathbb{N}^+$, la consulta *kNN join* de Q y S, denotado como $Q \ltimes S$, combina cada objeto $q \in Q$ con sus k vecinos más cercanos de S, tal que:

$$Q \ltimes S = \{(q,s) \in Q \text{ x } S \mid \forall q \in Q, \forall s \in kNN(S,q)\} \tag{8}$$

2.3.4 Range join

Sea $Q \subseteq U$ y $S \subseteq U$ dos subconjuntos finitos de U y $r \in \mathbb{R}^+$ un umbral, la operación *range join* se define como:

$$Q \bowtie_r S = \{(q,s) \in Q \text{ x } S \mid d(q,s) \leq r, q \in Q, s \in S\} \tag{9}$$

2.3.5 Distance join

Sea $Q \subseteq U$ y $S \subseteq U$ dos subconjuntos finitos de U y un entero k, la consulta *k distance join* de Q y S, regresa un conjunto de k pares (p_Q, p_S) tal que $p_Q \in Q$, $p_S \in S$, y las distancias entre los k pares sean las más pequeñas entre todos los pares de $Q \text{ x } S$.

2.4 Evaluación de un sistema de recuperación de información

Existen diversas formas de evaluar el desempeño de un sistema de recuperación de información (IR). Cuando un usuario realiza una consulta en un sistema de IR, el sistema responde con una lista de resultados y la consulta realiza una partición de la base de datos en cuatro conjuntos (J. Zhang, 2008):

- A – Recuperados y relevantes
- B – Recuperados y no relevantes

- C – No recuperados pero relevantes

- D – No recuperados y no relevantes

Entre los parámetros más utilizados para medir el desempeño de los sistemas de recuperación de información se encuentran: precisión y exhaustividad.

$$Precisión = \frac{|A|}{|A \cup B|} \tag{10}$$

$$Exhaustividad = \frac{|A|}{|A \cup C|} \tag{11}$$

Donde la precisión indica la proporción entre lo que el usuario necesita y lo que entrega la búsqueda y la exhaustividad indica la proporción entre lo que el usuario necesita y lo que debería de entregar el sistema.

Un parámetro adicional utilizado para medir el desempeño de estos sistemas es $Exhaustividad@k$, el cual es la proporción del número de elementos relevantes dentro de los primeros k resultados entregados al usuario, entre la cantidad de elementos relevantes de la base de datos.

Otros aspectos a considerar para la evaluación del desempeño de un sistema de recuperación de información son: el número de evaluaciones de distancias necesarias para obtener un resultado satisfactorio, tiempo promedio de consulta, tiempo de construcción de la estructura de datos y la cantidad de memoria que utiliza el sistema.

Capítulo 3. Índices

Una solución trivial al problema de búsqueda de proximidad es realizar una búsqueda lineal de todos los elementos en la base de datos y entregar todos los elementos similares respecto al elemento de consulta proporcionado por el usuario. Sin embargo, esta solución solo es viable cuando tenemos bases de datos muy pequeñas. En bases de datos de mediano o gran tamaño, es necesario utilizar algoritmos o estructuras de datos que permitan reducir el espacio de búsqueda para reducir el tiempo de consulta. Si bien existen algoritmos para la reducción del espacio de búsqueda en consultas de proximidad para casos donde no se tiene disponible un índice (Fredriksson y Braithwaite, 2013), la mayoría de los trabajos del estado del arte utilizan estructuras de datos para la reducción de este.

Un índice es una estructura de datos cuyo objetivo es la reducción de evaluaciones de la función de distancia al momento de realizar una consulta de proximidad. Estas estructuras de datos se construyen con anticipación, debido a que el proceso de construcción de estas es costoso. Sin embargo, una vez construido el índice, la reducción en el número de evaluaciones de la función de distancia para cada consulta, amortiza el costo de construcción de este (Chávez et al., 2001).

Un índice cumple con dos funciones fundamentales (Chávez et al., 2001):

1. Genera una relación de equivalencia, la cual induce una partición de la base de datos en subconjuntos.

2. Construye una estructura de datos que dada una consulta, permite identificar el conjunto de subconjuntos donde pueden existir objetos relevantes, los cuales tienen que ser revisados en forma exhaustiva.

Sea X un conjunto, una relación de equivalencia es una relación $\sim$, tal que para todo $x, y, z \in \mathrm{X}$, la relación $\sim$ cumple con la propiedad de reflexividad $(x \sim x)$, simetría $(x \sim y \iff y \sim x)$ y transitividad $(x \sim y \wedge y \sim z \implies x \sim z)$. Por ejemplo, sea $=$ la relación de igualdad en $\mathbb{R}$ (números reales) y $x, y, z \in \mathbb{R}$, observe que la relación de igualdad $=$ es una relación de equivalencia, ya que cumple con la propiedad de reflexividad $(x = x)$, simetría $(x = y \Leftrightarrow y = x)$ y transitividad $(x = y \wedge y = z \implies x = z)$.

Sea $x \in \mathrm{X}$ y $\sim$ una relación de equivalencia, la clase de equivalencia de x se define como $[x] = \{y \mid x \sim y\}$ y el conjunto de las clases de equivalencia de X es llamado el conjunto cociente, denotado como $\mathrm{X}/\sim$, el cual es una partición del conjunto X.

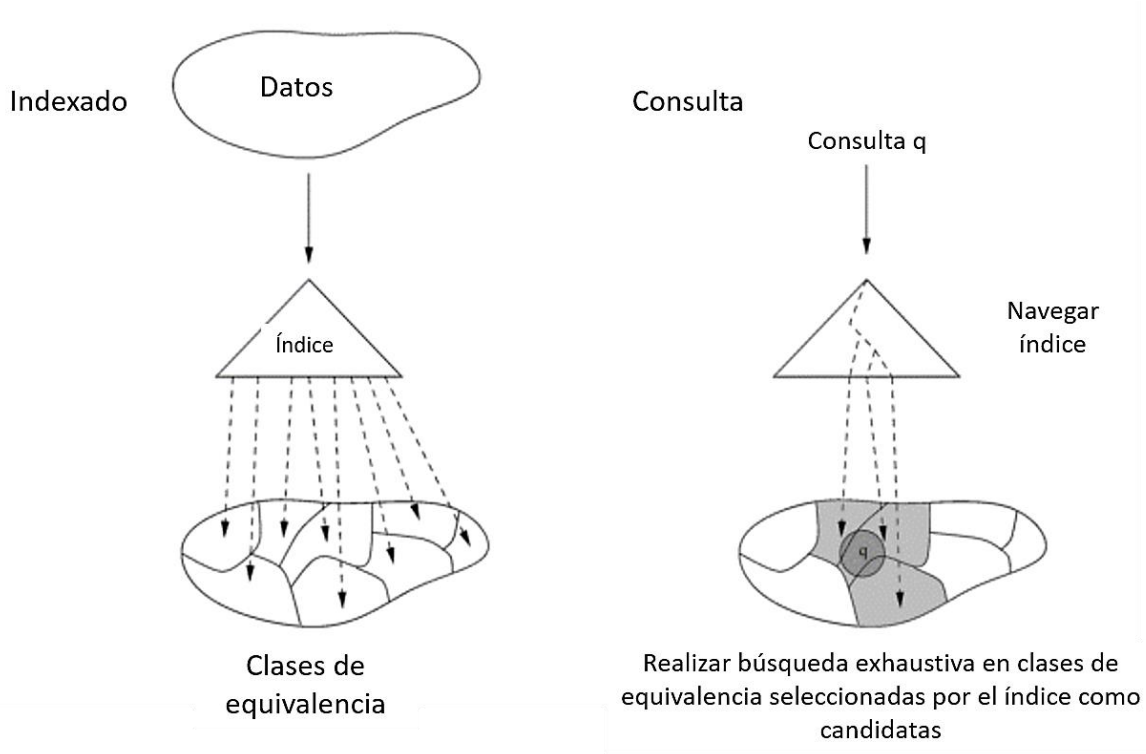


Figura 2.- Proceso de construcción de índice y su uso de este durante el proceso de consulta. Figura modificada de (Chávez et al., 2001).

Las relaciones de equivalencia utilizadas en los índices pueden clasificarse en dos tipos:

1. Relación de equivalencia de pivotes.

2. Relación de equivalencia compacta.

Los índices que utilizan la relación de equivalencia de pivotes, utilizan las distancias entre elementos de la base de datos y un conjunto de elementos llamados “pivotes” (que pueden o no pertenecer a la base de datos) para generar las particiones (Chávez et al., 2001). Si $\{p_1, \ldots, p_k\}$ es el conjunto de pivotes, la relación de equivalencia de pivotes se define como:

$$x \sim_{\{p_i\}} y \iff \forall i, d(x, p_i) = d(y, p_i) \quad (12)$$

Los índices que utilizan la relación de equivalencia compacta, utilizan un conjunto de puntos llamados "centros" y asignan elementos de acuerdo a la cercanía del elemento de la base de datos con el centro más cercano. Si $\{c_1, \dots, c_k\}$ es el conjunto de centros, la relación de equivalencia compacta se define como:

$$x \sim_{\{c_i\}} y \iff closest(x, \{c_i\}) = closest(y, \{c_i\}) \quad (13)$$

Donde:

$$closest(z, S) = \{w \in S, \forall w' \in S, d(z, w) \leq d(z, w')\} \quad (14)$$

En algunas aplicaciones con bases de datos de gran tamaño, realizar consultas de proximidad utilizando índices no es viable, debido a que el tiempo de consulta de estos sistemas puede ser elevado aun utilizando índices para la reducción del espacio de búsqueda. En estas situaciones, se pueden utilizar índices aproximados, los cuales tienen como objetivo reducir la complejidad del tiempo de consulta realizando una aproximación a la respuesta verdadera (Chávez et al., 2001). Los índices aproximados pueden clasificarse en dos tipos: probabilísticos, los cuales proporcionan una cota inferior de la probabilidad de que se encuentre la respuesta correcta y aproximados, los cuales proporcionan una garantía de que la respuesta entregada al usuario se encuentre dentro de un cierto factor de la distancia de la respuesta verdadera.

3.1 Estrategias de partición

Una de las funciones principales de un índice es generar una partición de la base de datos mediante una relación de equivalencia, la cual induce esta partición. Sea U el universo de objetos válidos y $S \subseteq U$ un subconjunto finito de U (base de datos). En esta sección se describen dos de las estrategias de partición más comunes utilizadas en índices: partición por bola y partición por hiperplano.

3.1.1 Partición por bola

Un ejemplo de una relación de equivalencia de pivotes es el esquema de partición por bola. El objetivo de la partición por bola es generar una partición de S en dos subconjuntos S_1 y S_2 respecto a un pivote $p \in U$ (Zezula, 2006). Sea d_m la mediana de $\{d(x,p), \forall x \in S\}$, entonces:

$$S_1 = \{x \in S \,|d(x,p) \leq d_m\} \tag{15}$$

$$S_2 = \{x \in S \,|d(x,p) \geq d_m\} \tag{16}$$

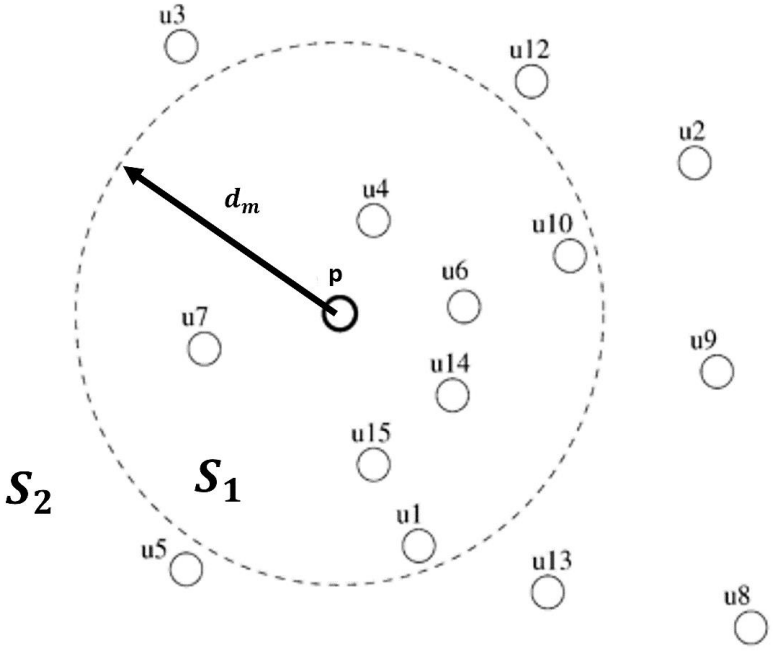


Figura 3.- Ejemplo de partición por bola. Figura modificada de (Chávez et al., 2001).

Existe un gran número de heurísticas para la selección de pivotes, sin embargo, una de las más comunes es la selección aleatoria. De igual manera, la distancia d_m puede elegirse en base a diferentes heurísticas, por ejemplo: media, moda, máximo de las distancias, etc.

3.1.2 Partición por hiperplano

Un ejemplo de una relación de equivalencia compacta es el esquema de partición por hiperplano. El objetivo del esquema de partición por hiperplano es generar una partición de S en dos subconjuntos S_1 y S_2, respecto a dos centros c_1 y c_2, donde:

$$S_1 = \{x \in S \,|d(x, c_1) \leq d(x, c_2)\} \tag{17}$$

$$S_2 = \{x \in S \,|d(x, c_2) \leq d(x, c_1)\} \tag{18}$$

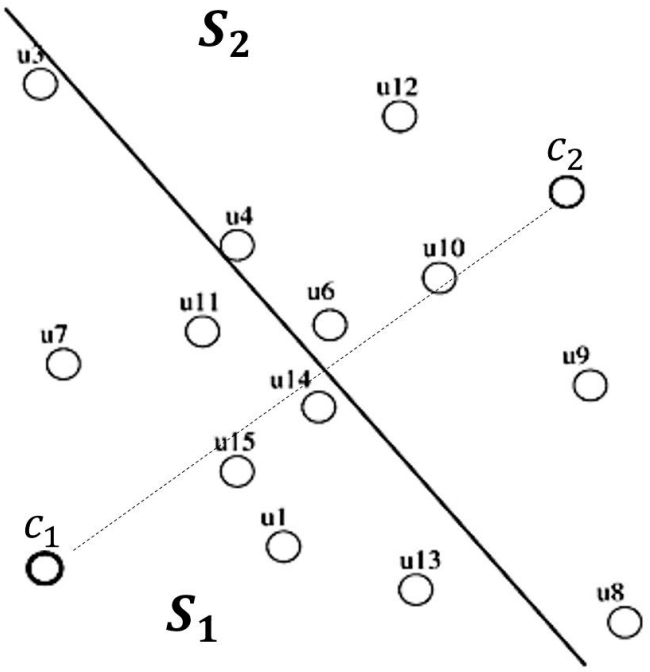


Figura 4.- Ejemplo de partición por hiperplano. Figura modificada de (Chávez et al., 2001).

3.2 Índices para espacio métrico

Sea (U, d) un espacio métrico, donde U es el universo de objetos válidos, $S \subseteq U$ un subconjunto finito de U (base de datos) y $d: U \times U \rightarrow \mathbb{R}^+$ una función de distancia la cual modela el concepto de similitud. Los índices para espacio métrico son estructuras de datos que tienen disponible como única información para la reducción del espacio de búsqueda la distancia entre los objetos, es decir, no se tiene disponible información adicional de los objetos como coordenadas, propiedades geométricas, etc.

Debido a que generalmente el costo computacional de la evaluación de la función de distancia es elevado, el objetivo de los índices es reducir el número de evaluaciones de la función de distancia durante el proceso de consulta.

3.2.1 Estrategias para reducir la cantidad de evaluaciones de la función de distancia

Existen diversas estrategias para reducir el número de evaluaciones de la función de distancia durante el proceso de consulta. En espacios métricos, como la única información disponible es la distancia entre los objetos, las estrategias consisten en guardar las distancias entre los objetos calculadas previamente durante el proceso de indexado, y utilizar la desigualdad del triángulo para determinar cotas inferiores y superiores de distancias entre otros objetos (Zezula, 2006). En el caso de espacios vectoriales, se puede utilizar como información adicional las coordenadas de los objetos para explotar propiedades geométricas y reducir aún más el número de evaluaciones de la función de distancia (Chávez et al., 2001).

A continuación se muestran algunas de las reglas más comunes para la reducción del número de evaluaciones de la función de distancia basadas en la desigualdad del triángulo.

3.2.1.1 Objeto-Pivote

Sea (U, d) un espacio métrico, $d: U \times U \rightarrow \mathbb{R}^+$ una función de distancia y $q, p, o \in U$ (Zezula, 2006), entonces:

$$|d(q,p) - d(p,o)| \leq d(q,o) \leq d(q,p) + d(p,o) \tag{19}$$

3.2.1.2 Objeto- Rango

Sea (U, d) un espacio métrico, $d: U \times U \rightarrow \mathbb{R}^+$ una función de distancia y $p, o \in U$ tal que $r_l \leq d(o,p) \leq r_h$, $q \in U$ y la distancia $d(q,p)$ (Zezula, 2006), entonces:

$$max\{d(q,p) - r_h, r_l - d(q,p), 0\} \leq d(q,o) \leq d(q,p) + r_h \tag{20}$$

3.2.1.3 Doble pivote

Sea (U, d) un espacio métrico, $d: U \times U \rightarrow \mathbb{R}^+$ una función de distancia y $p_1, p_2, o \in U$ tal que $d(o, p_1) \leq d(o, p_2)$. Sea $q \in U$ y la distancia $d(q, p_1)$ y $d(q, p_2)$ (Zezula, 2006), entonces:

$$max\left\{\frac{d(q,p_1) - d(q,p_2)}{2}, 0\right\} \leq d(q,o) \tag{21}$$

3.2.1 Vantage Point Tree

El índice métrico Vantage Point Tree (Yianilos, 1993), es un índice para funciones de distancia continuas (aunque puede utilizarse para funciones de distancia discretas), el cual realiza una descomposición de los datos en forma de árbol binario balanceado y utiliza la relación de equivalencia de pivotes, realizando una partición de un conjunto S en dos subconjuntos S_1 y S_2, respecto a un pivote $p \in U$. La construcción de este índice consiste en seleccionar un elemento como raíz del árbol (pivote) y calcular la distancia de este elemento a todos los elementos de la base de datos. Posteriormente, se calcula la mediana de las distancias d_m y la partición se realiza de la siguiente forma:

$$S_1 = \{x \in S \,|d(x,p) \leq d_m\} \tag{22}$$

$$S_2 = \{x \in S \,|d(x,p) \geq d_m\} \tag{23}$$

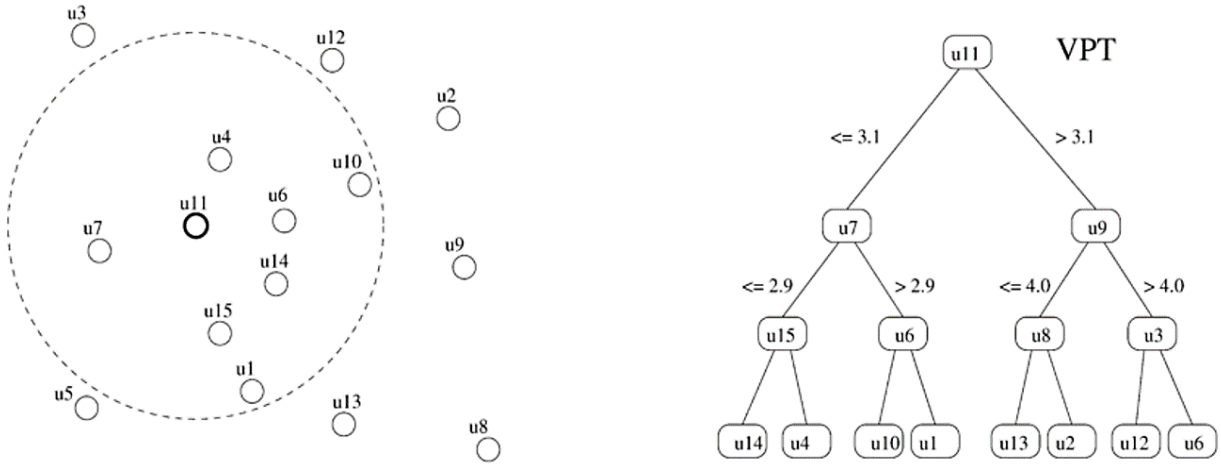


Figura 5.- Partición de los datos y estructura de Vantage Point Tree. Figura de (Chávez et al., 2001).

Diversas heurísticas pueden ser aplicadas para la selección de la distancia d_m, por ejemplo, se puede utilizar la media de las distancias para datos de dimensión elevada (Chávez et al., 2001), sin embargo, esto tiene como consecuencia que la estructura de datos no sea balanceada.

El proceso de consulta para una consulta de rango $R(S, q)$ con radio r, consiste en un recorrido del árbol donde para cada nodo se evalúa la distancia $d(q, p)$, si $d(q, p) \leq r$, p se agrega a la lista de resultados. Para determinar si se debe de visitar un subárbol se utiliza la estrategia de reducción de evaluación de distancia objeto-rango, visitando el subárbol izquierdo si $\max\{d(q, p) - d_m, 0\} \leq r$ y el subárbol derecho si $\max\{d_m - d(q, p), 0\} \leq r$. Sea n el total de elementos a indexar, el espacio utilizado por el índice VPT es $O(n)$, el tiempo de construcción $O(n \log n)$ y el tiempo de consulta $O(\log n)$ para un radio de búsqueda muy pequeño.

3.2.2 BKT

El índice BKT (Burkhard y Keller, 1973), es un índice diseñado para funciones de distancias discretas, el cual utiliza la relación de equivalencia de pivotes. La construcción de este índice se realiza de la siguiente manera: sea S nuestra base de datos, se selecciona un pivote $p \in S$ como la raíz del árbol, después se calcula la distancia de p a todos los elementos de la base de datos. Posteriormente, para cada distancia $i \geq 0$ se genera un subconjunto $S_i = \{o \in S \mid d(o, p) = i\}$. Finalmente, para cada S_i, se repite el mismo procedimiento en forma recursiva.

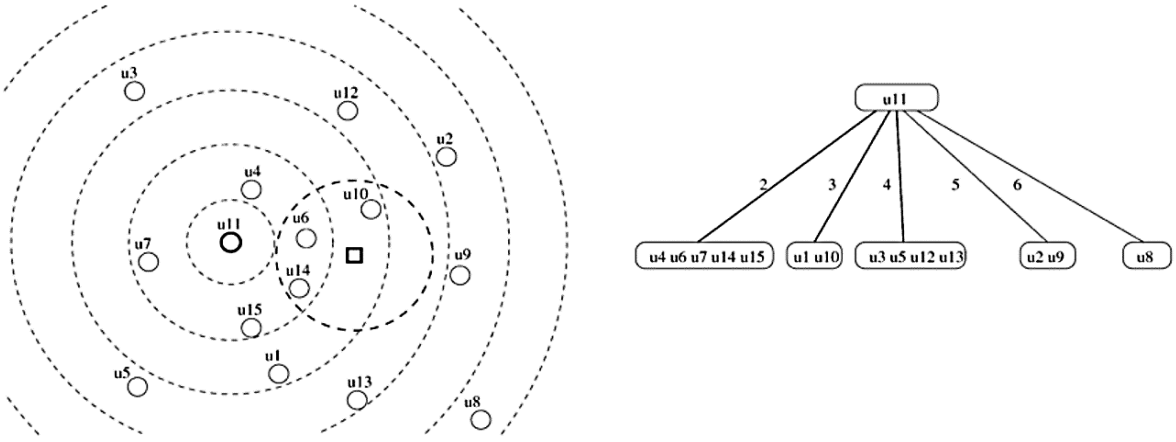


Figura 6.- Partición de los datos y estructura Burkhard Keller Tree. Figura de (Chávez et al., 2001).

El proceso de consulta de rango $R(S, q)$ con radio r, consiste en un recorrido del árbol donde para cada nodo se evalúa la distancia $d(q, p)$. Si $d(q, p) \leq r$, p se agrega a la lista de resultados. Se visitan todos los subárboles i tal que $d(p, q) - r \leq i \leq d(p, q) + r$, continuando la búsqueda en forma recursiva. Sea n el total de elementos a indexar, el espacio utilizado por el índice BKT es $O(n)$, el tiempo de construcción $O(n \log n)$ y el tiempo de consulta $O(n^{\alpha})$ para $0 < \alpha < 1$.

3.2 Índices para espacio vectorial

Un espacio vectorial es un caso específico de un espacio métrico (U, d), donde los elementos de este espacio son tuplas de números reales. En el caso de espacios vectoriales, además de las distancias entre los objetos, tenemos como información adicional las coordenadas de los objetos, las cuales pueden ser utilizadas para explotar propiedades geométricas y reducir aún más el número de evaluaciones de la función de distancia.

Una de las funciones de distancias más utilizadas para espacios vectoriales es la familia de funciones L_p o distancias Minkowski. Dado un espacio vectorial de k dimensiones (cada elemento de este espacio es una tupla de k números reales), la familia de distancias L_p se define como:

$$L_p[(x_1, \dots, x_k), (y_1, \dots, y_k)] = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \tag{24}$$

A continuación se muestran dos índices para espacio vectorial los cuales son ampliamente utilizados en la literatura.

3.2.1 R-Tree

El índice R-Tree (Guttman, 1984), es una estructura de datos que tiene como objetivo realizar una descomposición jerárquica de los datos en forma de árbol. Los nodos hojas de este árbol tienen forma (I, *object_id*), donde I es un rectángulo delimitador mínimo (*MBR – Minimum bounding rectangle*) y *object_id*

es un apuntador al objeto. Los nodos internos del árbol tienen forma (I, child-pointer), donde I es un MBR y child-pointer es un apuntador a cada nodo hijo. Esto es, cada MBR se almacena como un nodo interno.

El árbol se puede construir insertando un nodo a la vez utilizando una estrategia *bottom-up*. También puede construirse utilizando el método *bulkloading*. El primer paso de este método es seleccionar una dimensión y ordenar los datos de esta, para posteriormente realizar una partición de los datos. Este proceso se realiza en forma recursiva utilizando las dimensiones restantes. Una vez realizado este proceso de partición, cada partición se mapea a un nodo hoja y se construye el árbol utilizando una estrategia bottom-up (Kim y Patel, 2010).

Esta estructura tiene dos parámetros: máxima cantidad de elementos en un nodo (M) y mínima cantidad de elementos en un nodo (m).

El proceso de búsqueda se realiza de la siguiente manera: dado un rectángulo de búsqueda R, se revisa el nodo raíz y se verifican los MBR que intersectan a R. Para cada MBR que intersecta a S, se repite el procedimiento en forma recursiva. Sea n el total de elementos a indexar, el espacio utilizado por el índice R-Tree es $O(n)$, el tiempo de construcción $O(n \log n)$ y el tiempo de consulta $O(n)$ para búsquedas de intersección.

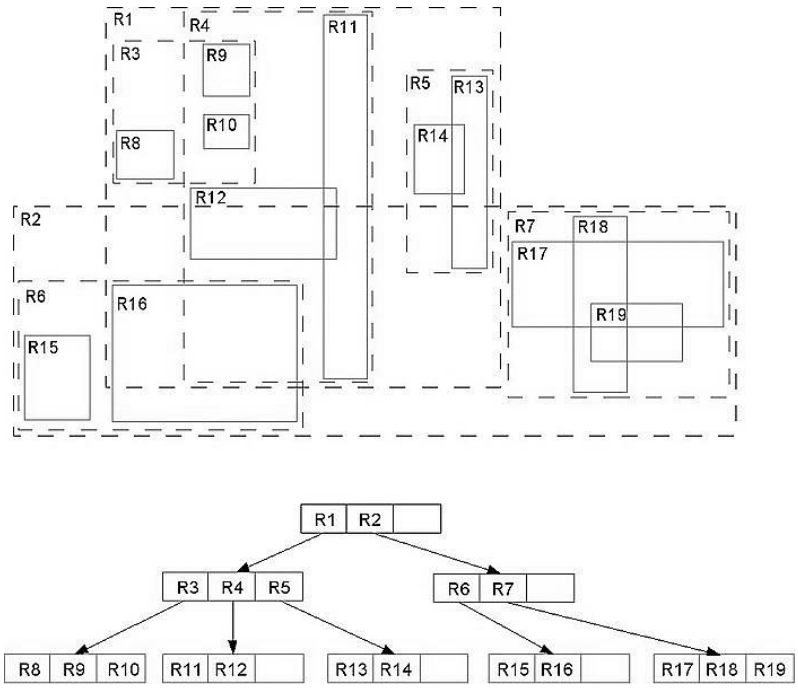


Figura 7.- Partición de los datos y estructura de R-Tree. Figura recuperada el 29/07/17 de: https://en.wikipedia.org/wiki/File:R-tree.svg

3.2.2 KD-Tree

El índice KD-*Tree* o árbol de k dimensiones (Bentley, 1975), es una estructura de datos que tiene como objetivo generalizar el concepto de árbol binario de búsqueda para objetos de k dimensiones. En la **Figura 8** se puede observar la estructura de un KD-*Tree* de 2 dimensiones.

La construcción de este árbol se realiza en forma recursiva de la siguiente manera: en la raíz del árbol se realiza una partición de los objetos en dos conjuntos mediante un hiperplano. Cada partición se asigna a cada hijo del nodo. Este procedimiento se realiza en forma recursiva. Un nodo interno tiene dos nodos hijos, a excepción de las hojas que pueden tener uno o dos.

En cada nodo se selecciona un eje (usualmente el que tiene mayor varianza) y un valor de partición, tal que el hiperplano de partición es perpendicular al eje de partición. Existen diferentes formas de seleccionar el valor de partición pero comúnmente se utiliza la mediana de los datos en el eje seleccionado.

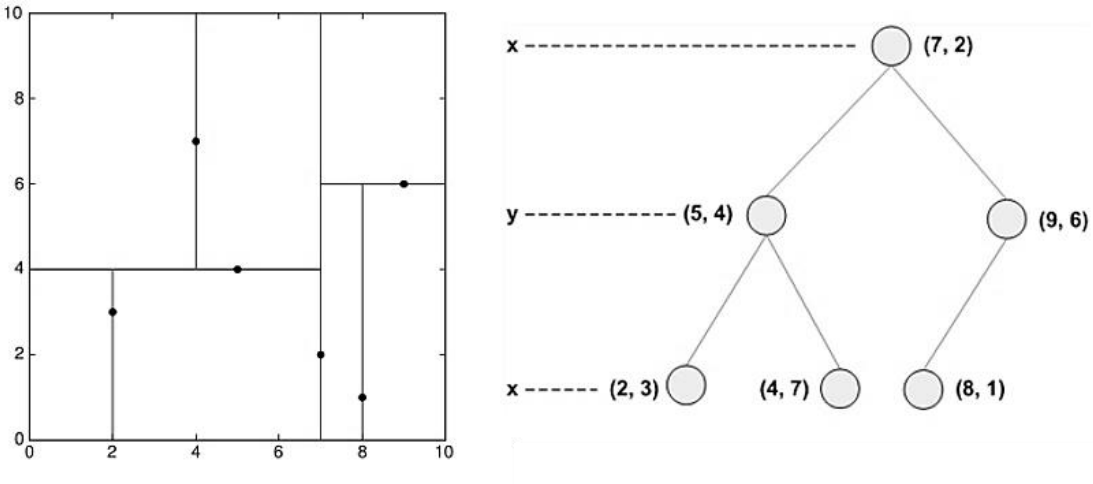


Figura 8.- Partición de los datos y estructura de KD Tree. Figuras recuperadas el 29/07/17 de: https://commons.wikimedia.org/wiki/File:Tree_0001.svg, https://commons.wikimedia.org/wiki/File:Kdtree_2d.svg

El proceso de búsqueda del vecino cercano de un punto, consiste en realizar un recorrido desde la raíz. En cada nodo interno, se compara el valor de partición del eje de partición para identificar a que conjunto pertenece. Este procedimiento se repite hasta llegar a una hoja. Sin embargo, en este punto se tiene solo un candidato al vecino cercano, por lo que hay que recorrer otros nodos hojas mediante *backtracking*. Una consulta de rango rectangular de dos dimensiones en un conjunto de puntos P, entrega

el conjunto de puntos que se encuentran dentro de un rectángulo de consulta $[x, x'] \times [y, y']$, donde un punto $p = (p_x, p_y) \in P$ se encuentra dentro del rectángulo de consulta si $p_x \in [x, x'] \wedge p_y \in [y, y']$ (Berg et al., 2008).

Sea n el total de elementos a indexar, el espacio utilizado por esta estructura es $O(n)$, el tiempo de construcción $O(n \log n)$ y el tiempo de consulta para una consulta de rango rectangular es $O(n^{1-\frac{1}{d}} + k)$, donde d es la dimensión de los datos y k es el número de elementos entregados al usuario (Berg et al., 2008).

3.3 Índices Invertidos

Un índice invertido es una estructura de datos que realiza un mapeo de contenido a identificadores de un objeto de una base de datos, los cuales son almacenados en listas llamadas listas invertidas. Este índice está compuesto por un vocabulario y una lista por cada elemento del vocabulario (lista invertida), que contiene los identificadores de todos objetos que contienen esta palabra, así como información adicional (D. Zhang et al., 2009).

En una base de datos de archivos de texto, un vocabulario está compuesto por el conjunto de todas las palabras diferentes de toda la colección de archivos de texto en la base de datos, y las listas invertidas contienen los identificadores de los documentos que tienen esa palabra. En la **Figura 9** se muestra un ejemplo de un índice invertido para archivos de texto. Por ejemplo, de acuerdo con la **Figura 9**, los archivos de texto 2, 31, 54 y 101 contienen el termino Calpurnia.

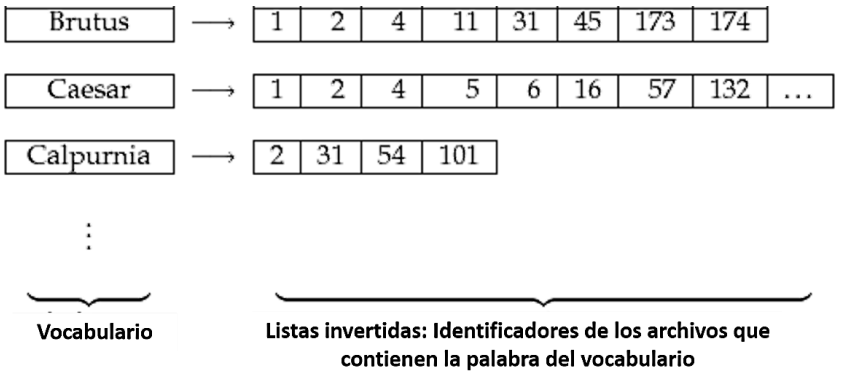


Figura 9.- Ejemplo de índice invertido para archivos de texto. Figura recuperada y modificada el 29/07/17 de: https://i2.wp.com/nlp.stanford.edu/IR-book/html/htmledition/img43.png

Dado el índice invertido de la figura anterior, el proceso de consulta se realiza de la siguiente manera. Para cada palabra del documento de consulta proporcionado por el usuario, se localiza su lista invertida, lo cual puede realizarse en tiempo constante $O(1)$ por palabra (en el caso promedio), utilizando una tabla hash. Posteriormente, se puede realizar la intersección o unión de las listas invertidas de identificadores y puede entregarse al usuario el documento cuyo identificador tuvo la máxima frecuencia, para el caso de una búsqueda aproximada del vecino más cercano.

Una alternativa para reducir el tiempo de consulta y tiempo de construcción de los índices invertidos (e índices en general), es utilizar varios procesadores para estas estructuras de datos. Uno de los métodos más utilizados para la construcción y consulta de índices invertidos utilizando varios procesadores es la utilización de la metodología MapReduce.

3.3.1 MapReduce

MapReduce es un modelo de programación orientado a aplicaciones de cómputo distribuido donde es necesario realizar operaciones en bases de datos de gran tamaño (Jang et al., 2015). Este modelo divide un trabajo en múltiples tareas independientes, las cuales pueden ser ejecutadas en paralelo en sistemas de cómputo distribuido.

Es importante señalar que para realizar un trabajo mediante MapReduce, este trabajo debe ser planteado como una secuencia de operaciones *Map* y *Reduce*. La operación *Map*, toma un conjunto de tuplas (llave, valor) y aplica una función a cada tupla, transformando el conjunto de tuplas inicial en otro conjunto intermedio de tuplas (llave, valor). Posteriormente, todos los pares con la misma llave son agrupados, generando tuplas (llave, lista). La operación *Reduce* toma esta salida y combina estas tuplas en un conjunto de tuplas más pequeño (IBM, 2017).

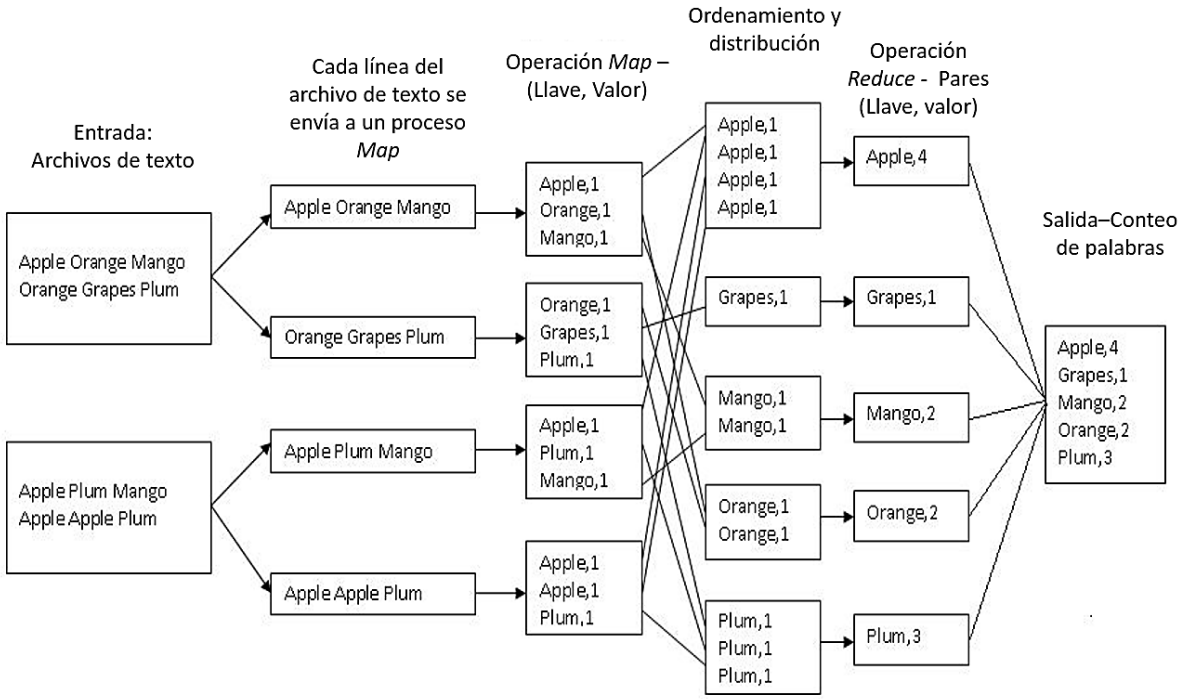


Figura 10.- Conteo de palabras mediante metodología MapReduce. Figura recuperada el 29/07/17 de: http://i.stack.imgur.com/DBu97.jpg

Apache Hadoop es uno de los conjuntos de herramientas de trabajo para cómputo distribuido más utilizados, diseñado para realizar operaciones en bases de datos de gran tamaño utilizando el modelo MapReduce. Apache Hadoop tiene tres componentes principales: *Hadoop Distributed File System* (sistema de archivos distribuidos), Hadoop YARN (conjunto de herramientas de trabajo para la calendarización y manejo de recursos en clusters) y Hadoop MapReduce (librería basada en YARN para aplicaciones de MapReduce) (Foundation, 2017c). Una de las desventajas de Hadoop es que durante sus primeros años, se realizaban escrituras y lecturas a disco duro durante operaciones intermedias *Map* y *Reduce*, incrementando el tiempo de estas. En los últimos años, un gran número de librerías y proyectos fueron incorporados al ecosistema de Apache Hadoop para mejorar el desempeño de este. Uno de estos proyectos es Apache Spark.

Apache Spark es un sistema de propósito general para el procesamiento de bases de datos de gran tamaño en sistemas de cómputo distribuido. Una de las ventajas de Apache Spark ante Apache Hadoop, es que Apache Spark permite realizar las operaciones MapReduce tanto en disco duro como en memoria RAM, realizando estas operaciones 100 veces más rápido en memoria RAM y 10 veces más rápido en disco

duro en comparación con Apache Hadoop (Foundation, 2017a). En la **Figura 11** se muestra la arquitectura de una aplicación de Apache Spark.

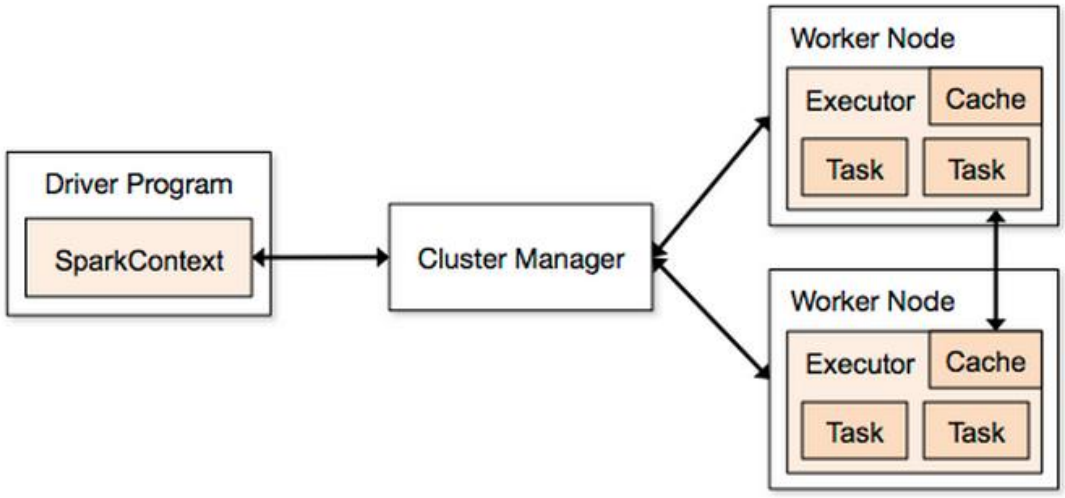


Figura 11.- Arquitectura de aplicación de Apache Spark. Figura recuperada el 29/07/17 de: http://i.stack.imgur.com/tjSdG.png

Para ejecutar un programa de Apache Spark (*Driver Program*), el objeto *SparkContext* se comunica con el manejador de *clusters*, para que este asigne los recursos necesarios para la ejecución de la aplicación. Después, Apache Spark adquiere *executors* en los nodos del *cluster*, donde un *Worker Node* puede ser un nodo físico o una máquina virtual, los cuales ejecutarán las operaciones y guardarán los resultados de estas. Finalmente, se envía el código de la aplicación a los *executors* y el objeto *SparkContext* envía las tareas a los *executors* para ser ejecutadas.

3.4 Índices sucintos

Un índice sucinto es una estructura de datos que utiliza una cantidad de información cercana a la entropía del peor de los casos (mínimo teórico de información) y que permite realizar un conjunto de operaciones de consulta (Navarro y Sadakane, 2014).

La entropía del peor de los casos H_{wc} (Navarro, 2016), es el mínimo número de bits requeridos para asignar un identificador a un elemento de un conjunto U, tal que todos los códigos sean de la misma longitud y se define como:

$$H_{wc} = log_2\lceil|U|\rceil \quad (25)$$

3.4.1 Índice Sarray

El índice Sarray (Okanohara y Sadakane, 2007), es un índice sucinto para bitmaps (vectores de bits). Sea B un bitmap, este índice sucinto requiere que el bitmap no sea denso, es decir, si m es el número de unos del bitmap B y n es la longitud de B, entonces $m \ll n$.

Las operaciones de consulta que soporta esta estructura son:

- $access(B, i)$: regresa el bit $B[i]$para $1 \leq i \leq n$
- $rank_v(B, i)$: regresa el número de ocurrencias del bit $v \in \{0,1\}$ en $B[1, i]$ para $0 \leq i \leq n$
- $select_v(B, j)$: regresa la posición en B del j-ésimo bit $v \in \{0,1\}$ para cualquier $j \geq 0$

Esta estructura utiliza $m(1.92 + \log\frac{n}{m}) + o(m)$ de espacio de almacenamiento. La complejidad en el peor de los casos para la operación $select_v(B, j)$ es $O(\frac{log\ ^4 m}{\log n})$.

3.5 Maldición de la dimensión

Es bien conocido que el desempeño de las búsquedas de proximidad de objetos representados mediante vectores de dimensión elevada se aproxima al desempeño de una búsqueda secuencial conforme aumenta la dimensión de estos, tanto para índices de espacio métrico o espacio vectorial; a este efecto se le conoce como la maldición de la dimensión (Chávez et al., 2001). En el caso de espacios vectoriales, en general los índices tienen una dependencia exponencial en relación con la dimensión de la representación del objeto, como en el caso del índice KD-Tree.

Un punto importante es que la dimensión intrínseca (real) de los datos, no siempre corresponde con la dimensión de la representación de estos. Por ejemplo, se pueden tener objetos con una representación de dimensión elevada pero que los datos se encuentren en un plano de 2 dimensiones,

logrando un buen desempeño en tiempo de consulta utilizando índices de espacio vectorial, debido a que la dimensión intrínseca de los datos es pequeña (Chávez et al., 2001).

Si realizamos un histograma de las distancias de todos los elementos de la base de datos $x \in X$ respecto a un elemento pivote p, cuando aumenta la dimensión intrínseca (real) de los datos, la media de las distancias de los elementos de la base de datos respecto a p aumenta y la varianza disminuye. Dado que todas las distancias tienden a aglomerarse muy cerca de la media, nuestro índice no puede discriminar los elementos, ya que todos se encuentran prácticamente a la misma distancia y ante una consulta de radio, solo una cantidad pequeña de puntos puede descartarse.

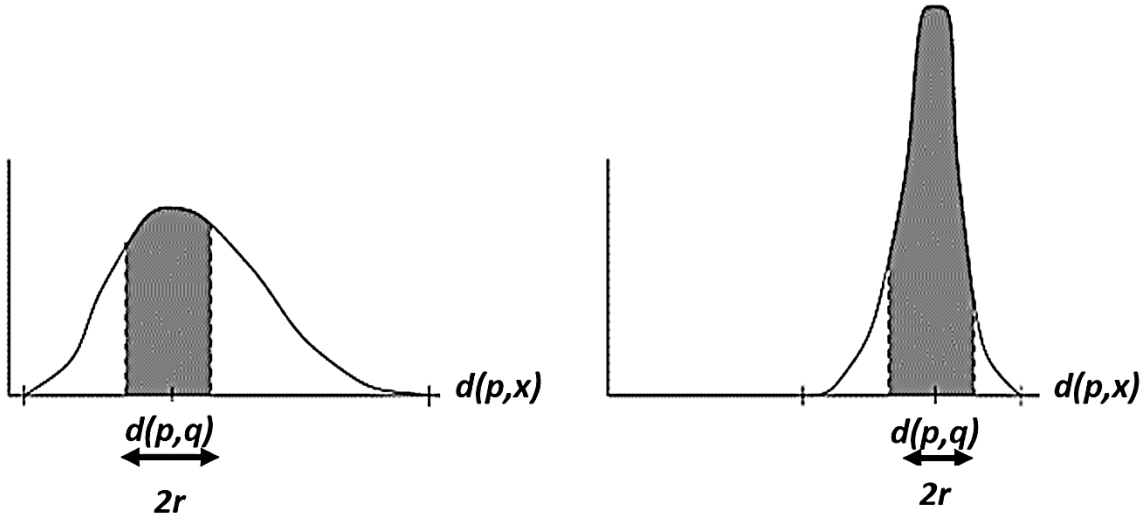


Figura 12.- Histograma de distancias. Izquierda: dimensión intrínseca pequeña. Derecha: dimensión intrínseca alta. Figura de (Chávez et al., 2001).

La dimensión intrínseca de un espacio métrico se define como (Chávez et al., 2001):

$$\rho = \frac{\mu^2}{2\sigma^2} \tag{26}$$

Donde μ y σ es la media y la desviación estándar del histograma de distancias, respectivamente.

Existen diversas opciones para reducir este efecto, por ejemplo, aplicar técnicas de reducción de dimensión, el diseño de estructuras de datos especializadas para datos de dimensión elevada, entre otras (D. Zhang et al., 2009).

Capítulo 4. Nubes de puntos

Este capítulo presenta algunas representaciones de objetos multimedia mediante nubes de puntos que actualmente son utilizadas en la literatura. Además, se muestran algunos de los desafíos que se presentan al utilizar esta representación. Finalmente, se dedica una sección a una representación alternativa de una nube de puntos mediante una cadena binaria.

4.1 Nubes de puntos para audio

En 2003, Wang presenta un sistema de recuperación de audio basado en contenido, el cual tiene como objetivo identificar por medio de un segmento de audio proporcionado por el usuario (consulta), el archivo de audio de la base de datos al que pertenece este segmento, donde este segmento de audio puede estar sujeto a ruido y/o distorsión.

Wang representa cada archivo de audio de la base de datos mediante una nube de puntos (constelación de puntos). Para calcular la nube de puntos de un archivo de audio, Wang calcula el espectrograma del archivo de audio, el cual es una representación en tiempo, frecuencia y amplitud de la energía de la señal.

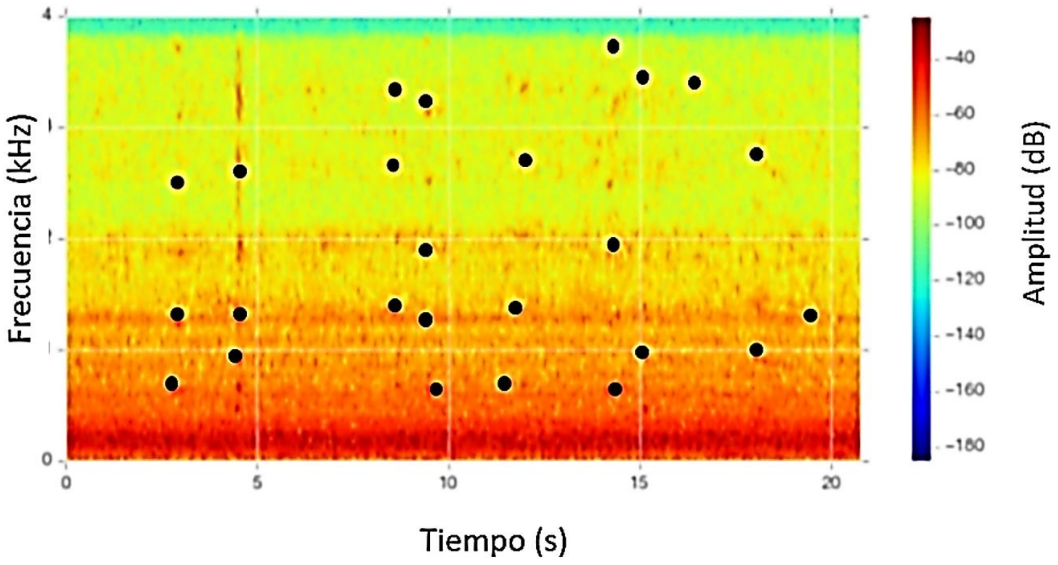


Figura 13.- Nube de puntos generada mediante el esquema propuesto por Wang para un archivo de audio. Figura obtenida el 29/07/17 y modificada de: http://www.mathisintheair.org/wp/wp-content/uploads/2017/01/spectrogram-744x298.png

Dado este espectrograma, se considera como candidato un punto si tiene una cantidad elevada de energía comparada con su vecindario (se genera una región alrededor de este). Posteriormente, para generar la nube de puntos, se eligen los puntos con mayor amplitud del conjunto de puntos candidatos, ya que estos tienen una mayor probabilidad de mantenerse ante distorsión o ruido.

Para realizar el proceso de búsqueda, Wang construye a partir de la nube de puntos del archivo de audio un conjunto de tuplas (hash combinatorio), las cuales se utilizan para representar el archivo de audio. Para generar una tupla, se selecciona un punto de la nube de puntos (punto ancla) y se asocia una región objetivo. Posteriormente, cada punto ancla se asocia con los puntos de su región objetivo formando tuplas de la forma $(freq_1, freq_1, \Delta_{time})$. Esto se realiza para cada archivo de la base de datos. Las tuplas posteriormente se indexan en una tabla hash. En la **Figura 14** se muestra un ejemplo de la generación de tuplas para una nube de puntos.

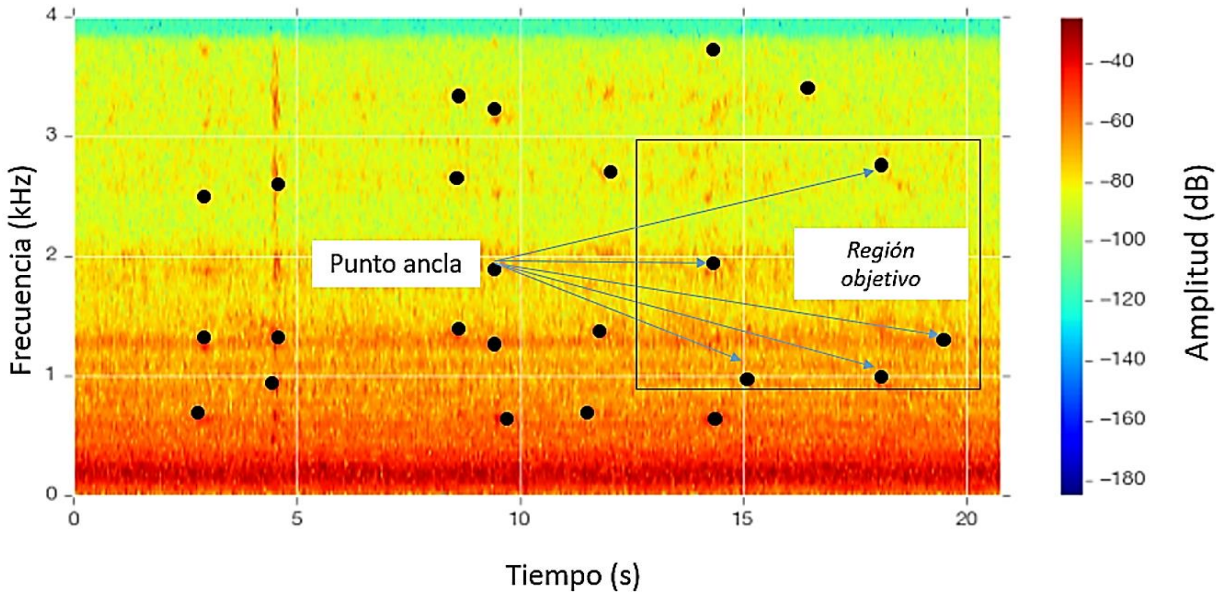


Figura 14.- Generación de hash combinatorio. Figura obtenida el 29/07/17 y modificada de: http://www.mathisintheair.org/wp/wp-content/uploads/2017/01/spectrogram-744x298.png

4.2 Nubes de puntos para imágenes

En el área de procesamiento digital de imágenes, un punto de interés es un punto que captura información relevante de una característica de la imagen. Estos puntos tienen que ser definidos formalmente, tienen que tener una posición (coordenadas) bien definidas y deben ser estables ante

variaciones locales o globales de la imagen como transformaciones, cambios en iluminación, entre otros (Lindeberg, 2013).

El conjunto de puntos de interés de una imagen, junto con la información del vecindario de cada punto de interés, son utilizados para generar un vector de características o descriptor. El objetivo del descriptor es ser robusto ante transformaciones locales/globales de la imagen y ante ruido.

En nuestro caso en específico, solo estamos interesados en la detección de puntos de interés de imágenes y no en el descriptor, ya que el conjunto de puntos de interés de cada imagen se utilizará como una nube de puntos que representa a la imagen. Una de las desventajas de utilizar descriptores, es que estos tienen dimensión elevada, lo que puede ocasionar que índices métricos o de espacio vectorial no tengan un buen desempeño debido al problema de la maldición de la dimensión.

4.2.1 Descriptor SIFT

El algoritmo SIFT (Lowe, 2004), es detector de puntos de interés y descriptor, utilizado ampliamente en la literatura, cuyos puntos de interés son invariantes a rotación, cambios de escala y parcialmente invariantes ante iluminación, ruido y transformaciones afines. El descriptor SIFT (vector característico) es un vector de números reales de 128 dimensiones.

El algoritmo SIFT puede dividirse en las siguientes etapas:

- Detección de máximos en espacio de escalas.
- Localización de puntos de interés.
- Asignación de orientación.
- Descriptor de punto de interés.

El proceso de detección de máximos en el espacio de escalas, utiliza un filtrado en cascada para identificar puntos de interés candidatos. El espacio de escala se define como:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{27}$$

Donde $G(x, y, \sigma)$ es una Gausiana, σ es un parámetro de escala, $I(x, y)$ es la imagen de entrada y $*$ es la operación de convolución. Para calcular candidatos de puntos de interés invariantes a cambios de escala de manera eficiente, Lowe propone utilizar diferencias de gaussianas, el cual se utiliza como una aproximación al Laplaciano Gaussiano, ya que este es invariante a escala. Las diferencias de gaussianas se calculan para diferentes octavas de la imagen (**Figura 15**).

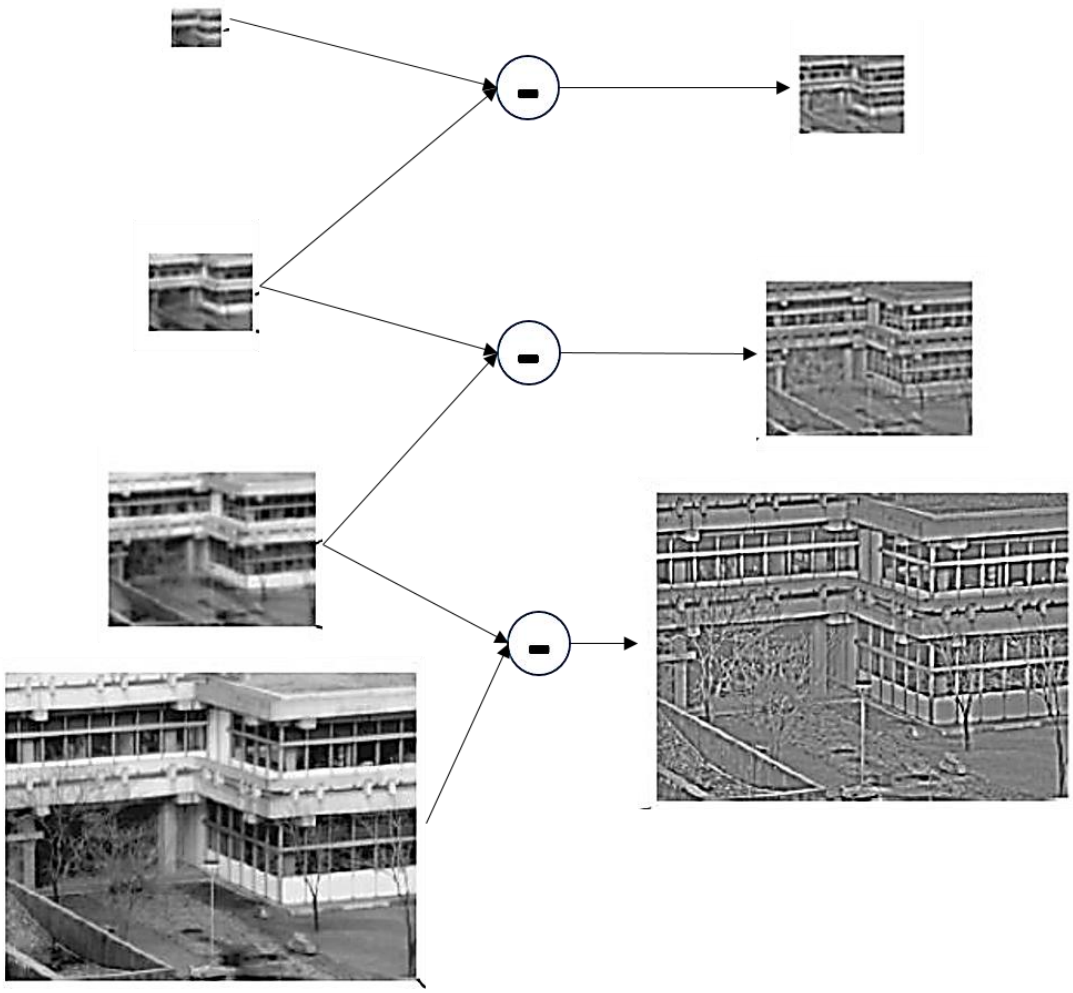


Figura 15.- Aproximación al Laplaciano Gaussiano. Figura obtenida y modificada el 29/07/17 de: https://raw.githubusercontent.com/LoserSun/Computer-Vision-Course-Note/master/relevant%20materials/picture/5.5.JPG

La localización de los puntos de interés consiste en identificar de las diferencias de gaussianas, los máximos locales. Para esto, cada punto se compara con sus ocho vecinos (de su misma imagen) y con nueve vecinos en una escala inferior y superior (diferencias de gaussianas). Este punto se considera como punto de interés solo si es mayor a todos los puntos vecinos, considerando los vecinos de la imagen a la que pertenece y sus nueve vecinos de una escala inferior y superior. En la siguiente imagen se muestra un ejemplo de este proceso.

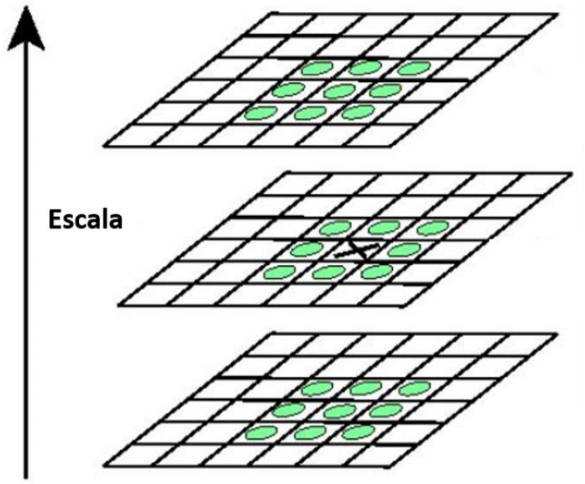


Figura 16.- Identificación de puntos de interés de diferencias de gaussianas para algoritmo SIFT. Figura recuperada el 29/07/17 de: https://raw.githubusercontent.com/LoserSun/Computer-Vision-Course-Note/master/relevant%20materials/picture/5.20.JPG

Figura 17.- Puntos de interés de imagen calculados mediante algoritmo SIFT. Los puntos de interés son los centros de cada círculo. Figura recuperada el 29/07/17 de: http://i.stack.imgur.com/0SuqI.jpg

4.2.2 Descriptor SURF

El algoritmo SURF (Bay et al., 2008), es un detector de puntos de interés y un descriptor invariante a cambios de escala y rotación. Uno de los objetivos del algoritmo SURF, es reducir el tiempo de ejecución de descriptores propuestos en la literatura como el algoritmo SIFT, GLOH o el detector Harris, entre otros.

Para esto, Bay et al. aproximan el Laplaciano Gaussiano, el cual es invariante a cambios de escala, mediante la matriz Hessiana:

$$\mathcal{H}(x,\sigma) = \begin{bmatrix} L_{xx}(\boldsymbol{x},\sigma) & L_{xy}(\boldsymbol{x},\sigma) \\ L_{xy}(\boldsymbol{x},\sigma) & L_{yy}(\boldsymbol{x},\sigma) \end{bmatrix} \tag{28}$$

Donde $\boldsymbol{x} = (x, y)$, $L_{xx}(x,\sigma)$ es la convolución de la derivada de la función gausiana de segundo orden $\frac{\partial^2}{dx^2} g(\sigma)$ con la imagen I en el punto $\boldsymbol{x}$ y σ es un factor de escala. $L_{xy}(\boldsymbol{x},\sigma)$ y $L_{yy}(\boldsymbol{x},\sigma)$ se definen de manera similar.

Para reducir aún más la complejidad del cálculo de la aproximación del Laplaciano Gaussiano, Bay et al. aproximan la matriz hessiana mediante un conjunto de filtros y utilizan imagen integrales. El algoritmo utilizado para calcular las imágenes integrales (Crow, 1984), permite obtener la suma de intensidades en cualquier región rectangular de esta en tiempo constante. Sea I una imagen, T su imagen integral y $p \in T$ un punto de la imagen integral. El punto $p = (x, y)$ se define como la suma de todos los pixeles en un rectángulo definido por el punto de interés p y la esquina inferior izquierda de la imagen integral T. La suma de intensidades de cualquier región rectangular de I acotada por los puntos $x_{izquierda}$, $x_{derecha}$, y_{abajo} y y_{arriba} es $T[x_{derecha}, y_{arriba}] - T\left[x_{derecha}, y_{abajo}\right] - T\left[x_{izquierda}, y_{arriba}\right] + T[x_{izquierda}, y_{abajo}]$.

La ventaja de este acercamiento es que la convolución de una imagen con estos filtros puede realizarse en forma eficiente utilizando imágenes integrales.

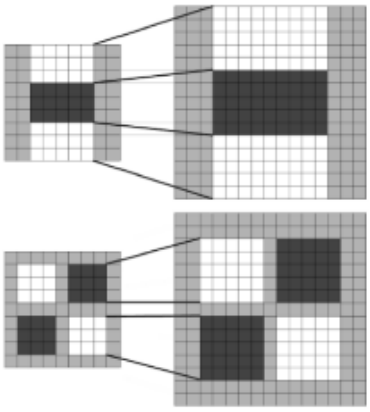


Figura 18.- Filtros para aproximar matriz hessiana. Figura recuperada el 29/07/17 de: https://imgur.com/bPc8bfN

Para localizar los puntos de interés en la imagen, el algoritmo SURF trabaja en el espacio de escalas, al igual que el algoritmo SIFT de Lowe. En el caso de SURF, en lugar de reducir el tamaño de la imagen, este algoritmo aumenta el tamaño de la imagen en cada nivel; el objetivo de este acercamiento es reducir la complejidad computacional utilizando las imágenes integrales y explotando que cada filtro puede aumentar su tamaño en tiempo constante. El espacio de escala se divide en octavas, donde una octava representa la convolución de la misma imagen y un filtro que incrementa de tamaño.

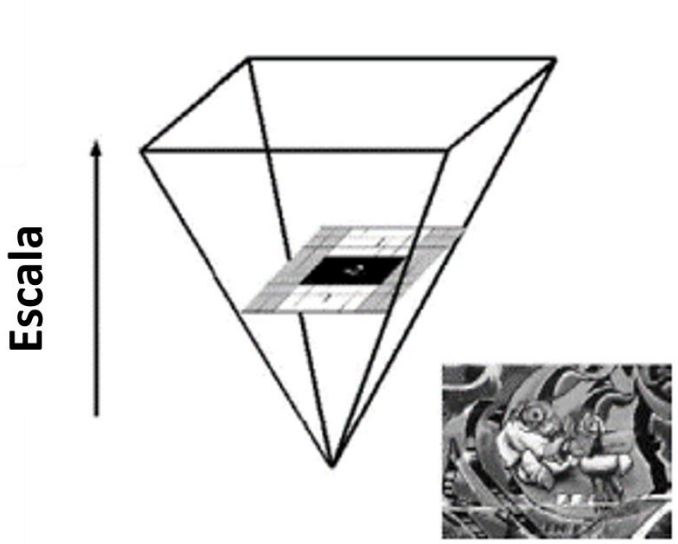


Figura 19.- Espacio de escala en algoritmo SURF. Figura recuperada el 29/07/17 de: http://imgur.com/Xw9dJSb

Figura 20.- Puntos de interés calculados con algoritmo SURF. Los puntos de interés son los centros de cada círculo. Figura recuperada el 29/07/17 de: http://opencv-python-tutroals.readthedocs.io/en/latest/_images/surf_kp1.jpg

Para localizar los puntos de interés, se seleccionan vecindarios de tamaño $3x3x3$ (similar al algoritmo SIFT) dentro del espacio de escala y se aplica un algoritmo. En la **Figura 20** se muestra un ejemplo de los puntos de interés de una imagen calculados mediante el algoritmo SURF.

4.3 Desafíos de representación

La representación de objetos multimedia mediante nubes de puntos presenta diversos desafíos. Algunos de los principales desafíos es que las nubes de puntos pueden estar sujetas a inserciones, borrados, ruido, transformaciones de similitud, afines y proyectivas.

En la **Figura 21**, se muestra un ejemplo del problema de inserciones y borrados. Suponga que tenemos un sistema de recuperación de audio basado en contenido que utiliza la representación de nubes de puntos propuesta por Wang (4.1 Nubes de puntos para audio).

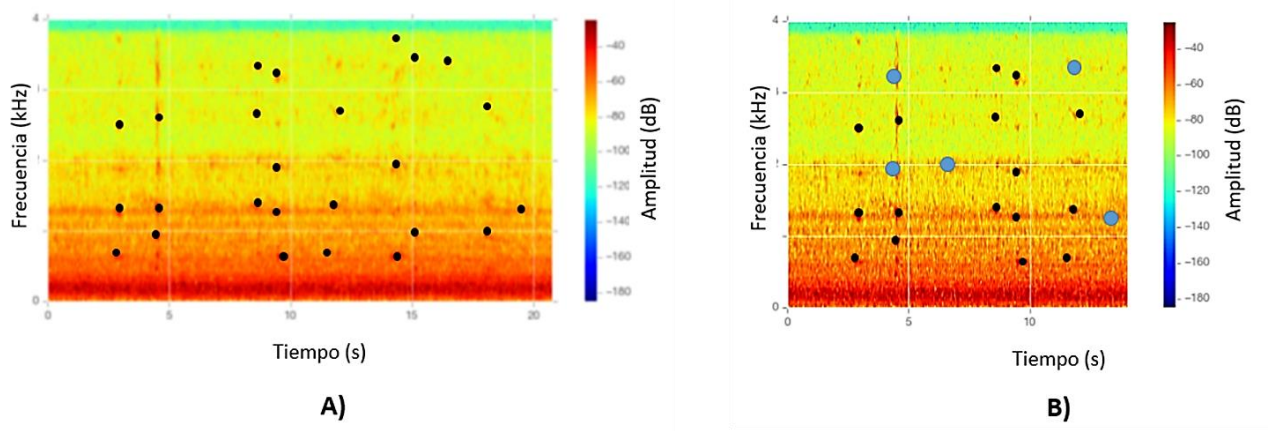


Figura 21.- Ejemplo del problema de inserciones y borrados para nubes de puntos en un sistema de recuperación de audio basado en contenido. A) Nube de puntos de archivo de audio original de la base de datos, B) Nube de puntos de archivo de audio de consulta. Figura obtenida el 29/07/17 y modificada de: http://www.mathisintheair.org/wp/wp-content/uploads/2017/01/spectrogram-744x298.png

En la **Figura 21** A), se muestra la nube de puntos del archivo de audio original de la base de datos del objeto que estamos buscando. Cuando el usuario realiza una consulta en este tipo de sistemas, se solicita que proporcione un segmento de audio como consulta, el cual puede ser sujeto a ruido ambiental por las condiciones de la grabación.

En la **Figura 21** B), se presenta el segmento de audio proporcionado por el usuario. Observe que el usuario solo proporciona los primeros quince segundos respecto al archivo de audio original, esto tiene como consecuencia que al momento de calcular los puntos de interés, el efecto sea equivalente a borrar los puntos de interés del archivo de audio original del segundo quince en adelante. Además, ya que durante la grabación del archivo de audio de consulta del usuario puede existir ruido ambiental, puntos que no se encuentran en la nube de puntos original pueden aparecer en la nube de puntos del archivo de consulta (**Figura 21** B) – puntos azules).

Otro de los desafíos que se presentan al utilizar nubes de puntos como representación de objetos multimedia es el ruido. Por ejemplo, en el caso de imágenes, factores como la calibración del lente de cámara, iluminación y/o exposición, pueden ocasionar que los algoritmos para calcular puntos de interés registren u omitan puntos de interés, comparados con los puntos de interés calculados en condiciones ideales. El ruido ambiental y el ruido ocasionado por sistemas eléctricos son algunos ejemplos de ruido en aplicaciones de audio.

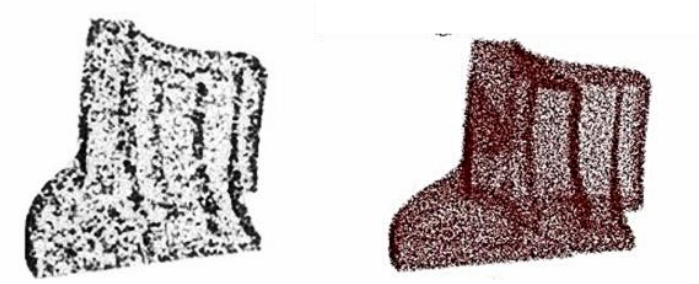


Figura 22.- Ejemplo de nube de puntos sujeta a ruido (sin y con pre-procesamiento). Figura obtenida el 29/07/17 y modificada de: https://www.intechopen.com/source/html/18899/media/image161.jpeg

Las nubes de puntos pueden estar sujetas a diferentes tipos de transformaciones. Una transformación proyectiva, es una transformación que involucra rotación, cambio de escala, traslación, sesgo y proyección perspectiva. Un subconjunto de estas son las transformaciones afines. Una transformación afín (involucra rotación, traslación, cambio de escala y/o sesgo), es cualquier transformación F: $\mathbb{R}^n \rightarrow \mathbb{R}^n$ que preserva colinealidad y proporciones de distancias de la forma:

$$F(p) = \boldsymbol{A}p + q \tag{29}$$

Donde $p \in \mathbb{R}^n$, **A** es una transformación lineal de $\mathbb{R}^n$

Una transformación de similitud es un subconjunto del conjunto de transformaciones afines, la cual puede consistir de rotaciones, cambio de escala y traslación. En la **Figura 23** se muestra un ejemplo de una imagen bajo una transformación de similitud, afín y proyectiva.

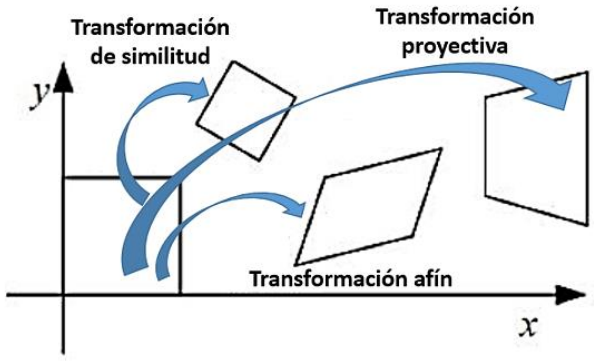


Figura 23.- Ejemplo de transformación de similitud, afín y proyectiva. Figura recuperada y modificada el 29/07/17 de: https://i.publiclab.org/system/images/photos/000/004/507/original/geometric-transformations.jpg

Una alternativa para mitigar el ruido es preprocesar los objetos multimedia antes de calcular su representación de nubes de puntos. Por ejemplo, en el caso de imágenes se pueden utilizar filtros de suavizado, normalización de parámetros como contraste, transformación de espacio de color a escala de grises, binarizacion, entre otros. Para el caso archivos de audio, se pueden aplicar filtros pasa baja o pasa banda para eliminar ruido de alta frecuencia.

Sin embargo, si no es posible realizar un pre-procesamiento de los objetos multimedia antes de obtener su nube de puntos, se puede discretizar las coordenadas de los puntos de interés de las nubes de puntos mediante una rejilla discreta, transformando los puntos de interés de $\mathbb{R}^2$ a $\mathbb{N}^2$.

Una invariante es una propiedad de un objeto la cual no cambia cuando se aplica una transformación a este. Existe un gran número de invariantes que pueden ser utilizadas para nubes de puntos en $\mathbb{R}^2$. Por ejemplo, sea $A, B, C, D, E \in \mathbb{R}^2$. Sea AC y AB la distancia entre estos elementos y $S(A, B, C)$ el área del triángulo con vértices A, B, C. Una invariante ante transformaciones de similitud para los puntos A, B, C (Iwamura et al., 2007) se define como:

$$\frac{AC}{AB} \tag{30}$$

Para cuatro puntos en el plano $(A, B, C\ D)$ (Nakai et al., 2006), una invariante a transformaciones afines es:

$$\frac{S(A, C, D)}{S(A, B, C)} \tag{31}$$

Para cinco puntos en el plano (A, B, C, D, E) (Nakai et al., 2006), la siguiente propiedad es invariante a transformaciones proyectivas:

$$\frac{S(A, B, C)S(A, D, E)}{S(A, B, D)S(A, C, E)} \tag{32}$$

En 2016, Chávez et al. proponen utilizar coeficientes de Fourier como invariantes para polígonos. Dada una nube de puntos en el plano complejo $\mathbb{C}$, es decir, cada punto (x, y) es un número complejo $z =$

$x + iy$ donde $z \in \mathbb{C}$. Un polígono en el plano es un ordenamiento de un conjunto de puntos $(z_1, z_2, \dots, z_n)$, donde el orden de estos especifica el orden de los vértices el polígono.

Dado lo anterior, Chávez et al. proponen la familia de funciones para polígonos φ_j, las cuales son invariantes ante transformaciones de similitud. La función φ_j: $\mathbb{C}^n \to \mathbb{C} \cup \{\infty\}$ se define para cada entero j como:

$$\varphi_j(z_1, z_2, \dots, z_n) = \frac{\sum_{k=1}^{n} \lambda^{jk} z_k}{\sum_{k=1}^{n} \lambda^{-jk} z_k} \tag{33}$$

Donde:

$$\lambda = e^{\frac{2\pi i}{n}} \text{ y } n \geq 3 \tag{34}$$

Dado que el problema de ruido puede ser mitigado mediante un pre-procesamiento o mediante una discretización utilizando una rejilla discreta y que existen invariantes en la literatura para transformaciones de similitud, afines y proyectivas de puntos en $\mathbb{R}^2$, esta tesis se enfoca en el problema de inserciones y borrados, en particular, en el diseño de índices para nubes de puntos que sean robustos ante este problema.

Es importante señalar que una de las desventajas de las invariantes presentadas anteriormente, es que son susceptibles ante el problema de inserciones y borrados, por lo que es fundamental el diseño de índices robustos ante este problema.

4.4 Nubes de puntos en rejillas discretas

Hasta este momento, las nubes de puntos en las secciones anteriores utilizan puntos en $\mathbb{R}^2$ para representar los objetos multimedia. En esta sección se presenta una representación alternativa para nubes de puntos en $\mathbb{R}^2$.

Suponga que las nubes de puntos se encuentran en una rejilla discreta, es decir, una nube de puntos es un subconjunto finito de $\mathbb{N}^2$. Dada una rejilla discreta R donde se ubican las nubes de puntos, sea c_{max} la coordenada máxima permitida en la rejilla, $\delta * \delta$ la dimensión de una celda uniforme y $D =$

$\left\lfloor \frac{c_{max}}{\delta} \right\rfloor^2$ el número de celdas de R. Dado lo anterior, podemos representar una nube de puntos mediante un bitmap $B = \{0,1\}^D$, asignando un 1 a $B[i]$ si existe al menos un punto en la celda $R[i]$ y cero en caso contrario.

Existe un compromiso entre δ (tamaño de la celda) y la dimensión de la representación de la nube de puntos mediante un bitmap. Asignar un δ muy grande reduce la dimensión y el espacio de almacenamiento de B, sin embargo, la resolución de la rejilla disminuye considerablemente y se vuelve difícil discriminar los objetos de la base de datos. En el caso contrario, si seleccionamos un δ muy pequeño, aumenta la dimensión y el espacio de almacenamiento de B. Además deben de considerarse factores adicionales para la determinación de δ que dependen del dominio del problema en específico.

Una de las ventajas de esta representación es que permite utilizar directamente índices para espacios métricos y vectoriales. Además, ya que el bitmap se genera a partir de puntos que se encuentran en una rejilla discreta, se reduce el efecto de ruido y se mitiga el efecto de inserciones/borrados (7.6.1 Índices Invertidos). Sin embargo, uno de los problemas de esta representación es que bitmaps de dimensiones elevadas presentan el problema de la maldición de la dimensión y a medida que aumenta la resolución de la rejilla, el tamaño en memoria del bitmap aumenta. Además, esta representación solo indica ausencia o presencia de puntos en una celda, es decir, no captura el número de puntos presentes en una.

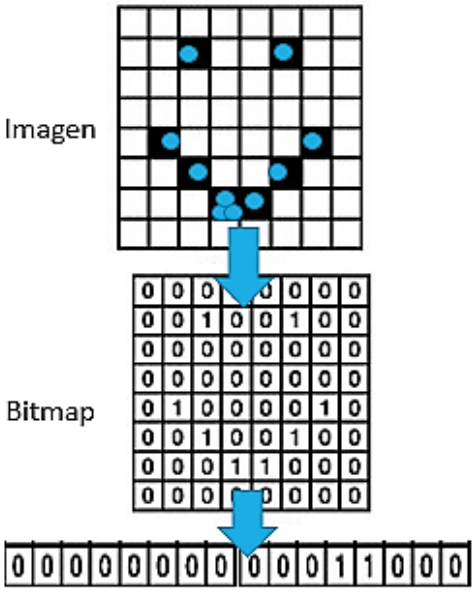


Figura 24.- Representación de una nube de puntos mediante una cadena binaria (bitmap).

Capítulo 5. Recuperación de nubes de puntos

En este capítulo presentamos el planteamiento formal del problema de recuperación de nubes de puntos, las soluciones existentes en el estado del arte y se presentan dos acercamientos considerados en esta tesis para la solución del mismo.

5.1 Planteamiento del problema

Considere una nube de puntos A_i como un subconjunto finito de $\mathbb{R}^2$. Sea $E = \{A_1, A_2, \dots, A_{|E|}\}$ una colección de nubes de puntos, Q una nube de puntos de consulta y d una métrica. Sea δ la tolerancia de cuanto un punto individual de una nube de puntos puede diferir respecto a un punto de otra nube para empatar.

Sea $A = \{x_1, x_2, \dots, x_{|A|}\}$ y $Q = \{y_1, y_2, \dots, y_{|Q|}\}$. Definiremos el conjunto C_{AQ} como el *conjunto soporte* para las nubes de puntos A y Q de la siguiente manera:

$$C_{AQ} = \{x_i, d(x_i, y_j) < \delta\} \forall\, x_i \in A \wedge y_j \in Q \tag{35}$$

Dada la definición anterior, el *soporte* se define como:

$$S_{AQ}(A) = \frac{|C_{AQ}|}{|A|} \tag{36}$$

$$S_{AQ}(Q) = \frac{|C_{AQ}|}{|Q|} \tag{37}$$

$$S_{AQ}(AQ) = \min\left(S_{AQ}(A), S_{AQ}(Q)\right) \tag{38}$$

Podemos observar que el conjunto soporte C_{AQ} depende fuertemente de la tolerancia δ. Para un δ suficientemente grande el soporte es máximo.

El problema de recuperación de nubes de puntos es el siguiente. Sea $0 \leq \epsilon \leq 1$ un umbral proporcionado por el usuario, E una colección de nubes de puntos (base de datos), una nube de puntos de consulta Q y d una métrica, se desea encontrar todas las nubes de puntos $A_i \in E$ tal que $S_{A_iQ}(A_iQ) \geq \epsilon$. De igual manera, nos interesa encontrar todas las nubes de puntos $A_i \in E$ tal que $S_{A_iQ}(A_i) \geq \epsilon$.

Un caso especial del problema de recuperación de nubes de puntos es cuando las nubes de puntos se encuentran sobre una rejilla discreta. En este caso, las nubes de puntos son un subconjunto finito de $\mathbb{N}^2$ y el *conjunto soporte* se define como:

$$C_{AQ} = \{x_i, x_i = y_j\} \forall\, x_i \in A \wedge y_j \in Q \tag{39}$$

La definición de soporte es idéntica que en el caso de puntos $\mathbb{R}^2$.

5.2 Estado del Arte

Dado que hasta donde tenemos conocimiento, solo existe un índice diseñado específicamente para nubes de puntos, en esta sección se presenta una descripción de este índice y un estudio de índices para consultas *kNN join* y *range Join*. El problema de recuperación de nubes de puntos puede modelarse como un caso específico de consultas *kNN join* o *range join*. Sin embargo, como se mostrará más adelante en esta sección, estos índices no pueden aplicarse directamente al problema de recuperación de nubes de puntos.

5.2.1 Índices para el problema de recuperación de nubes de puntos

El índice presentado por Wang en 2003, caracteriza una base de datos de canciones y segmentos de audio mediante nubes de puntos de dos dimensiones. Wang selecciona puntos de alta energía en un espectrograma y por cada punto de esta nube de puntos, se genera un conjunto de tuplas de la forma $(freq_1, freq_1, \Delta_{time})$, las cuales se utilizan para representar el archivo de audio. El emparejamiento de la nube de puntos del segmento de audio proporcionado por el usuario, se puede realizar utilizando una ventana deslizante sobre la base de datos hasta que el máximo número de emparejamientos sea

encontrado. Un esquema alternativo es generar un índice invertido de las tuplas mediante una tabla hash (Müller, 2015), donde el proceso de consulta consiste en calcular la nube de puntos del segmento de audio, generar su conjunto de tuplas y finalmente realizar una consulta por cada tupla en el índice invertido.

5.2.2 Índices para consultas range join y kNN-Join

Una observación interesante es que podemos plantear el problema de recuperación de nubes de puntos como un problema de búsqueda *range join* (2.3.4 Range join), donde el conjunto Q representa la nube de puntos de consulta y $S = E$ la colección de todas las nubes de puntos de la base de datos. Aunque esta operación es muy importante, la operación *range join* (*similarity join*) tiene un costo computacional muy elevado.

Existen diversas estructuras de datos para consultas *range join*, *kNN join* y *distance join* propuestas en la literatura. A continuación se presenta un diagrama de algunos de los trabajos actuales en la literatura organizado por tipo de índice y orden cronológico.

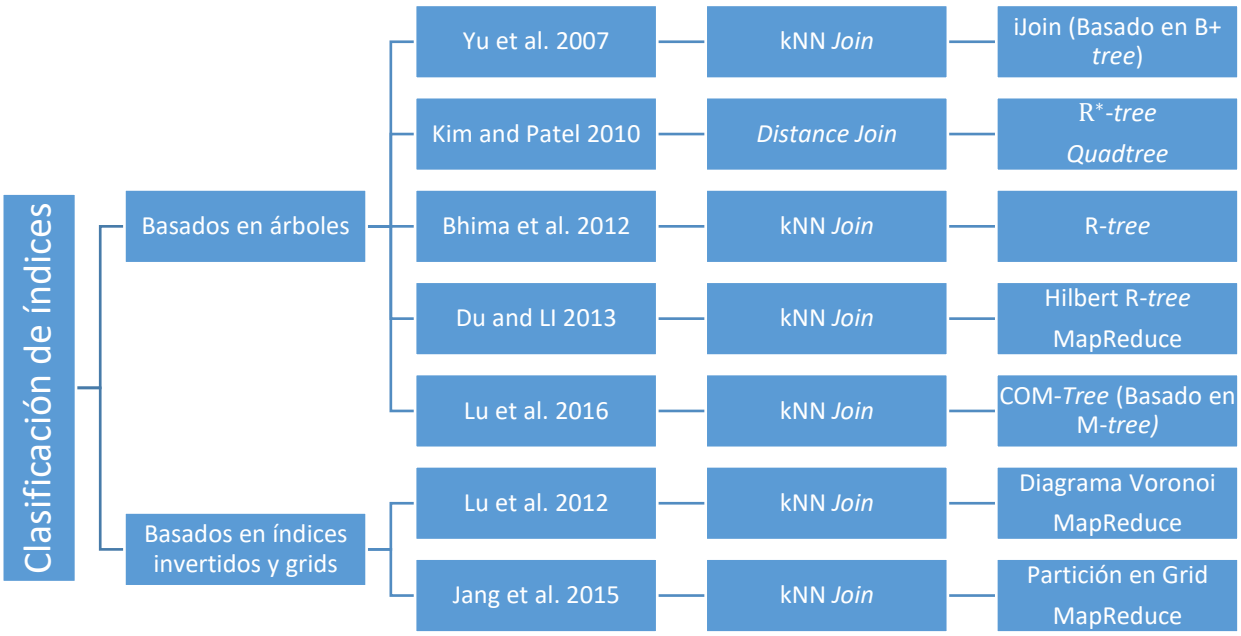


Figura 25.- Clasificación de índices para consultas KNN/Range Join del estado del arte.

El índice iJoin basado en el índice iDistance, fue propuesto con el objetivo de mejorar el desempeño de consultas kNN Join (Yu et al., 2007). Este índice realiza una partición de los datos utilizando k-means. Para cada elemento, se calcula la distancia de este al punto de referencia de su partición. Posteriormente se indexan estas distancias mediante un B+tree. Este procedimiento se realiza para ambos conjuntos y en el proceso de búsqueda se recorren ambos árboles.

En 2010, Kim y Patel, realizaron un estudio comparativo entre los índices R*-tree y Quadtree para consultas kNN y *distance join* con el objetivo de identificar las ventajas y desventajas de estas estructuras respecto a algoritmos de construcción, comparando tiempo de construcción y tiempo de consulta. Uno de los resultados más interesantes de este estudio es que en general Quadtree tiene un mejor desempeño a comparación del índice R*-tree para consultas *distance join* cuando los índices se construyen en forma dinámica, lo cual es muy interesante ya que Quadtree no es ampliamente utilizado a comparación de R*-tree.

En 2012, Bhima et al. presentan una estructura de datos que utiliza el índice R-Tree para realizar consultas *similarity join* y kNN en bases de datos espaciales. Para realizar la consulta *similarity join*, este método construye un índice R-Tree por conjunto. Para el proceso de consulta se realiza un recorrido desde las raíces de los árboles nivel por nivel. Cada nodo del primer R-Tree se compara con los nodos del otro R-Tree para identificar si los nodos se traslapan. Este proceso se realiza utilizando una estrategia de búsqueda en anchura. Si se identifica un traslape, este nodo se convierte en candidato, se agrega a una cola de prioridad y se revisan todos los nodos de esta cola.

En 2013, Du y Li proponen utilizar el paradigma MapReduce en conjunto con un índice Hilbert R-Tree para consultas kNN Join. En un Hilbert R-Tree se utiliza la curva de Hilbert para dar un ordenamiento de los rectángulos delimitadores mínimos (MBR), con el objetivo reducir el tamaño de los nodos del árbol. Dado dos conjuntos R y S, en la etapa Map se realiza una partición de los datos en bloques. En la fase reduce, se genera un Hilbert R-tree para cada bloque, el cual encuentra los k vecinos más cercanos de los elementos del conjunto S para cada elemento del conjunto R. Al final se ordena un conjunto de listas con los elementos de cada bin de acuerdo a una función de distancia (etapa reduce) y se regresan los k elementos más relevantes.

El índice métrico dinámico COM-tree (Chen et al., 2016), es un índice métrico basado en M-tree, el cual incorpora información que se utiliza por un conjunto de reglas para reducir el espacio de búsqueda. Para realizar la consulta kNN Join, el autor presenta dos algoritmos: GBA (*Group based MAkNN Algorithm*)

e IPA (*Index and Preprocessing based MAkNN Algorithm*). El objetivo de GBA es descomponer un conjunto en varios conjuntos disjuntos, tal que objetos cercanos se encuentren en un mismo grupo, minimizando el número de conjuntos disjuntos. El algoritmo IPA utiliza un método de podado progresivo que disminuye el espacio de búsqueda de forma gradual, utilizando una estrategia de recorrido a profundidad. Una de las ventajas de estos algoritmos es que son compatibles con otros índices métricos.

En 2012, se presenta una implementación de kNN join utilizando la metodología MapReduce, la cual utiliza como método de partición un diagrama Voronoi (Lu et al., 2012). Sea S y R dos conjuntos sobre los cuales se realiza la consulta kNN join, primero se realiza una etapa de pre-procesamiento donde se elige un conjunto de puntos de referencia del conjunto R para generar un diagrama Voronoi. En la primera etapa de MapReduce se encuentra para cada elemento de $R \cup S$, el punto de referencia al que pertenece y se almacena la distancia entre el punto y su punto de referencia, teniendo como resultado una partición de R basada en el diagrama Voronoi. Dentro de la etapa Map se asigna un subconjunto $S_i \in S$ para cada subconjunto R_i. Finalmente en la etapa Reduce, se realiza la operación kNN join para cada par R_i y S_i de la etapa Map.

En 2015 se propone un nuevo índice basado en una rejilla discreta para kNN join, utilizando la metodología MapReduce (Jang et al., 2015). Sean R y S dos conjuntos, este índice realiza una partición de los datos de los conjuntos R y S, utilizando un conjunto de pivotes (etapa Map). En la etapa Reduce, se insertan todos los datos en un índice tipo grid (rejilla discreta), se calcula la densidad de cada celda y se realiza un índice invertido que indica para cada celda sus puntos correspondientes. Después, se realiza una segunda etapa de MapReduce en la cual el algoritmo busca los k vecinos más cercanos de S para todos los puntos en R. En esta etapa, se realiza una búsqueda de radio a partir de la celda de consulta, aumentando el radio de consulta y manteniendo una cola de prioridad de tamaño k, que almacena los kNN de la consulta.

5.3 Observaciones de las estructuras de datos del estado del arte

Algunas de las ideas implementadas en las estructuras de datos del estado del arte son:

- Generar un índice para cada conjunto y revisar ambos índices de manera simultánea durante el proceso de consulta (Bhima et al., 2012; Yu et al., 2007).

- Realizar una partición de los datos utilizando métodos como k-means, diagramas Voronoi, rejillas discretas o particiones en bloques(Du et al., 2013; Jang et al., 2015; Lu et al., 2012; Yu et al., 2007).

- Utilizar la metodología MapReduce utilizando marcos de trabajo como Apache Hadoop (Du et al., 2013; Jang et al., 2015; Lu et al., 2012).

- Generar un conjunto de reglas para reducir el espacio de búsqueda (Chen et al., 2016).

La principal característica de la mayoría de los índices propuestos en el estado del arte para consultas range join, kNN join y distance join, es que suponen que ambos conjuntos se encuentran disponibles al momento de indexado y en algunos casos también se supone que sus cardinalidades son similares. Es importante señalar que en el problema de recuperación de nubes de puntos, el conjunto que representa la nube de puntos de consulta no se encuentra disponible al momento de indexado (la nube de consulta es dinámica), además la cardinalidad de la nube de consulta es mucho menor que la cardinalidad de la colección de nubes de puntos (base de datos). Debido a estas consideraciones, la aplicación directa de los métodos anteriormente mencionados no es posible para este problema.

Capítulo 6. Índices para Nubes de Puntos

En este capítulo se proponen índices para el problema de recuperación de nubes de puntos basados en dos esquemas diferentes: índices invertidos e índices métricos. El objetivo de estas estructuras es que sean robustas ante el problema de inserciones y borrados.

Dado que el desempeño de los dos esquemas anteriores depende de la estructura de datos que se utilice (índice invertido o índice métrico), se realizó un estudio comparativo de diferentes estructuras de datos de la literatura y diferentes esquemas propuestos durante el desarrollo de esta tesis.

6.1 Basados en índices invertidos

Sea A_i un subconjunto finito de $\mathbb{R}^2$ una nube de puntos, $E = \{A_1, A_2, \dots, A_{|E|}\}$ una colección de nubes de puntos (base de datos), Q una nube de puntos de consulta, d una métrica, δ la tolerancia de cuanto un punto individual de una nube de puntos puede diferir con respecto al punto de otra nube para empatar y $0 \leq \epsilon \leq 1$ un umbral proporcionado por el usuario, se desea encontrar todas las nubes de puntos $A_i \in E$ tal que $S_{A_iQ}(A_iQ) \geq \epsilon$.

A continuación se presenta el esquema general de construcción y consulta de los índices para nubes de puntos basados índices invertidos.

6.1.1 Construcción

El índice invertido propuesto se construye de la siguiente manera: para cada punto $x \in A_i$ donde $A_i \in E$, se genera una lista invertida que contiene una etiqueta del identificador de la nube de puntos a la que pertenece. Por ejemplo si $x \in A_i$, se genera la lista invertida $\{i\}$.

6.1.2 Consulta

Dado un índice invertido, para cada $x \in Q$, se realiza una consulta de rango o una consulta kNN, lo cual activa un conjunto de listas invertidas, donde posteriormente se realiza la intersección o unión estas, calculando la frecuencia de los identificadores. Finalmente, se entrega como resultado una lista de identificadores de nubes de puntos, ordenadas en orden descendente en función de $S_{A_iQ}(A_iQ)$ o $S_{A_iQ}(A_i)$.

A continuación se presenta el pseudocódigo para el proceso de consulta del esquema propuesto para índices de nubes de puntos basados en índices invertidos.

1. $\forall x \in Q$ se realiza una búsqueda de rango con radio δ en el índice invertido y se almacenan las listas invertidas activadas en una lista de resultados preliminares.

2. Se calcula la frecuencia de cada identificador de la lista de resultados preliminares.

3. Se calcula el soporte $S_{A_iQ}(A_iQ)$ para cada identificador de la lista de resultados preliminares y se ordenan en orden descendente en función del soporte.

4. Se entrega al usuario una lista de todas las nubes de puntos $A_i \in E$ tal que $S_{A_iQ}(A_iQ) \geq \epsilon$

Observe que este esquema de consulta es bastante flexible. Por ejemplo, en lugar de que el usuario proporcione la tolerancia δ, el usuario puede proporcionar un parámetro k, tal que un punto individual de una nube de puntos q empata cuando un punto de otra nube de puntos se encuentra dentro de los k vecinos más cercanos de q. Para esto solamente se necesita modificar el paso 1, donde en lugar de realizar una búsqueda de rango, se realiza una búsqueda kNN.

De igual manera si se desea una consulta kNN tal que $S_{A_iQ}(A_iQ) \geq \epsilon$, la única modificación necesaria es en el paso 4, donde en lugar de entregar todas las $A_i \in E$ tal que $S_{A_iQ}(A_iQ) \geq \epsilon$, solamente se entregan las primeras k.

A continuación se muestra el análisis del tiempo de consulta en el peor de los casos de este esquema. Sea idx la complejidad del tiempo de consulta en el peor de los casos del índice invertido utilizado en el paso 1 y l la lista de resultados preliminares. Si $|Q|$ es $O(1)$, el tiempo de consulta en el peor de los casos de este esquema es:

- Paso 1: $O(idx)$
- Paso 2: $O(l)$
- Paso 3: $O(l \log l)$
- Paso 4: $O(l)$

Tiempo de consulta de esquema propuesto (peor de los casos): $O(idx + l \log l)$

De acuerdo al pseudocódigo anterior, para listas de resultados preliminares pequeñas, la mayor parte del tiempo de ejecución del proceso de consulta se concentra en el paso 1. Por esta razón, la selección del índice invertido utilizado es fundamental para el desempeño del tiempo de consulta. A continuación se presentan diferentes índices invertidos considerados para el problema de recuperación de nubes de puntos.

6.1.3 Índice para nubes de puntos basado en R*-Tree

Para la construcción de este índice para nubes de puntos utilizaremos el índice R*-Tree (Beckmann et al., 1990), el cual es unas de las variantes de la familia R-Tree con mejor desempeño. Este índice tiene la misma estructura que el índice R-Tree pero utiliza diferentes heurísticas con el objetivo de reducir el traslape y tamaño de los rectángulos delimitadores mínimos. Existen diversos algoritmos y estrategias para la construcción del índice R*-tree. En nuestro caso utilizaremos el algoritmo Bulkloading, el cual realiza un pre-procesamiento de los datos que tiene como objetivo reducir el tamaño de este índice; este pre-procesamiento puede consistir en realizar particiones y ordenamientos de los datos, así como estimar el número de rectángulos delimitadores mínimos necesarios (Kim et al., 2010). Este método de construcción es más rápido y generalmente genera índices R-Tree con mejores estructuras internas.

Sea n el total de elementos almacenados en el índice invertido R*-Tree. Dado que podemos controlar el mínimo y máximo número de elementos en cada nodo del R*-Tree, podemos suponer que $|l| = O(1)$. Por lo tanto:

Tiempo de consulta (peor de los casos): $O(idx + l \log l) = O(n)$

Tiempo de construcción: $O(n \log n)$

Memoria de almacenamiento: $O(n)$

6.1.4 Índice para nubes de puntos basado en Índice Métrico

Dado que el universo de objetos válidos es $\mathbb{R}^2$, una opción lógica es utilizar índices métricos, ya que estos índices tienen excelente desempeño cuando la representación de los objetos a indexar es pequeña. El índice Vantage Point Tree (Yianilos, 1993), es un índice de espacio métrico que realiza una descomposición de los datos en forma de árbol binario balanceado. Este índice presenta un buen desempeño en tiempo de consulta en búsquedas de proximidad para datos de dimensiones pequeñas, lo cual puede ser corroborado en base a los resultados experimentales que se presentan en el capítulo 8.

Sea n el total de elementos almacenados en el índice invertido VPT. Dado que cada nodo solo tiene un elemento en el índice VPT, podemos suponer que $|l| = O(1)$. Por lo tanto:

Tiempo de consulta (peor de los casos): $O(idx + l \log l) = O(\log n)$

Tiempo de construcción: $O(n \log n)$

Memoria de almacenamiento: $O(n)$

El tiempo de consulta del índice VPT es $O(\log n)$ solo para radios de búsqueda pequeños, por lo tanto, el tiempo de consulta para radios más grandes puede encontrarse entre $O(\log n)$ y $O(n)$.

6.1.5 Índice para nubes de puntos basado en Inverted Grid Index

El índice Inverted Grid Index o IGI (Ji et al., 2014), tiene como objetivo mejorar el desempeño de consultas kNN y RNN (*Reverse Nearest Neighbor*) en aplicaciones para bases de datos espaciales en sistemas distribuidos. Este índice realiza una partición de los datos mediante una rejilla discreta con celdas de dimensión uniforme y posteriormente genera un índice invertido con una lista invertida para cada celda; este es construido mediante la metodología MapReduce utilizando Apache Hadoop.

Sea $P \in R^2$ una base de datos, δ la longitud de una celda de la rejilla discreta en cada dimensión (cada celda es de tamaño $\delta * \delta$) y C el conjunto de celdas de la rejilla discreta. El índice IGI es un índice invertido donde para cada $p \in P$ con coordenadas $p.x$ y $p.y$, se calcula la celda $c[i,j]$ tal que $p \in c[i,j]$ si p se encuentra en la celda $c\left[\left\lfloor\frac{p.x}{\delta}\right\rfloor, \left\lfloor\frac{p.y}{\delta}\right\rfloor\right]$. Finalmente para cada celda c, se genera una lista invertida que contiene todos los puntos p tal que $p \in c$.

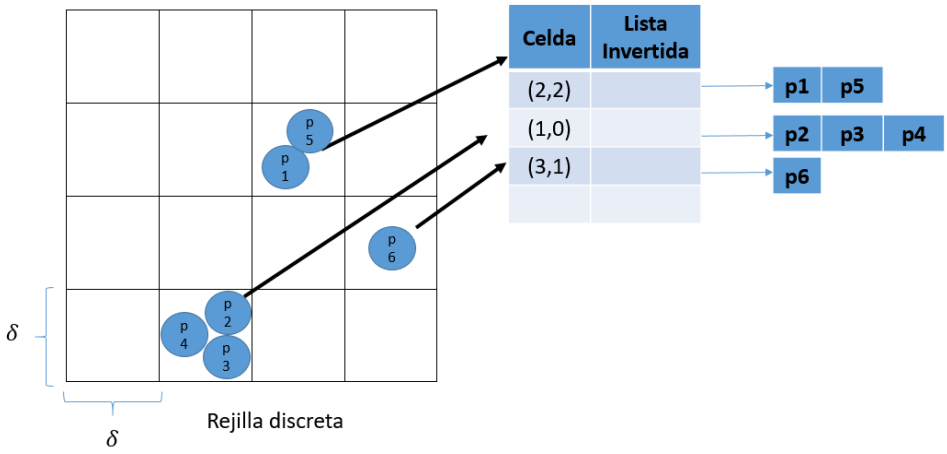


Figura 26.- Estructura del índice Inverted Grid Index (IGI): rejilla discreta e índice invertido.

Utilizaremos el índice IGI para el problema de recuperación de nubes de puntos de la siguiente manera. En nuestro caso, si $p \in E$ es un punto de la base de datos donde $p.x$ y $p.y$ son sus coordenadas, calcularemos su celda correspondiente $Celda(p) = \left[\left\lfloor\frac{p.x}{\delta}\right\rfloor, \left\lfloor\frac{p.y}{\delta}\right\rfloor\right]$ y para cada celda se genera una lista invertida que contiene el identificador de la nube de puntos a la que pertenece p.

Se utiliza el mismo proceso de consulta presentado en la sección 6.1.2, modificando el paso 1 de la siguiente manera:

1. $\forall q \in Q$ se calcula $Celda(q) = \left[\left\lfloor\frac{q.x}{\delta}\right\rfloor, \left\lfloor\frac{q.y}{\delta}\right\rfloor\right]$, se obtiene la lista invertida de esta celda y se almacena en una lista de resultados preliminares.

Sea n el total de elementos almacenados en el índice invertido IGI. El índice IGI puede implementarse mediante una tabla hash, por lo que en el peor de los casos una consulta se realiza en $O(n)$. Dependiendo de los parámetros utilizados para la rejilla discreta, en el peor de los casos $|l| = O(n)$, por lo tanto:

Tiempo de consulta (peor de los casos): $O(idx + l \log l) = O(n + n \log n) = O(n \log n)$

Tiempo de construcción: $O(n)$

Memoria de almacenamiento: $O(n)$

6.1.6 Índice para nubes de puntos basado en Inverted Grid Index y R*-Tree/VPT

Una extensión natural de los índices de las secciones anteriores es realizar una combinación de estos. Por ejemplo, el índice de consultas KNN Join propuesto en 2013 por Du y Li, realiza una descomposición de los datos en bloques y para cada uno de estos genera un Hilbert R-Tree para encontrar los k vecinos más cercanos de los elementos de un conjunto S para cada elemento de un conjunto R.

Utilizando la misma estrategia, realizaremos una descomposición de los datos mediante una rejilla discreta utilizando el esquema del índice Inverted Grid Index, donde para cada celda generaremos un índice invertido utilizando el índice de R*-Tree o Vantage Point Tree. Observe que tenemos un índice invertido de dos niveles, donde en el primer nivel se tiene un índice invertido para los índices R*-Tree o VPT de cada celda y el segundo nivel es un índice invertido donde para cada punto del índice R*-Tree/ VPT se tiene una lista invertida del identificador de la nube a la que pertenece cada punto. Una ventaja de este esquema es que se puede implementar en paralelo fácilmente.

Sea $E = \{A_1, A_2, \dots, A_{|E|}\}$ una colección de nubes de puntos (base de datos) y $C = \{c_1, c_2, \dots, c_{|C|}\}$ el conjunto de celdas de la rejilla discreta, el proceso de construcción se realiza de la siguiente manera:

1. $\forall \boldsymbol{x} \in E$ calcular su celda correspondiente de acuerdo con $Celda(x) = \left[\left\lfloor \frac{\boldsymbol{x}.x}{\delta} \right\rfloor, \left\lfloor \frac{\boldsymbol{x}.y}{\delta} \right\rfloor\right]$

2. $\forall c_i \in C$ construir un índice invertido (de espacio métrico o vectorial) con los puntos que pertenecen a c_i.

El proceso de consulta se realiza de igual manera que en la sección 6.1.2, modificando solamente el paso 1:

1. $\forall x \in Q$ se calcula la celda $Celda(x) = \left[\left\lfloor \frac{x.x}{\delta} \right\rfloor, \left\lfloor \frac{x.y}{\delta} \right\rfloor\right]$ y se realiza una búsqueda de rango con radio δ en el índice invertido de la celda a la que pertenece, almacenando las listas invertidas activadas en una lista de resultados preliminares.

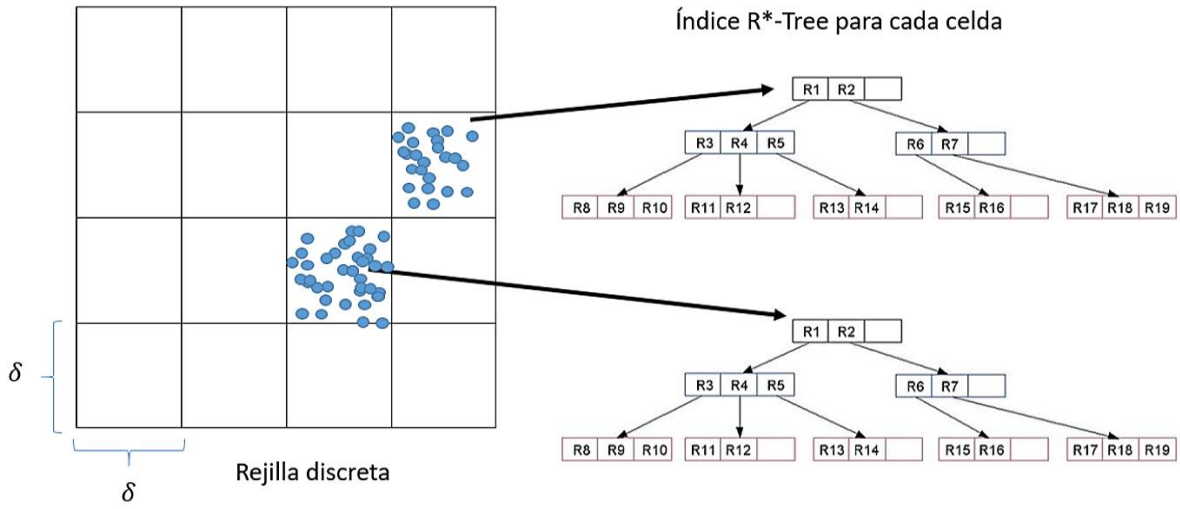


Figura 27.- Índice Invertido IGI + R*-Tree. Figura obtenida el 29/07/17 y modificada de: https://en.wikipedia.org/wiki/File:R-tree.svg

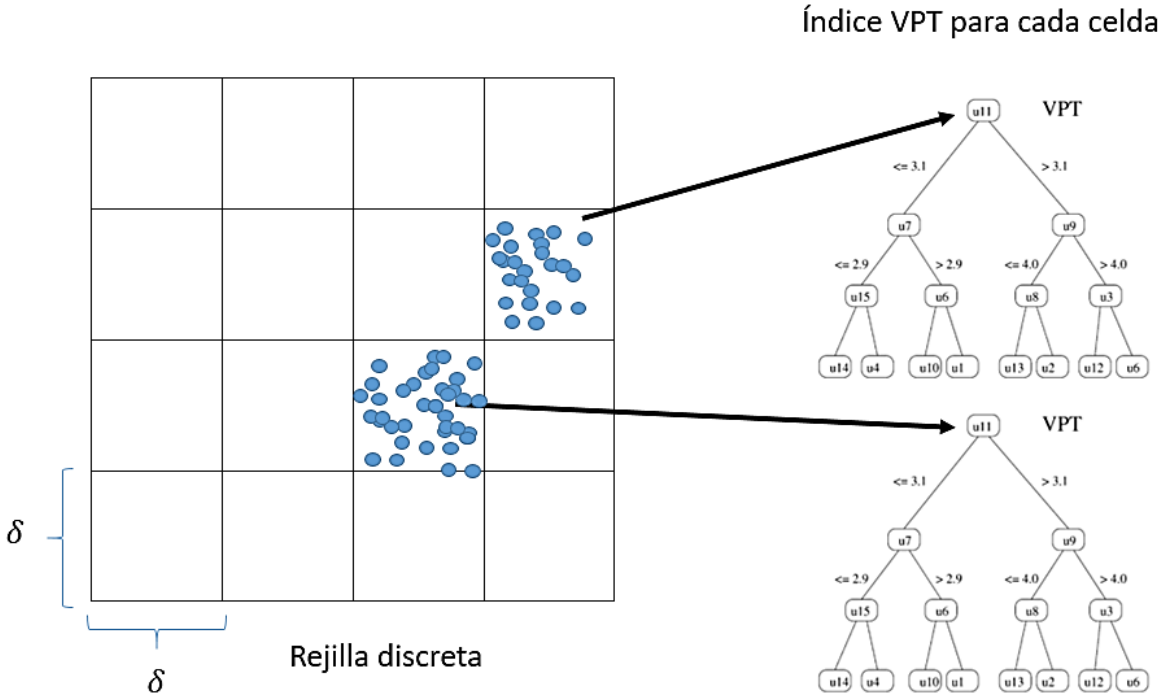


Figura 28.- Índice Invertido IGI + VPT. Figura modificada de (Chávez et al., 2001).

Sea n el total de elementos almacenados en los índices de todas las celdas. Para determinar la celda a la que pertenece un elemento se utiliza una función, por lo que en el peor de los casos el determinar la celda a la que pertenece un punto se realiza en $O(1)$.

Observe que tanto para el índice R*-Tree y VPT, podemos suponer que $|l| = O(1)$ y que en el peor de los casos, dependiendo de los parámetros de la rejilla discreta, una celda puede tener un número de elementos de orden $O(n)$. Por lo tanto:

Tiempo de consulta (peor de los casos): $O(idx + l \log l)$

- Para R*-Tree: $O(n + 1 \log 1) = O(n)$
- Para VPT: $O(\log n + 1 \log 1) = O(\log n)$

Tiempo de construcción: $O(n \log n)$

Memoria de almacenamiento: $O(n)$

El tiempo de consulta del índice VPT es $O(\log n)$ solo para radios de búsqueda pequeños, por lo que el tiempo de consulta para radios más grandes puede encontrarse entre $O(\log n)$ y $O(n)$.

6.1.7 Succinct Inverted Grid Index

En esta sección se propone una modificación al índice Inverted Grid Index (Ji et al., 2014) llamado Succinct Inverted Grid Index (Succinct IGI). Suponga que las nubes de puntos se encuentran en una rejilla discreta. Sea ρ la cantidad promedio de puntos por celda de la rejilla discreta y $|E|$ el número de nubes de puntos a indexar. Observe que podemos ajustar la cantidad de elementos promedio por celda tal que $\rho \ll |E|$ mediante una rejilla discreta adaptativa.

Dado lo anterior, convertiremos las listas invertidas de identificadores en bitmaps. Observe que $\rho \ll |E|$ (utilizando una rejilla discreta adaptativa), por lo tanto, el bitmap no es denso y podemos utilizar el índice sucinto sarray para representar los bitmaps con una cantidad de memoria cercana a la entropía del peor de los casos, reduciendo el espacio de almacenamiento.

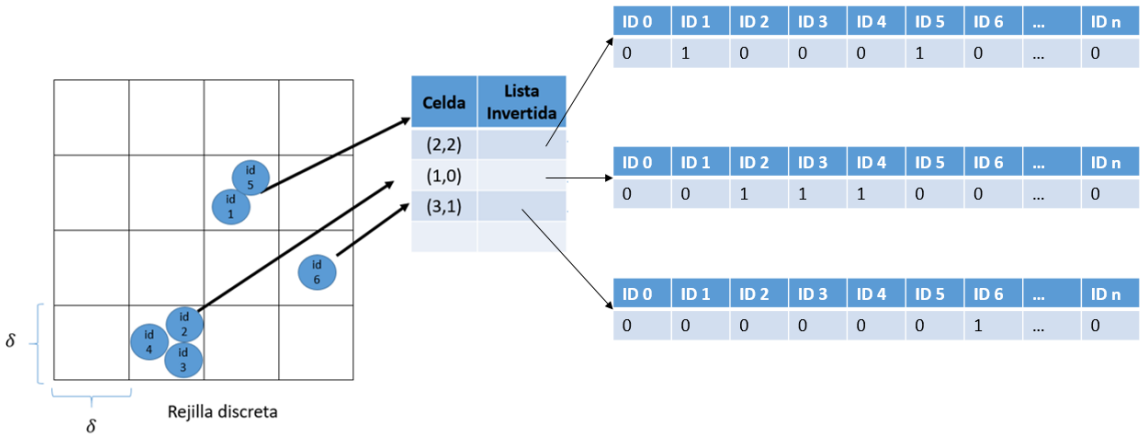


Figura 29.- Succinct Inverted Grid Index.

El proceso de construcción es el mismo que el índice para nubes de puntos basado en Inverted Grid Index presentado en la sección 6.1.5, a excepción de un paso adicional que consiste en convertir las listas de identificadores en bitmaps y representar cada uno de estos mediante el índice sucinto sarray.

Se utiliza el mismo proceso de consulta presentado en la sección 6.1.2, modificando el paso 1 de la siguiente manera:

1. $\forall q \in Q$ se calcula $Celda(q) = \left[\left\lfloor \frac{q.x}{\delta} \right\rfloor, \left\lfloor \frac{q.y}{\delta} \right\rfloor\right]$, se obtiene el índice sarray de esta celda y se utiliza la operación $select_1$ para recuperar los identificadores almacenados en este. Posteriormente se almacenan estos identificadores en una lista de resultados preliminares.

Una de las desventajas de este índice es que durante el proceso de consulta se necesitan recuperar los identificadores de los índices sarrays. Sea m el número de unos de un bitmap, observe que para obtener los identificadores almacenados en el índice sarray, realizaremos m consultas $select_1$ por bitmap. Sin embargo, de acuerdo con los resultados de las pruebas experimentales realizadas, bajo ciertas condiciones este índice propuesto utiliza hasta un 54% menos de espacio de almacenamiento y presenta una mejora de hasta un 37% en tiempo de consulta comparado con el índice para nubes de puntos basado en IGI.

Sea n el total de elementos almacenados en el índice Succinct IGI y C el conjunto de celdas de la rejilla discreta. El índice Succinct IGI puede implementarse mediante una tabla hash, por lo que en el peor

de los casos, una consulta se realiza en $O(n)$. Dependiendo de los parámetros utilizados para la rejilla discreta, en el peor de los casos $|l| = O(n)$. Finalmente, para obtener los identificadores realizamos m consultas $select_1$ de complejidad $O\left(\frac{log^4 m}{\log|E|}\right) \approx O(1)$, ya que $m = O(1)$. Por lo tanto:

Tiempo de consulta (peor de los casos): $O(1 + n + n\log n) = O(n\log n)$

Tiempo de construcción: $O(n)$

Memoria de almacenamiento: $O\left(|C| * \left(m(1.92 + \log\frac{|E|}{m}) + o(m)\right)\right)$

6.2 Basados en índices métricos

Se utilizará la representación de nubes de puntos mediante bitmaps (presentado en la sección 4.4 Nubes de puntos en rejillas discretas), donde U es el conjunto de todos los bitmaps de dimensión D ($\{0,1\}^D$) y d una métrica. Dada esta representación se resolverá el problema de recuperación de nubes de puntos mediante búsqueda de proximidad en espacio métrico (U, d).

Se utilizarán las siguientes métricas:

- Similitud coseno: $d(A,B) = arccos\left(\frac{A\cdot B}{||A||||B||}\right)$
- Coeficiente de Jaccard: $d(A,B) = 1 - \frac{|A\cap B|}{|A\cup B|}$
- Distancia de Hamming: $d(A,B) = \sum Bitwise\ XOR(A,B)$

6.2.1 Índice para nubes de puntos basado en Sarray e Índice Métrico

Una alternativa para representar las nubes de puntos mediante bitmaps utilizando una cantidad pequeña de memoria, es representar cada bitmap de la base de datos mediante un índice sucinto sarray.

Utilizando este índice sucinto es posible reducir el espacio de almacenamiento de 1 Mbit por bitmap (rejilla de 1000x1000) a tan solo 12 Kbits.

Dado lo anterior, el esquema de búsqueda de proximidad en espacio métrico para el problema de recuperación de nubes de puntos está compuesto por:

- Universo de objetos validos U: todas las nubes de puntos representadas por $\{0,1\}^D$, es decir, bitmaps de dimensión D, los cuales se almacenan utilizando un índice sarray.

- Métrica d: similitud coseno, coeficiente de Jaccard o distancia de Hamming

Dado el espacio métrico (U, d), se utilizará el índice para espacio métrico Vantage Point Tree para realizar búsquedas de proximidad.

El cálculo de las funciones de distancias anteriores involucra el cómputo del producto interno. Sea m_1, m_2 y n el número de unos del bitmap B_1, el número de unos del bitmap B_2 y la longitud de los bitmaps, respectivamente. Dado lo anterior, el cálculo del producto interno entre los bitmaps B_1 y B_2 es $O(n)$.

Sin embargo, dado que los bitmaps de la base de datos son almacenados utilizando índices sarray, es posible realizar el cálculo del producto interno de B_1 y B_2 en tiempo sublineal, utilizando la operación $select_1$ y el algoritmo clásico de intersección, calculando el producto interno en tiempo $O(m_1 + m_2)$, donde $m_1 \ll n \ y \ m_2 \ll n$. Por lo tanto, el uso del índice sarray para representar las nubes de puntos mediante bitmaps, permite utilizar una cantidad de memoria cercana al mínimo teórico de información y realizar el cálculo del producto interno entre dos bitmaps en tiempo sublineal, acelerando el cálculo de las funciones de distancia.

6.2.2 Índice para nubes de puntos basado en listas de posiciones relativas e Índice Métrico

Otra alternativa para representar las nubes de puntos mediante bitmaps utilizando una cantidad pequeña de memoria, es utilizar una lista de posiciones relativas. Sea $B = \{0,1\}^n$ un bitmap de longitud n, m el número de unos de B y j el j-ésimo 1 del bitmap B.

Para cada bitmap de la base de datos, se generará una lista de posiciones relativas l, donde se almacenará la posición de cada 1 en B, es decir:

$$l = \{select_1(B, j) \forall j \ \leq m \ \wedge j \geq 1\} \tag{40}$$

Para reducir el tiempo de ejecución del cálculo del producto interno utilizando esta representación, se utilizará el algoritmo de intersección SIMD Galloping (Lemire et al., 2016). Este algoritmo utiliza instrucciones SIMD para trabajar en bloques de 4 enteros de 32 bits simultáneamente, utilizando registros de 128 bits.

Capítulo 7. Resultados

En este capítulo se presentan los procedimientos utilizados para la evaluación del desempeño de las estructuras de datos propuestas para el problema de recuperación de nubes de puntos, las bases de datos utilizadas, una descripción de las pruebas experimentales realizadas y los resultados de las pruebas experimentales de los índices para nubes de puntos propuestos; enfocados en el tiempo de consulta, tiempo de construcción, memoria de almacenamiento utilizada y exhaustividad@k ante el problema de inserciones y borrados. Nuestro objetivo es realizar consultas en bases de datos de contenido multimedia de gran tamaño en menos de 1 segundo.

Para todas las pruebas experimentales realizadas, todos los índices propuestos para el problema de recuperación de nubes de puntos, incluyendo el índice de Wang, presentan $exhaustividad@1 = 1$ para las bases de datos sintéticas y $exhaustividad@1 \geq 0.989$ para la base de datos MIR Flickr-1M. Esto indica que la abstracción de nubes de puntos y los índices para nubes de puntos propuestos son robustos ante el problema de inserciones y borrados. Por lo tanto, solo nos enfocaremos en el tiempo de consulta, tiempo de construcción y memoria utilizada.

7.1 Entorno de desarrollo y equipo

Todos los índices propuestos fueron implementados en C++ utilizando el compilador G++ 4.7 con la bandera de optimización –O3 activada, con excepción de una segunda versión del índice Inverted Grid Index donde se utilizó la metodología MapReduce mediante Apache Spark 2.0.1, en específico la API Pyspark y Spark SQL. El código fuente de estas estructuras de datos se encuentra disponible en GitHub en la URL https://github.com/miramirezch/2DPointCloudIndexing. Además de Apache Spark, se utilizaron las siguientes librerías: Boost 1.6.2, SDSL Lite 2.0 (Gog et al., 2014) y SIMD Compression and Intersection (Lemire et al., 2016).

Todos los experimentos se realizaron en un servidor con 64 núcleos Intel Xeon CPU E7-4809 v3 @ 2.00GHz con 1.5 TB de memoria RAM, utilizando Ubuntu Linux 14.04.5 LTS.

Los índices presentados previamente utilizan un solo núcleo en todos los casos, a excepción de la segunda versión del índice Inverted Grid Index implementada en Apache Spark, la cual utiliza 60 núcleos del servidor. Esta versión se presenta con el nombre IGI Spark en la sección de resultados.

7.2 Bases de datos sintéticas

Se generaron dos bases de datos sintéticas de 10 millones de nubes de puntos de dos dimensiones, con 1000 puntos por nube, en un intervalo de $[0,10000) \times [0,10000)$, para la evaluación de los índices propuestos. Se consideraron dos distribuciones de datos: distribución uniforme y una mezcla de gaussianas de 3 componentes (GM).

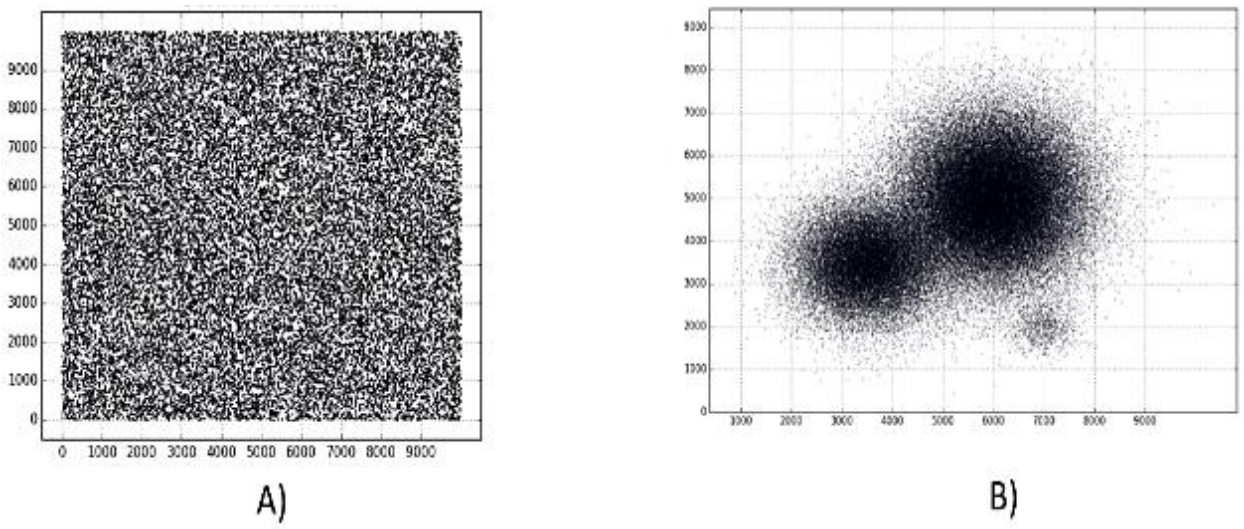


Figura 30.- Bases de datos sintéticas: puntos de interés generados en intervalo [0,10000)x[0,10000). A) Distribución uniforme - B) Distribución GM.

El objetivo de la base de datos con distribución GM es simular un caso adverso para datos que se encuentran en una rejilla discreta, donde un gran número de celdas están vacías y un pequeño número de celdas contienen una cantidad muy grande de datos.

Se generaron subconjuntos de estas bases de datos de 10K, 100K y 1M de nubes de puntos para ambas distribuciones.

7.3 Base de datos de Imágenes

Un punto fundamental es evaluar el desempeño de las estructuras de datos propuestas en situaciones realistas. Aunque existe un gran número de objetos multimedia que pueden ser representados mediante nubes de puntos en el plano, nos enfocaremos en la recuperación de imágenes en bases de datos de gran tamaño. A continuación se presenta información sobre la base de datos de imágenes utilizada y los diferentes algoritmos utilizados para generar la nube de puntos de cada imagen de la base de datos.

7.3.1- MIR FLICKR-1M

La base de datos de imágenes MIR Flickr-1M (Huiskes et al., 2010), es una extensión de la base de datos MIR Flickr que consiste de un millón de imágenes de alta calidad que provienen de usuarios de Flickr y que se encuentra disponible bajo la licencia Creative Commons. Esta base de datos cuenta información y herramientas adicionales; por ejemplo, anotaciones, metadatos (características de cámara, tiempo, fecha, localización, etc.), descriptores y herramientas basadas en la representación de bag-of-words para el análisis y la clasificación de estas imágenes.

Figura 31.- Imágenes de base de datos MIR Flickr-1M.

7.4 Evaluación del desempeño de estructuras propuestas ante inserciones y borrados

Para evaluar el desempeño de los índices propuestos ante el problema de inserciones y borrados, se seleccionó un subconjunto aleatorio de 1000 nubes de puntos de las bases de datos sintéticas para ambas distribuciones y un subconjunto aleatorio de 1000 imágenes de la base de datos MIR Flickr-1M.

Para ambos subconjuntos, datos sintéticos e imágenes, se realizaron las siguientes modificaciones:

- 333 nubes de puntos se modificaron con 1% de borrados y 1% de inserciones
- 333 nubes de puntos se modificaron con 5% de borrados y 5% de inserciones
- 334 nubes de puntos se modificaron con 10% de borrados y 10% de inserciones

Las inserciones y borrados fueron generadas con distribución uniforme. Estas nubes de puntos modificadas con inserciones/borrados serán utilizadas como nubes de puntos de consulta para la evaluación del desempeño de las estructuras de datos propuestas ante el problema de inserciones y borrados. Para evaluar el desempeño de las estructuras de datos propuestas, los siguientes parámetros serán evaluados:

- Tiempo de construcción
- Tiempo de consulta
- Memoria de almacenamiento
- $Exhaustividad@k$

7.5 Parámetros de estructuras de datos

Dado que hasta donde tenemos conocimiento, solo el índice de Wang está diseñado para el problema de recuperación de nubes de puntos, la evaluación se realiza comparando el desempeño de las estructuras propuestas en relación con nuestra implementación del índice de Wang implementada en C++. En esta implementación utilizamos un índice invertido para encontrar el máximo número de coincidencias (Müller, 2015).

Se utilizaron los siguientes parámetros para las estructuras de datos (para las bases de datos sintéticas):

- R*-tree: Max-Elementos/Nodo = 20, Min-Elementos/Nodo = 10
- IGI: $\delta = 10$
- Wang: $t_{offset} = 0, \Delta_{time} = 500, \Delta_{freq} = 500, \frac{Tuples}{Anchor} = 5$

Los siguientes parámetros se utilizaron para las estructuras de datos propuestas en la base de datos MIR Flickr-1M:

- R*-tree: Max-Elementos/Nodo = 20, Min-Elementos/Nodo = 10
- IGI: $\delta = 1$
- Wang: $t_{offset} = 0, \Delta_{time} = 50, \Delta_{freq} = 50, \frac{Tuples}{Anchor} = 5$

Para la versión del índice IGI Spark, realizado mediante la metodología MapReduce en Apache Spark, se utilizó la siguiente configuración de sesión:

- Spark.worker.cores: 60
- Spark.Worker.memory: 60 GB
- Spark.executor.cores: 5

- Spark.executor.instances: 12

- Spark.executor.memory: 20GB

En todas las pruebas experimentales realizadas, en lugar de proporcionar la tolerancia δ, se proporcionó el parámetro k, tal que un punto q de una nube de puntos empata cuando un punto de otra nube de puntos se encuentra dentro de los k vecinos más cercanos de q (ver sección 6.1.2). Se utilizó $k = 1$ para todas las pruebas experimentales, con excepción de una prueba experimental realizada con el índice basado en R*-Tree, donde se utilizó $\delta = 10$ (este último se muestra en la sección de resultados con la leyenda Rtree – Range).

7.6 Resultados en bases de datos sintéticas

A continuación se muestra el desempeño de los índices propuestos para el problema de recuperación de nubes de puntos en las bases de datos sintéticas con distribución uniforme y distribución GM. Las gráficas que se muestran en esta sección tienen una escala semi-logaritmica (base 10).

7.6.1 Índices Invertidos

7.6.1.1 Tiempo de Consulta

En la **Figura 32** y **Figura 33** se muestra el tiempo de consulta promedio de los índices propuestos basados en invertidos, para bases de datos de 10K, 100K, 1M y 10M de nubes de puntos con distribución uniforme y GM respectivamente.

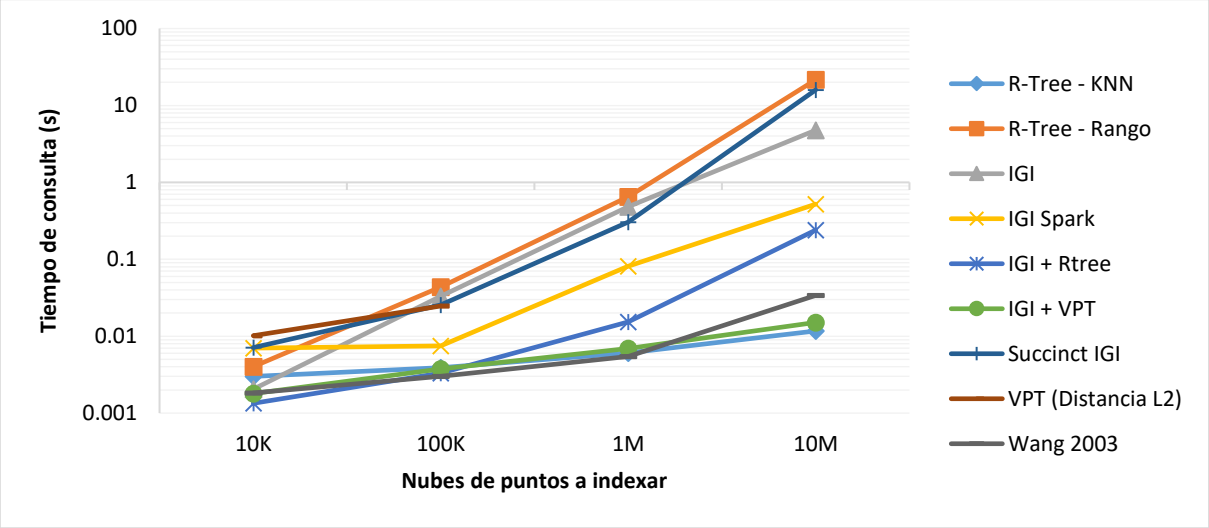


Figura 32.- Tiempo de consulta de índices para nubes de puntos basados en índices invertidos para consulta 1NN - Distribución Uniforme.

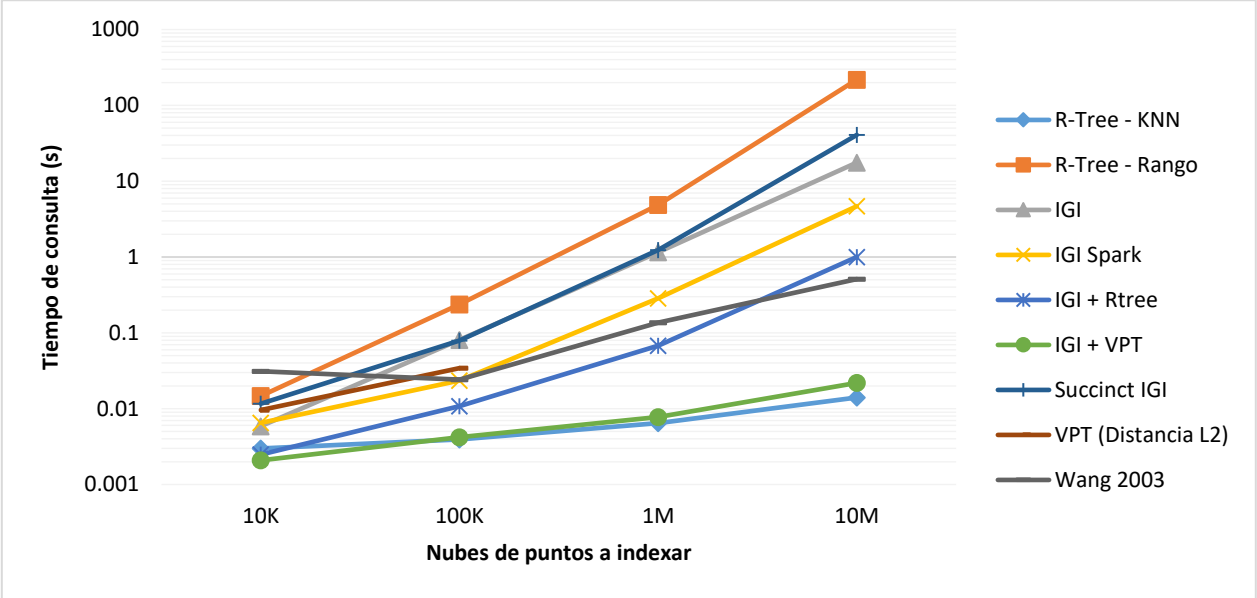


Figura 33.- Tiempo de consulta de índices para nubes de puntos basados en índices invertidos para consulta 1NN - Distribución GM

Podemos observar que el índice con mejor desempeño es el índice basado en R*-tree presentado en la página 49, con un tiempo de consulta promedio para una base de datos de 10M de nubes de puntos de 12ms y 14ms comparado con el tiempo de consulta promedio de nuestra implementación del índice de Wang de 34ms y 512ms para datos con distribución uniforme y GM, respectivamente.

Observe que el tiempo de consulta de la versión del índice IGI de Apache Spark supera los 4.5 segundos para la base de datos de 10M de nubes con distribución GM. Aunque esta versión utiliza los 60

núcleos del servidor, el módulo Spark SQL utilizado para el proceso de construcción y consulta, no permite índices SQL, lo que en algunas ocasiones tiene como resultado que se realicen lecturas secuenciales de todos los datos (Foundation, 2017b).

Un resultado muy interesante es el desempeño del índice propuesto en 6.1.7 Succinct Inverted Grid Index, el cual presenta un tiempo de consulta comparable y en algunos casos menor a la versión original del índice IGI, utilizando hasta 50% menos de memoria de almacenamiento, en las pruebas experimentales de 100K y 1M de nubes de puntos. En la **Tabla 1** y **Tabla 2** se muestra una comparación de la memoria de almacenamiento utilizada por la versión original del índice IGI y nuestra propuesta Succinct IGI.

Tabla 1.- Comparación de espacio de almacenamiento (MB) - IGI / Succinct IGI para distribución uniforme.

Índice	10K Nubes de puntos	100K Nubes de puntos	1M Nubes de puntos	10M Nubes de puntos
IGI	38	380	3796	37962
Succinct IGI (propuesto)	209	392	1727	15998

Tabla 2.- Comparación de espacio de almacenamiento (MB) - IGI / Succinct IGI para distribución GM.

Índice	10K Nubes de puntos	100K Nubes de puntos	1M Nubes de puntos	10M Nubes de puntos
IGI	38	379	3788	37875
Succinct IGI (propuesto)	91	244	1372	14274

7.6.1.2 Tiempo de Construcción

De acuerdo con las pruebas experimentales realizadas, el índice IGI Spark realizado con Apache Spark tiene el mejor tiempo de construcción comparado con el resto de los índices propuestos (diferencia de un orden de magnitud). Este es un resultado esperado, ya que esta versión utiliza 60 núcleos del servidor para el proceso de construcción.

Observe que los índices con mayor tiempo de construcción son el índice basado en VPT y nuestra implementación del índice propuesto por Wang para ambas distribuciones de datos. Con excepción de estos índices y el índice IGI Spark, el resto de los índices propuestos tienen tiempos de construcción comparables en ambas distribuciones como se puede observar en la **Figura 34** y **Figura 35**.

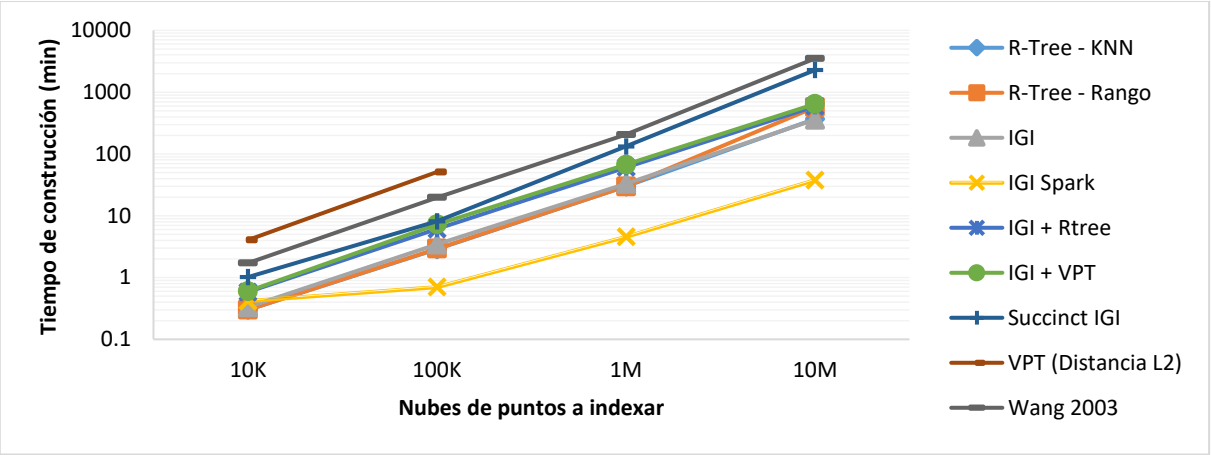


Figura 34.- Tiempo de construcción de índices para nubes de puntos basados en índices invertidos para consulta 1NN - Distribución Uniforme.

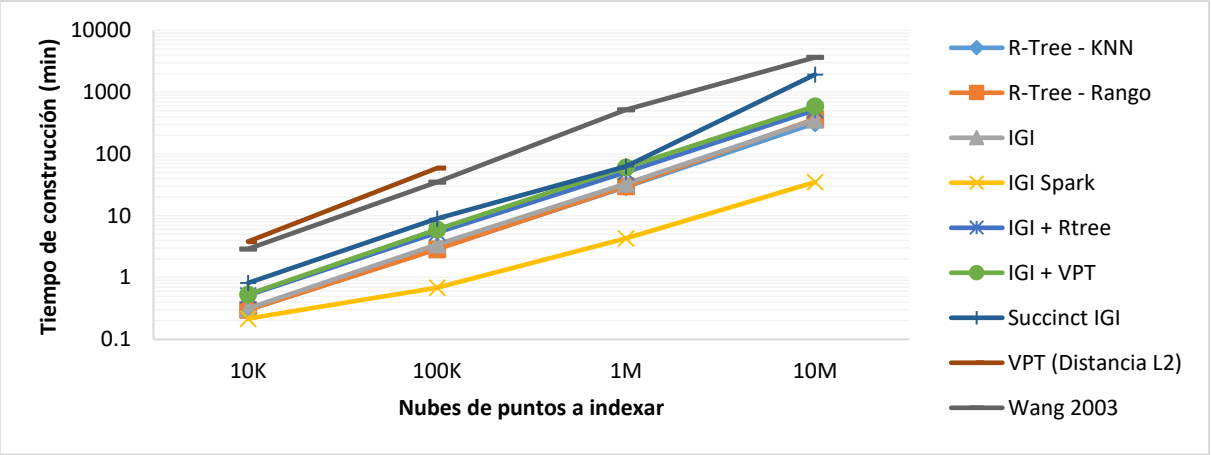
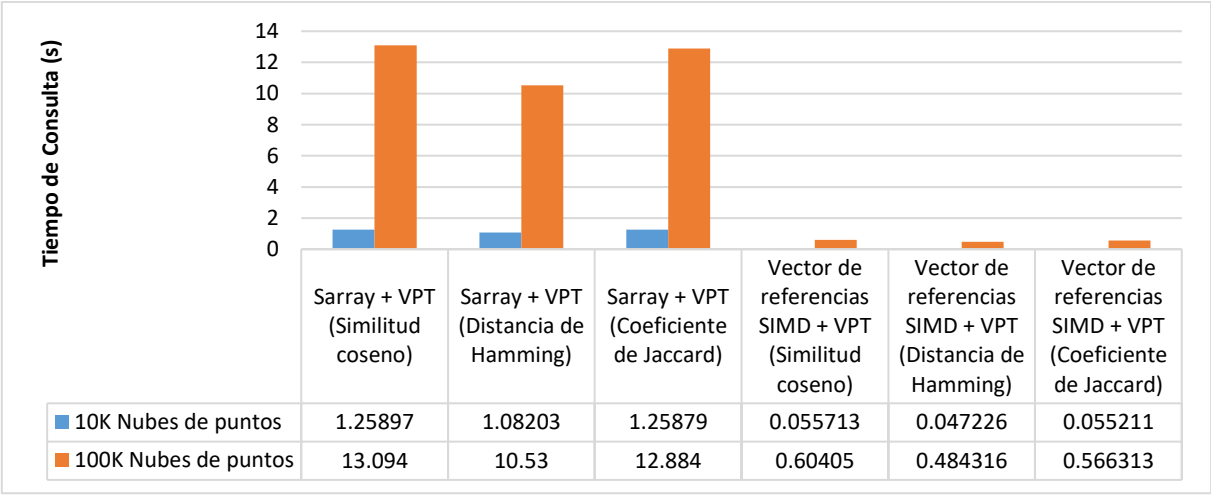


Figura 35.- Tiempo de construcción de índices para nubes de puntos basados en índices invertidos para consulta 1NN - Distribución GM.

7.6.2 Índices Métricos

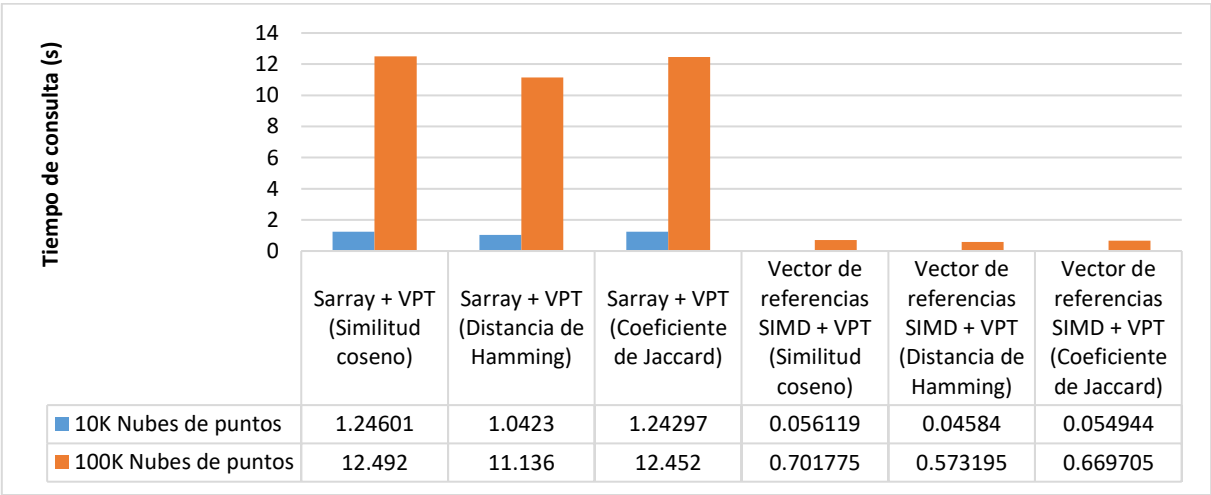
7.6.2.1 Tiempo de Consulta

En la **Figura 36** y **Figura 37** podemos observar que los tiempos de consulta de la mayoría de los índices propuestos basados en índices métricos, tienen un tiempo de consulta superior a un segundo en las bases de datos de 10K y 100K nubes de puntos. Debido a que nuestra meta es realizar consultas de proximidad en menos de un segundo en bases de datos de gran tamaño, estos índices muestran un pobre desempeño comparado con los resultados de los índices para nubes de puntos basados en índices invertidos.



	Sarray + VPT (Similitud coseno)	Sarray + VPT (Distancia de Hamming)	Sarray + VPT (Coeficiente de Jaccard)	Vector de referencias SIMD + VPT (Similitud coseno)	Vector de referencias SIMD + VPT (Distancia de Hamming)	Vector de referencias SIMD + VPT (Coeficiente de Jaccard)
10K Nubes de puntos	1.25897	1.08203	1.25879	0.055713	0.047226	0.055211
100K Nubes de puntos	13.094	10.53	12.884	0.60405	0.484316	0.566313

Figura 36.- Tiempo de consulta de índices para nubes de puntos basados en índices métricos para consulta 30NN - Distribución Uniforme.



	Sarray + VPT (Similitud coseno)	Sarray + VPT (Distancia de Hamming)	Sarray + VPT (Coeficiente de Jaccard)	Vector de referencias SIMD + VPT (Similitud coseno)	Vector de referencias SIMD + VPT (Distancia de Hamming)	Vector de referencias SIMD + VPT (Coeficiente de Jaccard)
10K Nubes de puntos	1.24601	1.0423	1.24297	0.056119	0.04584	0.054944
100K Nubes de puntos	12.492	11.136	12.452	0.701775	0.573195	0.669705

Figura 37.- Tiempo de consulta de índices para nubes de puntos basados en índices métricos para consulta 30NN - Distribución GM.

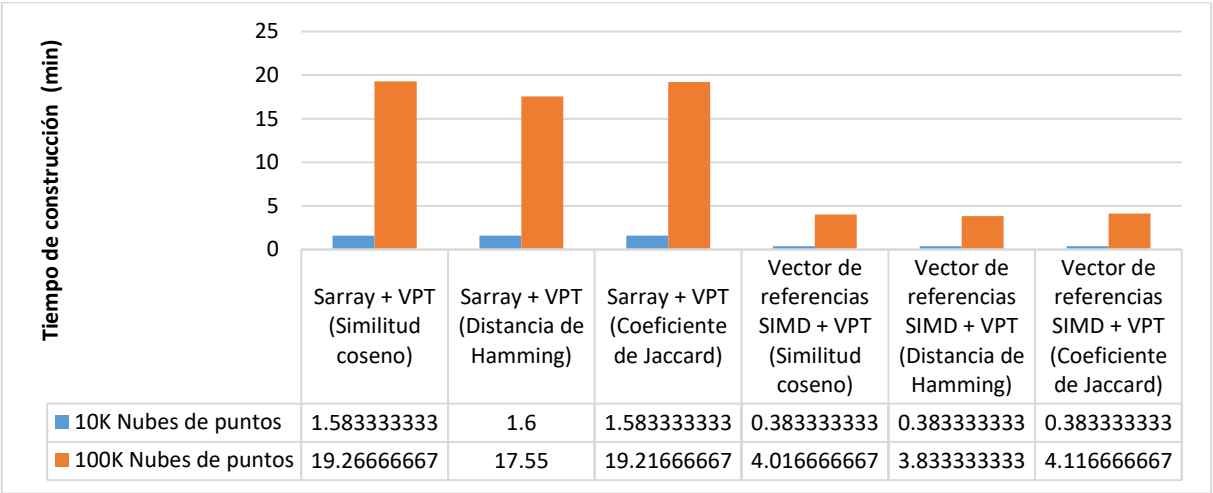
Dado que las nubes de puntos se representan mediante bitmaps de dimensión elevada (Dimensión 1×10^6), los índices para nubes de puntos basados en índices métricos se enfrentan al problema de la maldición de la dimensión y los tiempos de consulta no son competitivos comparados con los índices para nubes de puntos basados en índices invertidos.

De acuerdo a los resultados obtenidos, el uso de estos índices para nubes de puntos basados en índices métricos, puede ser viable solo para base de datos pequeñas. Algunas observaciones importantes son: no existe una diferencia significativa del tiempo de consulta de estos índices respecto a la distribución de los datos, existe una mejora del tiempo de consulta de los índices que utilizan listas de posiciones relativas de un orden de magnitud respecto a los que utilizan sarrays para representar los bitmaps.

Es importante señalar que los resultados experimentales presentados son para bitmaps con dimensión 1×10^6, sin embargo, se realizaron pruebas experimentales con bitmaps de diferentes dimensiones (10K, 40K, 1M y 100M) obteniendo resultados similares en tiempo de consulta.

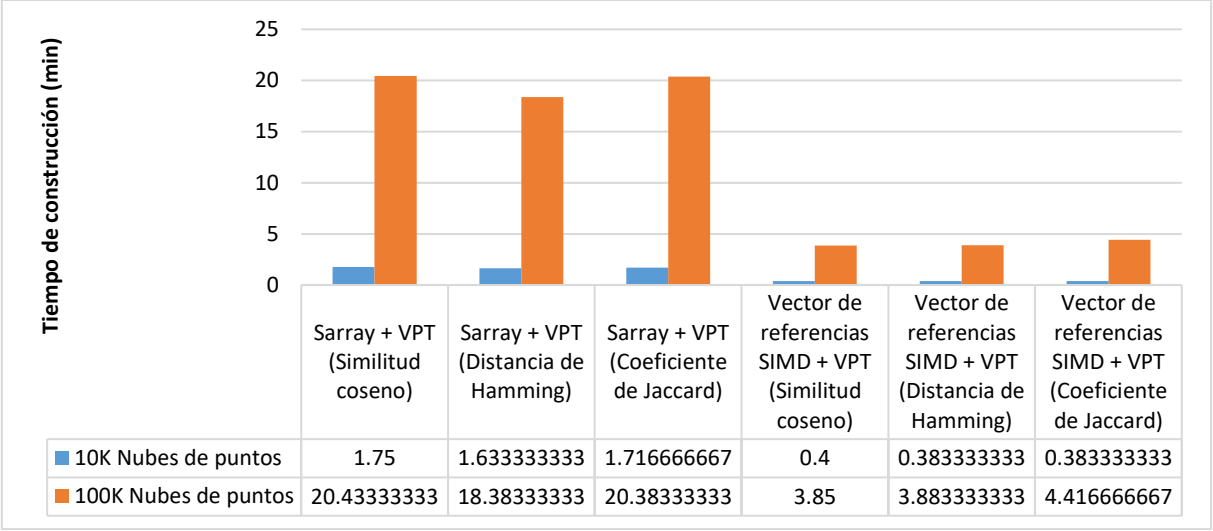
7.6.2.2 Tiempo de Construcción

Podemos observar en la **Figura 38** y **Figura 39**, que el tiempo de construcción de los índices que utilizan índices sucintos sarrays es aproximadamente 4 veces mayor que los índices que utilizan listas de referencias e instrucciones SIMD. Una observación adicional es que no existe una diferencia significativa en el tiempo de construcción de estos índices respecto a la distribución de los datos (uniforme o GM).



	Sarray + VPT (Similitud coseno)	Sarray + VPT (Distancia de Hamming)	Sarray + VPT (Coeficiente de Jaccard)	Vector de referencias SIMD + VPT (Similitud coseno)	Vector de referencias SIMD + VPT (Distancia de Hamming)	Vector de referencias SIMD + VPT (Coeficiente de Jaccard)
10K Nubes de puntos	1.583333333	1.6	1.583333333	0.383333333	0.383333333	0.383333333
100K Nubes de puntos	19.26666667	17.55	19.21666667	4.016666667	3.833333333	4.116666667

Figura 38.- Tiempo de construcción de índices para nubes de puntos basados en índices métricos para consulta 30NN - Distribución Uniforme.



	Sarray + VPT (Similitud coseno)	Sarray + VPT (Distancia de Hamming)	Sarray + VPT (Coeficiente de Jaccard)	Vector de referencias SIMD + VPT (Similitud coseno)	Vector de referencias SIMD + VPT (Distancia de Hamming)	Vector de referencias SIMD + VPT (Coeficiente de Jaccard)
10K Nubes de puntos	1.75	1.633333333	1.716666667	0.4	0.383333333	0.383333333
100K Nubes de puntos	20.43333333	18.38333333	20.38333333	3.85	3.883333333	4.416666667

Figura 39.- Tiempo de construcción de índices para nubes de puntos basados en índices métricos para consulta 30NN - Distribución GM.

7.6.3 Resumen de Resultados

En la **Tabla 3** se presenta un resumen de los resultados de las pruebas experimentales para la base de datos sintéticos de 10M de nubes de puntos con distribución GM (peor de los casos), donde se muestra el tiempo promedio de construcción, tiempo promedio de consulta, desviación estándar de tiempo promedio de consulta, memoria utilizada, dependencias y dificultad de implementación.

Se definen dos niveles para la dificultad de implementación:

1. Fácil

    a. La instalación de dependencias es sencilla.

    b. Estructura de datos cuenta con una API disponible.

2. Moderada

    a. Instalación de dependencias es sencilla pero se tiene que configurar el entorno de ejecución. Ejemplo: configuración de los parámetros de la sesión de Apache Spark.

    b. La estructura de datos no cuenta con una API disponible. Se proporciona un Script para realizar las pruebas experimentales.

Tabla 3.- Comparación de índices para nubes de puntos propuestos (Distribución GM) – 10M de nubes de puntos (~10,000 millones de puntos).

Clasificación	Índice	Construcción (min)	Consulta (s)	Consulta Desviación Estándar (s)	Memoria (GB)	Implementación
Esquema publicado	A. L. Wang, 2003	3654.68	0.5118	0.0344	465	1
	Rtree KNN	314.45	0.014041	0.00122508	495	1
	Rtree Rango	368.7	218.447	98.7248	495	1
	IGI(Ji et al., 2014)	357.2833	17.5925	0.468111	39	1
	IGI + R-Tree (Du et al., 2013)	515.9833	1.00137	0.146236	367	1
Esquema no publicado	IGI Spark	35.033	4.68	N/A	71	2
	IGI + VPT	592.7666	0.021884	0.00329599	630	1
	Succinct IGI	1925.7	40.727	3.20516	16	1
	Sarray + VPT	N/A	N/A	N/A	N/A	1
	Vector de referencias SIMD + VPT	N/A	N/A	N/A	N/A	1

7.7 MIR Flickr-1M

En esta sección se presentan los resultados de las pruebas experimentales de índices propuestos para el problema de recuperación de nubes de puntos, utilizando la base de datos de imágenes MIR Flickr-1M (Huiskes et al., 2010), la cual consiste en un conjunto de un millón de imágenes de alta calidad.

Se utilizó la librería OpenCV 3.2.0 (Python) para obtener los puntos de interés de las imágenes de la base de datos MIR Flickr-1M mediante los algoritmos SIFT y SURF. Para el descriptor SIFT, se generaron un máximo de 1000 puntos de interés por imagen y para el algoritmo SURF se seleccionó *hessianThreshold*=500 (promedio de 1000 puntos por imagen).

Para evaluar el desempeño de los índices ante el problema de inserciones y borrados, se seleccionó un subconjunto aleatorio de 1000 imágenes de la base de datos MIR Flickr-1M y se obtuvieron sus puntos de interés mediante el algoritmo SIFT y SURF. Posteriormente estas nubes de puntos se modificaron de la siguiente manera:

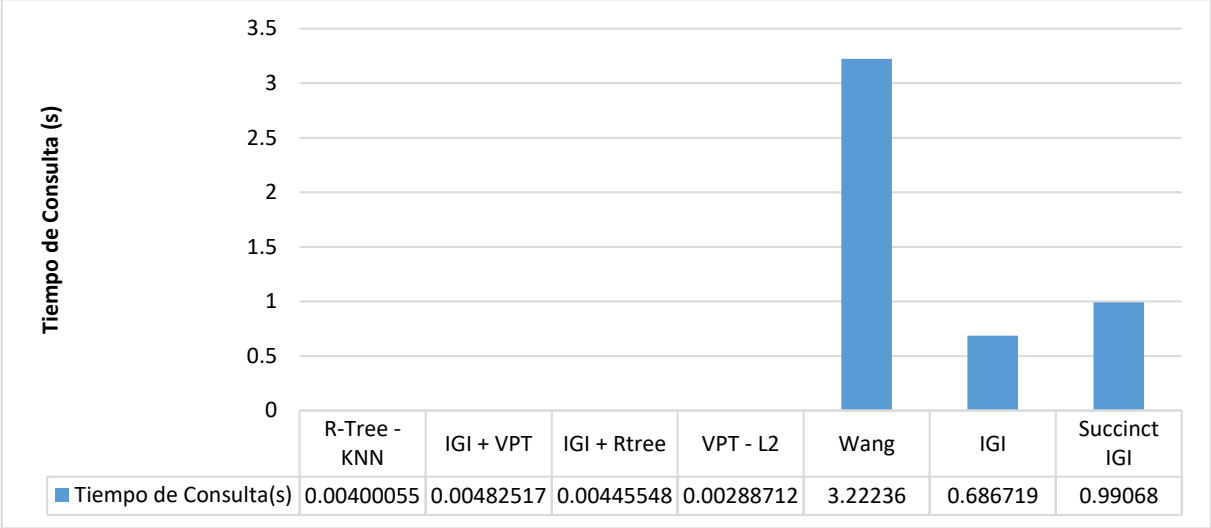
- 333 nubes de puntos se modificaron con 1% de borrados y 1% de inserciones
- 333 nubes de puntos se modificaron con 5% de borrados y 5% de inserciones
- 334 nubes de puntos se modificaron con 10% de borrados y 10% de inserciones

7.7.1 Resultados para Índices Invertidos

7.7.1.1 SIFT

En la **Figura 40** se presenta una gráfica con los tiempos de consulta promedio para nubes de puntos obtenidas mediante el algoritmo SIFT. Podemos observar que el tiempo de consulta del índice de Wang, tiene el mayor tiempo de consulta promedio de todos los índices para nubes de puntos. Además, el tiempo de consulta del índice de Wang, es mayor por un factor de 3 órdenes de magnitud respecto a los índices para nubes de puntos basados en R*-tree, VPT, IGI+Rtree e IGI+VPT. El índice basado en VPT presenta el mejor tiempo promedio de consulta con 2.8 ms comparado con 3.2s del índice de Wang.

Observe que los índices IGI y Succinct IGI presentan un tiempo de consulta promedio elevado. La razón por la cual el índice de Wang, IGI y Succinct IGI presentan tiempo de consulta elevados (respecto a los índices basados en Rtree, VPT, IGI+Rtree e IGI+VPT), es que estos índices utilizan un esquema basado en una tabla hash. La densidad de los datos en la rejilla discreta de la base de datos MIR Flickr-1M, es mayor comparada con la base de datos sintética, dado que los puntos de interés de las imágenes de MIR Flickr-1M se encuentran dentro del intervalo $[0,500) \times [0,500)$, en comparación con el intervalo $[0,10000) \times [0,10000)$ de la base de datos sintética. Por lo tanto, existe un mayor número de colisiones y listas invertidas de mayor tamaño. Una posible solución es aumentar la resolución de la rejilla para disminuir el número promedio de elementos en una celda, sin embargo, se debe de considerar el ruido dado que el intervalo de los puntos de interés $[0,500) \times [0,500)$ es muy pequeño.



	R-Tree - KNN	IGI + VPT	IGI + Rtree	VPT - L2	Wang	IGI	Succinct IGI
Tiempo de Consulta(s)	0.00400055	0.00482517	0.00445548	0.00288712	3.22236	0.686719	0.99068

Figura 40.- Tiempo de consulta de índices para nubes de puntos basados en índices invertidos para consulta 1NN - Base de datos MIR Flickr-1M con algoritmo SIFT.

Es importante mencionar que todos los índices para nubes de puntos basados en índices invertidos, incluyendo el índice de Wang, presentan $Exhaustividad@1 \geq 0.99$, confirmando el excelente desempeño de estas estructuras ante el problema de inserciones y borrados. En la **Tabla 4** se muestra un resumen de los resultados de las pruebas experimentales en relación con el tiempo promedio de consulta y exhaustividad.

Tabla 4.- Resumen de tiempo de consulta y exhaustividad de índices para nubes de puntos basados en índices invertidos para consulta 1NN - Base de datos MIR Flickr-1M utilizando algoritmo SIFT.

Índice	Consulta (s)	Exhaustividad@1	Exhaustividad@5	Exhaustividad@30
Rtree KNN	0.0040055	1	1	1
IGI+VPT	0.00482517	1	1	1
IGI+Rtree	0.00445548	1	1	1
VPT (Distancia L2)	0.00288712	1	1	1
A. L. Wang, 2003	3.22236	0.995	0.995	0.995
IGI	0.686719	0.998	1	1
Succinct IGI	0.99068	1	1	1

7.7.1.2 SURF

En esta sección se presentan los resultados del tiempo promedio de consulta para nubes de puntos obtenidas mediante el algoritmo SURF.

Podemos observar que en la **Figura 41**, el tiempo de consulta del índice de Wang tiene uno de los mayores tiempos de consulta promedio de todos los índices para nubes de puntos, con una diferencia de 3 órdenes de magnitud respecto a los índices para nubes de puntos basados en Rtree, VPT, IGI+Rtree e IGI+VPT. El índice basado en VPT presenta el mejor tiempo promedio de consulta con 3ms, comparado

con 2.06s del índice de Wang. Nuevamente podemos observar que los índices basados en tabla hash tienen los tiempos promedio de consulta más elevados.

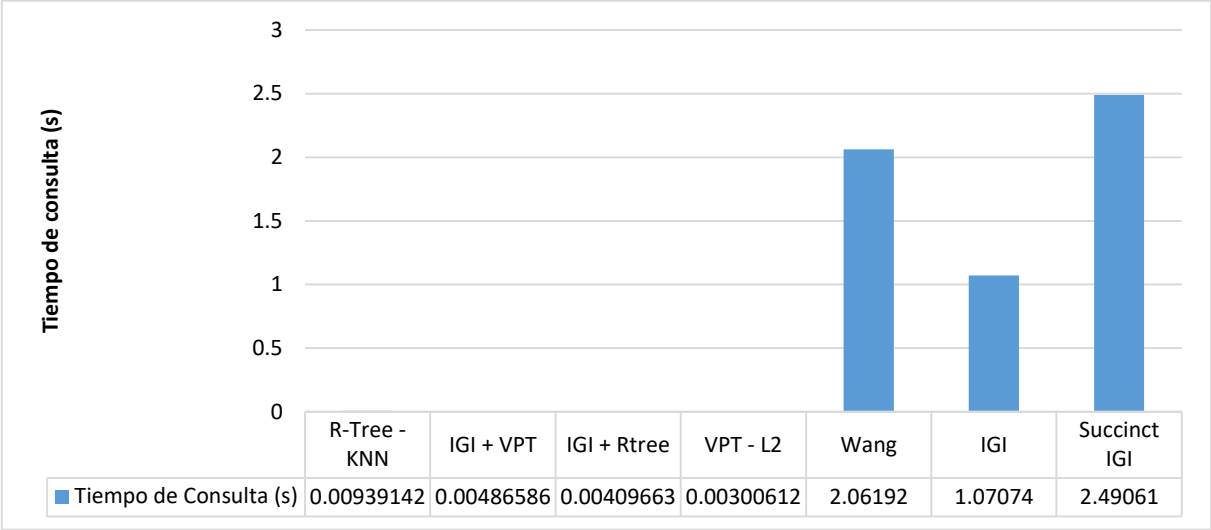


Figura 41.- Tiempo de consulta de índices para nubes de puntos basados en índices invertidos para consulta 1NN - Base de datos MIR Flickr-1M con algoritmo SURF.

Una diferencia respecto a las nubes de puntos obtenidas mediante el algoritmo SIFT, es que la Exhaustividad@1 de todos los índices se encuentra en el intervalo [0.997,0.999], alcanzando Exhaustividad@5=1 para todos los índices propuestos, con excepción del índice de Wang. En la **Tabla 5** se muestra un resumen de los resultados de las pruebas experimentales en relación con el tiempo de consulta y exhaustividad para el algoritmo SURF.

Tabla 5.- Resumen de tiempo de consulta y exhaustividad de índices para nubes de puntos basados en índices invertidos para consulta 1NN - Base de datos MIR Flickr-1M utilizando el algoritmo SURF.

Índice	Consulta (s)	Exhaustividad@ 1	Exhaustividad@ 5	Exhaustividad@ 30
Rtree KNN	0.00939142	0.998	1	1
IGI+VPT	0.00486586	0.999	1	1
IGI+Rtree	0.00409663	0.999	1	1
VPT (Distancia L2)	0.00300612	0.998	1	1
Wang	2.06192	0.997	0.997	0.997
IGI	1.07074	0.999	1	1
Succinct IGI	2.49061	0.999	1	1

7.7.2 Resumen de Resultados

De acuerdo con las pruebas experimentales realizadas en la base de datos MIR Flickr-1M, los índices para nubes de puntos basados en índices invertidos, tienen un desempeño superior en tiempo promedio de consulta respecto al índice para nubes de puntos de Wang (3.2 segundos para SIFT), en comparación con el índice propuesto para nubes de puntos basado en VPT (2 mili-segundos para SIFT). De igual manera, los resultados confirman que estos índices son robustos ante el problema de inserciones y borrados ($Exhaustividad@1 \geq 0.99$), validando los resultados obtenidos en las pruebas experimentales para datos sintéticos.

7.8 Puntos de interés necesarios para el problema de recuperación de nubes de puntos en imágenes

Hasta este momento, se utilizaron aproximadamente 1000 puntos de interés por objeto en las pruebas realizadas con las bases de datos sintéticas y la base de datos MIR Flickr-1M, con el objetivo de tener una cota superior en el tiempo promedio de consulta y tiempo de construcción. Dado que en la práctica, el tiempo promedio de consulta es proporcional al número de puntos de interés de la nube de puntos, es fundamental determinar el número de puntos de interés necesario para recuperar una nube de puntos de una base de datos, manteniendo $exhaustividad@1 \approx 1$, bajo el efecto de inserciones y borrados.

Dado lo anterior, se realizaron dos pruebas experimentales en la base de datos MIR Flickr-1M. En la primera prueba utilizamos el algoritmo SIFT para calcular un máximo de 100 puntos de interés por imagen. Para evaluar el desempeño ante inserciones y borrados, se seleccionó un subconjunto de 1000 imágenes de la base de datos y se modificaron sus nubes de puntos de la siguiente manera: 333 nubes de puntos se modificaron con 1% de borrados y 1% de inserciones, 333 nubes de puntos se modificaron con 5% de borrados y 5% de inserciones, 334 nubes de puntos se modificaron con 10% de borrados y 10% de inserciones.

Tabla 6.- Resultados de búsqueda en base de datos MIR Flickr 1M utilizando nubes de puntos con 100 puntos de interés calculados mediante el algoritmo SIFT.

Índice	Construcción (s)	Consulta (s)	Exhaustividad@1	Exhaustividad@5	Exhaustividad@30
Rtree KNN	253	0.000477	0.999	1	1
IGI+VPT	112	0.000315	0.999	1	1
IGI+Rtree	38	0.000202	1	1	1
VPT (Distancia L2)	165	0.000159	1	1	1
A. L. Wang, 2003	220	0.034356	0.998	0.998	0.998
IGI	28	0.011627	0.999	0.999	0.999
Succinct IGI	340	0.020629	0.999	0.999	0.999

De acuerdo con los resultados de la **Tabla 6**, podemos observar que existe una mejora del tiempo promedio de consulta de al menos un orden de magnitud en todos los índices, comparado con el tiempo

de consulta de las pruebas realizadas que utilizan 1000 puntos de interés por imagen (**Tabla 4**), manteniendo $exhaustividad@1 \geq 0.99$.

En la segunda prueba utilizamos el algoritmo SIFT para calcular un máximo de 25 puntos de interés por imagen. Para evaluar el desempeño ante inserciones y borrados, se seleccionó un subconjunto de 1000 imágenes de la base de datos y se modificaron sus nubes de puntos de la siguiente manera: 333 nubes de puntos se modificaron con 10% de borrados y 10% de inserciones, 333 nubes de puntos se modificaron con 20% de borrados y 20% de inserciones, 334 nubes de puntos se modificaron con 40% de borrados y 40% de inserciones.

Tabla 7.- Resultados de búsqueda en base de datos MIR Flickr 1M utilizando nubes de puntos con 25 puntos de interés calculados mediante el algoritmo SIFT.

Índice	Construcción (s)	Consulta (s)	Exhaustividad @1	Exhaustividad @5	Exhaustividad @30
Rtree KNN	7	0.000089243	1	1	1
IGI+VPT	24	0.000056	1	1	1
IGI+Rtree	9	0.000031	1	1	1
VPT (Distancia L2)	36	0.000036	1	1	1
A. L. Wang, 2003	36	0.00018957	0.989	0.989	0.989
IGI	6	0.00032246	1	1	1
Succinct IGI	121	0.00107162	0.998	0.999	0.999

En la **Tabla 7** se muestran los resultados de la segunda prueba experimental. De acuerdo a los resultados obtenidos, existe una mejora en el tiempo promedio de consulta de dos órdenes de magnitud utilizando 25 puntos de interés comparados con las pruebas experimentales que utilizan 1000 puntos de interés por imagen, logrando un tiempo promedio de consulta de 36us para el índice basado en VPT, comparado con 189us para el índice de Wang. Observe que en esta prueba experimental, se aumentó el efecto de inserciones y borrados pasando de un promedio de 5.33% de inserciones/borrados de las pruebas anteriores, a un 23.33% de inserciones/borrados manteniendo $exhaustividad@1 \geq 0.999$ para los índices propuestos y $exhaustividad@1 = 0.989$ para el índice de Wang.

Capítulo 8. Conclusiones

8.1 Resumen

En esta tesis se abordó el problema de recuperación de nubes de puntos para objetos multimedia, enfocados en el diseño de estructuras de datos que sean robustas ante el problema de inserciones y borrados. Para esto, se propusieron dos esquemas de solución para los índices de nubes de puntos: índices invertidos e índices métricos.

Se realizaron pruebas experimentales en una base de datos de datos sintéticas de 10 millones de nubes de puntos y en la base de datos MIR Flickr-1M (1 millón de imágenes de alta calidad). Los resultados obtenidos muestran que los índices para nubes de puntos propuestos basados en índices invertidos, tienen un desempeño superior respecto al índice para nubes de puntos de Wang, el cual hasta donde tenemos conocimiento es el único índice diseñado para nubes de puntos, logrando en pruebas experimentales una mejora de hasta tres órdenes de magnitud en tiempo promedio de consulta (ver **Tabla 4** y **Tabla 5**). Los índices para nubes de puntos propuestos basados en índices métricos, no tienen un buen desempeño en tiempo de consulta para bases de datos de gran tamaño, ya que la representación de las nubes de puntos mediante bitmaps es de dimensión elevada y se enfrentan al problema de la maldición de la dimensión.

Finalmente, se realizó una prueba experimental donde se determinó que 25 puntos de interés por imagen son suficientes para resolver el problema de recuperación de nubes de puntos manteniendo una $exhaustividad@1 \approx 1$ aun bajo el efecto de inserciones y borrados.

8.2 Conclusiones

A continuación se muestran las conclusiones obtenidas durante el desarrollo de esta tesis:

1. El problema de recuperación de nubes de puntos puede responderse en 14ms (peor de los casos) para bases de datos de 10 millones de nubes de puntos con 1000 puntos por nube, logrando $exhaustividad@1 = 1$ incluso bajo el efecto de inserciones y borrados.

2. Los índices para nubes de puntos basados en índices invertidos, presentan un desempeño general superior en tiempo promedio de consulta comparado con el índice para nubes de

puntos de Wang, logrando mejoras en tiempo promedio de consulta de hasta 3 órdenes de magnitud.

3. Los índices para nubes de puntos basados en índices métricos, los cuales utilizan cadenas binarias para representar una nube de puntos, tienen un desempeño inferior comparado con los índices propuestos basados en índices invertidos y con el índice propuesto por Wang, debido a que se enfrentan al problema de la maldición de la dimensión.

4. La abstracción de nubes de puntos para objetos multimedia y los índices propuestos para nubes de puntos, son robustos ante inserciones y borrados, logrando $exhaustividad@1 \geq 0.99$ para nubes de puntos incluso con un promedio 23% de inserciones/borrados en bases de datos de gran tamaño (1M de nubes de puntos).

5. De acuerdo con las pruebas experimentales realizadas en la base de datos MIR Flickr-1M, no existe una diferencia significativa en tiempo de consulta o exhaustividad en relación con el algoritmo utilizado para calcular los puntos de interés de la imagen (SIFT o SURF).

6. Aunque los métodos de indexados para consultas kNN join no pueden aplicarse directamente para resolver el problema de recuperación de nubes de puntos, las estrategias e ideas de estos índices se utilizaron para los índices de nubes de puntos propuestos en esta tesis. Por ejemplo, la partición de los datos mediante una rejilla discreta, el uso de la metodología MapReduce para el proceso de construcción y consulta, entre otras.

7. De acuerdo con las pruebas experimentales realizadas, en los índices para nubes de puntos basados en índices métricos, la distancia de Hamming en general presenta mejores resultados en tiempo promedio de consulta, aunque la diferencia no es significativa, ya que el tiempo de consulta de la distancia de Hamming, la similitud coseno y el coeficiente de Jaccard es proporcional a el tamaño del bitmap. Es importante señalar que estos índices presentan $exhaustividad@1 = 1$ para todas las distancias.

8. La principal ventaja de utilizar una rejilla discreta es que facilita la implementación en paralelo de los índices propuestos. Por ejemplo, el índice IGI implementado mediante Apache Spark, explota esta característica, lo cual permite utilizar el paradigma MapReduce

para el proceso de construcción y el proceso de consulta. Esto se debe a que la información de cada celda es independiente de sus celdas adyacentes, lo que permite distribuir la información a diferentes nodos en un sistema distribuido.

8.3 Contribuciones

A continuación se presentan las contribuciones de este trabajo de tesis:

- Los índices para nubes de puntos basados en índices invertidos presentados en esta tesis (Basados en R*-Tree, VPT, IGI, IGI+R*-Tree, IGI+VPT) tienen un desempeño superior comparado con el índice para nubes de puntos de Wang, logrando una mejora de hasta tres órdenes de magnitud en tiempo promedio de consulta (2.8 ms para VPT comparado con 3.2s del índice de Wang en la base de datos MIR Flickr-1M).

- Los índices para nubes de puntos basados en índices invertidos e índices métricos, son robustos ante el problema de inserciones y borrados, logrando en todas las pruebas experimentales en bases de datos sintéticos y de imágenes una $exhaustividad@1 \geq 0.99$ utilizando nubes de puntos de consulta hasta con un promedio de 23% de inserciones y borrados.

- Se propone el índice Succinct IGI, el cual es una versión sucinta del índice IGI, el cual utiliza en algunos casos un 50% menos de memoria de almacenamiento, con un incremento en tiempo de consulta de tan solo 6% respecto al índice IGI.

- Se generó una librería con todos los índices implementados en el desarrollo de esta tesis. Los índices se implementaron en C++ y se encuentra disponible en: https://github.com/miramirezch/2DPointCloudIndexing

8.4 Trabajo futuro

A continuación se presenta una lista de algunos puntos que pueden ser estudiados a partir de los resultados de esta tesis:

- Dado que el tiempo promedio de consulta de los índices propuestos basados en índices invertidos, es dominado por el tiempo de consulta del índice invertido seleccionado, es fundamental utilizar el índice cuyo tiempo de consulta en el peor de los casos sea sublineal y que mejor se aproxime a $O(1)$. Si bien, el índice VPT tiene tiempo de consulta en el peor de los casos de $O(\log n)$, esto solo se cumple para un radio de búsqueda muy pequeño. En el año 1998, se presenta una estructura de datos para resolver el problema *Halfspace Range Reporting* (Chan, 1998), la cual puede utilizarse para resolver consultas de rango en $\mathbb{R}^2$ en tiempo $O(\log n + k)$ en el peor de los casos, donde k es la cantidad de puntos entregada por la consulta. Un siguiente paso sería implementar esta estructura de datos para el problema de recuperación de nubes de puntos utilizando esta estructura como índice invertido.

- Dado que los índices propuestos en esta tesis trabajan solo en memoria principal (memoria RAM), esto presenta una desventaja al momento de aumentar el tamaño de las bases de datos, ya que una vez que se agote la memoria RAM disponible, no es posible escalar estas estructuras. Una alternativa para mejorar la escalabilidad de las estructuras propuestas, es implementar versiones sucintas de los índices invertidos utilizados, por ejemplo, versiones sucintas del índice R*-Tree o Vantage Point Tree. Otra alternativa para bases de datos de mayor tamaño, es el diseño de estructuras de datos para nubes de puntos que trabajen en memoria secundaria (disco duro). En el caso específico de la versión del índice IGI en Apache Spark, se puede almacenar y trabajar en memoria secundaria utilizando el formato ORC o Parquet. Estos formatos están diseñados para reducir las operaciones de lectura/escritura y el espacio de almacenamiento.

- Dado que los índices para nubes de puntos propuestos son robustos ante el problema de inserciones y borrados, el siguiente paso es generar un sistema de recuperación de imágenes cuyas nubes de puntos sean invariantes a transformaciones de similitud, afines o proyectivas, utilizando alguna de las invariantes del estado del arte.

- Todos los índices presentados en esta tesis (a excepción del índice IGI en su versión de Apache Spark) utilizan un solo procesador. Por lo tanto, una alternativa para mejorar el desempeño de los índices para nubes de puntos propuestos, es paralelizar estas estructuras utilizando diferentes herramientas como MapReduce, CUDA o multi-threading. Un análisis respecto a la mejora en tiempo de consulta, construcción y exhaustividad es necesario para evaluar el porcentaje de mejora respecto a las estructuras que utilizan un solo procesador.

- En relación con la librería de índices para nubes de puntos implementada durante el transcurso de esta tesis disponible en GitHub, una mejora es implementar un módulo para serializar estos índices. En este momento, esta librería no permite almacenar los índices (con excepción de la versión del índice IGI en Apache Spark), por lo tanto, es necesario construir el índice cada vez que se inicia una aplicación. El proceso de serialización puede realizarse mediante el módulo de serialización de la librería Boost, disponible en C++.

- Una alternativa para agilizar el tiempo promedio de consulta de la versión del índice IGI Spark, es utilizar marcos de trabajo como Apache Ignite o Apache Kudu para mejorar el tiempo de consultas SQL, ya que estos permiten utilizar índices, los cuales evitan realizar en algunos casos búsqueda lineal.

Literatura citada

Alzu'bi, A., Amira, A., Ramzan, N. 2015. Semantic content-based image retrieval: A comprehensive study. *Journal of Visual Communication and Image Representation*, *32*, 20–54. https://doi.org/10.1016/j.jvcir.2015.07.012

Bay, H., Ess, A., Tuytelaars, T., Van Gool, L. 2008. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, *110*(3), 346–359. https://doi.org/10.1016/j.cviu.2007.09.014

Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B. 1990. The R*-tree: An efficient and Robust Access Method for Points and Rectangles. En *Proceedings of the 1990 ACM SIGMOD international conference on Management of data - SIGMOD '90* (Vol. 19, pp. 322–331). New York, New York, USA: ACM Press. https://doi.org/10.1145/93597.98741

Bentley, J. L. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, *18*(9), 509–517. https://doi.org/10.1145/361002.361007

Berg, M., Cheong, O., van Kreveld, M., Overmars, M. 2008. Orthogonal Range Searching. En *Computational Geometry* (3a ed., pp. 95–120). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-77974-2_5

Bhima, K., Aruna Sri, T., Ramaiah, K. D., Jagan, A. 2012. Exerting Spatial Join and KNN Queries on Spatial Database. En *2012 International Conference on Recent Advances in Computing and Software Systems* (pp. 260–266). IEEE. https://doi.org/10.1109/RACSS.2012.6212678

Burkhard, W. A., Keller, R. M. 1973. Some Approaches to Best-Match File Searching. *Communications of the ACM*, *16*(4), 230–236. https://doi.org/10.1145/362003.362025

Chan, T. M. 1998. Random Sampling, Halfspace Range Reporting, and construction of (<k)-Levels in Three Dimensions. En *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280)* (pp. 586–595). IEEE Comput. Soc. https://doi.org/10.1109/SFCS.1998.743509

Chávez, E., Chávez Cáliz, A. C., López-López, J. L. 2016. Affine invariants of generalized polygons and matching under affine transformations. *Computational Geometry*, *58*, 60–69. https://doi.org/10.1016/j.comgeo.2016.06.003

Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J. L. 2001. Searching in Metric Spaces. *ACM Computing Surveys*, *33*(3), 273–321. https://doi.org/10.1145/502807.502808

Chen, L., Gao, Y., Chen, G., Zhang, H. 2016. Metric All-k-Nearest-Neighbor Search. *IEEE Transactions on Knowledge and Data Engineering*, *28*(1), 98–112. https://doi.org/10.1109/TKDE.2015.2453954

Crow, F. C. 1984. Summed-area tables for texture mapping. En *Proceedings of the 11th annual conference on Computer graphics and interactive techniques - SIGGRAPH '84* (Vol. 18, pp. 207–212). New York, New York, USA: ACM Press. https://doi.org/10.1145/800031.808600

Del Bimbo, A. 1999. *Visual information retrieval* (1st ed.). San Francisco, California: Morgan Kaufmann Publishers. Recuperado a partir de http://dl.acm.org/citation.cfm?id=319280

Du, Q., Li, X. 2013. A Novel KNN Join Algorithms based on Hilbert R-tree in MapReduce. En *Proceedings of 2013 3rd International Conference on Computer Science and Network Technology* (pp. 417–420). IEEE. https://doi.org/10.1109/ICCSNT.2013.6967143

Foundation, A. S. 2017a. Apache Spark - Lightning-Fast Cluster Computing. Recuperado el 28 de junio de 2017, a partir de https://spark.apache.org/

Foundation, A. S. 2017b. Fast Apache Spark SQL Queries - Apache Ignite. Recuperado el 29 de abril de 2017, a partir de https://ignite.apache.org/use-cases/spark/sql-queries

Foundation, A. S. 2017c. Welcome to Apache Hadoop! Recuperado el 28 de junio de 2017, a partir de http://hadoop.apache.org/

Fredriksson, K., Braithwaite, B. 2013. Quicker Similarity Joins in Metric Spaces. En *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 8199 LNCS, pp. 127–140). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-41062-8_13

Gog, S., Beller, T., Moffat, A., Petri, M. 2014. From Theory to Practice: Plug and Play with Succinct Data Structures. En *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 8504 LNCS, pp. 326–337). Springer, Cham. https://doi.org/10.1007/978-3-319-07959-2_28

Guttman, A. 1984. R-TREES: A DYNAMIC INDEX STRUCTURE FOR SPATIAL SEARCHING. En *Proceedings of the 1984 ACM SIGMOD international conference on Management of data - SIGMOD '84* (Vol. 14, p. 47). New York, New York, USA: ACM Press. https://doi.org/10.1145/602259.602266

Hong, L., Wan, Y., Jain, A. 1998. Fingerprint Image Enhancement: Algorithm and Performance Evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*(8), 777–789. https://doi.org/10.1109/34.709565

Huiskes, M. J., Thomee, B., Lew, M. S. 2010. New Trends and Ideas in Visual Concept Detection. En *Proceedings of the international conference on Multimedia information retrieval - MIR '10* (p. 527). New York, New York, USA: ACM Press. https://doi.org/10.1145/1743384.1743475

IBM. 2017. What is MapReduce? – IBM Analytics. Recuperado el 28 de junio de 2017, a partir de https://www.ibm.com/analytics/us/en/technology/hadoop/mapreduce/

Iwamura, M., Nakai, T., Kise, K. 2007. Improvement of Retrieval Speed and Required Amount of Memory for Geometric Hashing by Combining Local Invariants. En *Procedings of the British Machine Vision Conference 2007* (p. 103.1-103.10). British Machine Vision Association. https://doi.org/10.5244/C.21.103

Jang, M., Shin, Y.-S., Chang, J.-W. 2015. A Grid-based k-Nearest Neighbor Join for Large Scale Datasets on MapReduce. En *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems* (pp. 888–891). IEEE. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.189

Jegou, H., Douze, M., Schmid, C., Perez, P. 2010. Aggregating local descriptors into a compact image representation. En *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 3304–3311). IEEE. https://doi.org/10.1109/CVPR.2010.5540039

Ji, C., Li, Z., Qu, W., Xu, Y., Li, Y. 2014. Scalable nearest neighbor query processing based on Inverted Grid Index. *Journal of Network and Computer Applications*, *44*, 172–182. https://doi.org/10.1016/j.jnca.2014.05.010

Kim, Y. J., Patel, J. 2010. Performance Comparison of the R*-Tree and the Quadtree for kNN and Distance Join Queries. *IEEE Transactions on Knowledge and Data Engineering*, *22*(7), 1014–1027. https://doi.org/10.1109/TKDE.2009.141

Lemire, D., Boytsov, L., Kurz, N. 2016. SIMD compression and the intersection of sorted integers. *Software:*

Practice and Experience, *46*(6), 723–749. https://doi.org/10.1002/spe.2326

Lindeberg, T. 2013. Scale Selection Properties of Generalized Scale-Space Interest Point Detectors. *Journal of Mathematical Imaging and Vision*, *46*(2), 177–210. https://doi.org/10.1007/s10851-012-0378-3

Lowe, D. G. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, *60*(2), 91–110. https://doi.org/10.1023/B:VISI.0000029664.99615.94

Lu, W., Shen, Y., Chen, S., Ooi, B. C. 2012. Efficient Processing of k Nearest Neighbor Joins using MapReduce. *Proceedings of the VLDB Endowment*, *5*(10), 1016–1027. https://doi.org/10.14778/2336664.2336674

Moise, D., Shestakov, D., Gudmundsson, G., Amsaleg, L. 2013. Indexing and searching 100M images with map-reduce. En *Proceedings of the 3rd ACM conference on International conference on multimedia retrieval - ICMR '13* (p. 17). New York, New York, USA: ACM Press. https://doi.org/10.1145/2461466.2461470

Müller, M. 2015. Content-Based Audio Retrieval. En *Fundamentals of Music Processing* (1a ed., pp. 355–413). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-21945-5_7

Nakai, T., Kise, K., Iwamura, M. 2006. Use of Affine Invariants in Locally Likely Arrangement Hashing for Camera-Based Document Image Retrieval. En *Document Analysis Systems VII* (pp. 541–552). Springer, Berlin, Heidelberg. https://doi.org/10.1007/11669487_48

Navarro, G. 2016. *Compact Data Structures*. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9781316588284

Navarro, G., Sadakane, K. 2014. Fully Functional Static and Dynamic Succinct Trees. *ACM Transactions on Algorithms*, *10*(3), 1–39. https://doi.org/10.1145/2601073

Nguyen Mau Toan, Yasushi, I. 2016. Audio Fingerprint Hierarchy Searching on Massively Parallel with multi-GPGPUs using K-modes and LSH. En *2016 Eighth International Conference on Knowledge and Systems Engineering (KSE)* (pp. 49–54). IEEE. https://doi.org/10.1109/KSE.2016.7758028

Okanohara, D., Sadakane, K. 2007. Practical Entropy-Compressed Rank/Select Dictionary. En *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)* (pp. 60–70). Philadelphia, PA: Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611972870.6

Salakhutdinov, R., Hinton, G. 2009. Semantic hashing. *International Journal of Approximate Reasoning*, *50*(7), 969–978. https://doi.org/10.1016/j.ijar.2008.11.006

Sun, X., Wang, C., Xu, C., Zhang, L. 2013. Indexing Billions of Images for Sketch-based Retrieval. En *Proceedings of the 21st ACM international conference on Multimedia - MM '13* (pp. 233–242). New York, New York, USA: ACM Press. https://doi.org/10.1145/2502081.2502281

Wan, J., Wang, D., Hoi, S. C. H., Wu, P., Zhu, J., Zhang, Y., Li, J. 2014. Deep Learning for Content-Based Image Retrieval: A Comprehensive Study. En *Proceedings of the ACM International Conference on Multimedia - MM '14* (pp. 157–166). New York, New York, USA: ACM Press. https://doi.org/10.1145/2647868.2654948

Wang, A. L. 2003. An Industrial-Strength Audio Search Algorithm. En *Proceedings of the 4th International Society for Music Information Retrieval Conference (ISMIR 2003), Baltimore, Maryland (USA), 26-30 October 2003* (pp. 7–13). https://doi.org/10.1.1.217.8882

Wang, J., Liu, W., Kumar, S., Chang, S.-F. 2016. Learning to Hash for Indexing Big Data - A Survey.

Proceedings of the IEEE, *104*(1), 34–57. https://doi.org/10.1109/JPROC.2015.2487976

Yandex, A. B., Lempitsky, V. 2016. Efficient Indexing of Billion-Scale datasets of deep descriptors. En *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 2055–2063). IEEE. https://doi.org/10.1109/CVPR.2016.226

Yianilos, P. N. 1993. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, 311–321. Recuperado a partir de http://dl.acm.org/citation.cfm?id=313789

Yu, C., Cui, B., Wang, S., Su, J. 2007. Efficient index-based KNN join processing for high-dimensional data. *Information and Software Technology*, *49*(4), 332–344. https://doi.org/10.1016/j.infsof.2006.05.006

Zezula, P. 2006. *Similarity Search : The Metric Space Approach*. Springer.

Zhang, D., Tsotras, V. J., Levialdi, S., Grinstein, G., Berry, D. A., Gouet-Brunet, V., … Pitoura, E. 2009. Indexing and Similarity Search. En *Encyclopedia of Database Systems* (pp. 1438–1442). Boston, MA: Springer US. https://doi.org/10.1007/978-0-387-39940-9_615

Zhang, J. 2001. Information Retrieval and Visualization. En *Visualization for Information Retrieval* (pp. 1–20). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-75148-9_1

Zhang, J. 2008. Information Retrieval Preliminaries. En J. Zhang (Ed.), *Visualization for Information Retrieval* (pp. 21–46). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-75148-9_2