UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

Facultad de ciencias físico matemáticas

Diseño y creación de un índice para clasificación de imágenes digitales por similitud

Tesis
Para obtener el grado de
Licenciado en Ciencias Físico Matemáticas

Presenta
Diego Antonio Calzadilla Hernández

Asesor:
Dra. Karina Mariela Figueroa Mora

Morelia, Michoacán
Junio 2016

Agradecimientos

Dedico primeramente esta tesis a mi familia, en particular a mi Madre que ha sido un auténtico ejemplo en mi vida, agradezco también a mis hermanos René y Marisol por su apoyo en esta última etapa de mi carrera. Lua, no fue fácil concluir esta tesis, sin embargo siempre estuviste ahí para motivarme, comprenderme y ayudarme en los momentos más complicados, muchas gracias. Quiero terminar agradeciendo a la Dra. Karina Figueroa por su paciencia y apoyo durante este trabajo además de ser una guía en mi carrera profesional, pero sobre todo por brindarme su valiosa amistad.

Índice general

Índice de figuras

Resumen

Conforme el paso del tiempo el uso de las ciencias computacionales se vuelve cada vez más indispensable en la vida cotidiana. En este trabajo se aborda el problema de búsqueda por similaridad en imágenes digitales, particularmete fue creado un índice que mejora el comportamiento de algunos algoritmos existentes. Primeramente se presentan los conceptos de espacios métricos y tipos de consulta en búsqueda por similaridad, ádemas se exponen los índices existentes para resolver problemas de búsqueda por similaridad. En la segunda parte se expone la propuesta del nuevo índice, se describe el concepto partición del espacio métrico en zonas y su combinación con el índice de permutantes, aprovechando la estructura de esta combinación se ejemplifica la parcialización de este índice. Para finalizar se presenta el resultado de los experimentos hechos con el nuevo índice en bases de datos sintéticas y reales.

Palabras clave: Espacio métrico, Similaridad, Permutantes, Consulta, Índice, Zonas.

Abstract

As time goes by computer science has become more essential in real life. This work addresses the problem of digital images similarity search, particularly a new index that improves some existing algorithms behaviour was created. Firstly the concepts of metric spaces and types of proximity queries are presented, indexes that solve proximity searching problems are also exposed. In the second part the proposed new index is exposed, the concept of metric space partition zones, its combination with permutants index and its division are also presented. Finally the results made by the new index using real and synthetic databases are presented.

Parte I

INTRODUCCIÓN

Capítulo 1

Introducción

Las bases de datos están presentes en muchos ámbitos de la vida cotidiana y realizar búsqueda en ellas es un problema fundamental de las ciencias computacionales, sus aplicaciones son tan diversas que díficilmente podríamos englobarlas todas en un sólo trabajo, sin embargo, en esta tesis abordaremos su aplicación en imágenes digitales.

Existen 2 tipos de búsqueda: búsqueda exacta y búsqueda por similaridad, la exacta consiste en, dada una consulta, encontrar los elementos idénticos a la consulta, mientras que en la búsqueda por similaridad o proximidad se deben encontrar los elementos más parecidos a la consulta dada, teniendo aplicaciones en tipos de datos que pueden ser díficilmente estructurados como sonido, videos, documentos de texto, cadenas de ADN e imágenes.

Una de las aplicaciones de las bases de datos digitales es en la conservación de las especies silvestres, pues permite tener evidencia de animales sin necesidad de técnicas intrusivas, como marcas o implantes de un chip. Sin embargo, este tipo de bases de datos son enormes y la técnica clásica (una persona clasifica visualmente cada fotografía) tiene muchos errores humanos, de manera que la clasificación es muy inexacta.

Un sistema automatizado para el reconocimiento de estos animales debería contribuir a tener censos más exactos así como en tomar políticas públicas al respecto. Un sistema completo de este tipo debe pasar por varias etapas: segmentación (aislar el animal fotografiado de su entorno), caracterización (obtener los puntos característicos del animal), indexación (en bases de datos enormes, es imposible comparar uno a uno para su clasificación) y finalmente un sistema que a partir del índice concluya con una clasificación para distintos fines como el censo.

En este trabajo, se presentará una parte de este enorme y complejo sistema, esto es la creación de un índice de búsqueda por similitud que permita la clasificación de animales basada en imágenes caracterizadas.

1.1. Motivación

Obtener las imágenes que caracterizan a estos animales no es una tarea sencilla, es necesario instalar cámaras fotográficas en puntos estratégicos de su hábitat y automatizar el proceso de fotografiar las zonas en que los sensores de movimiento son activados, esto, sin embargo resulta más simple y menos dañino que utilizar técnicas intrusivas, ya que estas implican realizar una expedición a su hábitat, capturar a los animales silvestres, insertarles un chip de identificación para finalmente liberarlos, poniendo en riesgo su integridad física, además de alterar su comportamiento e incluso modificar su hábitat.

En la etapa de indexación de este sistema se propone modelar el problema como un espacio métrico, el cual consta de 2 partes, una base de datos y una función de distancia que mide la similitud entre elementos. Para el tratamiento de este problema se diseñan algoritmos que permitan indexar la base de datos. Existen muchos algoritmos para la búsqueda por similitud en bases de datos métricas, en [CNBY01] se tiene una revisión completa de este tipo de índices. En particular un índice muy efectivo es el basado en permutaciones el cual es muy efectivo en dimensiones altas, sin embargo, hace falta mejorar su velocidad de respuesta en dimensiones bajas por lo que en esta tesis se proponen algunas modificaciones para que pueda ser un buen algoritmo de indexamiento en el problema que nos ocupa.

1.2. Estructura de la tesis

Esta tesis está organizada de la siguiente manera:

- **Capítulo 2. Conceptos básicos** En este capítulo hablaremos de definiciones necesarias para el lector. Además damos un estado del arte de los algoritmos de espacios métricos para la búsqueda por similaridad.
- **Capítulo 3. Propuesta** En este capítulo describimos la propuesta de este trabajo.
- **Capítulo 4. Experimentación** Nuestra propuesta fue evaluada de manera experimental y los resultados son presentados en este capítulo. Las bases de datos con las que se trabajó fueron sintéticas (vectores uniformemente distribuidos en el cubo unitario) y reales.
- **Capítulo 5. Conclusiones** Finalmente, en este capítulo se muestran las conclusiones y trabajo futuro de este trabajo.

Al final de la tesis se pueden consultar las referencias usadas en este trabajo.

Parte II

CONCEPTOS BÁSICOS

Capítulo 2

Conceptos Básicos

En este capítulo se presentarán los conceptos usados en este trabajo a través de los cuales el lector logrará entender la búsqueda por similiaridad en espacios métricos. Se explicarán algunos de los índices ya existentes y también los tipos de consulta en búsqueda por proximidad.

2.1. Espacios Métricos

Formalmente, un espacio métrico se define como un par $(\mathbb{X}, d)$, donde el conjunto $\mathbb{X}$ denota un universo de objetos del espacio métrico. La función

$$d : \mathbb{X} \times \mathbb{X} \to \mathbb{R}^+$$

define una medida de distancia entre elementos del espacio métrico. Un subconjunto finito $\mathbb{U}$ del conjunto universo $\mathbb{X}$, $\mathbb{U} \subset \mathbb{X}$, es el conjunto donde la búsqueda es realizada. Es importante destacar que entre más grande sean los valores toma d, más distintos o lejanos son los elementos en la base de datos. La función de distancia debe satisfacer las siguientes propiedades:

P1. $\forall\ x, y \in \mathbb{X},\ d(x, y) \geq 0$ **No negatividad**

P2. $\forall\ x, y \in \mathbb{X},\ d(x, y) = d(y, x)$ **Simetría**

P3. $\forall\ x, y \in \mathbb{X},\ x \neq y \Rightarrow\ d(x, y) > 0$ **Positividad**

P4. $\forall\ x, y \in \mathbb{X},\ d(x, z) \leq d(x, y) + d(y, x)$ **Desigualdad triangular**

En esta tesis se usará el conjunto finito de objetos $\mathbb{U} \subset \mathbb{X}$ de tamaño $n = |\mathbb{U}|$. A este conjunto de elementos $\mathbb{U}$ lo llamaremos base de datos. El término distancia será usado para referirnos a la función de distancia de un espacio métrico.

2.2. Funciones de distancia

Hay varias distancias que pueden ser usadas en la base de datos, la elección de la distancia depende de la búsqueda a realizar y debe ser determinada por un experto. Las propiedades mencionadas

anteriormente sólo aseguran una definición consistente de la distancia, aunque cabe destacar que **P4** permite descartar elementos sin ser comparados directamente contra la consulta. A continuación se mencionarán algunos tipos de distancias.

2.2.1. Distancia de edición

Documentos de texto o cadenas de caracteres son usualmente comparadas por medio de la distancia de edición, también llamanda distancia Levenshtein. La distancia entre dos cadenas $x = x_1 \cdots x_n$ y $y = y_1 \cdots y_m$ es el mínimo número de operaciones necesarias para transformar la cadena x en la cadena y. Las operaciones posibles son:

- Inserción: Insertar el caracter c dentro de la cadena x en la posición i.
  $ins(x, i, c) = x_1 \cdots x_{i-1} c x_{i+1} \cdots x_n$
- Destrucción: Borrar el caracter c de la posición i dentro de la cadena x.
  $des(x, i, c) = x_1 \cdots x_{i-1} x_{i+1} \cdots x_n$
- Substitución: Substituir el caracter en la posición dentro de la cadena x con el nuevo caracter c.
  $sub(x, i, c) = x_1 \cdots x_{i-1} c x_{i+1} \cdots x_n$

2.2.2. Distancias en espacios métricos infinitos

En general, la distancia d entre cualesquiera dos puntos en un espacio métrico puede ser calculada por la ecuación dada por Minkowski:

$$L_p[(x_1, x_2, ..., x_n), (y_1, y_2, ..., y_n)] = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}$$

Hay 3 casos especiales de la distancia de Minkowski:

- $p = 1$, esta función de distancia se conoce generalmente como **distancia Manhattan** y expresa la distancia entre 2 puntos si se sigue un camino en forma de rejilla.
- $p = 2$, la distancia de Minkowski se reduce a la muy conocida **distancia Euclidiana**.
- $p \to \infty$, para este caso $\lim_{p\to\infty} \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}$ define la **distancia máxima** entre 2 puntos.

La figura 2.1 muestra la representación geométrica de los casos explicados anteriormente (las curvas están centradas en el mismo punto).
Hay varias aplicaciones de la distancia de Minkowski, entre ellas, podemos destacar el cálculo de distancias entre histogramas de colores o entre vectores característicos extraídos de imágenes. En este trabajo usaremos varias de estas distancias. Es importante mencionar que la distancia de Minkowski es calculada en tiempo lineal. Los algoritmos de programación dinámica que calculan la distancia de edición en cadenas de caracteres tienen complejidad $O(mn)$, donde m es el largo de una cadena y n el largo de la otra.

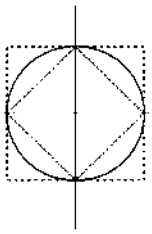

Figura 2.1: Conjunto de puntos a la misma distancia el centro para las distancias p_1(- - -), p_2(—), p_∞(...).

2.3. Consultas por proximidad

Existen básicamente 3 tipos de consultas en espacios métricos:

- **Consulta de rango**. Su objetivo es recuperar todos los elementos que están dentro de una distancia r de q, dónde $q \in \mathbb{X}$ denota la consulta. Es decir, $\{u \in \mathbb{U} \mid d(q,u) \leq r\}$
- **Consulta del vecino más cercano**. Este tipo de consulta recupera el elemento más cercano a q en $\mathbb{U}$, esto es, $\{u \in \mathbb{U} \mid \forall v \in \mathbb{U},\ d(q,u) \leq d(q,v)\}$
- **Consulta de los k vecinos más cercanos**. En esta consulta se obtienen los K elementos más cercanos a q en $\mathbb{U}$, es decir, recuperamos un conjunto $A \subset \mathbb{U}$ tal que $|A|$ =k y $\forall u \in A,\ v \in \mathbb{U} \setminus A$ se cumple que $d(q,u) \leq d(q,v)$.

A continuación, en la figura 2.2, se muestra gráficamente este tipo de consultas, en el lado izquierdo la consulta por rango (un espacio métrico en el plano, considerando la distancia Euclidiana) y en el derecho la consulta de los dos vecinos más cercanos.

El tiempo que se ocupa para evaluar una consulta puede ser expresado de la forma

$$T = \text{cálculos de distancia} \times \text{complejidad de } d() + \text{tiempo extra } CPU + \text{tiempo } E/S$$

siendo el objetivo principal de todo índice de búsqueda por proximidad minimizar T. Sucede en muchas aplicaciones que evaluar $d()$ es tan costoso, que las demás variables en T pueden ser ignoradas y por tanto el **número** de distancias evaluadas es la medida de complejidad de los algoritmos presentados en este trabajo.

Es claro que cualquier tipo de consulta puede ser respondida examinando la base de datos $\mathbb{U}$ en su totalidad, de hecho, si no fuera posible preprocesar la información (i.e. construir un índice de la base de datos), entonces la examinación exhaustiva uno a uno de los elementos de la base de datos es la única forma de realizar la búsqueda.

Un algoritmo de indexamiento es un procedimiento para construir una estructura de datos (índice) diseñada para posteriormente ahorrar cálculos de distancia cuando sean realizadas las consultas. La construcción de esta estructura puede ser costosa, pero es compensada mediante el ahorro de evaluaciones de distancia durante muchas consultas a la base de datos. Es por ello

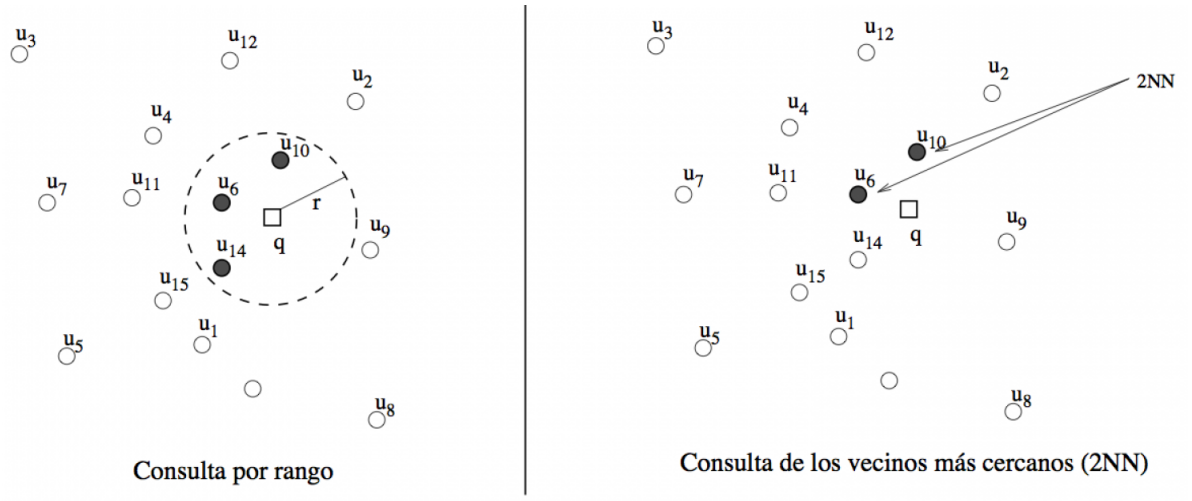

Figura 2.2: Tipos de consulta por similaridad en espacios métricos.

que el objetivo consiste en diseñar algoritmos de indexamiento que reduzcan las evaluaciones de distancia. Todas estas estructuras están fundamentadas en que la **desigualdad triangular** nos permite descartar elementos.

2.4. Problema de la dimensión intrínseca alta

En los espacios métricos usados suele existir una gran diferencia entre la dimensión representada y la dimensión intrínseca. La dimensión intrínseca representa el número real de dimensiones en las cuales los puntos pueden ser inmersos mientras mantienen (con poca distorsión) la distancia entre ellos. Por ejemplo, un plano inmerso en un espacio de 50 dimensiones tiene dimensión intrínseca 2, mientras que su dimensión representada es de 50.

Basado en esto varios autores han propuesto el uso de técnicas de indexación basadas en distancias (técnicas para espacios metricos), en un intento por evitar la maldición de la dimensionalidad. Estas técnicas usan sólo la distancia entre puntos y evitan cualquier referencia a coordenadas. Al usar sólo la información de la distancia es importante conocer el efecto que sufren los algoritmos cuando se incrementa la dimension intrínseca. La característica típica en espacios de dimension intrínseca alta es que la distribución de las distancias entre elementos tiene un histograma concentrado, con una media mayor a medida que la dimension crece. En un caso extremo tenemos un espacio donde $d(x, x) = 0$ y $\forall x \neq y, d(x, y) = 1$; allí es imposible evitar cálculos de distancia en una consulta, pues la distancia sólo nos dice si el elemento comparado es o no la respuesta.

En [CNBY01] proponen una manera de definir la dimensión intrínseca de un espacio métrico, como $\rho = \mu^2/2\sigma^2$ donde μ y σ^2 son la media y la varianza, respectivamente, de su histograma de distancias. Para ilustrar el problema con la dimension intrínseca alta, véase la figura 2.3, donde se

muestran dos histogramas de distancias de dos bases de datos distintas, con respecto a un elemento distinguido $p \in \mathbb{U}$. El histograma del lado derecho (dimension alta) está mucho más concentrado que el del lado izquierdo (dimensión baja). Piense que una consulta de rango q con radio r se aplica a estos dos espacios; y que la distancia entre p y q, representada por $d(p, q)$, se ubica al centro del histograma, que por cierto es su posición con mayor probabilidad. En ambos espacios las zonas sombreadas corresponden a los elementos que podrían ser descartados por la desigualdad triangular, sin compararse contra la consulta usando un índice (como se describe en la sección 2.3). Note que el número de elementos que podrían ser descartados es mayor en dimensión baja que en dimensión alta.

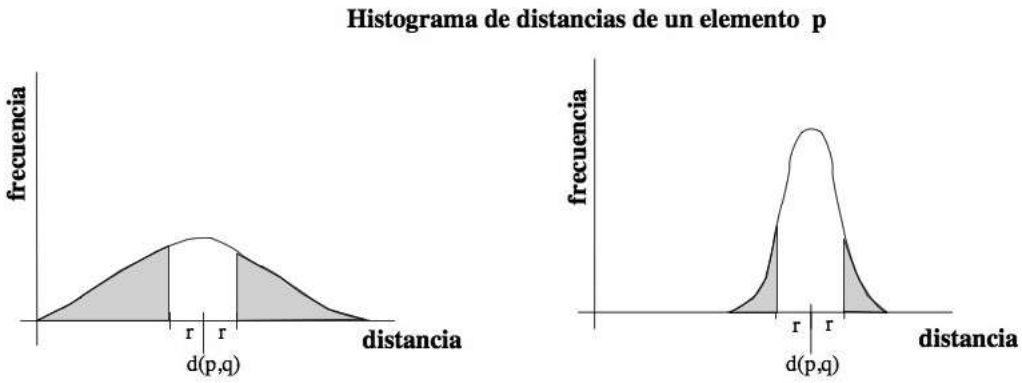

Figura 2.3: Histogramas de distancias de una consulta $(q, r)_d$ hacia un elemento p en dimensión baja y alta (izquierda y derecha, respectivamente). Las zonas sombreadas corresponden a los elementos que podrían ser descartados sin compararse contra la consulta usando un índice y la desigualdad triangular.

2.5. Estado del Arte

Los algoritmos que resuelven búsqueda por proximidad tienen generalmente dos fases: la de preprocesamiento y la de consulta. Durante la etapa de preprocesamiento hay comparaciones que son inevitables pues permiten la construcción del índice (**comparaciones internas**). El índice se crea una vez y se almacena, por tanto, el costo de su construcción no se considera al momento de resolver consultas. En la fase de consulta se recorre el índice para conocer la lista de candidatos que deberán ser comparados directamente contra la consulta (**comparaciones externas**) y los elementos que pueden ser descartados de manera segura. En la lista de candidatos puede haber elementos que no son relevantes para la consulta, pero que no pudieron ser distinguidos por el índice. El caso ideal es que todos los elementos en la lista de candidatos sean relevantes para la consulta.

2.5.1. Índices Basados en Pivotes

Construir un índice basado en pivotes consiste en elegir un conjunto $\mathbb{P} = \{p_1, p_2, ..., p_k\} \subset \mathbb{U}$ de tamaño $k = |\mathbb{P}|$, al que llamaremos conjunto de **pivotes**. Para cada elemento de la base de datos se precalculan las distancias hacia los elementos del conjunto $\mathbb{P}$. Este conjunto de distancias $\{d(p_1, u), d(p_2, u), ..., d(p_k, u)\}$, para todo $u \in \mathbb{U}$, es el que conformará el índice.

Dada una consulta q, se calcula $d(p_i, q)$ para todo $p_i \in \mathbb{P}$ (comparaciones internas). Con esto puede acotarse la distancia entre q y cualquier $u \in \mathbb{U}$ tal como se demuestra en el siguiente lema.

Lema 1 *Dados tres objetos $q \in \mathbb{X}, u \in \mathbb{U}, p \in \mathbb{P}$, entonces $|d(q,p) - d(p,u)| \leq d(q,u) \leq d(q,p) + d(p,u)$.*
Demostración: El límite superior se obtiene directamente de la desigualdad triangular,

$$d(q,u) \leq d(q,p) + d(p,u) \tag{2.1}$$

En el caso del límite inferior, de acuerdo a la desigualdad triangular, se tiene que:

$$d(p,u) \leq d(p,q) + d(q,u),$$
$$d(p,q) \leq d(p,u) + d(u,q)$$

Estas desigualdades implican:

$$d(p,u) - d(p,q) \leq d(q,u),$$
$$d(p,q) - d(p,u) \leq d(u,q)$$

Finalmente, combinando estas desigualdades y usando la simetría, obtenemos que:

$$|d(p,u) - d(p,q)| \leq d(q,u). \ \square \tag{2.2}$$

Los algoritmos basados en pivotes utilizan el **Lema 1** de la siguiente forma: en una consulta por rango, es posible descartar todos los elementos de la base de datos $u \in \mathbb{U}$ tales que $\exists p \in \mathbb{P}\ |d(q,p) - d(p,u)| > r$, pues es claro que si $|d(q,p) - d(p,u)| > r$, entonces $d(q,u) > r$. Los elementos no descartados de esta manera, serán comparados directamente contra la consulta(consultas externas).
En la figura 2.4 se muestra un ejemplo del uso de este tipo de algoritmos, en este ejemplo, p es un pivote. El anillo formado por los circulos centrados en p, a distancias $d(p,q)+r$ y $d(p,q)-r$, contiene a los elementos que no podrían ser descartados por p, estos son: $u_2, u_3, u_4, u_5, u_6, u_7$. El resto de la base de datos sí es descartada, por ejemplo, en el caso de u_1 , se cumple que $|d(p,q) - d(p,u_1)| > r$, lo mismo sucede con el resto. En [CNBY01] se ha demostrado que los algoritmos basados en pivotes son excelentes para espacios de dimensión baja (< 20). El algoritmo 1 muestra el proceso realizado durante el indexación, y el algoritmo 2 muestra el proceso de una consulta en esta familia de índices.

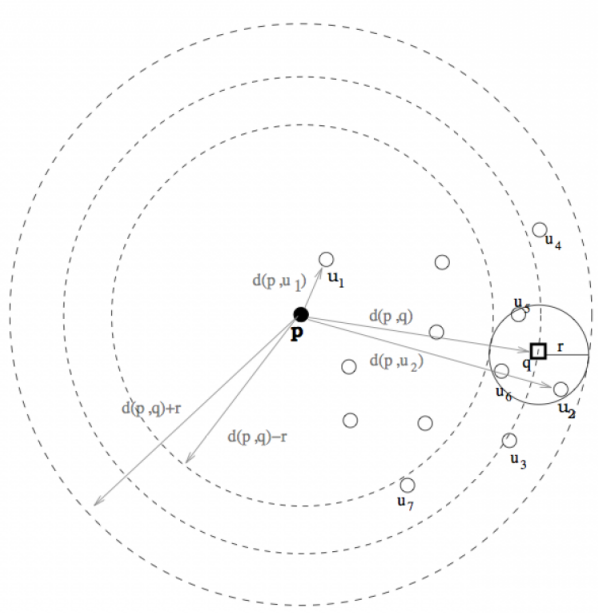

Figura 2.4: Ejemplo del algoritmo basado en pivotes para una consulta de rango.

Algoritmo 1 Algoritmo basado en pivotes, indexamiento

1: M matriz de distancias de tamaño $n \times k$
2: **for** *all* $u \in \mathbb{U}$ **do**
3: **for** *all* $p \in \mathbb{P}$ **do**
4: $M[u, p] \leftarrow d(u, p)$
5: **end for**
6: **end for**

Algoritmo 2 Algoritmo basado en pivotes, consulta

```
1: M matriz de distancias de tamaño n × k
2: 𝔽 ← 𝕌
3: for all p ∈ ℙ do
4:     d ← d(p, q)                                  ▷ Comparaciones internas
5:     for all u ∈ 𝔽 do
6:         𝔽 ← 𝔽 − u
7:         if |M[u, p] − d| > r then
8:             𝔽 ← 𝔽 − u                            ▷ Objetos filtrados
9:         end if
10:    end for
11: end for
```

El desempeño de los algoritmos en espacios métricos (consultas por rango y consultas de k vecinos más cercanos) decae a medida que la dimensión del espacio crece [BBK01], es por ello que en el presente trabajo se experimenta con espacios donde es posible variar la dimensión. Además en [BY97] y [BYN98] muestran que existe un número óptimo de pivotes que depende no sólo de la dimensión del espacio sino también del número de vecinos más cercanos (K) pues a medida que K aumenta, las comparaciones de distancia (externas) también aumentan.

2.6. Ejemplos de algoritmos basados en pivotes

En esta sección presentaremos los ejemplos más representativos de los índices basados en pivotes.

2.6.1. Índice BKT

La primera solución al problema de búsqueda por similaridad fue presentada en [BK73]. En esta propuesta los autores proponen un árbol (llamado árbol Burkhard-Keller o **BKT**) adaptable a distancias discretas y se define como sigue:
Un elemento $p \in \mathbb{U}$ arbitrario es elegido como raíz del árbol. Para cada distancia $i > 0$, definimos $\{u \in \mathbb{U} \mid \ d(u,p) = i\}$ como el conjunto de todos los elementos a distancia i de la raíz. Luego, para cualquier conjunto no vacío $\mathbb{U}_i$, agregamos un hijo de p (etiquetado i), de donde construimos recursivamente el **BKT** para $\mathbb{U}_i$. Este proceso puede ser repetido hasta que sólo queda un elemento por procesar o hasta que no hay mas de b elementos. Todos los elementos seleccionados como raíces de subarboles son los pivotes usados en este índice.

Dado un radio de consulta r y una consulta q, comenzamos en la raíz y accesamos a todos sus hijos i que cumplen la desigualdad $d(p,q) - r \leq i \leq d(p,q) + r$, y seguimos con este procedimiento recursivamente. Si llegamos a una hoja comparamos secuencialmente sus elementos.

La figura 2.5 muestra un ejemplo donde el elemento u_{11} ha sido seleccionado como la raíz. En este ejemplo sólo se ha construido el primer nivel del **BKT** por simplicidad. También es mostrada

la consulta q y las ramas del árbol que deben ser recorridas.

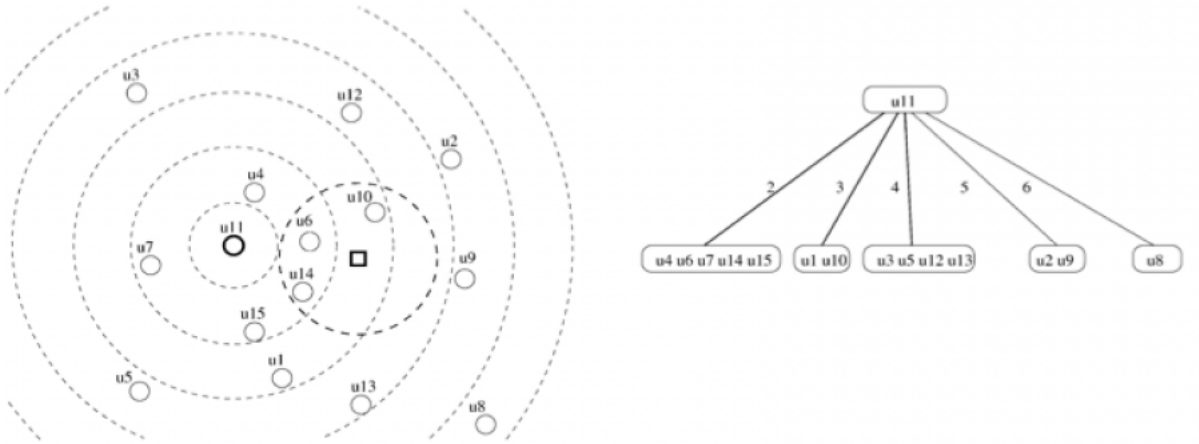

Figura 2.5: En la izquierda, la división del espacio obtenida al tomar a u_{11} como pivote. En la derecha, el primer nivel de un **BKT** con u_{11} como raíz. También se muestra la consulta q y las ramas que tienen que ser recorridas.

2.6.2. Índice FQT(Fixed Query Tree)

Este índice se trata básicamente de un **BKT** donde todos los pivotes alojados en los nodos del mismo nivel son iguales. Los elementos reales están todos guardados en las hojas. La ventaja de esta construcción es que algunas comparaciones entre la consulta y los nodos son guardadas durante el recorrido del árbol. Si visitamos muchos nodos del mismo nivel, necesitamos realizar sólo una comparación porque todos los pivotes en ese nivel son los mismos. Se ha demostrado experimentalmente en [BYCMW94] que un **FQT** realiza menos cálculos de distancia en el momento de la consulta. Demuestran además que un **FQT** construido sobre n elementos toma $O(log\ n)$ de altura promedio y que requiere $O(n\ log\ n)$ evaluaciones de distancia.

2.6.3. Índice FHQT(Fixed Height Query Tree)

FQT es un índice diseñado para espacios discretos y su proceso de construcción es similar al **BKT**. Tomamos un conjunto de k pivotes, inicialmente, con el pivote p_1 como raíz, y para cada distancia $i > 0$ determinamos el subconjunto de objetos que se encuentran a distancia i de p_1. Para cada subconjunto no vacío se genera una rama con etiqueta i. En cada hijo, con los elementos en éste se crea un **FQT**, tomando como raíz el siguiente pivote p_2. Nótese que todos los subárboles de un mismo nivel usan el mismo pivote como raíz, la altura del árbol es de tamaño k. La información de los elementos de la base de datos está almacenada en la hojas.
Para una consulta q y un radio r, se calcula la distancia de q hacia la raíz p_1, y se descartan aquellas ramas cuya etiqueta $i \notin [d(q,p_1)-r, d(q,p_1)+r]$. Se desciende por las ramas que no fueron

descartadas aplicando el mismo procedimiento.
En el **FHQT** todas las hojas tienen la misma profundida h. Cada pivote almacena todas las distancias hacia los elementos de la base de datos. El espacio ocupado se encuentra entre $O(n)$ y $O(nh)$. En la práctica el h óptimo es $O(log\ n)$, el cual rara vez puede ser alcanzado por limitaciones de espacio en bases de datos grandes. En la figura 2.6 mostramos un ejemplo del **FQT** y del **FHQT**.

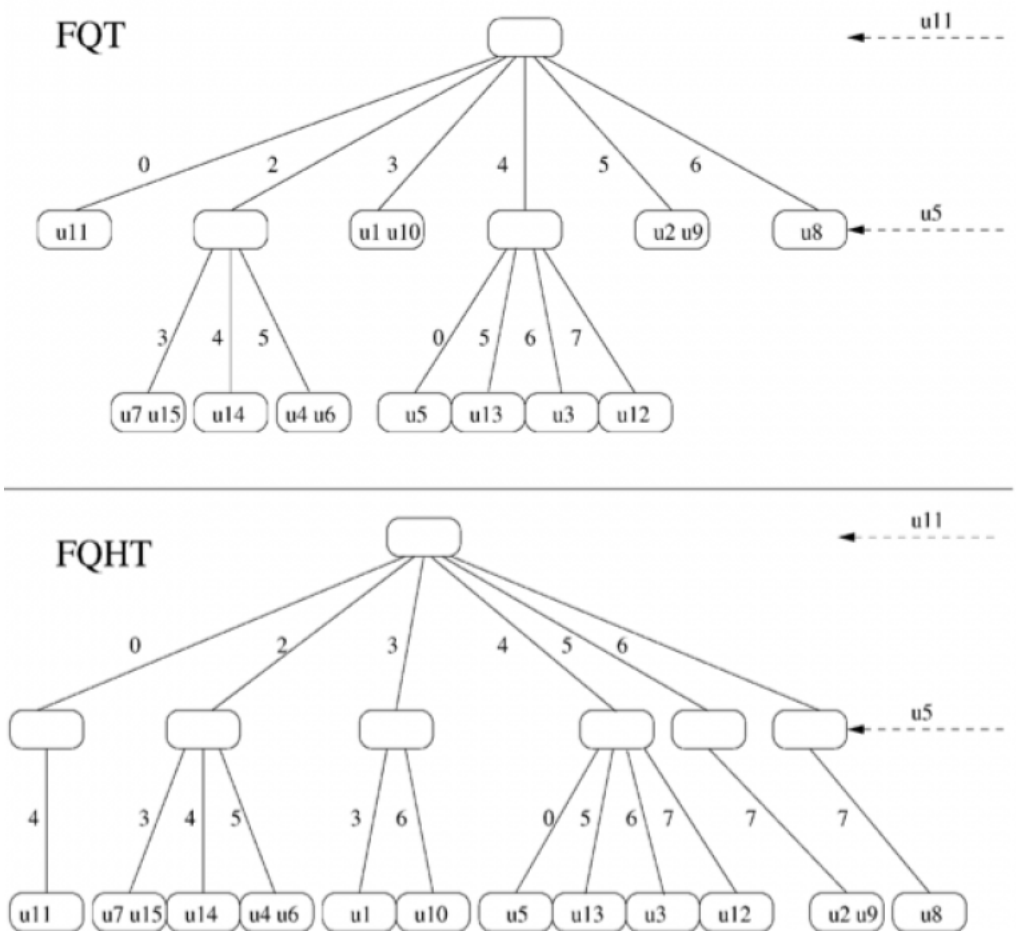

Figura 2.6: Ejemplo de **FQT** y **FHQT** para el conjunto de datos de la figura 2.5 tomando a u_{11} como primer pivote.

2.6.4. Índice FQA(Fixed Query Array)

En este índice, para cada elemento de la base de datos, se guarda una lista de distancias a los $\mathbb{P}$ pivotes. En el FQA, esta lista es considerada como una secuencia de $\mathbb{P}$ enteros. La estructura simplemente guarda los elementos de la base de datos ordenados lexicográficamente (obsérvese el cuadro 2.1, tabla (a)) por esta secuencia de distancias, esto es, los elementos son primero ordenados por su distancia al primer pivote (columna 1 de la tabla (a), cuadro 2.1), aquellos que tienen la misma distancia al primer pivote son ordenados por su distancia al segundo pivote (columna 2 de la tabla (a), cuadro 2.1), y así sucesivamente. A medida que más pivotes son agregados el arreglo está más ordenado.

El resultado del FQA tiene una fuerte relación al FQHT de altura $\mathbb{P}$. Si las hojas del FQHT son recorridas en pre-orden, la salida es precisamente el orden impuesto por el FQA, mas aún el algoritmo de búsqueda del FQHT es inherente al FQA. Cada nodo del FQHT corresponde a un rango de celdas en el FQA. Si un nodo desciende desde otro en el árbol, este rango es un subrango de otros en el arreglo. De aquí que, para cada vez, que el algoritmo del árbol se mueve desde un nodo a uno de sus hijos en el árbol, simulamos el movimiento en el arreglo, por búsqueda binaria al nuevo rango dentro del actual. Esta búsqueda binaria no desarrolla cálculos de distancia extra, sólo compara secuencias de enteros.

La complejidad de la construcción es $O(\mathbb{P}n)$ cálculos de distancia más el tiempo para ordenar el arreglo lexicográficamente. Esto es $O(k\ log\ n)$ ([CMN01]).

Para hacer más claros estos conceptos, mostreremos explícitamente el algoritmo de búsqueda. Dada una consulta q, un rango r y $\mathbb{P}$ pivotes $p_1 \cdots p_{\mathbb{P}}$, medimos $d_1(q, p_1), d_2(q, p_2), ..., d_P(q, p_P)$. Ahora, para cada i en el rango $d_i - r$ a $d_i + r$, hacemos una búsqueda binaria en arreglo del rango donde la primer coordenada es i. Una vez que el rango es calculado, para cada i continuamos recursivamente la búsqueda en el subarreglo encontrado, desde el pivote p_2. Esto es equivalente a visitar en el $i - ésimo$ nivel del subárbol en el FHQT. La búsqueda termina cuando usamos los $\mathbb{P}$ pivotes, y los puntos del subarreglo resultante son revisados secuencialmente. El procedimiento recursivo termina antes cuando el subarreglo resultante esta vacío.

Ejemplo

Consideremos el ejemplo de la figura 2.7. Cada rama descendiente de la raíz representa una distancia al pivote p_1. Ramas descendientes de los nodos del segundo nivel se refieren a las distancias a p_2, y así para las siguientes. Dada una consulta $(q, r)_d$, el algoritmo de búsqueda entra, en el nivel i del árbol, a aquellas ramas dentro del intervalo de interés $d(q, p_i) \pm r$. Tomemos $r = 2$ y $d(q, p_i) = \{3, 4, 5, 4\}$. Ramas etiquetadas $[1, 2, 3, 4]$ en el primer nivel serán examinadas y, recursivamente, todas las ramas debajo de ellas serán recorridas de acuerdo al intervalo apropiado para los respectivos niveles. Cuando una rama esta fuera del intervalo de interés, $[7, 8, 9]$ en nuestro ejemplo, no es tomada en cuenta. Al final, los elementos $\{4, 6, 7, 8\}$ quederán en la lista de candidatos, y

serán comparados contra la consulta para saber si formarán parte del conjunto respuesta.

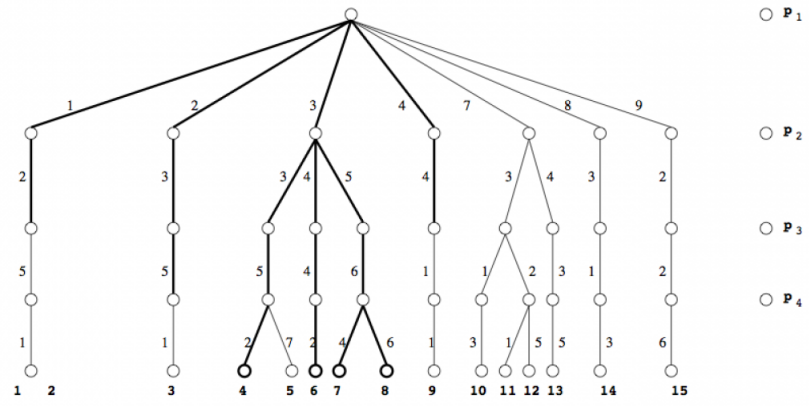

Figura 2.7: Representación de la búsqueda en FHQT.

El FQA equivalente guarda los elementos en el orden de ïzquierda a derecha"mostrado en la figura 2.7. El cuadro 2.1 muestra el proceso de búsqueda.

Tenemos 4 pivotes, y cada fila en las 4 tablas (a), (b), (c) y (d) representa una rama del árbol, las tablas del cuadro 2.1 representan las distancias de los puntos de la base de datos a los pivotes apropiados. Para una consulta q calculamos el vector $(d(q,p_1),...,d(q,p_4))$, en este caso $(3,4,5,4)$. El radio de búsqueda es 2. Tenemos que buscar los intervalos $(\{1,5\},\{2,6\},\{3,7\},\{2,6\})$ respectivamente, el proceso se encuentra representado en el cuadro 2.1. Usando búsqueda binaria encontramos los intervalos en la primera columna(filas en **negrita**). En cada una de las 4 tablas, mostramos en **negritas** los candidatos después de cada caso. La tabla (a) es quivalente al primer nivel en el árbol, mientras que la tabla (b) es equivalente al segundo nivel en el árbol y así para las demás. Podemos fácilmente verificar que aplicar búsqueda binaria en cada columna es equivalente a acotar la búsqueda en los niveles apropiados del árbol.

Es muy importante notar que el orden lexicográfico nos permite usar búsqueda binaria en las columnas subsecuentes. Considerando por ejemplo aquellas columnas que empiezan con un 3: los elementos de la segunda columna son también ordenados en orden creciente y así sucesivamente, esto puede notarse en el siguiente cuadro.

(a) $\{1,5\}$	(b) $\{2,6\}$	(c) $\{3,7\}$	(d) $\{2,6\}$
1 1 4 4	**1** 1 4 4	**1** 1 **4** 4	**1** 1 **4 4**
1 2 5 1	**1 2** 5 1	**1 2 5** 1	**1 2 5** 1
1 2 5 1	**1 2** 5 1	**1 2 5** 1	**1 2 5** 1
2 3 5 1	**2 3** 5 1	**2 3 5** 1	**2 3 5** 1
3 3 5 2	**3 3** 5 2	**3 3 5** 2	**3 3 5 2**
3 3 5 7	**3 3** 5 2	**3 3 5** 7	**3 3 5** 7
3 4 4 2	**3 4** 4 2	**3 4 4** 2	**3 4 4 2**
3 5 6 4	**3 5** 6 4	**3 5 6** 4	**3 5 6 4**
3 5 6 6	**3 5** 6 6	**3 5 6** 6	**3 5 6 6**
4 4 1 1	**4 4** 1 1	**4 4** 1 1	**4 4** 1 1
7 3 1 3	7 3 1 3	7 3 1 3	7 3 1 3
7 3 2 1	7 3 2 1	7 3 2 1	7 3 2 1
7 3 2 5	7 3 2 5	7 3 2 5	7 3 2 5
7 4 3 5	7 4 3 5	7 4 3 5	7 4 3 5
8 3 1 3	8 3 1 3	8 3 1 3	8 3 1 3
9 2 2 6	9 2 2 6	9 2 2 6	9 2 2 6

Cuadro 2.1: Representación de la búsqueda en FQA.

Es claro que la lista de candidatos usando cualquier representación no cambia. En la búsqueda basada en el FQA tenemos que pagar $O(log\ n)$ (el costo de la búsqueda binaria) para simular la visita a cada ramificación. Si visitamos m nodos en el árbol, usamos $O(m\ log\ n)$ tiempo en el arreglo.

2.7. Índice basado en particiones compactas

La idea central de este índice consiste en dividir el espacio en zonas tan compactas como sea posible. Por esta razón se selecciona un conjunto de centros $c_1, c_2, ..., c_k$ y se divide el espacio, es decir, se distribuyen los demás elementos entre las i zonas pertenecientes a cada c_k. El índice está compuesto por los centros, los elementos que pertenecen a cada zona y, en algunos casos, información adicional sobre las distancias.

Existen dos criterios posibles para descartar zonas durante una búsqueda:

- **Criterio del hiperplano:** Al momento de realizar una consulta de rango $(q, r)_d$, se evalúan las distancias entre q y los k centros, escogiéndose el centro más cercano a q, el cual se denominará c.

  Si la bola de consulta $(q, r)_d$ intersecta la zona de algún centro distinto a c, entonces se procede a revisar exhaustivamente esa zona por si hay objetos que caigan dentro de la bola de consulta. La figura 2.8 ilustra esta situación. Sea $x \in \mathbb{X}$ un objeto perteneciente a la zona cuyo centro es c_i y que se encuentra a una distancia menor o igual que r de la consulta q. Por la desigualdad triangular se tiene que:

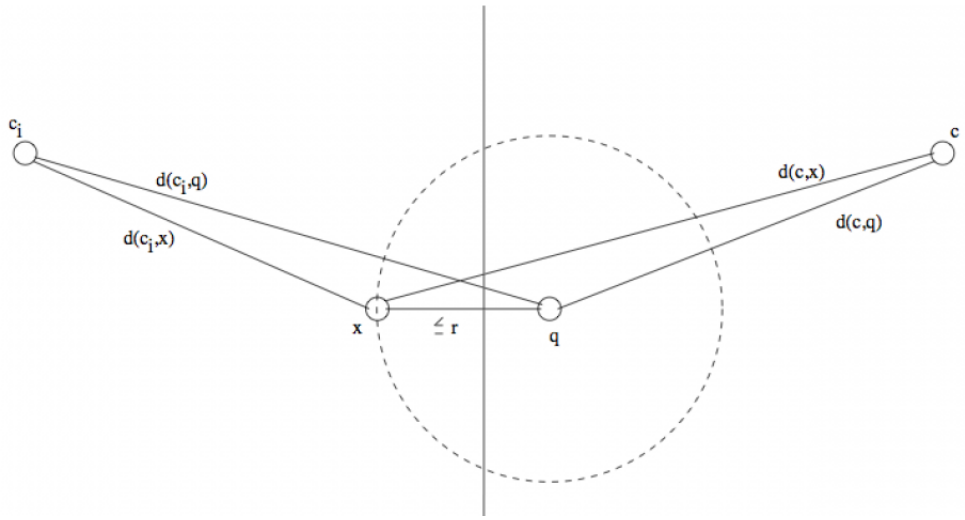

Figura 2.8: Representación del criterio de exclusión del hiperplano.

$$d(c, x) \leq d(c, q) + d(q, x) \leq d(c, q) + r \tag{2.3}$$

Por otro lado, se tiene que:

$$d(c_i, q) \leq d(c_i, x) + d(x, q) \leq d(c_i, x) + r \Rightarrow d(c_i, q) - r \leq d(c_i, x) \tag{2.4}$$

Como x pertenece a la zona de c_i, se cumple que $d(c_i, x) \leq d(c, x)$. De esta condición más

(2.3) y (2.4) se obtiene:

$$d(c_i, q) - r \leq d(c_i, x) \leq d(c, x) \leq d(c, q) + r \Rightarrow d(c_i, q) - r \leq d(c, q) + r \tag{2.5}$$

Dada la ecuación 2.5, se pueden descartar zonas cuyo centro c_i satisfaga la condición $d(c_i, q) > d(c, q) + 2r$, puesto que en ese caso dicha zona no puede tener intersección con la bola de consulta.

- **Criterio del radio de cobertura:** El radio de cobertura es la máxima distancia entre un centro c y algún objeto perteneciente a su zona, y se denotará como $cr(c)$.

  La figura 2.9 muestra cuál es el criterio de exclusión utilizando el radio de cobertura. Dada una consulta por rango $(q, r)_d$ se tiene que si $d(q, c) - r > cr(c)$, entonces la bola de consulta no puede tener intersección con la zona de centro c, y por lo tanto se pueden excluir de la respuesta todos los objetos pertenecientes a la zona de centro c sin necesidad de revisarlos directamente. En caso contrario, es necesario revisar exhaustivamente dicha zona por si hay objetos que se encuentran dentro de la bola de consulta. En la figura, la bola de consulta (q_1, r) no tiene intersección con la zona de centro c, por lo que no es necesario revisar los objetos que pertenecen a dicha zona. Por el contrario, en la consultas q_2 y q_3 sí hay intersección, lo que hace necesario revisar exhaustivamente dicha zona en busca de objetos que estén a distancia menor que r de la consulta.

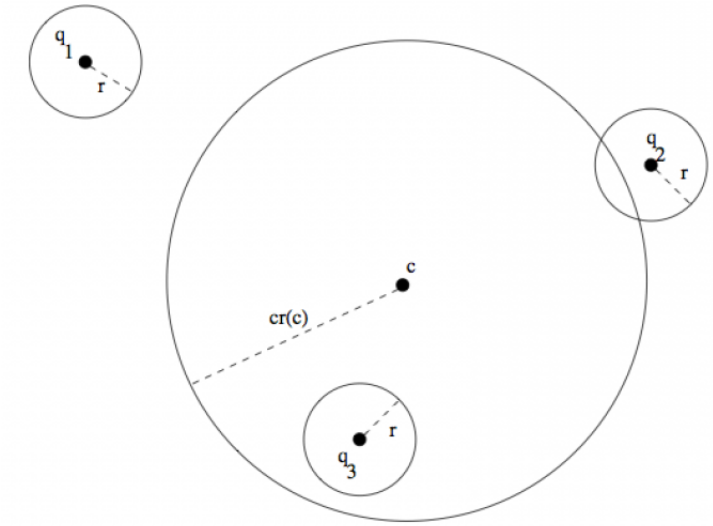

Figura 2.9: Condición de exclusión con criterio de radio de cobertura.

En [CNBY01] se reporta que esta técnica tiene un buen funcionamiento para espacios de dimensión menor a 64, aunque su construcción es cuadrática por tanto es un algoritmo que no se usa comunmente.

2.8. Índice basado en permutaciones

Los índices descritos anteriormente usan las cotas de distancia entre elementos para procesar la base de datos. El índice basado en permutaciones es una propuesta probabilística para resolver el problema de la búsqueda por similaridad en espacios métricos. Si bien para generar este índice también usamos las distancias, la diferencia radica en que almacenamos otro tipo de información sobre esas distancias.

La ventaja de esta técnica de permutaciones es que es posible obtener rápidamente un alto porcentaje de la respuesta correcta, lo que da como resultado un algoritmo probabilístico.

La idea general de este índice consiste en seleccionar un subconjunto $\mathbb{P} = \{p_1, p_2, ..., p_{|\mathbb{P}|}\} \subset \mathbb{U}$, llamados permutantes, en principio de manera aleatoria. Para cada $u \in \mathbb{U}$ formamos su permutación Π_u (vease la figura 2.10). Los empates se deciden en forma arbitraria pero consistente.

En la figura 2.10 se muestra un espacio métrico en $\mathbb{R}^2$. El conjunto de permutantes $\mathbb{P}$ son los círculos rellenos (u_1 a u_4, llamados p_1 a p_4 respectivamente). En el ejemplo se seleccionaron dos elementos para ilustrar la idea de las permutaciones, u_7 y u_{12}. El elemento u_7 tiene más cerca a p_4, después a p_3, luego p_1, y finalmente p_2. El elemento u_{12} tiene más cerca a p_4, después a p_2, p_3, p_1. Note que las permutaciones de u_7 y u_{12} son distintas.

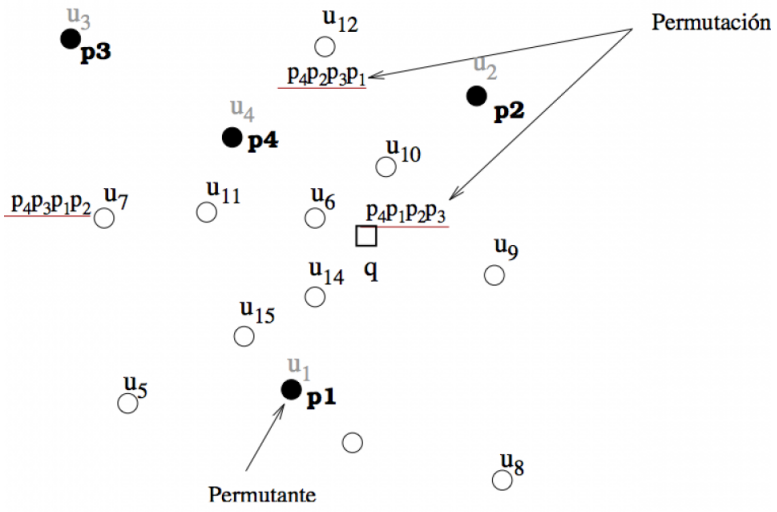

Figura 2.10: Los elementos u_7 y u_{12} perciben de manera diferente al conjunto de permutantes $\mathbb{P}$.

Intuitivamente, si dos elementos u y v son cercanos entre sí, sus permutaciones serán similares. En particular, si dos elementos u, v son tales que $d(u, v) = 0$, las permutaciones Π_u y Π_v seran iguales. Basándose en esta observación se quiere revisar primero aquellas permutaciones con mayor semejanza de la consulta, esperando que las permutaciones sean un buen predictor de cercanía entre elementos. Por lo tanto, el objetivo de esta técnica es identificar a los elementos más cercanos entre sí usando la semejanza entre las permutaciones.

2.8.1. Comparación entre permutaciones

Al momento de la consulta, calculamos Π_q para la consulta q. Luego, para saber en qué orden revisar los elementos, ordenamos los Π_u, $u \in \mathbb{U}$ de mayor a menor similaridad con respecto a Π_q. De esta forma, esperamos que los elementos con permutaciones muy similares a Π_q estén espacialmente cerca de q, y sean los más cercanos a la consulta. Como medida de similaridad entre permutaciones usamos Spearman Rho ([FKS03]), la cual denotaremos por $S_\rho(\Pi_q, \Pi_u)$. S_ρ es la suma de los cuadrados de las diferencias en las posiciones relativas de cada elemento en las dos permutaciones. Para cada $p_i \in \mathbb{P}$ calculamos su posición en Π_u y Π_q, esto es, $\Pi_u^{-1}(p_i)$ y $\Pi_q^{-1}(p_i)$, respectivamente. Formalmente,

$$S_\rho(\Pi_u, \Pi_q) = \sum_{i=1}^{k} |\Pi_u^{-1}(p_i) - \Pi_q^{-1}(p_i)|^2 \tag{2.6}$$

A continuación daremos un ejemplo de como se calcula $S_\rho(\Pi_u, \Pi_q)$. Sea $\Pi_q = p_1, p_2, p_3, p_4, p_5, p_6$ la permutación de la consulta, y sea $\Pi_u = p_3, p_6, p_2, p_1, p_5, p_4$ la permutación de un elemento u. El elemento p_3 dentro de la permutación Π_u está desfasado dos posiciones con respecto a su posición en Π_q. Las diferencias entre las permutaciones para cada elemento son: $|1-3|$, $|2-6|$, $|3-2|$, $|4-1|$, $|5-5|$, $|6-4|$, y la suma de los cudrados de todas las diferencias es $S_\rho(\Pi_u, \Pi_q) = 34$.

En la figura 2.11 se muestra un ejemplo de las dos fases del algoritmo basado en permutaciones; la fase de preprocesamiento (lado izquierdo) y la de consulta (lado derecho). En la fase de preprocesamiento se calculan las permutaciones para todos los elementos de la base de datos. En la fase de consulta, primero se calcula la permutación para q. Los elementos son ordenados respecto a la similaridad entre permutaciones (usando S_ρ) y finalmente comparados secuencialmente (primero u_6, luego u_{10}, etc.).

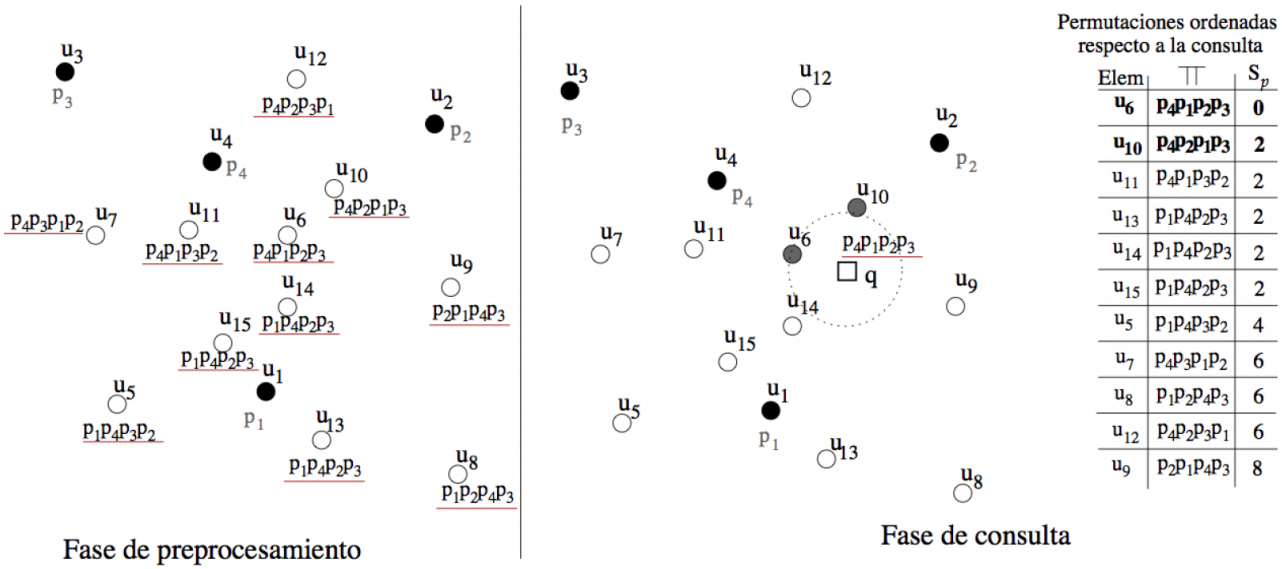

Figura 2.11: Ejemplo de las dos fases de un algoritmo basado en permutaciones. Las permutaciones fueron ordenadas por el valor de S_ρ respecto a la permutación de la consulta.

Existen otras medidas de similaridad entre permutaciones [FKS03], tales como Kendall Tau y Spearman Footrule. La métrica de Kedall Tau entre permutaciones está definida como sigue. Para cada par $p_i, p_j \in \mathbb{P}$, si p_i y p_j están en el mismo orden en Π_u y Π_q, (esto es $\Pi_u^{-1}(p_i) < \Pi_u^{-1}(p_j) \Leftrightarrow \Pi_q^{-1}(p_i) < \Pi_q^{-1}(p_j)$) entonces $K_{p_i,p_j}(\Pi_u, \Pi_q) = 0$; sino $K_{p_i,p_j}(\Pi_u, \Pi_q) = 1$. Kendall Tau está dada por $K(\Pi_u, \Pi_q) = \sum_{p_i,p_j \in \mathbb{P}} K_{p_i,p_j}(\Pi_u, \Pi_q)$. Esto es parecido a contar el número de intercambios necesarios en un ordenamiento con el método de la burbuja para convertir una permutación en la otra.

Por otro lado, Spearman Footrule es la distancia L_1 entre dos permutaciones. Formalmente,

$$F(\Pi_u, \Pi_q) = \sum_{i=1}^{k} |\Pi_u^{-1}(p_i) - \Pi_q^{-1}(p_i)| \tag{2.7}$$

Es importante mencionar que en [Fig07] se mostró que la eficiencia entre estas es pequeña y que la que tiene mejor *tradeodff* es Spearman Footrule.

2.8.2. Encontrando las permutaciones más similares

Como se explicó anteriormente, los elementos con las permutaciones más similares, tendrán altas probabilidades de formar parte del conjunto respuesta, en esta sección veremos como encontrar este conjunto.

Existen 2 procedimientos distintos para encontrar las permutaciones más similares a Π_q. El primero de ellos es usar un Trie con las permutaciones y recorrerlo calculando un valor principal

de S_ρ hasta llegar a una hoja; el otro procedimiento consiste en calcular todos los S_ρ y ordenar toda la base de datos por ese valor en forma creciente. En esta tesis optamos por ordenar la base de datos, a continuación presentaremos un análisis de este procedimiento.

Ordenando el conjunto

El algoritmo consiste en ordenar total o parcialmente la base de datos respecto al valor S_ρ. Este valor debe ser calculado previamente para cada elemento de la base de datos. Este proceso se muestra en el algoritmo 3.

Algoritmo 3 `Sort-búsqueda`(Permutación Π_q, Consulta q)

1: $A[1, n]$ arreglo
2: Sea $m \leq n$
3: $i \leftarrow 1$
4: **for** $u \in \mathbb{U}$ **do**
5: $\quad A[i] \leftarrow S_\rho(\Pi_u, \Pi_q)$
6: $\quad i \leftarrow i + 1$
7: **end for**
8: $sort(A)$ Por orden creciente de S_ρ
9: **for** $i \leftarrow 1$ to m **do**
10: $\quad$ *Comparar* $A[i]$ *con la consulta* q Comparaciones externas
11: **end for**

En el caso de un ordenamiento completo toda la base de datos quedará ordenada y se podrá comparar elemento a elemento de la base de datos contra la consulta hasta que se decida dejar de comparar elementos. En este ordenamiento generalmente se usa Quicksort($qsort$) de la librería estándar de lenguaje C cuya complejidad es $O(n\ log\ n)$.

Un ordenamiento parcial consiste en recuperar uno a uno los elementos de la base de datos más similares(primero el de S_ρ más pequeño, después el segundo más pequeño, etc.) sin tener que ordenar el conjunto completo. Una ventaja de este tipo de algoritmos es que ordena sólo una parte de los datos.

En [Fig07] se ha encontrado que este índice basado en permutaciones funciona bien para bases de datos de dimensión alta(se ha probado un óptimo funcionamiento para dimensión 1024), sin embargo para dimensiónes bajas, el número de evaluaciones de distancia es alto. Por lo que uno de los objetivos de este trabajo es encontrar una técnica que, en general, haga menos cálculos de distancia en dimensión baja.

2.8.3. Excepción del índice basado en permutaciones

Como comentamos en la sección anterior, el índice basado en permutaciones es una excelente alternativa en bases de datos de alta dimensión, sin embargo, hay que analizar detenidamente un caso de configuración del espacio(puede suceder en dimensiones bajas o altas) en el cual la distancia entre permutaciones no expresa con certeza la cercanía entre objetos de un espacio métrico. Para ello consideremos la figura 2.12 y calculemos Π_q, Π_{u_1} y Π_{u_2}:

- $\Pi_q = p_2 p_1$
- $\Pi_{u_1} = p_1 p_2$
- $\Pi_{u_2} = p_2 p_1$

De la figura 2.12 claramente el elemento u_1 es más cercano a la consulta q que el elemento u_2, ahora, usando la ecuación 2.6 calculamos la distancia de permutaciones con respecto a la consulta q y obtenemos los siguientes resultados:

$$S_\rho(\Pi_{u_2}, \Pi_q) = 0 \tag{2.8}$$

$$S_\rho(\Pi_{u_1}, \Pi_q) = 2 \tag{2.9}$$

La ecuación 2.8 nos dice que la distancia entre el elemento u_2 y la consulta q es cero, la ecuación 2.9 nos dice que el elemento u_1 está más lejos de la consulta q que el elemento u_2. Estas inconsistencias pueden provocar que los resultados obtenidos por este índice no sean los mas adecuados, por lo cual en este trabajo se busca una técnica que permita minimizarlas.

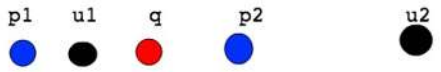

Figura 2.12: Ejemplo de una configuración donde el uso de permutaciones no reporta al elemento más cercano en primer lugar.

Parte III

PROPUESTA

Capítulo 3

Propuesta

Como previamente se mencionó, en dimensiones bajas, el índice basado en permutaciones no tiene un funcionamiento adecuado comparado con los índices basados en pivotes de la sección 2.5. Otra desventaja notable es que no siempre permite descartar objetos de forma segura(ver sección 2.8.3) cuando se resuelven consultas por proximidad.

La contribución principal de esta tesis consiste en garantizar que el algoritmo basado en permutaciones pueda descartar elementos acertadamente. Para lograr este objetivo, es primordial que en el índice exista más información sobre las distancias para tener más criterios que permitan descartar de forma adecuada, es importante también que esta adicíon de información extra no consuma mucho mas espacio en memoria. La técnica presentada en esta sección cumple con estos requisitos mejorando notablemente los resultados arrojados por el algoritmo basado en permutaciones usando bases de datos de dimensión mediana y baja(puede pensarse que un espacio de dimensión baja es aquel que puede ser embebido en un espacio vectorial uniformemente distribuido cuya dimensión es menor que 16).

3.1. Trabajo previo

La propuesta consiste esencialmente en agregar información cuantificada acerca de las distancias al índice basado en permutaciones([FP15]), una vez teniendo esta información extra será posible descartar objetos de forma adecuada logrando un mejor funcionamiento del algoritmo basado en permutaciones en espacios de dimensión media y baja.

3.1.1. Cuantización de zonas

Para cada objeto $u \in \mathbb{U}$, el índice guardará información cuantificada sobre las distancias además de la permutacion. Con el objetivo de guardar esta información, para cada $p \in \mathbb{P}$ definimos m zonas concéntricas limitadas por $m-1$ radios $r_1, ..., r_{m-1}$ y dos valores extras $r_0 = 0$ y $r_m = \infty$(para limitar la primera y la última zona). De esta manera cada objeto $u \in \mathbb{U}$ se encuentra localizado en la zona z_i donde se satisface la relación $r_{i-1} < d(u,p) \leq r_i$.

Es importante aclarar que no es necesario hacer mas cálculos de distancias para obtener las zonas, de hecho, las zonas son calculadas durante la obtención de la permutacion. En esta técnica el índice estará constituido por las siguientes partes:

- Π_u(permutacion para u)
- Z_u(información de zonas para u)

Geométricamente, esto puede verse como partir el espacio con m anillos de distancia centrados en cada permutante. Consideremos pues un ejemplo de esta técnica, en las siguientes figuras tenemos que los permutantes son los objetos $\mathbb{P} = p_1, p_2, p_3$ teniendo cada uno 3 zonas (que son denotadas por z_1,z_2,z_3). En la figura 3.1 se muestra la configuración del espacio para los permutantes p_1 y p_2.

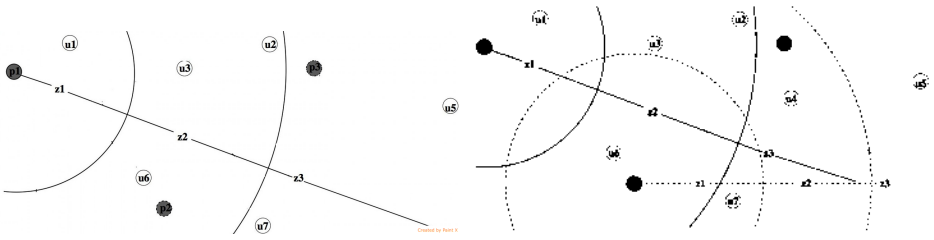

Figura 3.1: En la izquierda se muestra la partición del espacio correspondiente a p_1, mientras que en la derecha puede notarse la partición resultante al agregar los anillos correspondientes a p_2

Sean $u_1, ..., u_7$ elementos de la base de datos. Para cada uno de ellos, en la figura 3.2, mostramos su permutacion Π_u en la secuencia de números encerrada por [ ] y su información de zonas Z_u en la secuencia rodeada por ( ). Por ejemplo, u_6 se encuentra cerca de u_2, u_1, u_3 y pertenece a las zonas $1, 2, 2$ de los permutantes p_1, p_2, p_3 respectivamente. Nótese que con esta técnica podemos determinar si elementos con permutaciones iguales son cercanos o no.

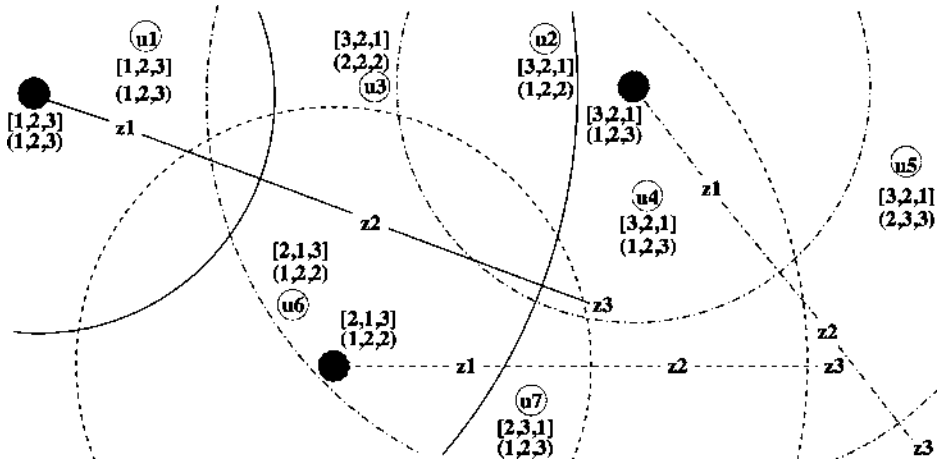

Figura 3.2: Partición del espacio resultante para 3 permutantes

Esta técnica tiene 2 ventajas importantes:

- Es posible descartar algunos elementos sin calcular sus distancias
- Mejorar la capacidad de predicción del algoritmo basado en permutaciones.

En términos de espacio, además de la tabla de permutaciones, necesitamos guardar la información de distancia. Sea $m = |\mathbb{P}|$. Cada objeto necesita $m\lceil log_2 m \rceil$ bits por permutacion. Por otro lado, con el fin de de representar m zonas, son necesarios $m\lceil log_2 z \rceil$ bits. Finalmente, para cada zona de un permutante, es necesario un número flotante para guardar el radio. Esto equivale a $mz32$ bits. Sumando para n objetos obtenemos que son necesarios $nm(\lceil log_2 m \rceil + \lceil log_2 z \rceil) + zm32$ bits.

El algortimo estándar basado en permutaciones usa $nm\lceil log_2 m \rceil$ bits. Con la memoria extra requerida por las zonas, es posible agregar mas permutantes al algoritmo estándar, pero no lo suficiente como para duplicar el número de permutantes. Así, para determinar la equivalencia en espacio requerido entre el algoritmo estándar y esta técnica seguimos el siguiente razonamiento .

Suponiendo que m es potencia de 2, la memoria extra fuerza el uso de otro bit. También se supone que z es potencia de 2, de esta forma, en el numerador tenemos el espacio usado por esta técnica y en el denominador el espacio usado por m^* permutantes, donde $m^* \in (m, 2m)$.
$\frac{nm(log_2 m + log_2 z) + zm32}{nm^*(log_2 m + 1)} = \frac{nm(log_2 m + log_2 z)}{nm^*(log_2 m + 1)} + \frac{zm32}{nm^*(log_2 m + 1)}$

El segundo término es $O(1/n)$, por lo que es despreciable. El primer término es
$\frac{nm(log_2 m + log_2 z)}{nm^*(log_2 m + 1)} = \frac{m}{m^*}(1 + \frac{log_2 z - 1}{log_2 m + 1})$. En el mismo espacio, esto es igual a 1, por lo que el número m^* de permutantes equivalentes para m permutantes y z zonas es:

$$m^* = m(1 + \frac{log_2 z - 1}{log_2 m + 1}) \tag{3.1}$$

3.2. Descripción de la propuesta

3.2.1. Etapa de indexamiento

En esta tesis, proponemos parcializar un índice con permutaciones e información de zonas, es decir, si Π_u consta de 4 elementos, tomaremos en cuenta sólo 2 o 3 elementos, de igual forma se reducirá el tamaño del vector de zonas. En la figura 3.3 se presenta un ejemplo de la propuesta, los permutantes son los objetos $\mathbb{P} = \{p_1, p_2, p_3, p_4\}$. Cada uno tiene 4 zonas (denotados por z_1, z_2, z_3, z_4).

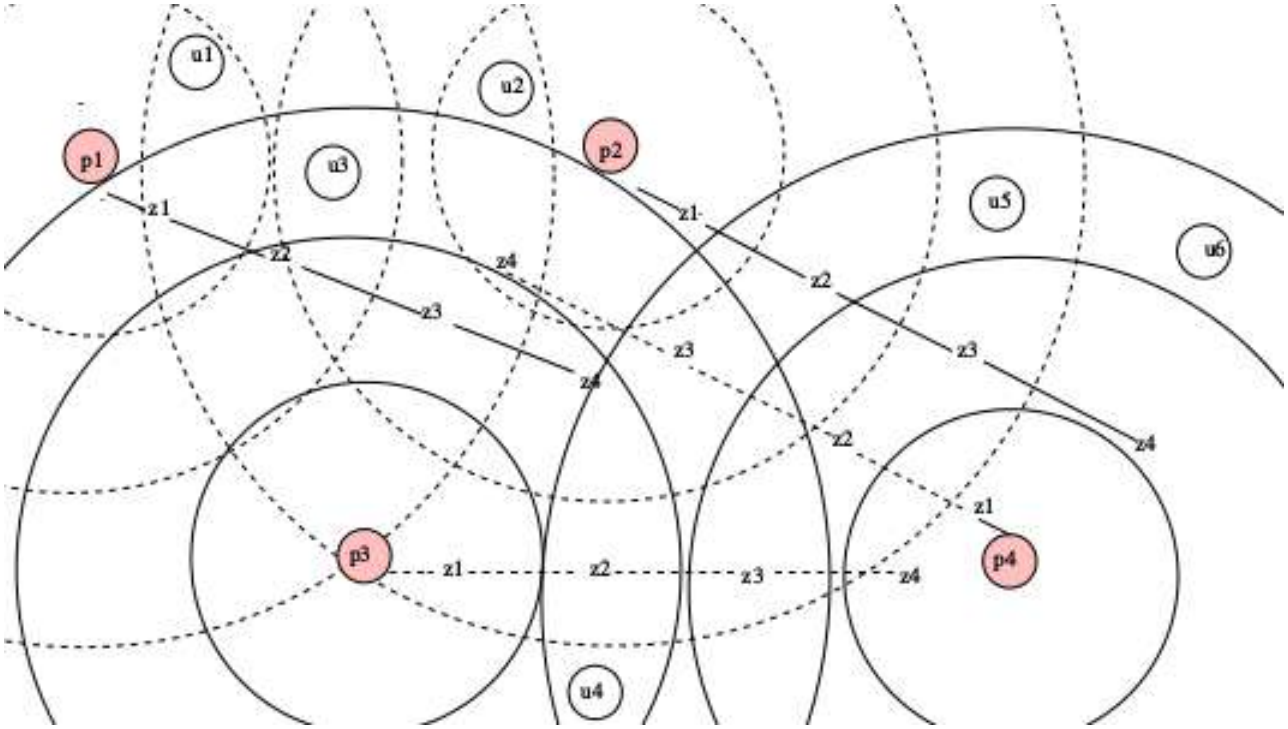

Figura 3.3: Partición del espacio resultante para 4 permutantes

Sean $u_1, ..., u_6$ elementos de la base de datos, en el cuadro 3.1 se muestra el índice consistente de la permutacion y el vector de información de zonas:

Elemento(u_i)	Permutacion(Π_{u_i})	Vector de zonas(Z_{u_i})
u_1	$p_1p_2p_3p_4$	$(1,3,4,4)$
u_2	$p_2p_1p_3p_4$	$(1,3,4,4)$
u_3	$p_1p_2p_3p_4$	$(2,2,3,4)$
u_4	$p_3p_4p_2p_1$	$(2,3,4,4)$
u_5	$p_4p_2p_3p_1$	$(3,3,4,4)$
u_6	$p_4p_2p_3p_1$	$(3,4,4,4)$

Cuadro 3.1: Ejemplo de índice conformado por permutaciones e información de zonas

Como se menciono anteriormente, la propuesta de esta tesis consiste en conservar solamente un porcentaje de la información compuesta por las permutaciones y el vector de zonas, por ejemplo, si consideramos el 50 % del índice mostrado en el cuadro 3.1 obtenemos el nuevo índice mostrado en el cuadro 3.2:

Elemento(u_i)	Permutacion(Π_{u_i})	Vector de zonas(Z_{u_i})
u_1	p_1p_2	$(1,3)$
u_2	p_2p_1	$(1,3)$
u_3	p_1p_2	$(2,2)$
u_4	p_3p_4	$(2,3)$
u_5	p_4p_2	$(3,3)$
u_6	p_4p_2	$(3,4)$

Cuadro 3.2: Ejemplo de índice parcializado al 50 %

Adicionalmente, si consideramos el 75 % del índice completo resulta el índice mostrado en el cuadro 3.3:

Elemento(u_i)	Permutacion(Π_{u_i})	Vector de zonas(Z_{u_i})
u_1	$p_1p_2p_3$	$(1,3,4)$
u_2	$p_2p_1p_3$	$(1,3,4)$
u_3	$p_1p_2p_3$	$(2,2,3)$
u_4	$p_3p_4p_2$	$(2,3,4)$
u_5	$p_4p_2p_3$	$(3,3,4)$
u_6	$p_4p_2p_3$	$(3,4,4)$

Cuadro 3.3: Ejemplo de índice parcializado al 75 %

A continuación describiremos como usar este índice durante la etapa de consulta.

3.2.2. Etapa de consulta

Descartando objetos Con el objetivo de descartar objetos, ha sido adaptado el criterio de exclusión usado por los pivotes. Es por ello que después de calcular la distancia q a todos los permutantes(para obtener Π_q), calculamos las zonas donde se encuentra q (Z_q) y las zonas donde intersecta la bola de consulta $(q, r)_d$. Por esta razón, definimos dos arreglos FZ y LZ. Para cada permutante, en FZ y LZ guardamos respectivamente la primera y la última zona donde intersecta la bola de consulta. Por tanto, dado el radio de consulta r, para cada permutante $p \in \mathbb{P}$, FZ_p y LZ_p guardan la zona que contiene $d(p, q) - r$ y $d(p, q) + r$ respectivamente. Con esto, cuando estamos revisando los objetos, descartamos cualquier elemento cuya zona no se encuentre en el rango $[FZ_p, LZ_p]$ para cada permutante $p \in \mathbb{P}$, esto es posible gracias a la desigualdad triangular.

Creando una nueva distancia Debido a que tenemos más información para cada objeto de la base de datos(permutacion y vector de información de zonas), usaremos la información de zonas para mejorar el orden de revisión de una consulta por similaridad. Se propone calcular la distancia entre vectores de información de zonas de la siguiente forma:

$$Z_D(Z_u, Z_v) = \sum_{j=[1,|\mathbb{P}|]} |Z_u(j) - Z_v(j)| \tag{3.2}$$

Z_D acumula la suma del valor absoluto de las diferencias en los identificadores de zonas entre objetos u, $v \in \mathbb{U}$ para todos los permutantes.

A continuación mencionaremos las posibles distancias a usar en esta propuesta:

- **Permutantes** Calcular Spearman Rho S_ρ como en el algoritmo basado en permutaciones original, ver ecuación 2.6.
- **sPZ** Es la suma de $S_\rho(\Pi_u, \Pi_q)$ y $Z_D(Z_u, Z_v)$.
- **PZ** Calculamos $S_\rho(\Pi_u, \Pi_q)$ y $Z_D(Z_u, Z_v)$ separadamente, ordenamos de acuerdo a S_ρ y usamos Z_D para descartar objetos.
- **ZP** Análogo a **PZ** pero ordenamos de acuerdo a Z_D y usamos S_ρ para descartar objetos.

3.2.3. Generación de radios

Los radios pueden ser asignados de las siguientes formas:

- **Uniformemente distribuidos por distancias(QD)** En este caso, obtenemos una muestra de objetos y calculamos su distancia hacia el permutante. Esto permite obtener una buena aproximación de la distribución de distancias con respecto al permutante usado. Llamemos r_{max} y r_{min} la máxima y la mínima distancia calculada en la muestra para ese permutante. Luego, básicamente se usa $(r_{max} - r_{min})/m$ como el incremento entre un radio y el siguiente.

- **Uniformemente distribuidos por elementos(QQ)** Una vez mas obtenemos una muestra de objetos y los comparamos con el permutante. Después, ordenamos las distancias y seleccionamos puntos tomados en intervalos regulares.

3.3. Mejorando el desempeño del algoritmo basado en permutaciones

En esta sección mostraremos la forma en que la información por zonas permite obtener resultados mas acertados a la hora de realizar búsquedas por proximidad. Recordemos el caso analizado en la sección 2.8.3 y analicemos que sucede al agregar la nueva información. En la figura 3.4 los permutantes son los objetos $\mathbb{P} = \{p_1, p_2\}$. Cada uno tiene 3 zonas denotadas por z_1, z_2, z_3. Sea q una consulta y u_1, u_2 elementos de la base de datos.

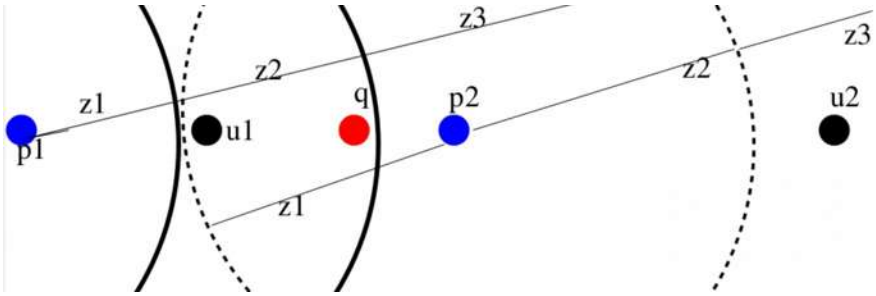

Figura 3.4: Agregando zonas al caso excepcional de la sección 2.7.3

Recordemos que los resultados de las ecuaciones 2.8 y 2.9 significaban que u_2 era más cercano a la consulta que u_1, siendo esto totalmente contradictorio a la configuración del espacio. Calculando información de zonas esperamos obtener mejores predicciones, los vectores con información de zonas son:

- $Z_q = (2, 1)$
- $Z_{u_1} = (2, 1)$
- $Z_{u_2} = (3, 3)$

Usando la ecuación 3.2 calculamos la distancia entre zonas respecto a la consulta q y obtenemos los siguientes resultados:

$$Z_D(Z_{u_2}, Z_q) = 3 \tag{3.3}$$

$$Z_D(Z_{u_1}, Z_q) = 0 \tag{3.4}$$

Los resultados de las previas ecuaciones son congruentes con la configuración de nuestra base de datos ya que el resultado expresado en la ecuación 3.3 afirma que efectivamente el elemento u_2 es el mas lejano a la consulta.

Sintetizando la información, para este ejemplo el índice será:

Elemento(u_i)	Permutacion(Π_{u_i})	Vector de zonas(Z_{u_i})
u_1	p_1p_2	$(1,1)$
u_2	p_2p_1	$(3,3)$

Mientras que la información de la consulta es:

Consulta	Permutacion(Π_q)	Vector de zonas(Z_q)
q	p_2p_1	$(2,1)$

Finalmente, haciendo uso de la nueva distancia **SPZ** definida en la sección 3.2.2 obtenemos los siguientes resultados:

$$Z_D(Z_{u_2}, Z_q) + S_\rho(\Pi_{u_2}, \Pi_q) = 3 \tag{3.5}$$

$$Z_D(Z_{u_1}, Z_q) + S_\rho(\Pi_{u_1}, \Pi_q) = 2 \tag{3.6}$$

Los valores arrojados por estas últimas ecuaciones significan que el objeto u_1 se encuentra efectivamente mas cerca de la consulta q que el objeto u_2, mejorando notablemente el resultado final del algoritmo basado en permutaciones.

Parte IV

EXPERIMENTACIÓN

Capítulo 4

Experimentación

En esta sección presentaremos los resultados de los experimentos hechos con el índice propuesto en la sección anterior, para llegar a estos resultados fueron utilizadas dos tipos de bases de datos usando la distancia euclidiana: bases de datos sintéticas(cubo unitario distribuido uniformemente en dimensión [2,20]) y bases de datos reales(imágenes de la nasa e histogramas de color). El primer tipo de base de datos nos permite conocer el comportamiento del algoritmo y cómo la variación de los párametros afecta los resultados. El segundo tipo de bases de datos es importante pues nos permite saber como funciona el nuevo índice en bases de datos reales.

4.1. Bases de datos sintéticas

Las gráficas a continuación muestran los resultados obtenidos para bases de datos sintéticas(80000 vectores distribuidos uniformemente en el cubo unitario), los parámetros a considerar fueron el número de vecinos más cercanos(1knn, 2knn, 4knn) y la forma de generación de radios para la información de zonas(percentiles o distancia uniforme), además el número de permutantes coincide con el valor de la dimensión. Las líneas en las gráficas representan distintos tipos de índices, por ejemplo, sPZ $z16$ 0,5 significa primeramente el tipo de distancia usada de acuerdo a la sección 3.2.2, después se representa el número de zonas asignadas y finalmente el rango del índice usado el cual va entre 0 y 1; 1 corresponde a la permutación completa. En las gráficas mostradas es hecha la comparación contra el método clásico de pivotes y en algunas también contra el método de permutantes. Es de notar que prácticamente en todas las gráficas el índice propuesto en esta tesis reduce considerablemente el número de cálculos de distancia con respecto a los algoritmos del estado del arte.

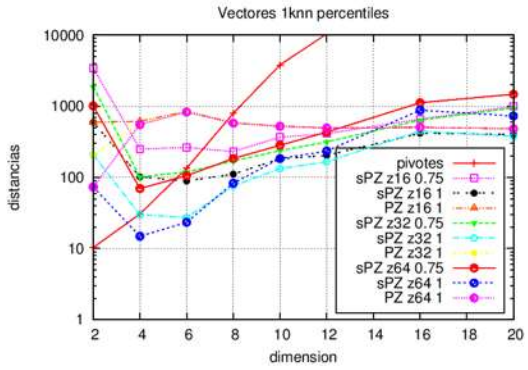

Figura 4.1: Comportamiento de búsqueda en el espacio de vectores usando percentiles y 1knn.

En la gráfica 4.1 es muy notario, a partir de dimensión 8, que el índice clásico de pivotes se dispara en el número de distancias calculadas. También podemos ver que para el rango de dimensión $[4-10]$ los índices propuestos en esta tesis tienen un funcionamiento adecuado. Nótese que para dimensión $[4-8]$ el índice con 64 zonas y usando consulta sPZ fue el que obtuvo mejores resultados, mientras que para dimensión $[10-20]$, los índices usando consulta sPZ con 32 y 16 zonas fueron los mas eficientes, en ambos casos considerando el índice completo. Otro dato a tomar en cuenta es que el índice parcial al 75 %(sPZ $z32$ 0,75) hizo en promedio 15.8 % más cálculos de distancia que los mejores índices.

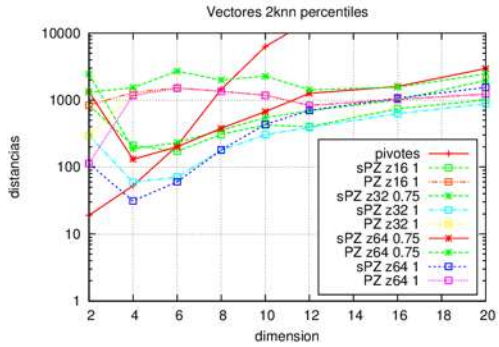

Figura 4.2: Comportamiento de búsqueda en el espacio de vectores usando percentiles y 2knn.

En la gráfica 4.2 vemos que los parámetros de mayor eficiencia son los mismo que para el caso de 1knn. Además el índice parcial sPZ $z64$ 0,75 hizo en promedio 50 % mas cálculos de distancia que los mejores índices.

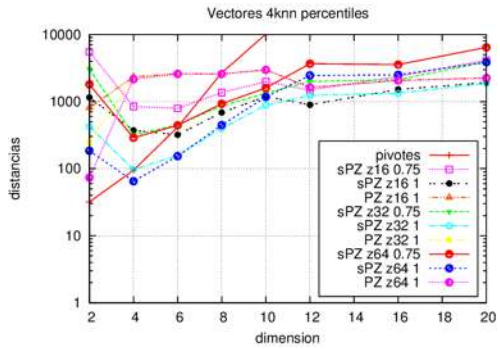

Figura 4.3: Comportamiento de búsqueda en el espacio de vectores usando percentiles y 4knn.

Para el caso de 4knn podemos observar, en la figura 4.3 que para dimensión $[2, 8]$ los mejores resultados fueron obtenidos usando sPZ y 64 zonas, para dimensiónes 10, 16 y 20 el índice más eficiente fue usando sPZ y 32 zonas, mientras que para dimensión 12 sPZ y 16 zonas obtuvo el mejor comportamiento. Por otro lado, los índices parciales sPZ $z64$ 0,75 y sPZ $z16$ 0,75 para rangos de dimensiones $[2-10]$ y $[12-20]$ hacen en promedio 56.14 % mas cálculos de distancia que los mejores índices.

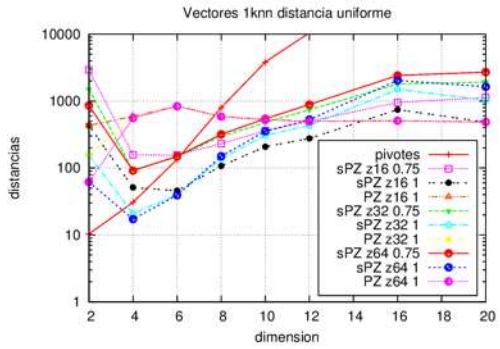

Figura 4.4: Búsqueda en el espacio de vectores usando distancia uniforme para 1knn.

Veremos ahora los resultados obtenidos para experimentos usando distancia uniforme en la generación de radios de las zonas. En la figura 4.4 se presenta el caso de 1knn y podemos ver que para el rango de dimensiones $[4-6]$, los mejores resultados fueron obtenidos usando el índice sPZ $z64$ 1, en el rango $[8-12]$ el índice que hizo menor cálculos de distancia fue sPZ $z16$ 1, mientras que para el rango $[16-20]$ el mejor índice fue PZ $z64$ 1, para este caso el índice parcial sPZ $z32$ 0,75 hizo en promedio 58.42 % mas evaluaciones que los índices de mejor comportamiento.

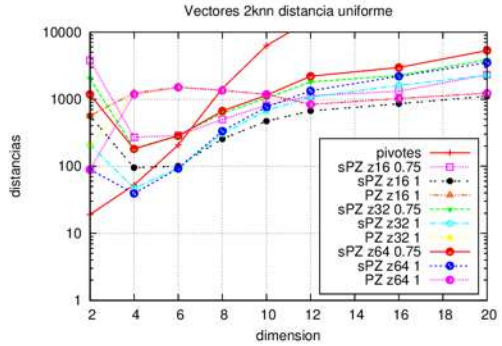

Figura 4.5: Búsqueda en el espacio de vectores usando distancia uniforme para 2knn.

Para 2knn, mostrada en la figura 4.5 en el rango de dimensiones $[4-6]$ el índice sPZ $z64$ 1 fue el índice con menor cálculos de distancia realizados, mientras que para $[8-20]$ el mejor índice fue sPZ $z16$ 1, además el índice sPZ $z32$ 0,75 realizó en promedio 58.42 % mas evaluaciones que los índices de mejor comportamiento.

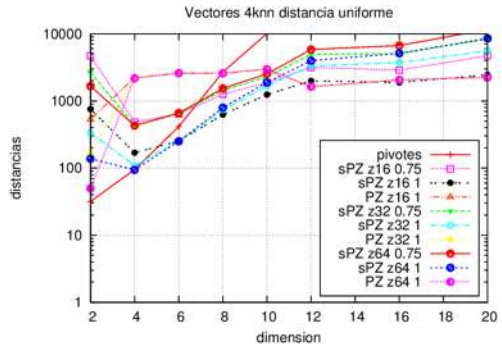

Figura 4.6: Búsqueda en el espacio de vectores usando distancia uniforme para 4knn.

En el caso de 4knn, mostrado en la figura 4.6, se obtuvo que para dimensiones en el rango $[4-6]$ el mejor índice fue sPZ $z64$ 1, mientras que para el rango $[8-20]$ el mejor índice fue sPZ $z16$ 1. Además el índice parcial sPZ $z32$ 0,75 realizó en promedio 68.28 % mas distancias que los mejores índices.

Procedemos ahora a presentar un análisis aún más detallado de los experimentos, en las siguientes gráficas hemos variado todos los parámetros(dimensión, número de permutantes, porcentaje del índice y número de zonas)

En las gráficas 4.7 y 4.8 se ha reducido el rango de la dimensión del espacio, podemos ver que para cualquier forma de generación de radios para la información de zonas(percentiles y distancia uniforme), los índices del estado del arte(permutantes y pivotes) calculan más distancias que el índice propuesto en esta tesis.

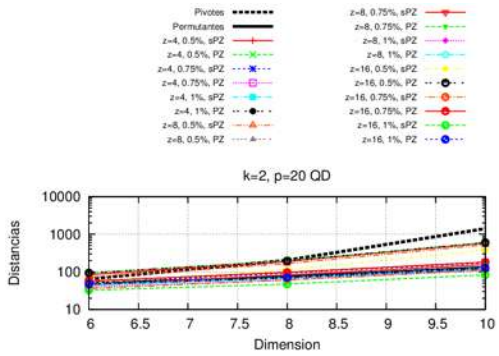

Figura 4.7: Comportamiento de búsqueda en el espacio de vectores usando percentiles para 2NN y 20 permutantes.

En la gráfica anterior se obtuvo que el índice con menor cálculos de distancia fue sPZ $z16$ 1, mientras que el índice parcial PZ $z16$ 0,75 realizó en promedio 10.33 % mas cálculos que el mejor índice. Este porcentaje es bastante aceptable, ya que con un poco más de cálculos y menos espacio en memoria se obtiene un índice bastante eficiente.

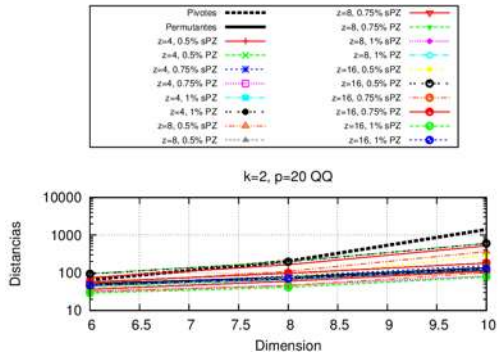

Figura 4.8: Comportamiento de búsqueda en el espacio de vectores usando distancia uniforme para 2NN y 20 permutantes.

Usando distancia uniforme resultó que el índice con menor cálculos de distancia fue sPZ $z16$ 1, mientras que el índice parcial PZ $z16$ 0,75 realizó en promedio 12 % mas cálculos que el mejor índice. De nueva cuenta, el índice parcial es bastante bueno por el espacio en memoria ahorrado.

En las gráficas 4.9 y 4.10 podemos ver el comportamiento en el espacio de vectores si el número de permutantes varía, de manera análoga al par de gráficas anteriores vemos que el índice propuesto en esta tesis está por debajo de los índices de permutantes y pivotes.

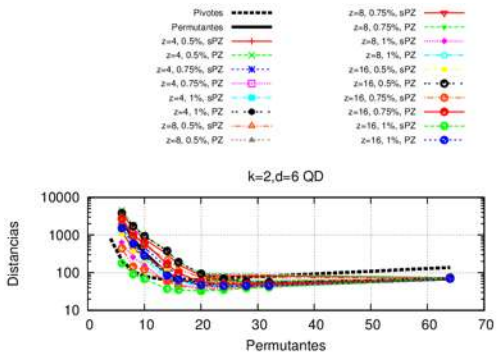

Figura 4.9: Comportamiento de búsqueda en el espacio de vectores usando percentiles para 2NN y dimensión 6.

Es posible observar que después de 30 permutantes todos los índices con excepción del índice de pivotes comienzan a estabilizarse en un sólo valor. Entre 2 y 20 permutantes vemos que el índice con menor cálculo de distancias fue sPZ $z16$ 1, mientras que el mejor índice parcial fue sPZ $z16$ 0,75, nótese también que usando 20 permutantes se obtuvieron el menor número de cálculos de distancia para todos los índices.

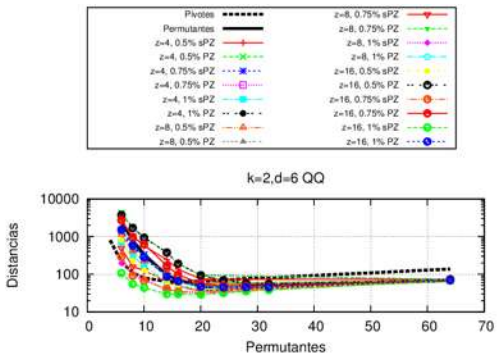

Figura 4.10: Comportamiento de búsqueda en el espacio de vectores usando distancia uniforme para 2NN y dimensión 6.

Usando distancia uniforme vemos que en general se hicieron menos cálculos de distancia que la gráfica de la figura 4.9, además los índices de mejor comportamiento son los mismos que en la gráfica anterior.

En las 2 gráficas mostradas a continuación(Figuras 4.11, 4.12) vemos el comportamiento de la búsqueda cuando varía el porcentaje del índice usado, cabe resaltar que, para algunos de los parámetros considerados, los índices parciales hacen prácticamente el mismo número de distancias que el índice completo, esto es muy importante ya que estos resultados demuestran que si usamos menos espacio en memoria para almacenar el índice se sigue teniendo un comportamiento muy eficiente.

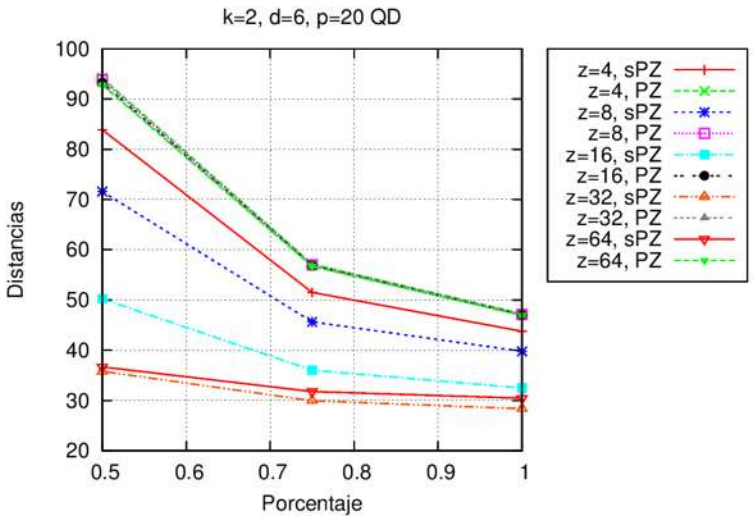

Figura 4.11: Resultados de búsqueda para 2NN en dimensión 6 usando 20 permutantes.

En la figura 4.11 podemos ver que los índices de mejor comportamiento fueron sPZ $z64$ y sPZ $z32$. Note que el valor tomado por el índice parcial 0.75 es muy similar al índice completo.

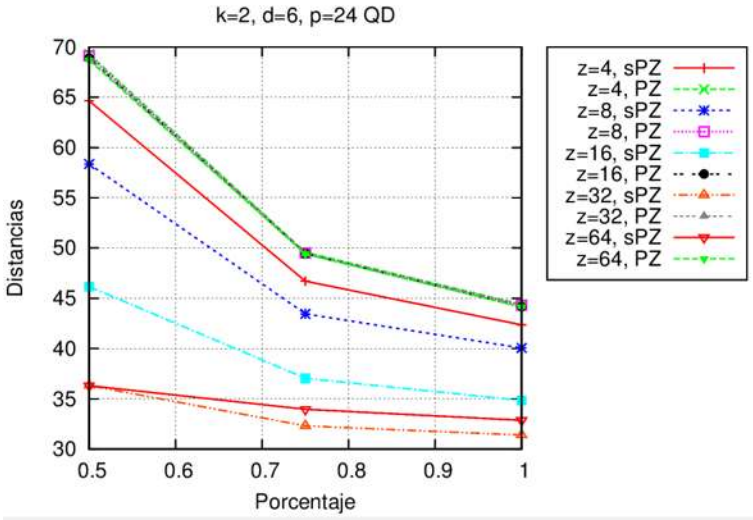

Figura 4.12: Resultados de búsqueda para 4NN en dimensión 6 usando 24 permutantes.

Vemos también en la figura 4.12 que los índices de mejor comportamiento fueron sPZ $z64$ y sPZ $z32$. Es importante resaltar que en este caso existe también poca diferencia entre el número de distancias hechas por el índice parcial(75 %) y las distancias hechas por el índice completo.

Ahora, analizaremos algunas gráficas en las cuales el parámetro que varía es el número de zonas usadas por el índice.

En las siguientes gráficas notamos que para todos aquellos índices de la forma PZ el número de distancias calculadas se mantienes constante, esto se debe a que PZ no usa la información de zonas como primera alternativa para encontrar los vecinos más cercanos.

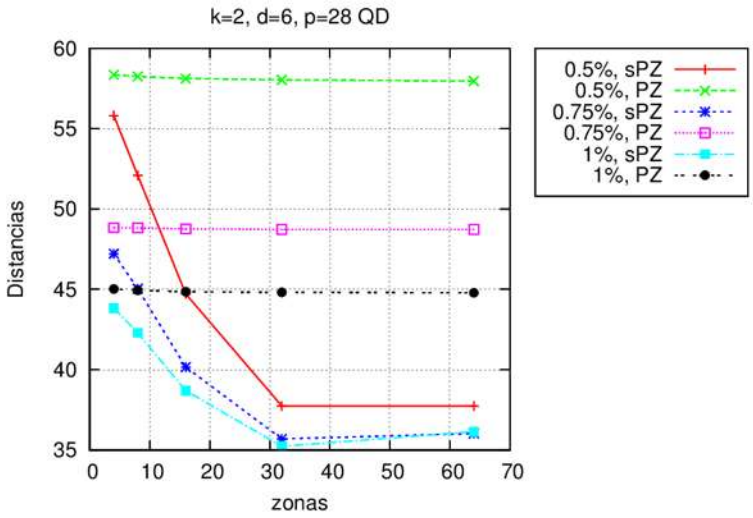

Figura 4.13: Resultados de búsqueda para 2NN en dimensión 6 usando 28 permutantes con percentiles.

En la gráfica anterior vemos que los índices de mejor comportamiento fueron aquellos que usaron sPZ. Notemos que a partir de 32 zonas el índice parcial(75 %) iguala al índice completo.

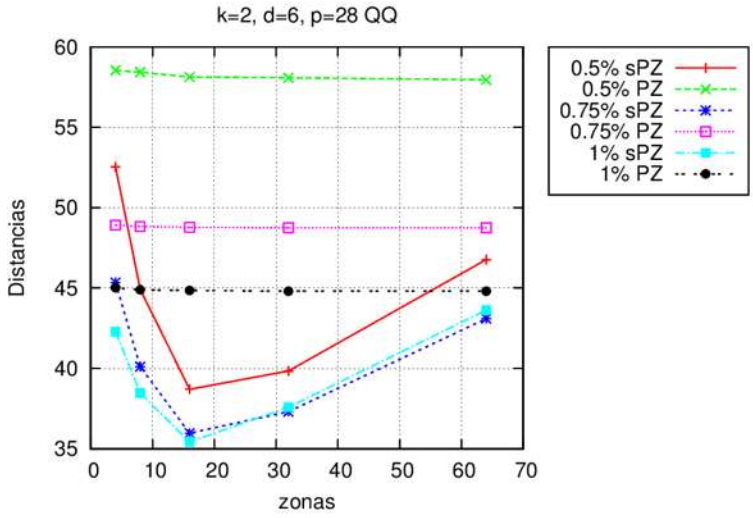

Figura 4.14: Resultados de búsqueda para 2NN en dimensión 6 usando 28 permutantes con distancia uniforme.

En la figura 4.14 vemos nuevamente que los índices de mejor comportamiento fueron los que usaron sPZ. Es muy interesante que en este caso, a partir de 32 zonas, el índice parcial al 75 % hizo menos calculos de distancia, es decir, que se redujo el número de distancias calculadas usando menos espacio en memoria.

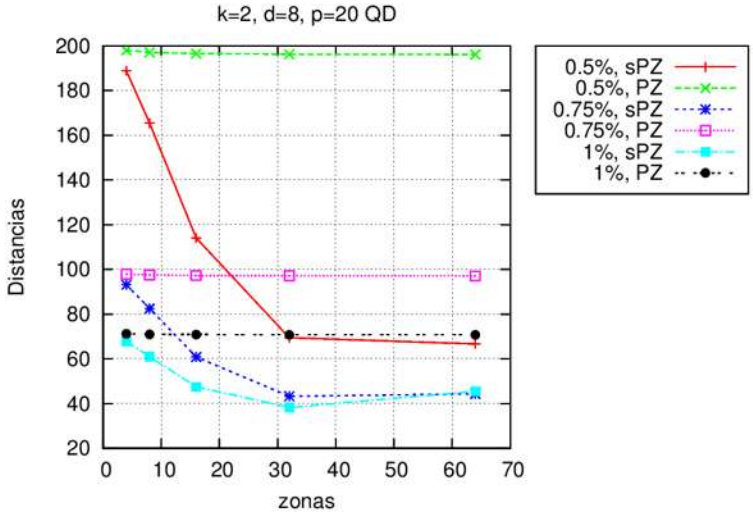

Figura 4.15: Resultados de búsqueda para 2NN en dimensión 8 usando 20 permutantes con percentiles.

En la figura 4.15, de igual forma que las gráficas anteriores a partir de 32 zonas el índice parcial al 75 % iguala el número de distancias hechas por el índice completo.

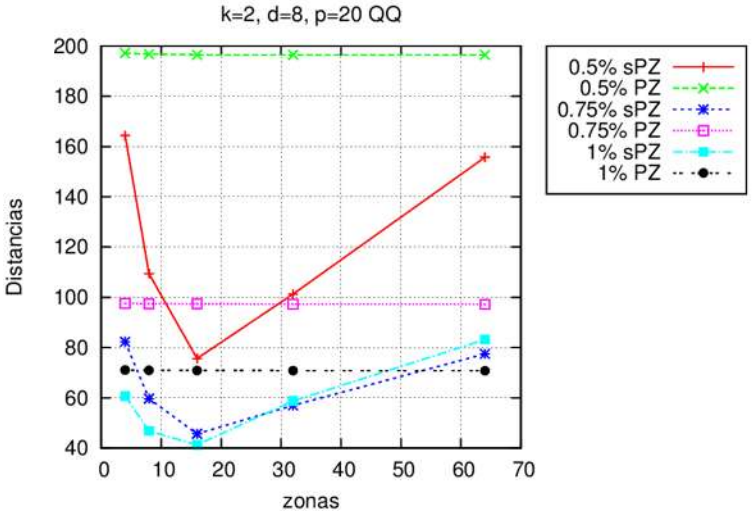

Figura 4.16: Resultados de búsqueda para 2NN en dimensión 8 usando 20 permutantes con distancia uniforme.

En la gráfica de la Figura 4.16 encontramos también que a partir de 32 zonas el índice parcial al 75 % hace inclusive menos calculos de distancia que el índice completo, esto es importantes, pues se logró reducir el número de distancias calculado y el espacio en memoria usado.

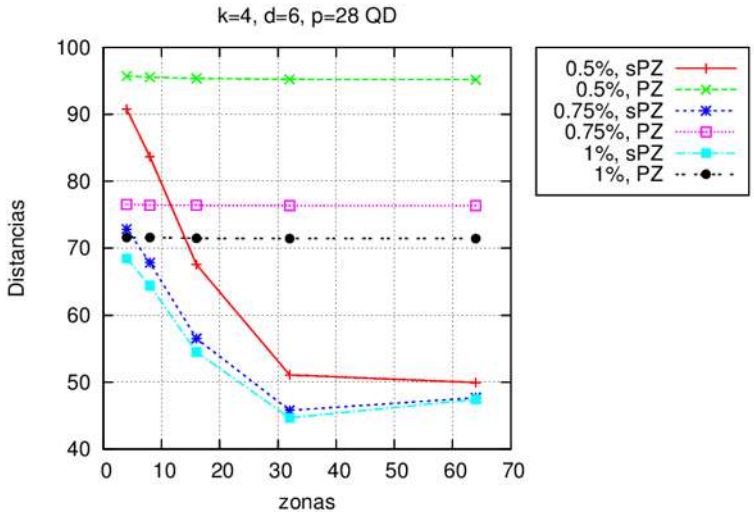

Figura 4.17: Resultados de búsqueda para 4NN en dimensión 6 usando 28 permutantes con percentiles.

En la gráfica 4.17 también podemos ver que a partir de 32 zonas el índice parcial iguala al índice completo.

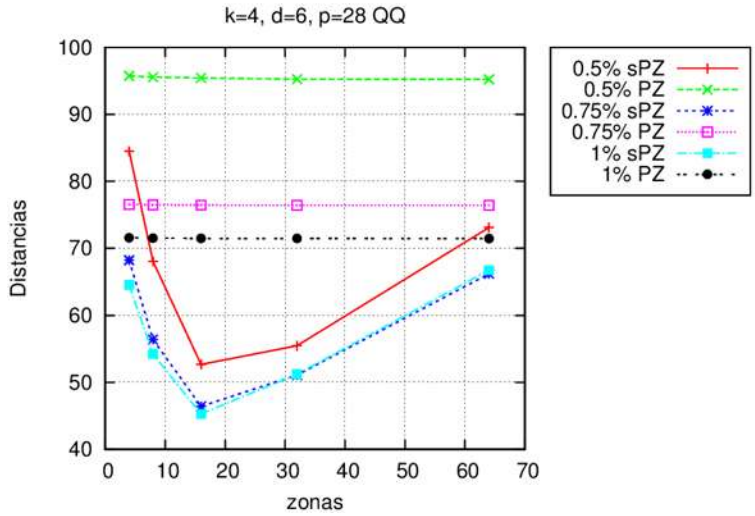

Figura 4.18: Resultados de búsqueda para 4NN en dimensión 6 usando 28 permutantes con distancia uniforme.

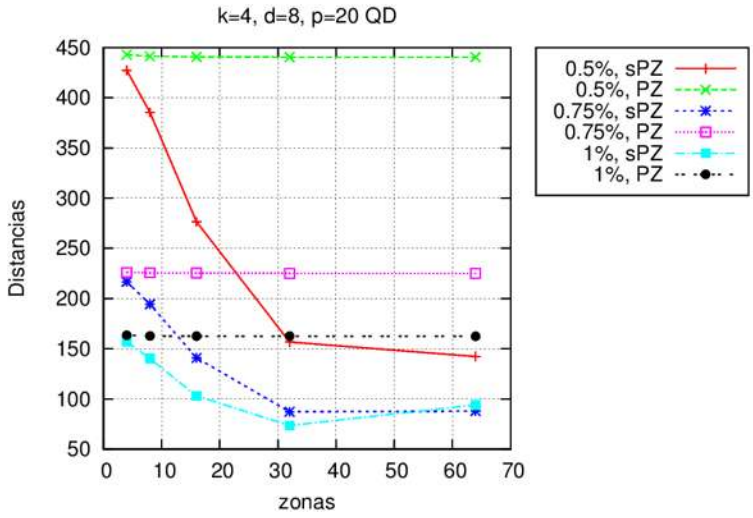

Figura 4.19: Resultados de búsqueda para 4NN en dimensión 8 usando 20 permutantes con percentiles.

En la figuras 4.18 y 4.19 se ve nuevamente que a partir de 32 zonas, el índice parcial al 75 % iguala al índice completo, quedando inclusive abajo del índice completo usando 64 zonas.

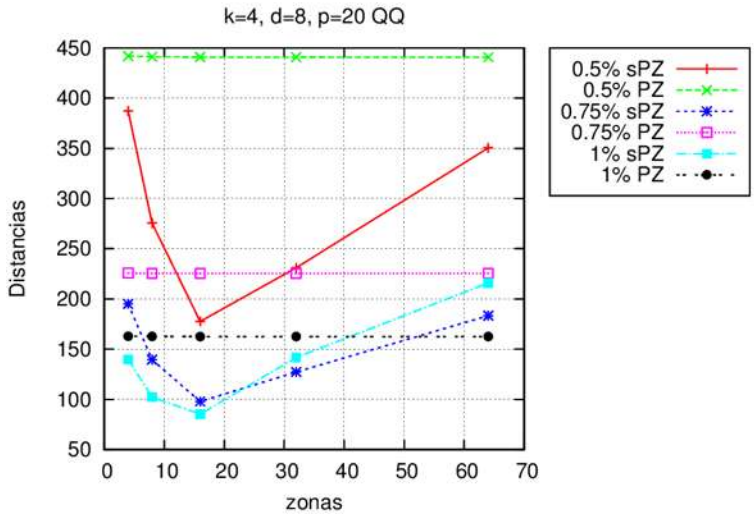

Figura 4.20: Resultados de búsqueda para 4NN en dimenesión 8 usando 20 permutantes con distancia uniforme.

En la gráfica 4.20 vemos como nuevamente el índice parcial al 75 %, a partir de 32 zonas comienza a hacer menos calculos de distancia que el índice completo. En resúmen, encontramos casos en los que el índice 0.75 % sPZ hace el mismo calculo de distancias que el índice completo y más aún, cuando se usa distancia uniforme queda por debajo del índice completo, esto significa que con menos

espacio en memoria es posible reducir las distancias calculadas gracias al índice propuesto en esta tesis.

4.2. Bases de datos reales

En esta sección fueron utilizadas imagenes de la nasa e histogramas de colores para la realización de estos experimentos.

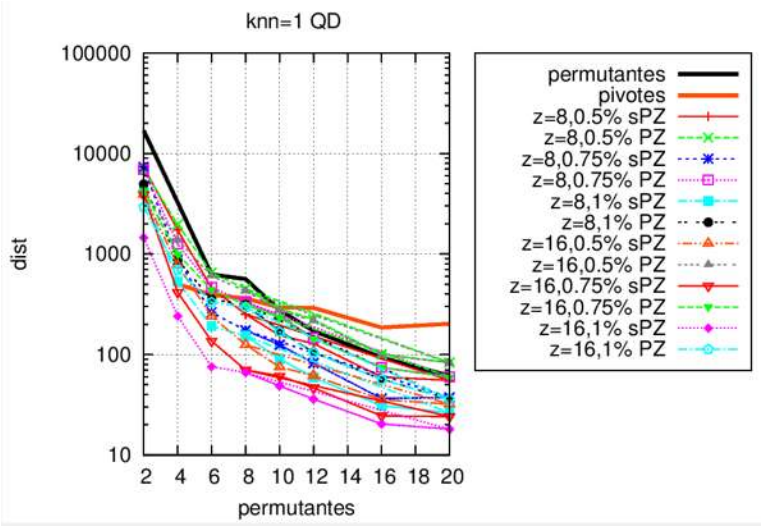

Figura 4.21: Búsqueda en el espacio de imagenes de la NASA para 1NN usando percentiles.

Podemos ver en la figura 4.21 que los índices del estado del arte están muy por encima de los índices propuestos en esta tesis, vemos tamién que el mejor índice fue el índice completo con 16 zonas y sPZ. Note que usando 8 permutantes, el índice parcial al 75 % realizó el mismo número de distancia que el índice completo.

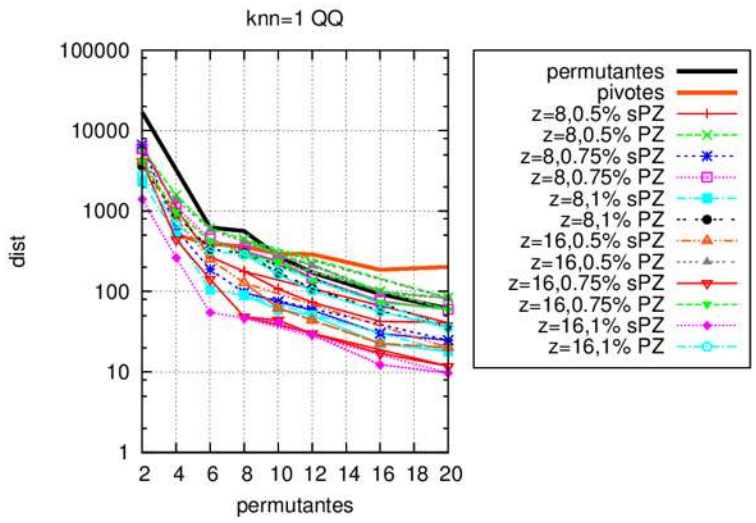

Figura 4.22: Búsqueda en el espacio de imagenes de la NASA para 1NN usando distancia uniforme.

En la figura 4.22 podemos ver que el mejor índice también fue el índice completo con 16 zonas y sPZ, sin embargo, si usamos 8, 10 o 12 permutantes el índice parcial al 75 % hace el mismo calculo de distancias que el índice completo. Vemos además que usando distancia uniforme se realizan menos calculos de distancias que usando percentiles.

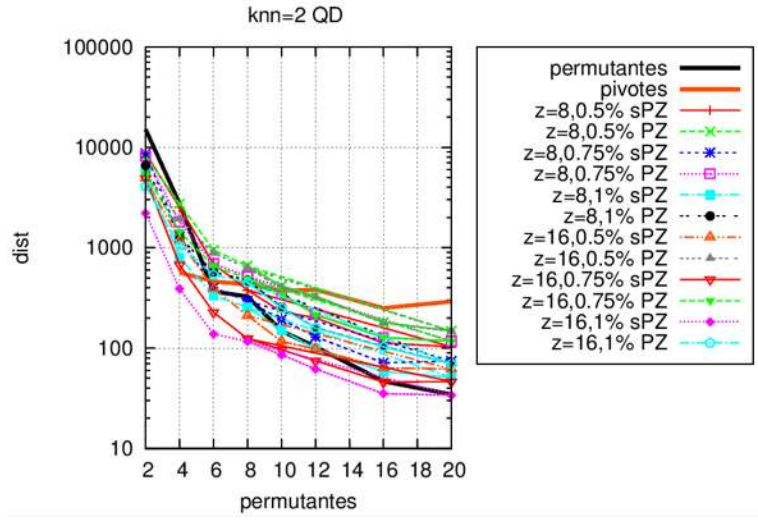

Figura 4.23: Búsqueda en el espacio de imagenes de la NASA para 2NN usando percentiles.

Vemos nuevamente, en la figura 4.23 que el índice que hizo el menor número de distancias fue el índice completo con 16 zonas y sPZ, tamién vemos que con 8 permutantes el índice parcial al 75 % hace el mismo número de calculos que el índice completo. Note que el índice de permutantes

comienza a hacer el mismo de distancias que los índices con información de zonas.

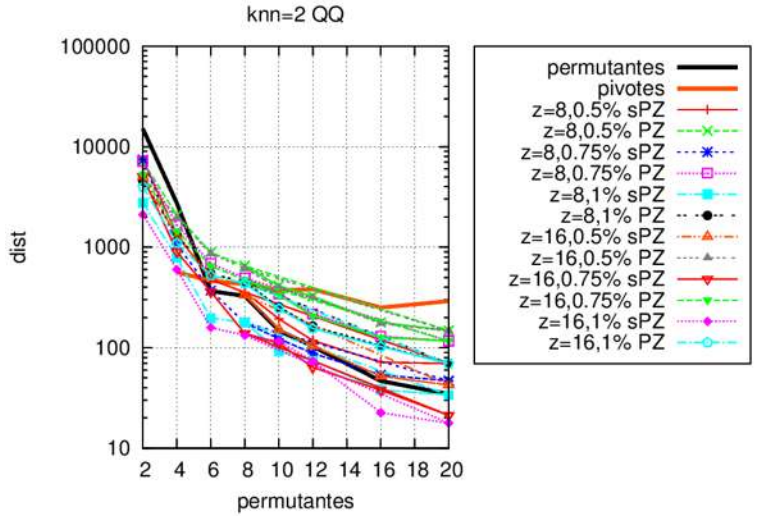

Figura 4.24: Búsqueda en el espacio de imagenes de la NASA para 2NN usando distancia uniforme.

En la figura 4.24 vemos que usando 8 permutantes el índice parcial al 75 % con 16 zonas y sPZ iguala al índice completo con 16 zonas y sPZ, usando 10 permutantes el mejor índice fue el índice completo con 8 zonas y sPZ, de igual forma que para 1NN usando distancia uniforme se hacen menos calculos de distancia.

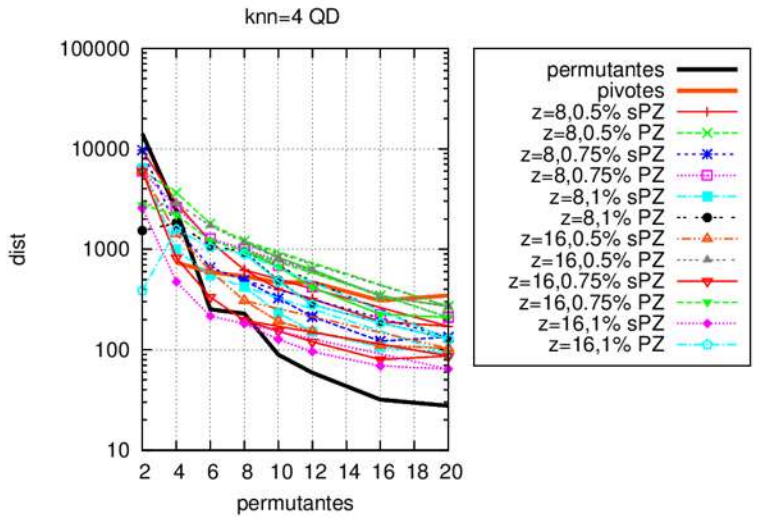

Figura 4.25: Búsqueda en el espacio de imagenes de la NASA para 4NN usando percentiles.

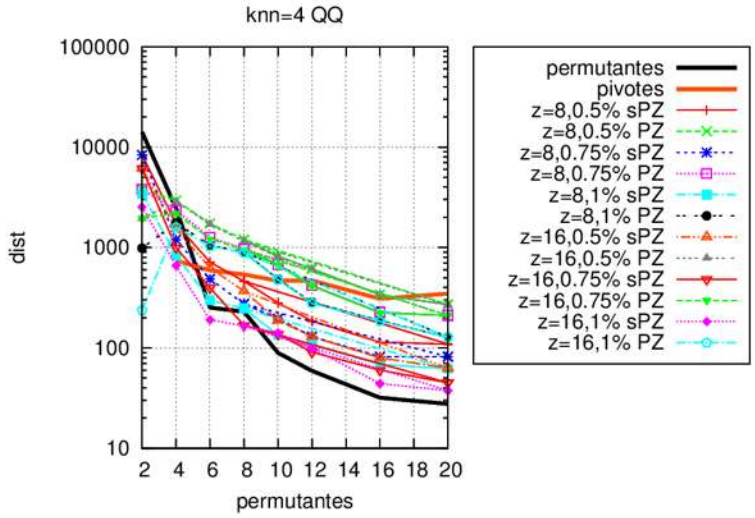

Figura 4.26: Búsqueda en el espacio de imagenes de la NASA para 4NN usando distancia uniforme.

Para 4NN, en las figuras 4.25 y 4.26 vemos que el índice de permutantes fue el que realizó menor número de distancias, por lo que para este caso el índice propuesto en esta tesis no es útil.

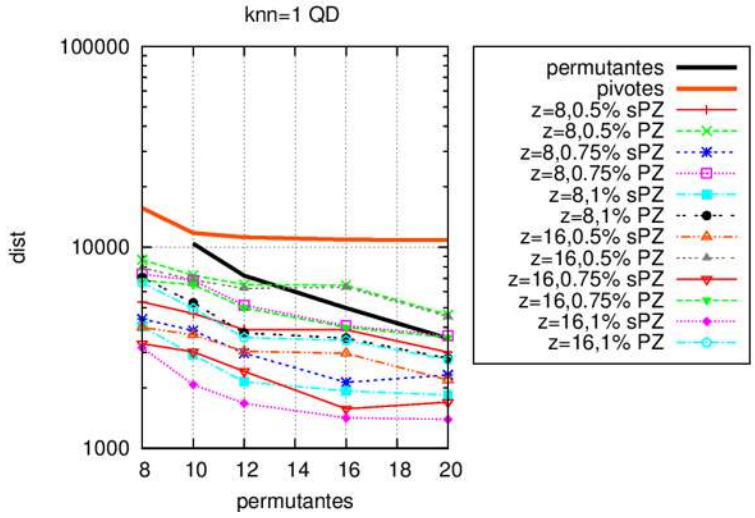

Figura 4.27: Búsqueda en el espacio de histogramas de colores para 1NN usando percentiles.

En la figura 4.27 vemos que el mejor índice fue el índice completo con 16 zonas y sPZ, usando 16 permutantes, el índice parcial al 75 % con 16 zonas y sPZ iguala al índice completo.

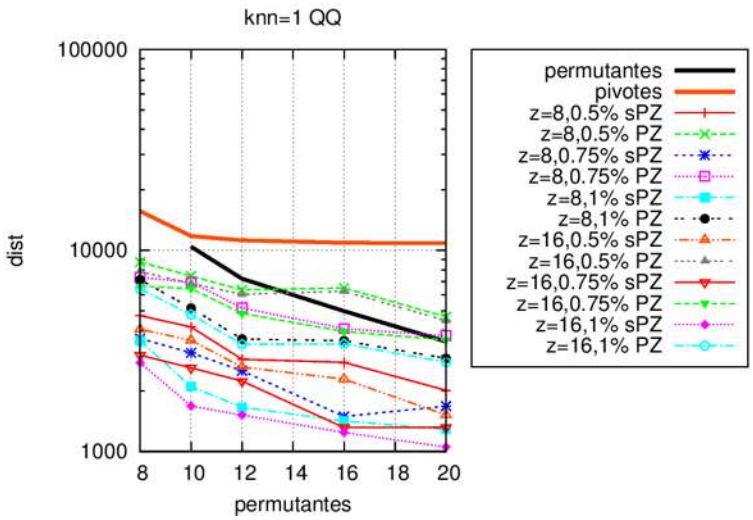

Figura 4.28: Búsqueda en el espacio de histogramas de colores para 1NN usando distancia uniforme.

En la figura 4.28 vemos que usando distancia uniforme se realizan menos cálculos de distancia que usando percentiles, tenemos que los índices que hacen menor cálculo de distancias son los mismos que los de la figura 4.27.

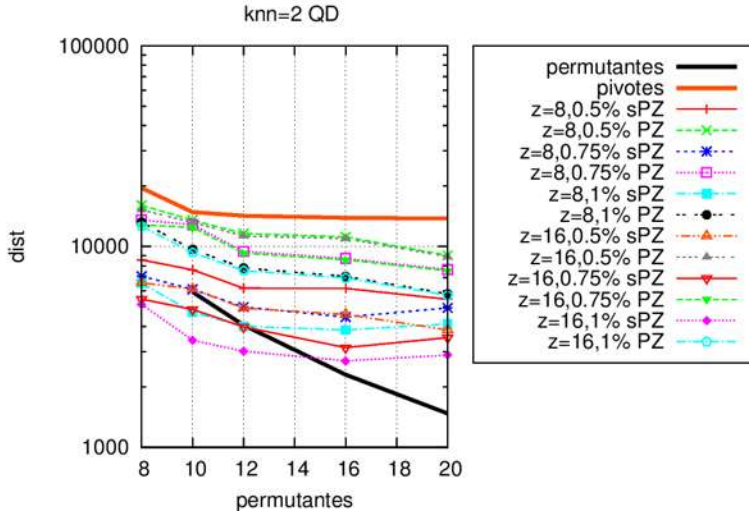

Figura 4.29: Búsqueda en el espacio de histogramas de colores para 2NN usando percentiles.

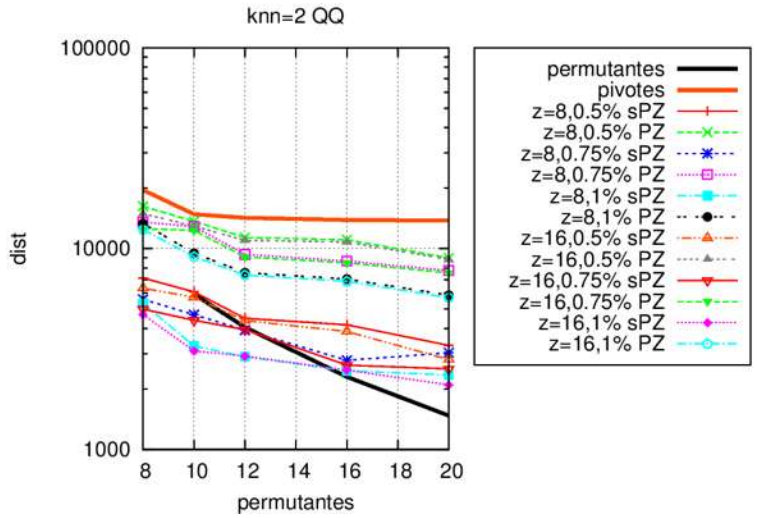

Figura 4.30: Búsqueda en el espacio de histogramas de colores para 2NN usando distancia uniforme.

En las figuras 4.29 y 4.30 vemos que a partir de 16 permutantes el mejor índice fue el de permutantes, mientras que hasta 16 permutantes el mejor índice fue el índice completo con 16 zonas y sPZ

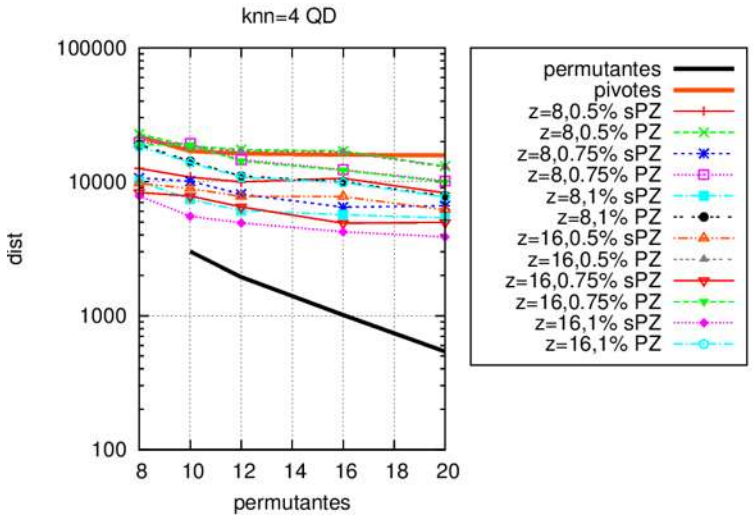

Figura 4.31: Búsqueda en el espacio de histogramas de colores para 4NN usando percentiles.

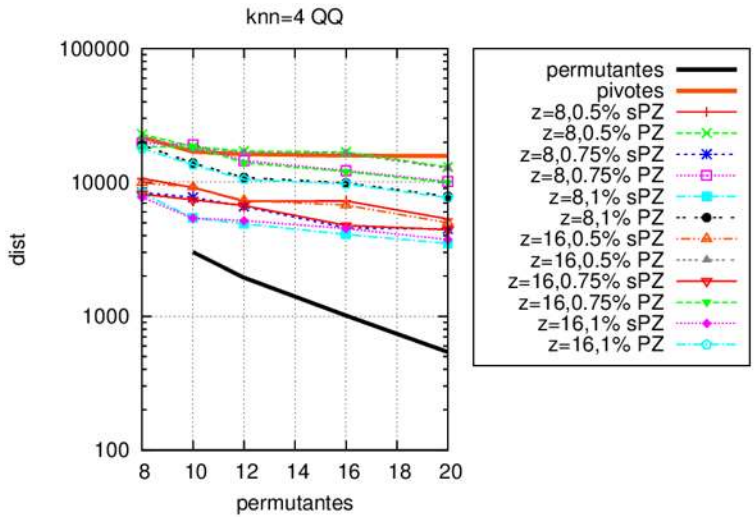

Figura 4.32: Búsqueda en el espacio de histogramas de colores para 4NN usando distancia uniforme.

Para 4NN el mejor índice fue el de permutantes, como puede verse en las figuras 4.31 y 4.32, tambieén es de notarse que en esta base de datos el índice propuesto en esta tesis no tiene ventaja.

Después de la realización de varios experimentos, fueron encontrados los parámentros para los cuáles el índice propuesto es efectivo comparado con el estado del arte.

Estos parámetros, en bases de datos génericas, son:

- **Variando dimensión.** $knn = 2$ $permutantes = 20$ $zonas = 16$
- **Variando permutantes.** $knn = 2$ $dimension = 6$ $zonas = 16$
- **Variando porcentaje de índice.** $knn = 2$ $dimension = 6, 8$ $permutantes = 20, 24, 32$
- **Variando zonas.** $knn = 2, 4$ $dimension = 6, 8$ $permutantes = 20, 28$

Los mejores resultados para bases de datos reales fueron obtenidos para 1NN usando 16 zonas.

Parte V

CONCLUSIONES

Capítulo 5

Conclusiones

En este trabajo se presentó una propuesta para los algoritmos de búsqueda por similaridad en espacios métricos. Cabe mencionar que los algoritmos existentes de este tipo, deterioran su funcionamiento en espacios de dimensión baja, en dimensión alta, su desempeño es competitivo.

En esta tesis se mostró una modificación a la técnica presentada en [FP15], ésta consiste en considerar porciones del índice compuesto por permutaciones e información de zonas, particularmente experimentamos usando el 50 % y 75 % del índice original. De manera experimental encontramos que, bajo ciertos parámetros, los índices parciales al 75 % son una excelente alternativa ya que igualan y en algunos casos reducen el número de distancias calculadas por el índice de permutaciones e información de zonas al 100 %. Es de aclarar que usando sólo el 75 % del índice se logra una reducción en memoria del 25 %.

La aportación de esta tesis consiste en lograr que un índice que trabaja de manera sobresaliente en alta dimensión ahora pueda ser usado en baja dimensión por lo que se vuelve un algoritmo competitivo en este rango de dimensiones. Otro punto a favor de la propuesta es que se logra reducir el tamaño del índice propuesto.

5.1. Trabajo a Futuro

Dado que nuestra propuesta abre una opción viable de los índices basados en permutaciones para trabajar en dimensiones bajas, un trabajo abierto sería reimplementar los algoritmos pivoteros para baja dimensión usando permutaciones.

Bibliografía

[Fig00] K. Figueroa. *Un Algoritmo Eficiente para el Problema de Todos los k Vecinos Más Cercanos en Espacios Métricos*, Tesis maestría, Facultad de Ciencias Físico Matemáticas. Universidad Michoacana de San Nicolás de Hidalgo. Morelia, Michoacán, México, 2000.

[CNBY01] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. *Proximity searching in metric spaces*, ACM Computing Surveys, 33(3):273-321, 2001.

[Fig07] K. Figueroa. *Indexación Efectiva en Espacios Métricos usando Permutaciones*, Tesis doctoral, Departamento de Ciencias de la Computación. Universidad de Chile, Santiago de Chile, 2007.

[FP15] K. Figueroa and R. Paredes. *Boosting Permutation Based Index for Proximity Searching*, Volume 9116 of the series Lecture Notes in Computer Science pp 103-112, 2015.

[CMN01] E. Chávez, J. Marroquín and G. Navarro. *Fixed queries array: A fast and economical data structure for proximity searching*, MultimediaTools and Applications(MTAP),14(2):113 135, 2001.

[ZADB06] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search. The Metric Space Approach*, Springer 2006.

[Nav02] G. Navarro. *Searching in Metric Spaces by Spatial Approximation*, The VLDB Journal 11(1):28-46, 2002.

[Doh04] V. Dohnal. *Indexing Structures for Searching in Metric Spaces*, Doctoral Thesis, Faculty of Informatics. Masaryk University, Brno, Czech Republic, 2004.

[BBK01] C. Böhm, S. Berchtold, and D. A. Keim. *Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases*, ACM Computing Surveys, 33(3): 322 373, 2001.

[BY97] R. Baeza-Yates. *Searching: an algorithmic tour*, In A. Kent and J.Williams, editors, Encyclopedia of Computer Science and Technology, volume 37, pages 331359. Marcel Dekker Inc., 1997.

[BYN98] R. Baeza-Yates and G. Navarro. *Fast approximate string matching in a dictionary*, In Proceedings 5th South American Symposium on String Processing and Information Retrieval (SPIRE98), pages 1422. IEEE CS Press, 1998.

[FKS03] R. Fagin, R. Kumar, and D. Sivakumar. *Comparing top k lists*, SIAM J. Discrete Math., 17(1):134160, 2003.

[CFN09] E. Chávez, K. Figueroa and G. Navarro, *Effective proximity retrieval by ordering permutations*, IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI), 30(9), 16471658, 2009.

[FP14] K. Figueroa and R. Paredes, *An effective permutant selection heuristic for proximity searching in metric spaces*, In: Proc. 6th Mexican Conf. on Pattern Recognition (MCPR14). pp. 102111. LNCS 8495, Springer 2014.

[BK73] W. A. Burkhard and R. M. Keller, *Some approaches to best-match file searching*, Communications of the ACM CACM Homepage archive, Volume 16 Issue 4, Pages 230-236, April 1973.

[BYCMW94] R.Baeza-Yates, W Cunto, U. Manber and S. Wu, *Proximity Matching Using Fixed-Queries Trees*, Proceeding CPM 1994, Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching, Pages 198-212.

[RPE14] P. Ramírez, D. Poaquiza and S. Espinza, *Similar but not the same: How to distinguish ocelot from margay with photographic captures?*, Revista Hippocampus 4 (4), 8-11, 2014.