# Universidad Michoacana de San Nicolás de Hidalgo

Facultad de Ciencias Física Matemáticas
Mat. Luis Manuel Rivera Gutiérrez

## Selección Efectiva de Permutaciones Candidatas para la Búsqueda por Similaridad en Espacios Métricos

Por

**Roxana Leonor Reyes Vera**

Tesis para optar al grado de Licenciada en Ciencias Físico-Matemáticas

Asesor : **Karina Figueroa Mora**
Miembros de la comisión : Fernándo Hernández Hernández
: Luis Valero Elizondo
: Francisco Domínguez Mota
: Jesús Ortiz Bejar

Morelia, Michoacán, México.

Febrero 2020

# Resumen

El creciente uso de la tecnología conlleva a un aumento masivo de datos. La mayoría de éstos son de tipo multimedia, es decir, videos, imágenes, etc. La necesidad de llevar acabo búsquedas en los datos es inminente, además, la principal consulta en este tipo de datos son las búsquedas por similitud; los usuarios estamos interesados en encontrar elementos similares, por ejemplo, tararear una canción y saber a cual corresponde, o pensar un concepto y encontrar una imagen que la relacione.

Una manera de enfrentar el problema es modelarlo como un espacio métrico, donde solo se requiere la base de datos y una función de distancia que *mida* la similitud entre los elementos. Una vez modelado de esta forma es posible crear estructuras de datos llamados *índices* que representen algún tipo de organización en los datos. Actualmente existen muchos tipos de índices.

En este trabajo se presenta una propuesta para mejorar el desempeño de uno de estos índices, el basado en permutaciones. La idea consiste en seleccionar alguno elementos de la base de datos y llamarlos permutantes. El resto de los elementos mide su distancia a los permutantes y los ordena de forma ascendente, a este orden se llamará permutación.

La hipótesis es que elementos iguales deberían tener exactamente la misma permutación, sin embargo, elementos similares deberían tener permutaciones similares.

El estudio que se presenta en esta tesis consiste en analizar el patrón para determinar, para una consulta dada, qué posibles permutaciones podrían formarse[1].

Palabras clave:

[1] Búsquedas, Permutantes, Espacios Métricos, Similitud, Índices.

# Abstract

The recently use of tech in our lives has increased the size of available data, specially those multimedia, i.e. video, images, etc. On the other hand, searching in those kind of data is inminent, moreover, the most valuable searching is by similarity; that is, users are interested in finding the most similar elements in a database, for example, I want to hum a song and retrieve the original one, or to think in a concept and retrieve the image that represent it.

A way to facing the problem is modelling as a metric space, where just a database and a distance function (where similarity can be represented) is requiered. In this model is possible to create a data structures (called index) that represent some kind of organization in data. Nowdays there are many proposals.

In this work, we propose a new index to increase the performance of one of those indexes, the permutation based algorithm. The main idea is to choose some elements in the database and called them permutants. The rest of the elements compute its distances to the permutants and sort them by proximity. This order is called permutation.

The hypothesis is that equal elements has exactly the same permutation, whereas similar objects must have similar permutations.

The main contribution of this work is to study the pattern to decide, for a given query, what are the possible permutations to search them.

# Índice general

# Índice de figuras

# Índice de cuadros

# Capítulo 1

# Introducción

En la actualidad con el desarrollo de la tecnología y el aumento del uso de ella los datos aumentan de manera inconcebible. Una vez generados los datos, lo natural es querer encontrar información sobre ellos. Para estos fines, los datos son almacenados en una base de datos (BD) y si se desea realizar una consulta, ésta debería revisar una base de datos. El escenario no suena mal excepto cuando la BD es masiva. En una BD tradicional que pueda cumplir el modelo relacional, es decir que la información pueda ser separada en dominios específicos, por ejemplo, podemos tener una tabla en la que guardemos por ejemplo nombre, edad , dirección y teléfono, así si una consulta pregunta por el nombre de la persona cuyo teléfono es 534218, la BD debería regresar el nombre de la persona ó de las personas con este número.

Por otro lado, existen datos que no pueden ser separados por dominios, por ejemplo, canciones, fotografías, etc. Las BD que almacenan este tipo de objetos se conocen como Bases de Datos Multimedia (BDM). En este tipo de BD las preguntas interesantes son del tipo: ¿en qué fotografías aparece un perro?, o ¿cuáles son las fotografías más similares a una de muestra?. Nótese que las consultas de tipo exacto no son interesantes, es decir, no es frecuente que se pida encontrar la fotografía que tengo como consulta.

Una manera de resolver cualquier búsqueda en una BD es de manera secuencial, esto es, comparar uno a uno todos los elementos de la base de datos contra la consulta, lo cual no es muy práctico para casos donde la cantidad de datos es muy grande. Una estrategia para evitar revisar toda la base de datos es establecer un *orden* en los datos. De manera que al momento de realizar la búsqueda no revise todos los datos, a esta estructura se le conoce como *índice*. Los índices sirven para navegar rápidamente entre los elementos de la BD.

Por supuesto, una pregunta interesante es cómo se comparan dos objetos de un BDM, ésto depende del dominio de los datos y se escapa de los alcances de esta tesis. Para la comparación entre dos objetos asumiremos que tenemos una función de distancia entre ellos. Con esta suposición

podemos hacer un mapeo del problema a un espacio métrico y generar índices que no dependan de la estructura de los datos conociendo únicamente la función de distancia entre ellos.

## 1.1. Aplicaciones

Existen muchas aplicaciones que hacen uso de la búsqueda por similaridad, algunos ejemplos son: recuperación de información, biología computacional, reconocimiento de patrones, y más. Se describen algunas de estas aplicaciones a continuación [CNBYM99].

1. El proceso clásico de reconocimiento de patrones tiene tres estados principales: segmentación, extracción de características, y clasificación. La segmentación consiste en la extracción de objetos individuales de los datos digitalizados. La extracción de las características consiste en hacer un mapeo del objeto digital sobre un espacio vectorial (usualmente de dimensión alta), donde cada coordenada representa el grado de presencia de cierta característica en el objeto. La clasificación consiste en asignar cada objeto a uno fuera del conjunto predefinido por las clases de objetos. Este modelo abarca patrones concretos de reconocimiento a tareas tales como reconocimiento de discursos, identificación de hablantes, pareo de firmas, reconocimiento de escritura, identificación biométrica, etc.

   La extracción de características convierte la clasificación original en un problema geométrico. Los objetos en la misma clase tienden a ser cercanos especialmente si las características son seleccionadas apropiadamente. Las técnicas de clasificación más populares, tales como máquinas de soporte vectorial, redes neuronales o $K$ vecinos más cercanos, son definidas en términos geométricos. Entre estos los clasificadores de los $K$ vecinos más cercanos ($K-NN$) son atractivos porque el entrenamiento es implícito.

2. Biología computacional. En esta área se utilizan secuencias (una secuencia de ADN, secuencia de neuclótidos o secuencia genética es una sucesión de letras representando una estructura primaria de una molécula real o hipótetica de ADN o banda, con la capacidad de transportar información) que definen objetos, por ejemplo proteínas, ADN, etc. Si las secuencias son modeladas como texto, el problema es llevado a búsquedas de cadenas de texto similares. Dependiendo de lo que se quiera evaluar al comparar secuencias, se utilizan distintas medidas de similitud entre cadenas, por ejemplo, una donde se pueda considerar probabilidades de mutaciones.

   Otras aplicaciones son los buscadores de internet, búsquedas de imágenes similares, reconocimiento facial, reconocimiento de huellas digitales; éstas son aplicaciones específicas que no se tratarán en esté trabajo, sin embargo es relevante mencionarlas por la importancia que representan en la actualidad.

## 1.2. Definición del Problema

El problema consiste en que dada una base de datos que contenga tipos como imágenes, documentos u otros datos multimedia, y dada una función de comparación de los elementos, es posible modelar esto como un espacio métrico. Posteriormente, se diseñará y se creará un índice que permita resolver el problema de la búsqueda por similaridad. Un espacio métrico está definido como una tupla $(\mathbb{X}, d)$, donde $\mathbb{X}$ representa el universo de objetos válidos y $d$ la función de distancia que denota que tan similar son los elementos. Por ejemplo, la aproximación de los $K-NN$ traslada el problema de clasificación a un problema de búsqueda por proximidad (encontrar los $K$ objetos cercanos representativos a nuevos elementos dados) en un espacio característico de dimensión alta.

Desafortunadamente los métodos actuales para la búsqueda por proximidad sufren de lo que es llamada la maldición de la dimensión [CNBYM01a]. Cualquier método para búsqueda por proximidad, no importa qué tan bien funcione en dimensiones bajas, termina escaneando el conjunto completo de objetos en dimensiones altas. Las técnicas de reducción de dimensiones son efectivas y bien conocidas, pero poseen un extra por encima del sistema cuando los datos son intrínsecamente de dimensiones altas, y la precisión de la clasificación caerá si las distancias en los espacios de dimensiones bajas no son bien preservados. Esto es, los datos pueden perder clasificación cuando se usa la aproximación $K-NN$ en un mapeo de espacio distorsionado de las distancias originales.

En todas las aplicaciones de ejemplo, el modelo general es de una caja negra de objetos (i.e la base de datos) que puede ser preprocesado para responder consultas de proximidad contra nuevos objetos. La única herramienta para obtener información de los objetos es mediante el cómputo de las distancias a través de los objetos.

## 1.3. Conceptos Básicos

Algunos conceptos básicos que serán de utilidad a lo largo de está lectura se mencionan enseguida:

Formalmente, un espacio métrico es un par $(\mathbb{X}, \delta)$ el cual está compuesto de un conjunto $\mathbb{X}$ que se denomina universo, cuyos elementos llamaremos elementos u objetos, y una función de distancia. La función de distancia $\delta$ debe cumplir las siguientes características:

1. Positividad estricta: $\delta(x, y) > 0 \ \forall x, y$ con $x \neq y$
2. Reflexividad: $\delta(x, y) = 0$ si y sólo si $x = y$
3. Simetría: $\delta(x, y) = \delta(y, x)$
4. Desigualdad triangular: $\delta(x, z) \leq \delta(x, y) + \delta(y, z)$

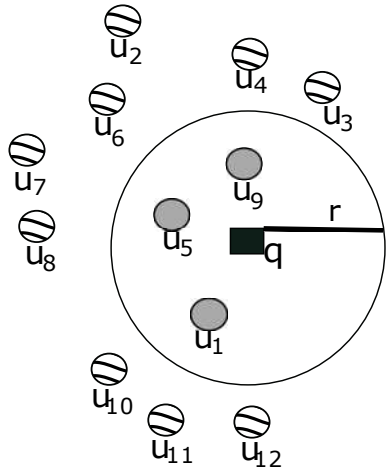



Figura 1.1: Tipo de consulta, consulta por rango.

Note que la desigualdad triangular es la única propiedad que permite descartar elementos sin ser comparados directamente contra la consulta. El detalle de este punto se explicará en las secciones siguientes.

Usaremos un conjunto finito de objetos $\mathbb{U} \subseteq X$ de tamaño $n = |\mathbb{U}|$. Donde al conjunto de elementos de interés $\mathtt{U}$ lo llamaremos diccionario, base de datos o simplemente conjunto de elementos. El término distancia se usará para referirnos a la métrica.

### 1.3.1. Consultas por proximidad

Hay dos tipos de consultas por similaridad, donde $q \in \mathbb{X}$ es el elemento de la consulta:

1. Consulta de rango. Recupera los elementos en un radio r, es decir, $(q,r)_d = \{u \in \mathbb{U} \mid d(q,r) \leq r\}$
2. Consulta de los $K$ vecinos más cercanos $KNN(q)_d$. Recupera los $K$ elementos de $\mathbb{U}$ más cercanos a $q$. Es decir, sea $\mathtt{A} \subseteq \mathtt{U}$ tal que $\forall u \in \mathtt{A}, \forall v \in \mathtt{U} - \mathtt{A}$, $d(q,u) \leq d(q,v), K = |\mathtt{A}|$

El caso más común en muchas aplicaciones es cuando se quiere recuperar el vecino más cercano (*NN* por su sigla en inglés Nearest Neighbor) esto es cuando $K = 1$. Gráficamente, estas consultas se muestran en la figura 1.3.1.

Tipos de consultas por proximidad en espacios métricos. Las figuras muestran los tipos de consulta vea que la figura 1.3.1 ejemplifica la búsqueda de los 2 vecinos más cercanos, mientras que la figura 1.3.1 expone una consulta por rango.

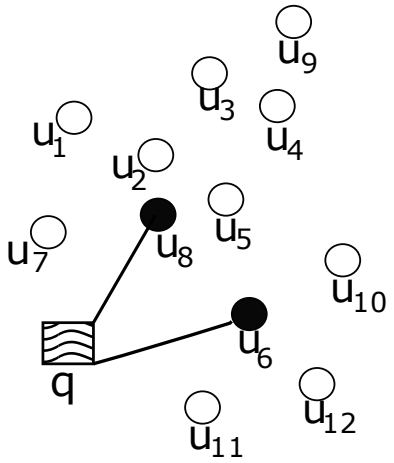



Figura 1.2: Tipo de consulta, consulta de los vecinos más cercanos.

### 1.3.2. Tipos de distancias

En los espacios métricos un caso particular son los espacios vectoriales $m$-dimensionales, en el cual los objetos son identificados con $m$ coordenadas de valores reales $(x_1, x_2, ..., x_m)$. Sean $u$ y $q$ dos vectores la distancia puede ser la métrica $L_p$ (también conocida como la métrica de Minkowski de orden $p$ ) definida como:

$$L_p = d_p(u,q) = \sqrt[p]{\sum_{i=1} |u_i - q_i|^p}$$

Cuyos casos especiales son:

Manhattan $L_1 = d_1(u,q) = \sum_{i=1}^{m} |u_i - q_i|$

Euclidiana $L_2 = d_2(u,q) = \sqrt{\sum_{i=1}^{m} |u_i - q_i|^2}$

Máxima $L_\infty = d_\infty(u,q) = \text{máx}_{i=1,...m} |u_i - q_i|$

### 1.3.3. Concepto de Índice Métrico

Los algoritmos que se emplean para realizar búsquedas por proximidad generalmente utilizan un índice para no comparar toda la base de datos. Estos algoritmos suelen tener dos partes, la de preprocesamiento y la de consulta. La base de datos se proyecta a un nuevo espacio en el que es posible crear un índice (fase de preprocesamiento). El índice se crea una vez y se almacena. En la parte de la consulta se recorre el índice para conocer la *lista de candidatos* que serán comparados directamente con la consulta (*comparaciones externas*) se descartan elementos de manera segura. Cabe mencionar que durante el recorrido del índice, es posible que se deban hacer algunas comparaciones por lo que, a éstas las llamaremos *comparaciones internas*.

Una proyección de un espacio métrico es proyectar el espacio original a un espacio vectorial mediante una función $\Phi$, cada elemento del espacio original es representado por un punto en el nuevo espacio. Los dos espacios están relacionados por la función $\Phi$ y la distancia del nuevo espacio $D(\Phi(x), \Phi(y))$, donde $x, y, \in \mathbb{X}$.

Si $D(\Phi(x), \Phi(y)) \leq d(x, y))$, entonces la búsqueda es posible en el espacio proyectado con el radio original, al ser posible tener elementos irrelevantes en este espacio será denominado como *lista de candidatos*, por lo tanto deberá ser verificada en el espacio original para obtener una *lista de resultados*.

# Capítulo 2

# Trabajo previo

Este capítulo describe los algoritmos existentes para la búsqueda por proximidad en espacios métricos, los cuales hacen uso de índices durante las consultas para reducir los costos en la búsqueda. Los índices son estructuras de datos usadas para recorrer los datos en cierto orden y tomar decisiones respecto a los elementos en la base de datos, es decir si los compara o los descarta. También en este capítulo se darán conceptos básicos para este trabajo.

## 2.1. Conceptos básicos

Formalmente, el problema de búsqueda por proximidad puede ser escrito como sigue:

Existe un universo $\mathbb{X}$ de objetos, y una función de distancia $d : \mathbb{X} \times \mathbb{X} \to \mathbb{R}^+ \cup \{0\}$ definida entre ellos. La distancia satisface los axiomas que hacen del conjunto un espacio métrico

- reflexivo: $d(x, y) = 0$ si y sólo si $x = y$
- estrictamente positivo: $x \neq y \Longrightarrow d(x, y) > 0$,
- simetría: $d(x, y) = d(y, x)$
- la desigualdad del triángulo: $d(x, z) \leq d(x, y) + d(y, z)$.

Se asume que esta distancia es cara computacionalmente (piense, por ejemplo, en comparar dos documentos). Tenemos una *base de datos* finita $\mathbb{U} \subseteq \mathbb{X}$, de tamaño $n$. La meta es preprocesar la base datos $\mathbb{U}$ para responder de manera eficiente (i.e. con el mínimo número de distancias posibles) consultas de rango y de $K$ vecinos más cercanos ($K - NN$). Las consultas de rango son expresadas como $(q, r)$ (un punto en $\mathbb{X}$ y una tolerancia de radio), la cual regresa todos los

puntos de la base de datos con distancia $r$, esto es $u \in \mathbb{U}$, $d(u,q) \leq r$. Por otro lado las consultas de los $K$ vecinos más cercanos regresa los elementos de $\mathbb{U}$ cercanos a $q$.

En [CNBYM01b],[HS00],[PZB06], varias propuestas de índices para espacios métricos son inspeccionadas y explicadas a detalle. En este capítulo explicaremos solo las más importantes.

## 2.2. Indexación

Todos los algoritmos de búsqueda en espacios métricos confían en un índice, esto es, una estructura de datos que mantiene cierta información en la base de datos de manera que guarda algunas evaluaciones de distancias en el tiempo de búsqueda. Existen dos tipos principales de organización de datos, que se cubrirán enseguida.

### 2.2.1. Esquemas de pivotes

Un *pivote* es un elemento seleccionado de la base de datos, cuyas distancias a *algunos* otros elementos fue precalculado (es decir, se evalúan todas las distancias de $u$ hacia cada $p$) y fue almacenado en un índice. Imagine que tenemos precalculado $d(p,u)$ para algún pivote $p$, donde $u \in \mathbb{U}$.

En tiempo de búsqueda, para una consulta de rango con radio $r$, computamos $d(p,q)$. Entonces para la desigualdad del triángulo $d(q,u) \geq |d(p,q) - d(p,u)|$, esto es, si $|d(p,q) - d(p,u)| > r$ sabemos que $d(q,u) > r$, por lo que $u$ puede ser filtrada fuera sin necesidad de calcular la distancia $d(q,u)$.

El esquema de pivotes más básico escoge $k$ pivotes $p_1, ..., p_k$ y computa todas las distancias $d(p_i,u)$, $u \in \mathbb{U}$, dentro de la tabla de $kn$ entradas. Entonces en tiempo de consulta, todas las $k$ distancias $d(p_i,q)$ son computadas y cada elemento $u$ tal que $D(q,u) = max_{i=1..k}|d(p_i,q) - d(p_i,u)| > r$ es descartado. Finalmente, $q$ es comparado contra los elementos que no fue posible descartar.

A medida que $k$ se incrementa, tenemos que pagar más comparaciones contra los pivotes, pero $D(q,u)$ se vuelve más cercano a $d(q,u)$, esto haría que más elementos pudieran ser descartados. Se puede mostrar que existe un número óptimo de pivotes $k^*$, que crece rápido con la dimensión y se vuelven rápidamente inalcanzables por las limitaciones de memoria. En los espacios métricos más fáciles, se usan tantos pivotes como la memoria lo permita. Existen diversas variaciones sobre esta idea básica, incluyendo diferentes maneras de organizar la tabla de las $kn$ entradas para reducir el tiempo extra de CPU.

Muchas estructuras de datos como los árboles son construidos de manera similar al concepto que usan los pivotes. En la mayoría de ellos, un pivote $p$ es elegido como la raíz de un árbol, y sus subárboles corresponden al rango de los valores de $d(p,u)$, siendo estructurados

recursivamente. En algunos casos las distancias exactas $d(p,u)$ son continuas, y el rango puede ser inferido de los elementos $u$ del subárbol. Aunque esto reduce la precisión del índice, el árbol usualmente toma $O(n)$ de espacio en lugar de $O(kn)$ necesario con $k$ pivotes. Además cada nodo interno es un pivote local (el cual conoce distancias a los elementos del subárbol únicamente), entonces automáticamente tenemos muchos pivotes (aunque local y con muchos datos). Finalmente los árboles pueden ser atravesados usando un tiempo extra sublineal.

Existen diferentes tipos de variantes de árboles de acuerdo a la función del mismo, el modo en que el rango de distancias es elegido (tratando de balancear el árbol o no), como los pivotes son locales (diferentes nodos pueden compartir pivotes, los cuales no pertenecen más al subárbol). Es muy poco conocido acerca de cuál es el mejor tipo de árbol, por ejemplo, la regla de oro de preferir árboles balanceados, la cual funciona bien para búsquedas exactas, se vuelve en una elección más apropiada contra árboles no balanceados cuando la dimensión incrementa. Para cada dimensión alta una buena estructura de datos es casi una lista ligada.

### 2.2.2. Esquemas de partición local

Otro esquema se construye partiendo de la idea de dividir la base de datos dentro de grupos espaciales compactos, esto quiere decir que los elementos en cada grupo son cercanos unos a otros. Un representativo de cada grupo es elegido, éste se compara contra $q$ para poder descartar al grupo completo sin más comparaciones. Usualmente estos esquemas son jerárquicos, y los grupos son divididos recursivamente en subgrupos.

Existen dos maneras principales de definir estos grupos. Uno puede definir "centros" con radio de cobertura, entonces todos los elementos en este grupo están dentro de la distancia del radio de cobertura al centro. Si un grupo tiene centro $c$ y radio de cobertura $r_c$ entonces, si $d(q,c) > r + r_c$, el grupo completo puede ser eliminado. La figura geométrica de este esquema corresponde a una bola centrada alrededor de $c$.

En la segunda aproximación un conjunto de centros es elegido y el resto de los elementos son adherido al grupo de su centro más cercano. En tiempo de consulta, si $q$ es cercano a $c_i$ y $d(q,c_j) - r > d(q,c_i) + r$, entonces podemos descartar el grupo completo de $c_j$. La figura geométrica en esta aproximación corresponde al domino de Dirichlet del espacio (una generalización del diagrama de Voronoi para espacios métricos), sin sobreponer entre los grupos.

## 2.3. Clasificación de algoritmos

A continuación se hará una explicación más detallada de los algoritmos del estado del arte.

### 2.3.1. Algoritmos basados en pivotes

Este tipo de algoritmos consisten en elegir un conjunto $\mathbb{P} = \{p_1, p_2, ...p_k\} \subset \mathbb{U}$, de tamaño $k = |\mathbb{P}|$ de pivotes. Donde para cada elemento de la base de datos se precalculan las distancias hacia los elementos del conjunto $\mathbb{P}$, cuyo conjunto de distancias $\{d(p_1,u), d(p_2,u), ..., (p_k,u)\} \forall u \in \mathbb{U}$, conformará el índice.

Dada una consulta $q$, se calcula $d(p,q)$ para todos los $p_i \in \mathbb{P}$ (comparaciones internas). Con esto se pueden acotar las distancias entre $q$ y cualquier $u \in \mathbb{U}$ tal como se demuestra en el siguiente lema.

**Lema 1**. Dados tres objetos $q \in \mathbb{X}, u \in \mathbb{U}, p \in \mathbb{P}$, sabemos que $|d(q,p) - d(p,u)| \leq d(q,u) \leq d(q,p) + d(p,u)$.

**Demostración:** El límite superior se obtiene directamente de la desigualdad triangular $d(q,u) \leq d(p,q) + d(p,u)$, 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: $d(p,u) - d(p,q) \leq d(q,u)$, $d(p,q) - d(p,u) \leq d(u,q)$, combinando estas desigualdades y usando la simetría, obtenemos que: $|d(p,u) - d(p,q)| \leq d(u,q)$.

Los algoritmos que usan este tipo de índices lo hacen de la siguiente manera: durante una consulta por rango, usando el lema 1, se pueden descartar todos los elementos de la base de datos $u \in U$ tales que $\exists p \in \mathbb{P}$, $|d(q,u) - d(p,u)| > r$. Los elementos no descartados se comparan directamente contra la consulta (comparaciones externas).

Para ver surveys sobre algoritmos basados en pivotes recomendamos [PZB06], [HS00], [CNBYM01a]. Por otro lado, algunos algoritmos interesantes con pivotes pueden ser consultados en [BYCMW94], [CMN99], por mencionar algunos.

### 2.3.2. Algoritmos basados en particiones compactas

Dividen el espacio en zonas tan compactas como sea posible. Selecciona un conjunto de objetos $p_1, p_2, ..., p_k \subset \mathbb{U}$, dividiendo el espacio, distribuyendo las $k$ zonas pertenecientes a los $p_i$, donde los $p_i$ son llamados "centros" de su zona. El índice esta compuesto por los centros. Un criterio de distribución, entre varios, es asignar cada $u$ a la zona cuyo $p_i$ sea el más cercano.

En este tipo de algoritmos se dividen de acuerdo a su procedimiento de búsqueda: los que usan radio de cobertura $r_c$, los que usan hiperplanos y los que usan ambos. El radio de cobertura $r_c$ es la distancia máxima del centro de la zona a los elementos en ella, mientras que un hiperplano es la frontera entre zonas cuando cada elemento es asignado a su centro más cercano.

**Lema 2**. Dados tres objetos $u, c \in \mathbb{U}, q \in \mathbb{X}$ si $c$ es el centro de la zona a la que pertenece $u$, con radio de cobertura $r_c$, sabemos que, $d(q,u)$ está acotada: máx $\{d(c,r) - r_c, 0\} \leq d(q,u) \leq$

$d(p,q) + r_c$.

**Demostración**. Ambos límites los obtenemos de la desigualdad triangular y de la cota superior: $d(p,q) \leq d(p,u) + d(u,q)$, $d(p,u) \leq r_c$ usando estas dos ecuaciones tenemos: $d(p,q) - d(q,u) \leq d(p,u) \leq r_c$, esto implica que: $d(p,q) - r_c \leq d(q,u)$. Sabemos que $d(q,u) \geq 0$, por lo tanto el límite inferior es máx$\{d(q,p) - r_c, 0\} \leq d(q,u)$. El límite superior lo obtenemos de la desigualdad triangular $d(q,u) \leq d(q,p) + d(p,u) \leq d(q,p) + r_c$.

**Lema 3**. Sea $u \in \mathbb{U}$ el objeto más cercano a $p_1$ que a $p_2$ o que equidista, es decir, $d(p_1,u) \leq d(p_2,u)$. Dados $d(q,p_1)$ y $d(q,p_2)$, podemos establecer la cota máx$\{\frac{d(q,p_1)-d(q,p_2)}{2}, 0\} \leq d(q,u)$.

**Demostración**. De la desigualdad triangular sabemos que $d(q,p_1) \leq d(q,u) + d(p_1,u)$, lo cual lleva a $d(q,p_1) - d(q,u) \leq d(p_1,u)$. Del mismo modo tenemos que $d(p_2,u) \leq d(q,p_2) + d(q,u)$, y por hipótesis $d(p_1,u) \leq d(p_2,u)$.

Combinando en las ecuaciones anteriores obtenemos $d(q,p_1) - d(q,u) \leq d(p_1,u) \leq d(p_2,u) \leq d(q,p_2) + d(q,u)$, $d(q,p_1) - d(q,p_2) \leq 2d(q,u)$.

Finalmente, sabemos que $d(q,u) \geq 0$, así la cota inferior es: máx$\{\frac{d(q,p_1)-d(q,p_2)}{2}, 0\} \leq d(q,u)$.

En una consulta de rango $(q,r)_d$ esta familia de algoritmos descarta aquellas zonas de centro $p$ tales que $d(p,q) - r_c > r$, pues usando el *lema 2* sabemos que cualquier elemento $u$ en esa zona no cumple con $d(q,u) > r$. En las zonas no descartadas se repite el proceso hasta que se llega a una zona sin divisiones.

Algunos algoritmos característicos de esta técnica se pueden leer en [CN00], [CPZ97], [Ben79], [DN87], [Uhl91], por mencionar algunos.

### 2.3.3. Algoritmos basado en permutaciones

Este tipo de algoritmo fue propuesto en [CFN05], donde cada elemento forma su permutación sobre un conjunto de elementos (a los que ahora llamaremos *permutantes*) y el orden de la permutación de este conjunto es dado por la distancia creciente al elemento. La ventaja es que es posible obtener un alto porcentaje de respuesta correcta revisando muy pocos elementos.

Seleccionamos un conjunto $\mathbb{P} = \{p_1, ..., p_k\} \subset U$, llamados permutantes, seleccionados en principio de manera aleatoria, se calcula su preorden $\leq_u$ en $\mathbb{P}$ y se forma su permutación $\Pi_u$. Si dos elemento $u$ y $v$ son iguales sus permutaciones serán iguales, se esperaría que si dos elementos son parecidos, sus permutaciones sean parecidas.

Al momento de la consulta se calcula $\Pi_q$ (permutación de la consulta) para una consulta $q$ usando $\leq_q$ sobre el conjunto $\mathbb{P}$, ordena los $\Pi_u$, $u \in U$ de mayor a menor similaridad con respecto a

| Elemento | Permutaciones ($P_i$) | $S_p$ |
|---|---|---|
| $u_6$ | $P_{4231}$ | 0 |
| $u_{10}$ | $P_{4213}$ | 2 |
| $u_{11}$ | $P_{4132}$ | 2 |
| $u_{13}$ | $P_{1423}$ | 2 |
| $u_{14}$ | $P_{1423}$ | 2 |
| $u_{15}$ | $P_{1423}$ | 2 |
| $u_5$ | $P_{1432}$ | 4 |
| $u_7$ | $P_{4312}$ | 6 |
| $u_8$ | $P_{1243}$ | 6 |
| $u_{12}$ | $P_{4231}$ | 6 |
| $u_9$ | $P_{2143}$ | 8 |

Cuadro 2.1: Permutaciones ordenadas respecto a la consulta.

$\Pi_q$.

Como medida de similaridad entre permutaciones se usa Spearman Rho [DG77], se denota por $S_p(\Pi_q, \Pi_u)$, $S_p$ es la suma de los cuadrados de las diferencias en las posiciones relativas de cada elemento en las dos permutaciones. Para cada $p_i \in \mathbb{P}$ se calcula su posición en $\Pi_u$ y $\Pi_q$, esto es $\Pi_u^{-1}(p_i)$ y $\Pi_q^{-1}(p_i)$, es decir:

$$S_p(\Pi_q, \Pi_u) = \sum_{i=1}^{k} |\Pi_u^{-1}(p_i) - \Pi_q^{-1}(p_i)|^2 \tag{2.1}$$

En la figura 2.1 y 2.2 se presenta un ejemplo de las 2 fases de este algoritmo. La figura 2.1 muestra la fase de construcción de las permutaciones, y la figura 2.2 muestra la consulta y su permutación. Note que la tabla 2.1 mostrada presenta la comparación entre cada permutación de la base de datos contra la permutación de la consulta.

#### PP-Index

El PP-Index [LI09] representa a cada objeto indexado con una muy pequeña parte de la permutación: el prefijo. La estructura de datos del PP-Index consiste de un árbol de prefijos que se mantiene en la memoria principal, indexando los prefijos de las permutaciones, y los datos almacenados que se mantienen en disco.

### 2.3.4. Estructuras de Datos

Dada una colección de objetos $\mathbb{U}$ para ser indexados, y una medida de similaridad $d$, el PP-Index genera las permutaciones, y con un parámetro $l$ decide el tamaño el prefijo (encontrado

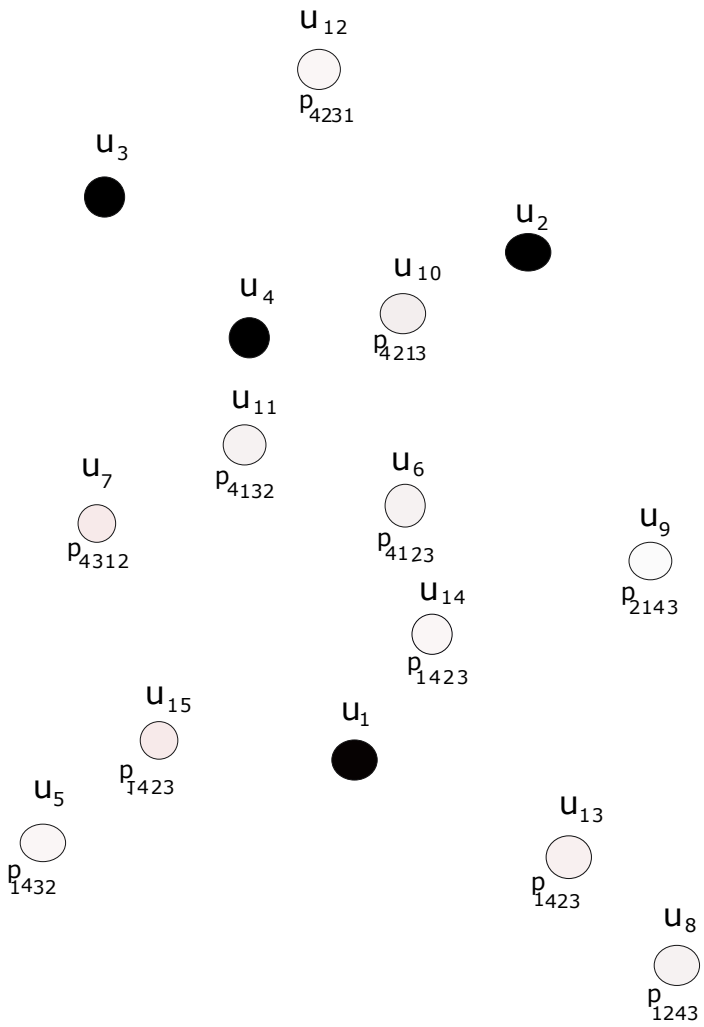



Figura 2.1: Ejemplo de la fase 1 del algoritmo basado en permutaciones.

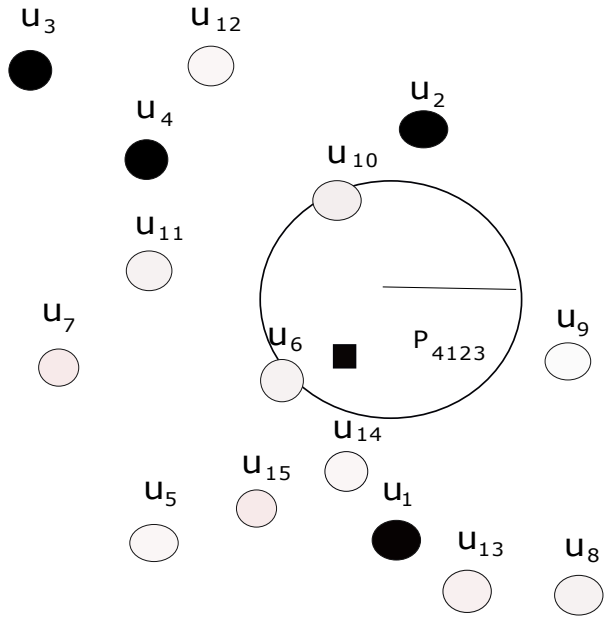



Figura 2.2: Ejemplo de la fase 2 del algoritmo basado en permutaciones.

experimentalmente). Cada objeto $o_i \in \mathbb{U}$ es representado por una permutacion de prefijo $w_{o_i}$, es decir los primeros $l$ elementos de la permutación, i.e, $w_{o_i} = p^l_{o_i}$.

El árbol de prefijos del PP-Index es construido con todas las permutaciones de prefijos generadas por el objeto indexado. La hoja al final de cada camino es el prefijo de una permutación $w$.

### 2.3.5. Construyendo un índice

El algoritmo de indexación primero inicializa con un árbol vacío. El algoritmo toma el prefijo de la permutación de un objeto $o_i \in \mathbb{U}$ y posteriormente agrega el resto uno a uno.

### 2.3.6. Función de búsqueda

La función de búsqueda esta diseñada para usar el índice para responder eficientemente la consulta del $K$ vecino más cercano. La estrategia consiste en un subárbol del árbol de prefijos que identifica un número específico de objetos candidatos, todos representados por prefijos de permutaciones que coinciden con el prefijo de la permutación de la consulta. Una consulta $K$-NN esta compuesta del objeto consulta $q$, el valor de $K$, y el valor de $z$, indicando el número mínimo de objetos candidatos entre los cuales el $K$ vecino más cercano tiene que ser seleccionado.

La idea general para descender por el árbol es usando el mismo permutante con el

que inició la permutación de la consulta. Ese subárbol contiene a todos los posibles elementos candidatos.

# Capítulo 3

# Propuesta

En este capítulo se describirá la propuesta que se tiene para mejorar las búsquedas por similaridad en espacios métricos. En el capítulo 2 se describieron algunos algoritmos que se utilizan para la búsquedas por similaridad. Esta propuesta puede ser vista como una mejora al algoritmo PPIndex.

## 3.1. Idea principal

Los algoritmos basados en permutaciones (como se vió en el capítulo 2) tienen para cada permutación de la base de datos, un tiempo de consulta. Se obtiene la permutación de la consulta y, la idea principal es que los vecinos más cercanos deberían tener permutaciones similares. Es decir, de todas las permutaciones, se llaman candidatas a las más parecidas.

En este trabajo se muestra un algoritmo eficiente para encontrar esas permutaciones candidatas sin tener que comparar la permutación de la consulta contra todas las de la base de datos y después ordenarlas para conocer aquellas más parecidas.

En la sección 2.3.3 se presentó formalmente la idea de los algoritmos basados en permutaciones. Como se pudo apreciar en el ejemplo de la figura 2.1 y la figura 2.2, se tienen 2 fases. La primera, se deben construir las permutaciones. En la segunda etapa, la de consulta, primero se calcula la permutación de la consulta (costo $O(k)$ distancias internas), posteriormente se deben encontrar las permutaciones mas parecidas a la de la consulta. Para esto, se comparan todas las permutaciones de la base de datos, es decir, para toda $u \in \mathbb{U}$, se evalúa $S_p(\Pi_u, \Pi_q)$, aquellos elementos con las permutaciones mas parecidas serán los elementos candidatos a compararse directamente contra la consulta (distancias externas).

Por otro lado, el *PP-Index* mostrado en la sección 2.3.3 crea un árbol con las permutaciones de la base de datos. Dicha estructura es recorrida de acuerdo a la permutación de la consulta.

Por ejemplo,

## 3.2. Propuesta

La propuesta de esta tesis consiste en definir un criterio para saber cuáles permutaciones serían candidatas sin tener que comparar todas contra la consulta.

Para explicar la propuesta comenzaremos con un ejemplo. Suponga que se conocen las distancias de la consulta a los permutantes, es decir $D_q = \{1,1,4,4\}$. La permutación de la consulta sería $\Pi_q = 1,2,3,4$, puesto que la distancia hacia los permutantes 1 y 2 son iguales, y sea $\Pi_u = 2,1,3,4$ la permutación al vecino más cercano, sus distancias serían $D_u = \{1,0,4,4\}$, si el radio de consulta fuera 1, significaría que efectivamente la consulta lo vería como vecino más cercano. De este ejemplo tenemos los siguiente:

- Las permutaciones que inician con el mismo permutante que la consulta deberían ser candidatos. Esto es debido a que ambos tienen muy cercano a ese permutante.
- Las permutaciones podrían iniciar con otro permutante "cercano" a la consulta. Como es el caso del ejemplo anterior.

Con esta consideración se propone usar parámetro denominado *holgura* que denotaremos como $\varphi$ para decidir si un permutante debería ser considerado como *permutante de inicio*. Formalmente:

Sea $q$ una consulta con $\Pi_q$ como su permutación, y $D_q$ como el vector de distancias de $q$ hacia cada $p_i \in \mathbb{P}$, es decir, $D_q = \{d(q,p_1), d(q,p_2), \ldots, d(q,p_k)\}$.

$$|d(q,p_i) - d(q,p_j)| \leq \varphi \tag{3.1}$$

Si para dos permutantes $p_i$ y $p_j$, la diferencia de distancia hacia $q$ es menor o igual que $\varphi$, entonces las permutaciones que inician con el permutante $p_j$ también deberían estar entre las permutaciones candidatas.

Con este parámetro será posible descartar aquellas permutaciones que no comiencen con alguno de esos permutantes. El proceso será descrito paso a paso mediante un ejemplo. Considérese el espacio representado por los puntos de la figura 3.1, donde los puntos representan los objetos de la base de datos en el espacio.

El proceso se lleva a cabo de la siguiente manera, de todos los objetos se eligen de manera aleatoria los objetos que ahora serán los permutantes, donde estos elementos que ahora son los permutantes permitiran hacer el calculo de las distancias del resto de los elementos y la

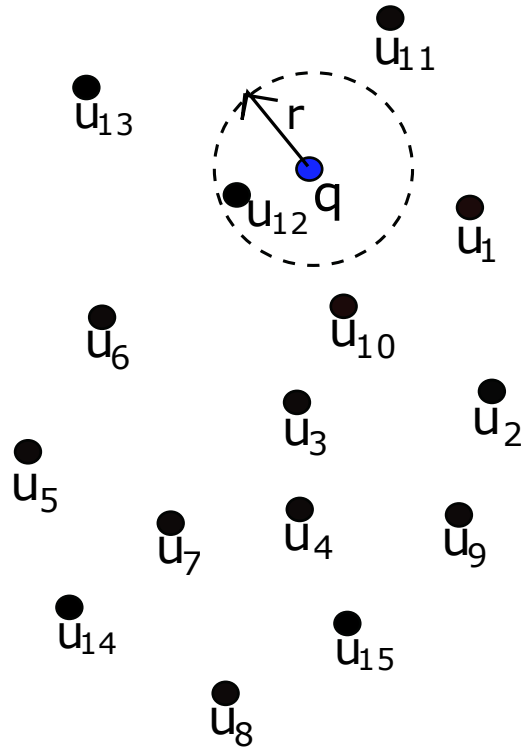



Figura 3.1: Espacio de ejemplo. Los círculos negros representan algún tipo de objeto con una función de distancia determinada.

consulta a dichos permutantes. En la figura 3.2 se muestran los permutantes seleccionados de manera aleatoria en la base de datos. En particular son los denominados $u_1, u_2, u_3, u_9$.

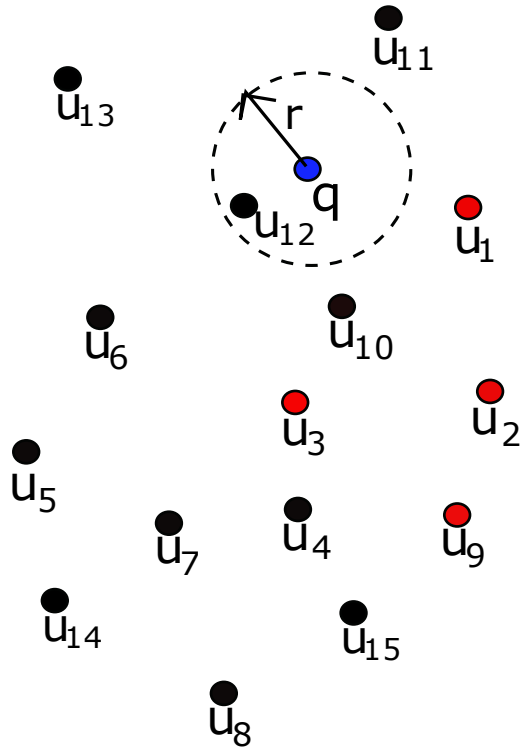



Figura 3.2: Permutantes seleccionados de manera aleatoria del espacio métrico.

En la figura 3.3 se muestra que cada objeto de la base de datos computa su distancia hacia los permutantes. Es decir, cada objeto calcula su distancia hacia los permutantes y crea la permutación de cada objeto. Esto permitirá conocer las permutaciones de cada uno. Para este ejemplo, las permutaciones se muestran en la tabla 3.1.

En la figura 3.4 se muestra una consulta cuyo rango de búsqueda es $r$. Puede observar que la línea punteada muestra el área delimitada por el radio $r$. Note que la consulta realiza el cómputo de las distancias hacia los permutantes, los ordena por cercanía y obtiene su permutación. Para este ejemplo es: $\Pi_q = u_1, u_3, u_2, u_9$, donde $D_q = \{11, 16, 12, 17\}$.

El algoritmo original PBA habría hecho la comparación de todas las permutaciones de la base de datos contra la permutación de la consulta. En la tabla 3.2 se muestra el valor del spearman rho (descrito en la ecuación 2.1). Esta tabla sólo es informativa, y se puede apreciar que de acuerdo a la propuesta de este trabajo, deberíamos calcular las diferencias de distancias como se muestran en las ecuaciones 3.2, 3.3, 3.4.

$$|11 - 12| = 1 \leq \varphi \tag{3.2}$$

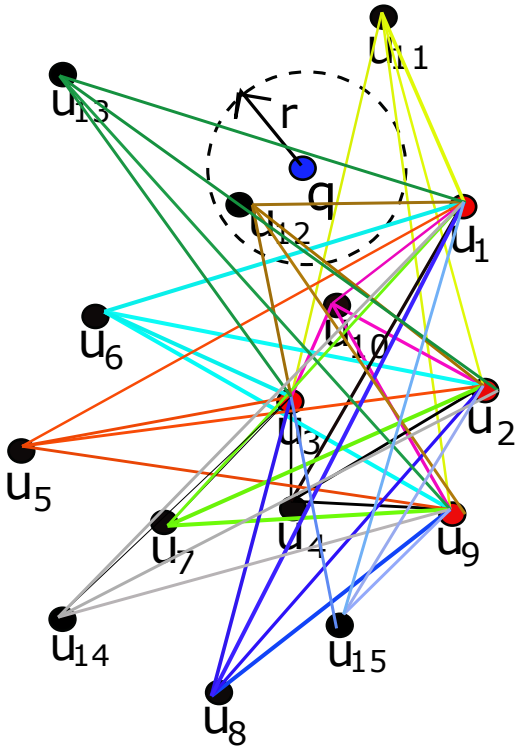



Figura 3.3: Cálculo de las distancias de cada objeto hacia el conjunto de permutantes.

| $\Pi_{u_i}$ | $u_i$ |
|---|---|
| $\Pi_{u_4}$ | $u_9, u_3, u_2, u_1$ |
| $\Pi_{u_5}$ | $u_3, u_9, u_2, u_1$ |
| $\Pi_{u_6}$ | $u_3, u_9, u_1, u_2$ |
| $\Pi_{u_7}$ | $u_3, u_9, u_2, u_1$ |
| $\Pi_{u_8}$ | $u_9, u_3, u_2, u_1$ |
| $\Pi_{u_{10}}$ | $u_3, u_1, u_2, u_9$ |
| $\Pi_{u_{11}}$ | $u_1, u_2, u_3, u_9$ |
| $\Pi_{u_{12}}$ | $u_3, u_1, u_2, u_9$ |
| $\Pi_{u_{13}}$ | $u_3, u_1, u_2, u_9$ |
| $\Pi_{u_{14}}$ | $u_3, u_9, u_2, u_1$ |
| $\Pi_{u_{15}}$ | $u_9, u_3, u_2, u_1$ |

Cuadro 3.1: Permutaciones de cada elemento de la base de datos.

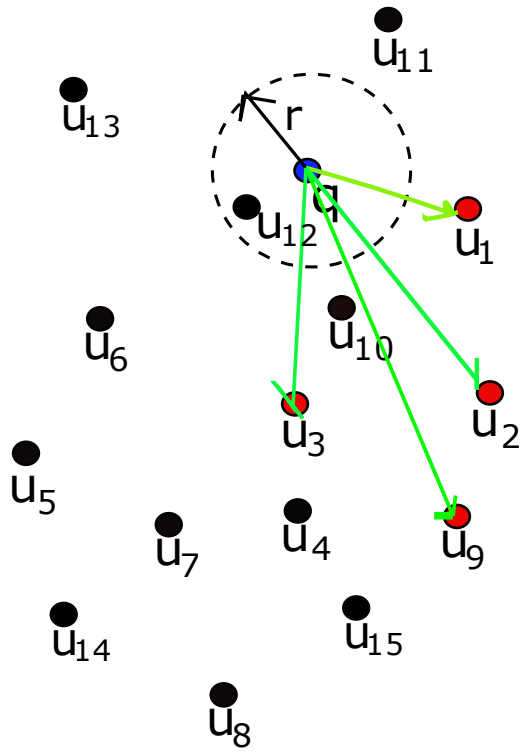



Figura 3.4: Cómputo de la distancia de la consulta hacia los permutantes. El radio de consulta es $r$ (círculo con la línea punteada).

| $\Pi_{u_i}$ | permutación | $S_\rho(q,u_i)$ |
|---|---|---|
| $\Pi_{u_4}$ | $u_9,u_3,u_2,u_1$ | 6 |
| $\Pi_{u_5}$ | $u_3,u_9,u_2,u_1$ | 6 |
| $\Pi_{u_6}$ | $u_3,u_9,u_1,u_2$ | 6 |
| $\Pi_{u_7}$ | $u_3,u_9,u_2,u_1$ | 6 |
| $\Pi_{u_8}$ | $u_9,u_3,u_2,u_1$ | 6 |
| $\Pi_{u_{10}}$ | $u_3,u_1,u_2,u_9$ | 2 |
| $\Pi_{u_{11}}$ | $u_1,u_2,u_3,u_9$ | 2 |
| $\Pi_{u_{12}}$ | $u_3,u_1,u_2,u_9$ | 2 |
| $\Pi_{u_{13}}$ | $u_3,u_1,u_2,u_9$ | 2 |
| $\Pi_{u_{14}}$ | $u_3,u_9,u_2,u_1$ | 6 |
| $\Pi_{u_{15}}$ | $u_9,u_3,u_2,u_1$ | 7 |

Cuadro 3.2: Evaluación de spearman rho entre la consulta y cada elemento de la base de datos.

$$|11 - 16| = 5 \leq \varphi \tag{3.3}$$

$$|11 - 17| = 6 \leq \varphi \tag{3.4}$$

De acuerdo a las ecuaciones anteriores, la permutaciones de los posibles vecinos más cercanos deberían comenzar con el permutante 1 o 3.

En este ejemplo se puede ver que el vecino más cercano a la consulta $q$ con radio $r$ es el objeto $u_{12}$ ya que se encuentra dentro del radio, sin embargo se puede observar que existen objetos cercanos a la consulta que no se encuentran dentro del radio y dichos objetos son considerados como vecinos más cercanos con este proceso de recuperación. Lo que determinara cuantos regresara es el número de vecinos que se desea encontrar. Se nota que el uso de este criterio en este ejemplo nos da un resultado optimo.

# Capítulo 4

# Experimentación

En este capítulo mostraremos el funcionamiento de nuestra propuesta de manera experimental. Para esto probaremos con dos tipos de bases de datos: las bases de datos sintéticas, es decir, vectores aleatorios distribuidos de manera uniforme en el cubo unitario; y bases de datos reales.

A pesar de que se realizaron exhaustivas pruebas, a continuación se presentan únicamente los resultados más sobresalientes para ambos tipos de bases de datos.

## 4.1. Ejemplo

Comenzaremos la explicación con un ejemplo con una base de datos de un diccionario en inglés. El ejercicio consistió en: tomar 32 permutantes, y construir la permutación de cada elemento (fase de preprocesamiento). Al momento de realizar una consulta se construyó la permutación respectiva y se probaron distintos tamaños de olguras. Los resultados obtenidos para buscar un vecino más cercano de la palabra *book* se muestra en la tabla 4.1. Esta palabra se quería encontrar un vecino más cercano. Los posibles inicios muestran por cuáles permuntates debería iniciar la permutación de los posibles vecinos. Sin embargo aún variando las olguras no se obtienen resultados.

Note por ejemplo en la tabla 4.2 que el uso de un radio mayor a 2 en cuya caso se uso radio igual a 3, la recuperación de 1 vecino es buena a partir de la holgura 0, sin embargo cambiando las palabras en las mismas condiciones, es decir, radio 3 y 1 vecino se considera un buen resultado de recuperación a partir de la holgura 1 con recuperaciones de un 80 porciento en adelante.

Para la recuperación de 2 vecinos con holgura 0 es un aproximado de un treinta a un ochenta por ciento de recuperación visitando un mínimo de inicios, sin embargo usando una holgura igual a 1 la recuperación de los objetos es de un noventa a un cien porciento y visita

| Book, r=1 | | | | |
|---|---|---|---|---|
| Olgura | Candidatos | Posibles inicios | Vecinos recuperados | % recuperación |
| 0 | 15 | 3 | 0 | 0 % |
| 1 | 16 | 6 | 0 | 0 % |
| 2 | 16 | 12 | 0 | 0 % |
| 3 | 16 | 14 | 0 | 0 % |
| 4 | 16 | 19 | 0 | 0 % |
| 5 | 16 | 24 | 0 | 0 % |
| 6 | 16 | 30 | 0 | 0 % |
| 7 | 16 | 32 | 0 | 0 % |
| 8 | 16 | 32 | 0 | 0 % |
| 9 | 16 | 32 | 0 | 0 % |
| 10 | 16 | 32 | 0 | 0 % |

Cuadro 4.1: Resultados para la palabra *book* con radio 1.

| monkey, r=3, NN=1 | | | | |
|---|---|---|---|---|
| Holgura | Candidatos | Posibles inicios | Vecinos recuperados | % recuperación |
| 0 | 217 | 5 | 3 | 60 % |
| 1 | 372 | 14 | 4 | 80 % |
| 2 | 375 | 20 | 5 | 100 % |
| 3 | 375 | 24 | 5 | 100 % |
| 4 | 375 | 28 | 5 | 100 % |
| 5 | 375 | 31 | 5 | 100 % |
| 6 | 375 | 32 | 5 | 100 % |
| 7 | 375 | 32 | 5 | 100 % |
| 8 | 375 | 32 | 5 | 100 % |
| 9 | 375 | 32 | 5 | 100 % |
| 10 | 375 | 32 | 5 | 100 % |

Cuadro 4.2: Resultados para la palabra *monkey* con 1 vecino y radio 3.

| book, r=4 | | | | |
|---|---|---|---|---|
| Holgura | Candidatos | Posibles inicios | Vecinos recuperados | % recuperación |
| 0 | 4044 | 3 | 26 | 22 % |
| 1 | 5530 | 6 | 115 | 98 % |
| 2 | 5994 | 12 | 117 | 100 % |
| 3 | 5998 | 14 | 117 | 100 % |
| 4 | 6027 | 19 | 117 | 100 % |
| 5 | 6028 | 24 | 117 | 100 % |
| 6 | 6028 | 30 | 117 | 100 % |
| 7 | 6028 | 32 | 117 | 100 % |
| 8 | 6028 | 32 | 117 | 100 % |
| 9 | 6028 | 32 | 117 | 100 % |
| 10 | 6028 | 32 | 117 | 100 % |

Cuadro 4.3: Resultados para la palabra *book* y 4 vecinos.

| universe, r=4 | | | | |
|---|---|---|---|---|
| Holgura | Candidatos | Posibles inicios | Vecinos recuperados | % recuperación |
| 0 | 89 | 1 | 91 | 100 % |
| 1 | 265 | 4 | 91 | 100 % |
| 2 | 334 | 8 | 91 | 100 % |
| 3 | 335 | 13 | 91 | 100 % |
| 4 | 336 | 19 | 91 | 100 % |
| 5 | 337 | 32 | 91 | 100 % |
| 6 | 337 | 32 | 91 | 100 % |
| 7 | 337 | 32 | 91 | 100 % |
| 8 | 337 | 32 | 91 | 100 % |
| 9 | 337 | 32 | 91 | 100 % |
| 10 | 337 | 32 | 91 | 100 % |

Cuadro 4.4: Resultado para la palabra *universe* con 4 vecinos.

| star, r=4 | | | | |
|---|---|---|---|---|
| Holgura | Candidatos | Posibles inicios | Vecinos recuperados | % recuperación |
| 0 | 2981 | 1 | 91 | 28 % |
| 1 | 4750 | 9 | 303 | 93 % |
| 2 | 6982 | 8 | 323 | 99 % |
| 3 | 7125 | 13 | 325 | 100 % |
| 4 | 7256 | 19 | 325 | 100 % |
| 5 | 7269 | 32 | 325 | 100 % |
| 6 | 7269 | 32 | 325 | 100 % |
| 7 | 7269 | 32 | 325 | 100 % |
| 8 | 7269 | 32 | 325 | 100 % |
| 9 | 7269 | 32 | 325 | 100 % |
| 10 | 7269 | 32 | 325 | 100 % |

Cuadro 4.5: Resultados para la palabra *star* con 4 vecinos.

| mister, r=4 | | | | |
|---|---|---|---|---|
| Holgura | Candidatos | Posibles inicios | Vecinos recuperados | % recuperación |
| 0 | 2358 | 2 | 34 | 93 % |
| 1 | 3386 | 6 | 35 | 96 % |
| 2 | 3795 | 13 | 36 | 100 % |
| 3 | 3942 | 20 | 36 | 100 % |
| 4 | 3954 | 26 | 36 | 100 % |
| 5 | 3954 | 28 | 36 | 100 % |
| 6 | 3954 | 31 | 36 | 100 % |
| 7 | 3954 | 32 | 36 | 100 % |
| 8 | 3954 | 32 | 36 | 100 % |
| 9 | 3954 | 32 | 36 | 100 % |
| 10 | 3954 | 32 | 36 | 100 % |

Cuadro 4.6: Resultados para la palabra *mister* con 4 vecinos.

| monkey, r=4 | | | | |
|---|---|---|---|---|
| Holgura | Candidatos | Posibles inicios | Vecinos recuperados | % recuperación |
| 0 | 1353 | 5 | 5 | 45 % |
| 1 | 2933 | 14 | 9 | 81 % |
| 2 | 2990 | 20 | 11 | 100 % |
| 3 | 3005 | 24 | 11 | 100 % |
| 4 | 3005 | 28 | 11 | 100 % |
| 5 | 3005 | 31 | 11 | 100 % |
| 6 | 3005 | 32 | 11 | 100 % |
| 7 | 3005 | 32 | 11 | 100 % |
| 8 | 3005 | 32 | 11 | 100 % |
| 9 | 3005 | 32 | 11 | 100 % |
| 10 | 3005 | 32 | 11 | 100 % |

Cuadro 4.7: Resultados para la palabra *monkey* con 4 vecinos.

aproximadamente de un sexto a un octavo de inicios para dicha recuperación. Nuevamente después de holgura mayor o igual a 2 la recuperación es de un cien porciento pero la visita de los inicios incrementa mucho.

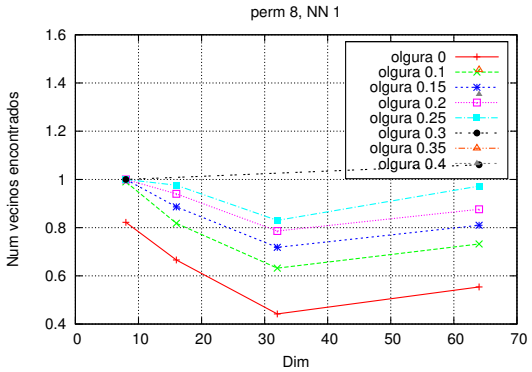
Para la obtención de 4 vecinos con holgura 0 la recuperación de objetos es de aproximadamente de un 20 porciento a un 100 porciento y visitando un aproximado de un treinta y doceavo a un sexto de posibles inicios. Para holgura igual a uno se recupera más del ochenta porciento hasta el 100 porciento y visitando los posibles inicios de a lo más un 45 porciento.

## 4.2. Bases de Datos Sintéticas

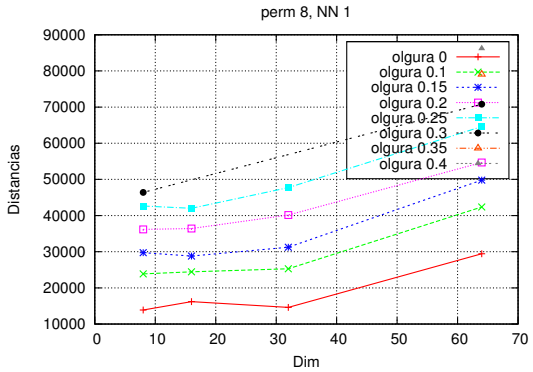
Las bases de datos sintéticas (BDS son vectores uniformemente distribuidos en el cubo unitario, de esta manera es posible controlar la dimensión intrínseca de los datos y la cantidad de datos, esto nos permite poder estudiar el comportamiento de nuestro algoritmo con distintos parámetros.

Las dimensiones utilizadas para realizar las pruebas correspondientes, fueron $d$ = 8, 16, 32 y 64, en una base de datos de 100,000 objetos. Los experimentos se realizando variando algunos parámetros para ver el comportamiento, por ejemplo: el número de permutantes, la holgura y el número de vecinos a recuperar.

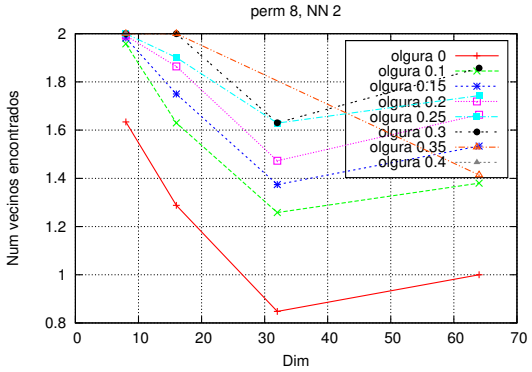
Las holguras utilizadas fueron 0, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, y los vecinos a recuperar fueron 1,2,4,8. A continuación se muestran los resultados gráficamente.



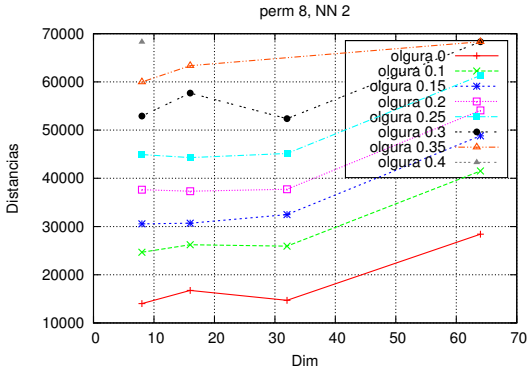

(a) Precisión en la recuperación, NN=1.




(b) Distancias calculadas, NN=1.




(c) Precisión en la recuperación, NN=2.



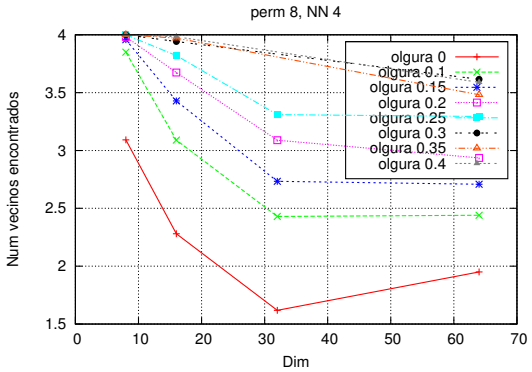

(d) Distancias calculadas, NN=2.

Figura 4.1: Gráficas variando las dimensiones usando 8 permutantes para recuperar 1 y 2 vecinos más cercanos.

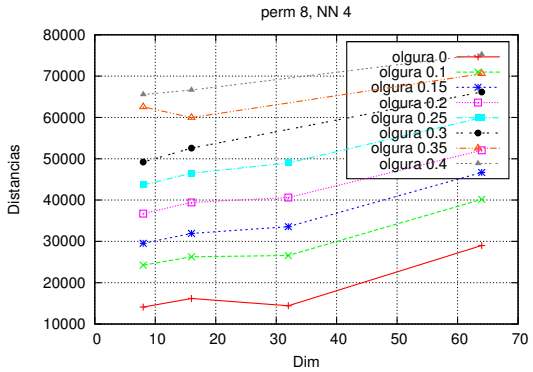
### 4.2.1. Variando la Dimensión de los Datos.

En las gráficas de la Figura 4.1 se muestra el desempeño de la propuesta para recuperar 1 y 2 vecinos con 8 pemutantes variando las dimensiones y el parámetro de holgura. En las gráficas se puede apreciar que para dimensión 8 a partir de holgura 0.1 se logra recuperar la totalidad de los vecinos buscados calculando únicamente un aproximado de 25000 distancias. En la Figura 4.1(a) se muestra la precisión y en la Figura 4.1(b), las distancias para el caso de 1 vecino. Para el caso de los 2 vecinos (Figuras 4.1(d) y 4.1(c)), como se esperaba, a medida que aumenta el tamaño de la holgura existen mas elementos a revisar, aunque por supuesto recupera mejor y el desempeño empeora a medida que aumenta la dimensión.

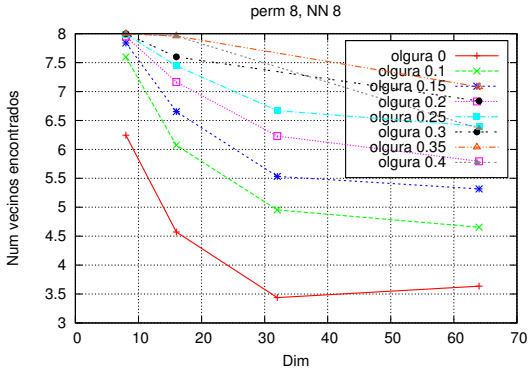
Para el caso de usar 8 permutantes y buscando 4 y 8 vecinos (ver las gráficas de la figura 4.2). En la Figura 4.2(a) se nota que para dimensiones pequeñas nuevamente las holguras pequeñas son buenas, pero conforme incrementa la dimensión la recuperación de vecinos con holguras pequeñas no es muy buena, el cálculo de distancias crece (ver figura 4.2(b)) casi a la mitad para holguras buenas como 0.25, sin embargo usar holgura de 0.15 es buena en cuestión de



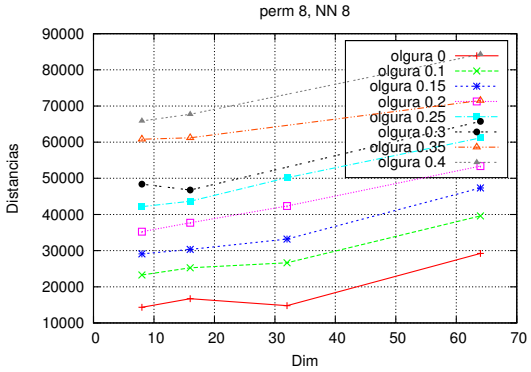

(a) Precisión en la recuperación, NN=4.




(b) Distancias calculadas, NN=4.




(c) Precisión en la recuperación, NN=8.



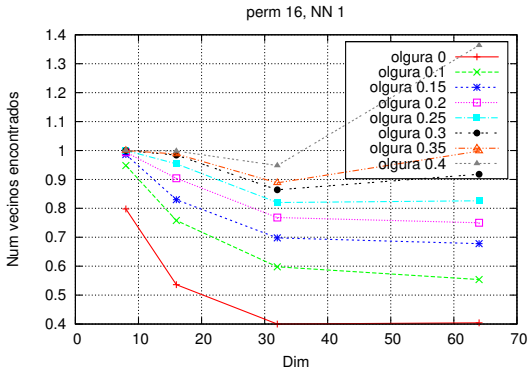

(d) Distancias calculadas, NN=8.

Figura 4.2: Gráficas variando las dimensiones usando 8 permutantes para recuperar 4 y 8 vecinos más cercanos.

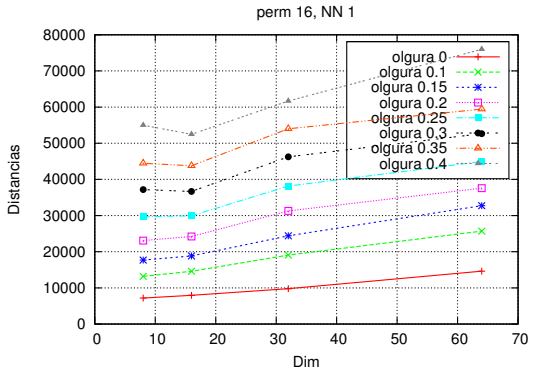
recuperación y cálculo de distancias, ya que para esta se hace un cálculo aproximado de 30000 distancias y se recupera un aproximado de 3.5 vecinos. En el caso de buscar 8 vecinos, note que la holgura 0.1 tiene un excelente desempeño (más de 7.5 vecinos en promedio) y sólo aumentan unas cuantas distancias.

Aumentar el número de permutantes podría mejorar las búsquedas, en las gráficas siguientes se mostrará el uso de 16 permutantes para distintos tamaños de vecinos más cercanos a recuperar. En las gráficas de la Figura 4.3 se muestra la recuperación para 1, 2 y 4 vecinos cercanos. Para 1 vecino se presenta la gráfica de precisión y las distancias calculadas en 4.3(b) y 4.3(a) respectivamente. En el caso de 2 vecinos véase las gráficas 4.3(d) y 4.3(c); y para el caso de 4 vecinos corresponden a las gráficas 4.3(f) y 4.3(e). Nótese que para todos estos experimentos, la mejor holgura es con 0.1 pues hay un equilibrio entre el número de distancias y tener una muy buena tasa de recuperación.

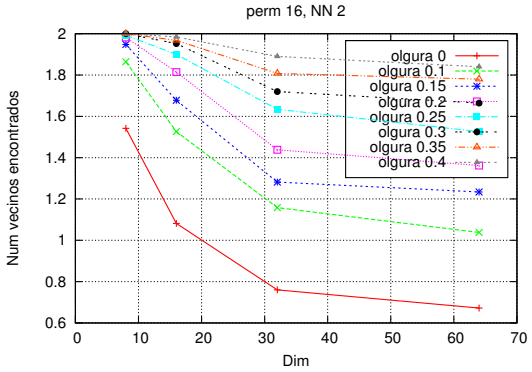
En las gráficas de la Figura 4.4, se muestran los resultados obtenidos pero ahora duplicando el número de permutantes, es decir 32. Note que para holguras mayores de 0 casi se recuperan todos los elementos buscados, sin embargo a medida que aumenta el tamaño de la holgura



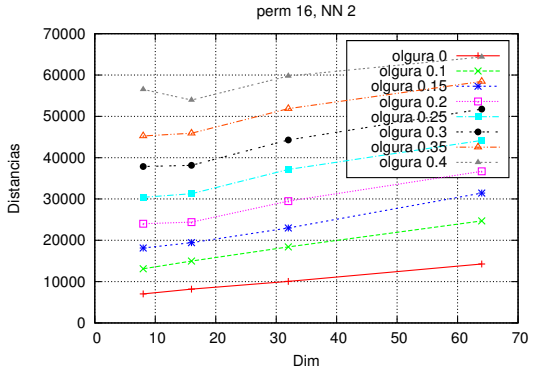

(a) Precisión en la recuperación, NN=1.



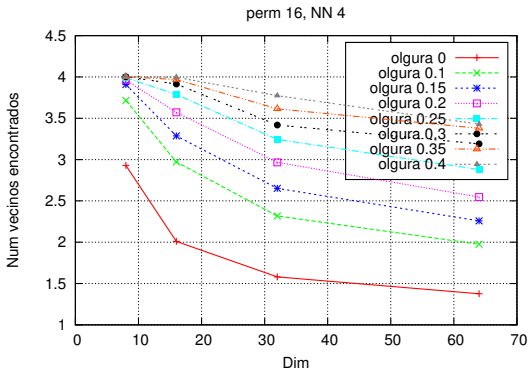

(b) Distancias calculadas, NN=1.



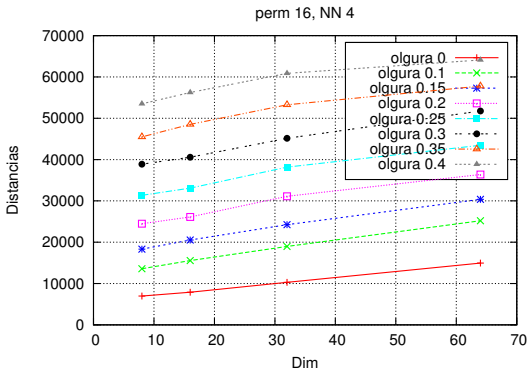

(c) Precisión en la recuperación, NN=2.




(d) Distancias calculadas, NN=2.



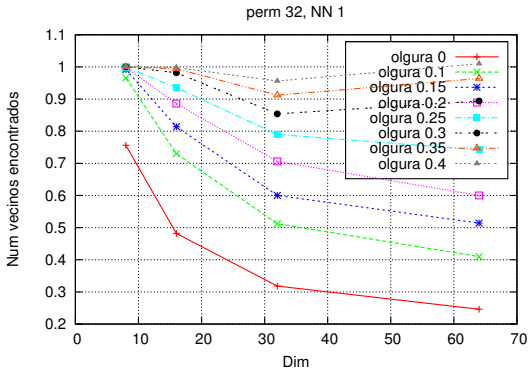

(e) Precisión en la recuperación, NN=4.



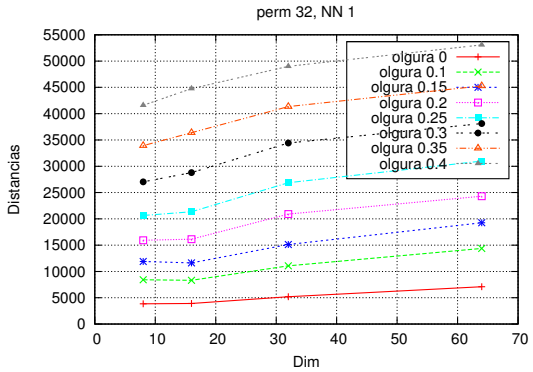

(f) Distancias calculadas, NN=4.

Figura 4.3: Gráficas variando las dimensiones usando 16 permutantes para recuperar 1, 2 y 4 vecinos más cercanos. La primer fila corresponde a los resultados de buscar 1 vecino, la segunda fila para 2 vecinos y la tercer fila para 4 vecinos.

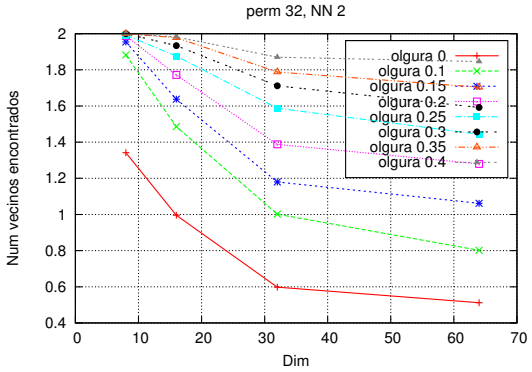
se incrementa el costo en cálculos de distancia; por lo tanto, la holgura que mejor balance tiene es 0.1 para el caso de recuperar 1 y 2 vecinos. Las gráficas de la primera fila corresponden a 1 vecino en precisión 4.4(a) y 4.4(b) en número de cálculos de distancia. Para 2 vecinos se muestran en las figuras 4.4(c) (para el caso de la recuperación) y en 4.4(d) para las distancias calculadas.



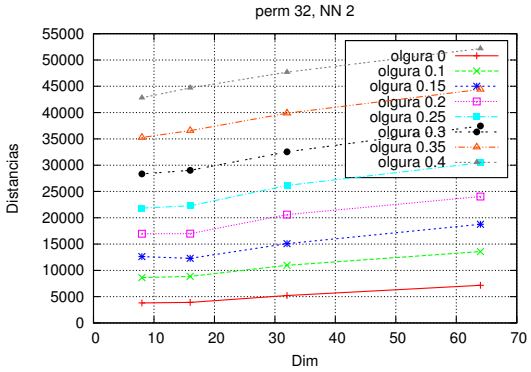

(a) Precisión en la recuperación, NN=1.




(b) Distancias calculadas, NN=1.




(c) Precisión en la recuperación, NN=2.



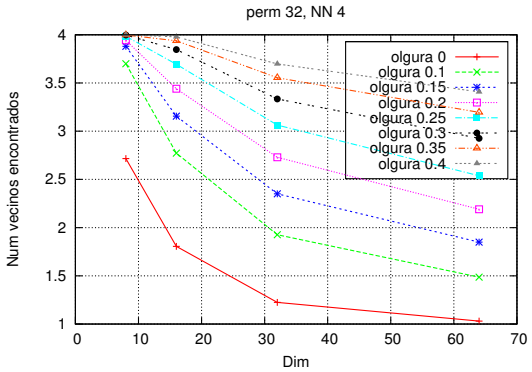

(d) Distancias calculadas, NN=2.

Figura 4.4: Gráficas variando las dimensiones usando 32 permutantes para recuperar 1 y 2 vecinos más cercanos.

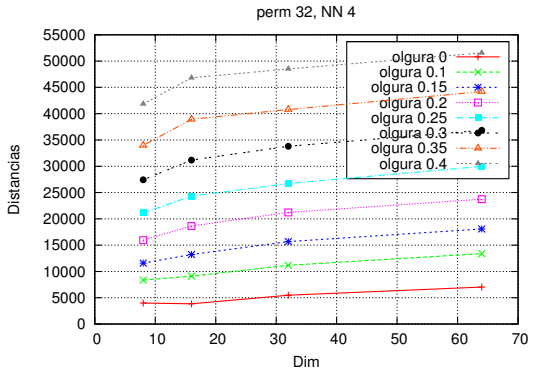
Para la recuperación de más vecinos (4 y 8), en las gráficas de la Figura 4.5. Para 4 vecinos son las gráficas de la primera columna 4.5(a) y 4.5(b) para recuperación y distancias respectivamente. Note que a medida que aumenta la dimensión empeora el desempeño rápidamente, sin embargo, aumentando el tamaño de la holgura es posible incrementar el número de distancias y mejorar el desempeño. Para el caso de 8 vecinos, véase las gráficas 4.5(c) y 4.5(d), precisión y recuperación respectivamente. En estas gráficas note que en dimensión alta (32) con solo 12,000 distancias se tiene casi el 50 % de los vecinos, es decir aproximandamente el 10 % de la base de datos.

Para las siguientes gráficas de la Figura 4.6 se duplicó el número de permutantes, es decir a 64. Las gráficas de la primer fila, esto es 4.6(a) y 4.6(b) muestran el desempeño para recuperar 1 vecino, por supuesto a medida que aumenta el número de permutantes mejora el desempeño, incluso disminuye el número de cálculos de distancia totales. En las Figuras 4.6(c) y 4.6(d) se muestra el desempeño para el caso de recuperar 2 vecinos. Note que para una holgura de 2.5 se tiene 1.6 vecinos recuperados en promedio y revisando solo el 25 % de la base de datos.

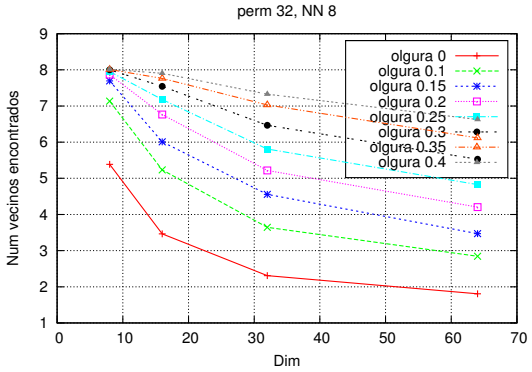
En el caso de recuperar más vecinos, 4 y 8, el desempeño se presenta en la Figura



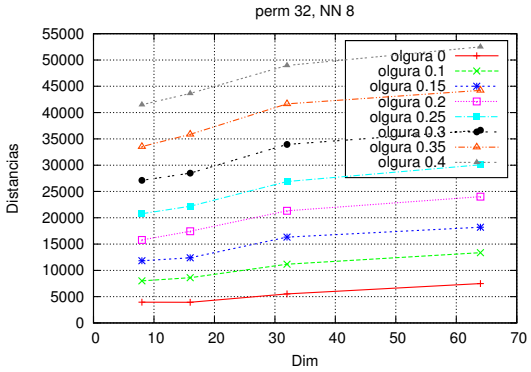

(a) Precisión en la recuperación, NN=4.




(b) Distancias calculadas, NN=4.




(c) Precisión en la recuperación, NN=8.



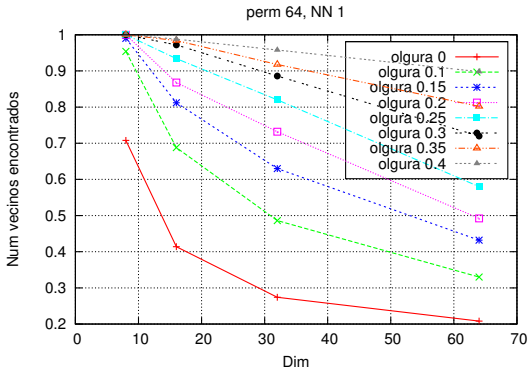

(d) Distancias calculadas, NN=8.

Figura 4.5: Gráficas variando las dimensiones usando 32 permutantes para recuperar 4 y 8 vecinos más cercanos.

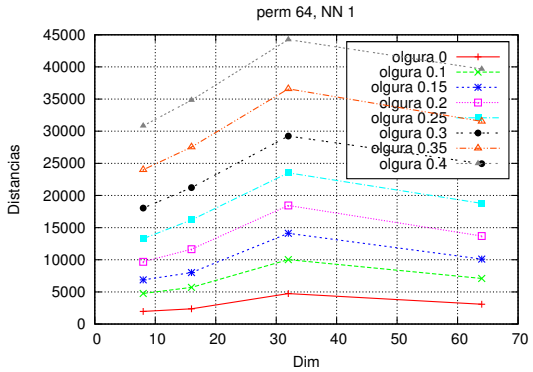
4.7. Para 4 vecinos, la primer fila presenta la recuperación y el costo en función de las distancias, como se puede ver en las Figuras 4.7(a) y 4.7(b). Para dimensiones bajas con una holgura de 0.1 se tiene un buen desempeño, para dimensiones altas, con una holgura de 0.2 o 0.25 se tiene una buena recuperación. Lo mismo sucede para la recuperación de 8 vecinos mas cercanos que se muestra en las Figuras 4.7(c) y 4.7(d), precisión y costo en distancias respectivamente.

Considerando muchos más permutantes, 128, para buscar 1 y 2 vecinos más cercanos (véase las gráficas de la Figura 4.8). Note que la cantidad empleada para resolver las consultas disminuye respecto a usar menos permutantes, la recuperación más o menos se mantiene en su comportamiento. En la primer fila se muestra el desempeño para 1 vecino (Figuras 4.8(a) y 4.8(b)), y la segunda fila para 2 vecinos (Figuras 4.8(c) y 4.8(d)), precisión en la recuperación al lado izquierdo y costo en distancias al lado derecho respectivamente.

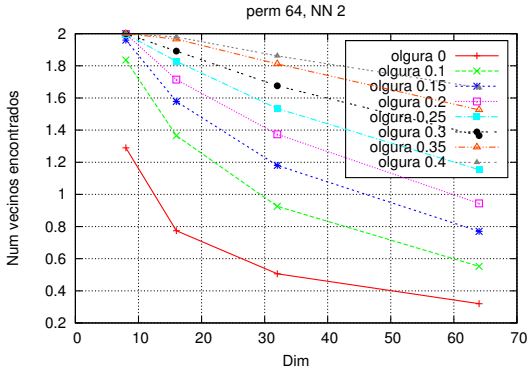
Para el caso de usar 128 permutantes y querer recuperar 4 y 8 vecinos el desempeño se muestra en la gráficas de la Figura 4.9. La primer fila es para 4 vecinos (véase las Figuras 4.9(a) y 4.9(b)) y la segunda para 8 vecinos (véase las Figuras 4.9(c) y 4.9(d)). Al lado izquierdo se muestra la precisión en la recuperación y en el lado derecho las distancias calculadas. En esta gráficas se



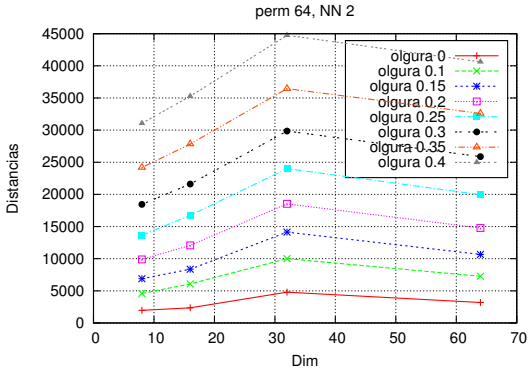

(a) Precisión en la recuperación, NN=1.




(b) Distancias calculadas, NN=1.




(c) Precisión en la recuperación, NN=2.




(d) Distancias calculadas, NN=2.

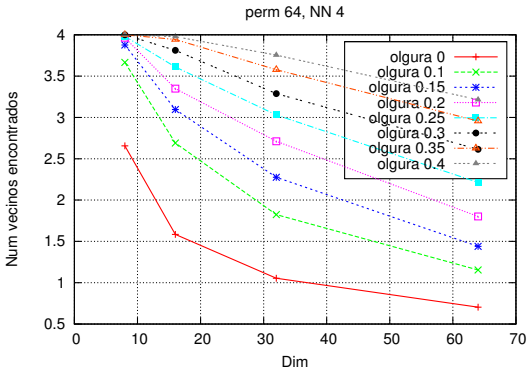
Figura 4.6: Gráficas variando las dimensiones usando 64 permutantes para recuperar 1 y 2 vecinos más cercanos.

muestra que con 64 permutantes más se disminuyen varios miles de distancias. Note que para dimensión 32, con sólo revisar el 20 % de la base de datos se tiene una recuperación del 75 %.

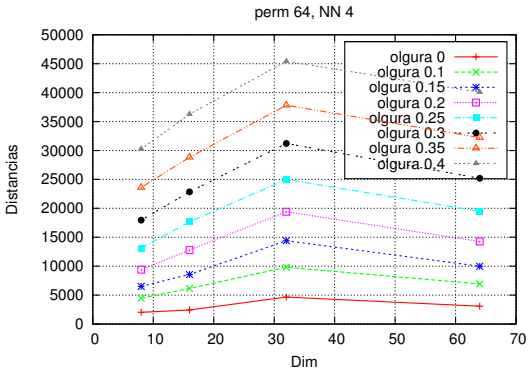
### 4.2.2. Variando la Cantidad de Permutantes.

Para esta sección, analizaremos los resultados considerando variar el parámetro de permutantes.

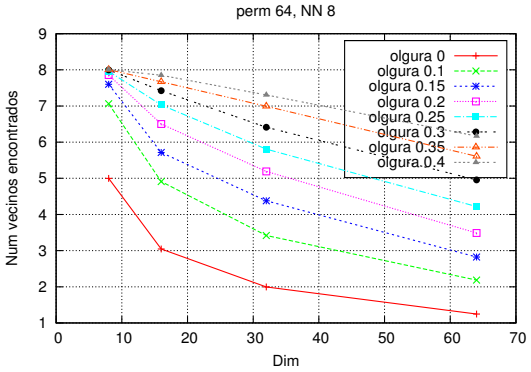
En las gráficas de la Figura 4.10 se muestra el desempeño de la propuesta al variar el número de permutantes, buscando recuperar 8 vecinos. Las gráficas de la primer fila (Figuras 4.10(a) y 4.10(b), precisión y distancias calculadas respectivamente) se muestra el comportamiento en dimension 8. La segunda fila muestra el comportamiento en dimensión 16 (Figuras 4.10(c) y 4.10(d), precisión y distancias calculadas respectivamente). En estas gráficas se puede apreciar que: a medida que se incrementa el número de permutantes, se disminuye el número de cálculos de distancia; sin embargo, también disminuye el número de vecinos encontrados, este fenómeno se debe a que hay más posibilidades para el inicio de una permutación similar.



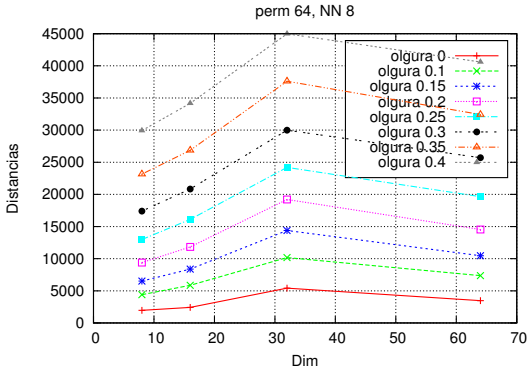

(a) Precisión en la recuperación.




(b) Distancias calculadas.




(c) Precisión en la recuperación.




(d) Distancias calculadas.

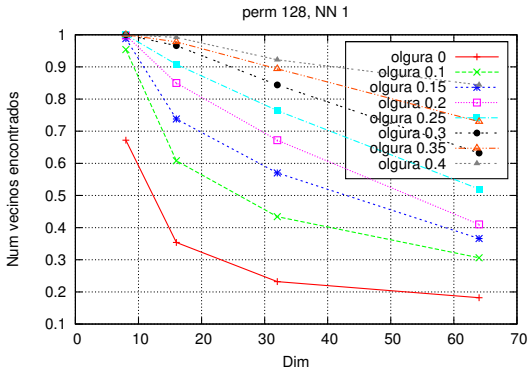
Figura 4.7: Gráficas variando las dimensiones usando 64 permutantes para recuperar 4 y 8 vecinos más cercanos.

Para el caso de incrementar la dimensión de los datos a 32 y 64 los resultados se pueden observar en la Figura 4.11. Para dimensión 32 en precisión y número de cálculos de distancia véase las Figuras 4.11(a) y 4.11(b) respectivamente. Para dimensión 64 corresponde a la segunda fila en las Figuras 4.11(c) y 4.11(d) de precisión y costo en cálculos de distancia. En estas gráficas se puede apreciar a mayor cantidad de distancias mejor es la recuperación, consideramos que el mejor equilibrio se encuentra con 16 permutantes.

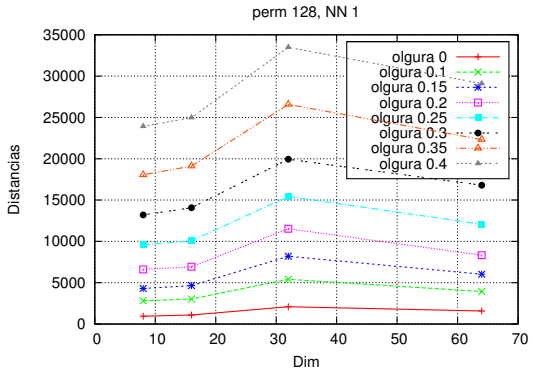
## 4.3. Variando el Parámetro de Holgura

A continuación se observarán los resultados variando el parámetro de la holgura contra los vecinos recuperados. Las gráficas se mostrarán para una dimensión en específico y una cantidad de permutantes fijas.

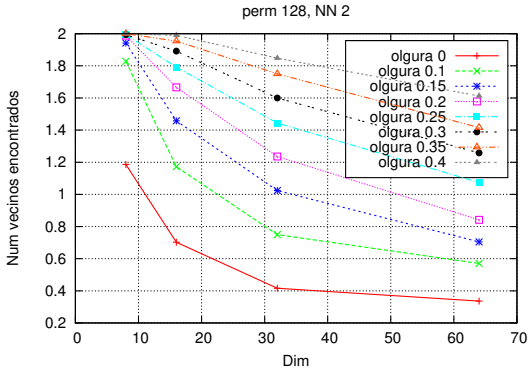
Dado que esta técnica tuvo mejores resultados en dimensión baja (8), se mostrará el detalle del desempeño para 8, 16,32 y 64 permutantes. Para el caso de 8 y 16 los resultados se pueden



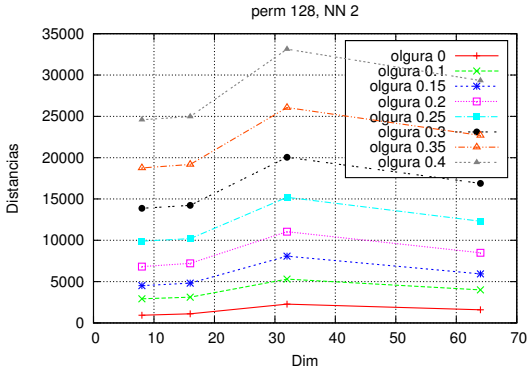

(a) Precisión en la recuperación, NN=1.




(b) Distancias calculadas, NN=1.




(c) Precisión en la recuperación, NN=2.



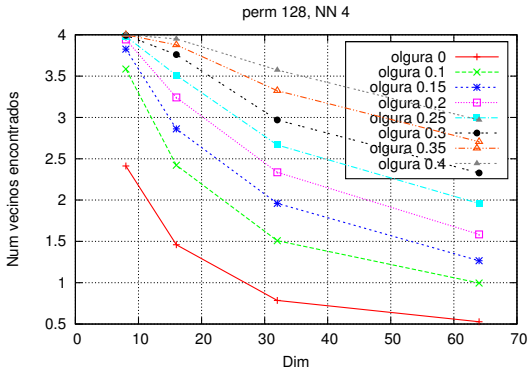

(d) Distancias calculadas, NN=2.

Figura 4.8: Gráficas variando las dimensiones usando 128 permutantes para recuperar 1 vecino más cercano.

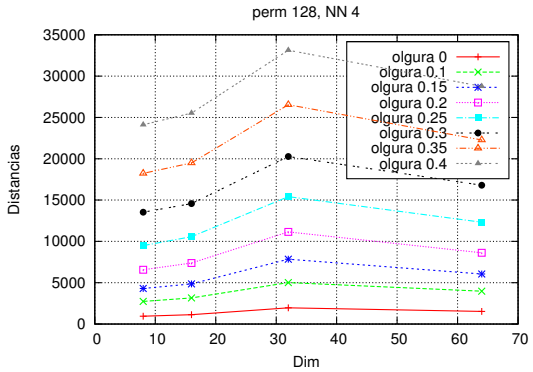
apreciar en la Figura 4.12. En la primer fila corresponde a 8 permutantes (Figuras 4.12(a) y 4.12(b)), mientras que la segunda fila es para 16 permutantes (Figuras 4.12(c) y 4.12(d)). En el lado izquierdo se muestra la recuperación y en el lado derecho la cantidad de cálculos de distancia respectivamente. Note que el cálculo de distancias no depende del número de vecinos a consultar, sólo del parámetro de holgura, es por esto que pareciera que tienen el mismo número de comparaciones. Otro aspecto relevante a mencionar es que a medida que aumenta la holgura mejora la recuperación, sin embargo a medida que aumentan los permutantes, se empeora un poco.

En las gráficas de la Figura 4.13 se muestra el desempeño de la técnica propuesta usando 32 y 64 permutantes para dimensión 8, buscando recuperar 8 vecinos mas cercanos. Para 32 permutantes son las Figuras 4.13(a) y 4.13(b). Para 64 permutantes, véase las Figuras 4.13(c) y 4.13(d). En el lado izquierdo la recuperación y el derecho para costo en distancias. Nóte que para esta dimensión con 64 permutantes tiene un buen desempeño a partir de la parámetro de holgura 0.25.

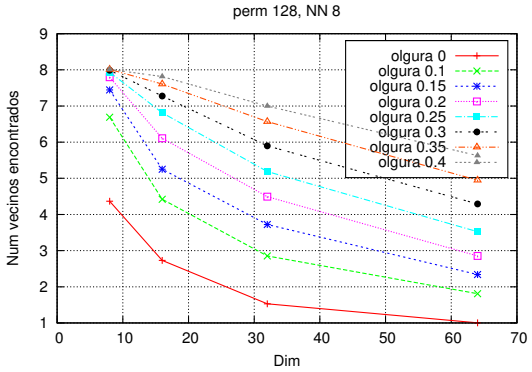
Para dimensión 16, sólo se presentará su desempeño usando 32 y 64 permutantes, esto se puede observar en las gráficas de la Figura 4.14. Esto experimentos corroboran el comportamiento



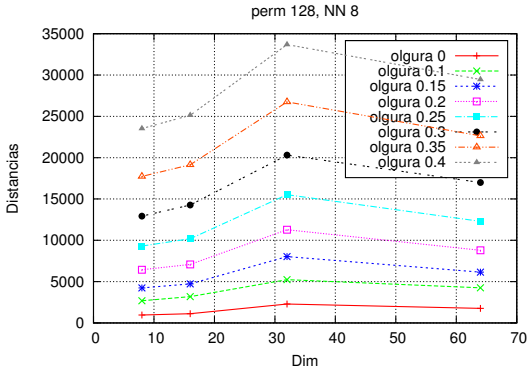

(a) Precisión en la recuperación.




(b) Distancias calculadas.




(c) Precisión en la recuperación.



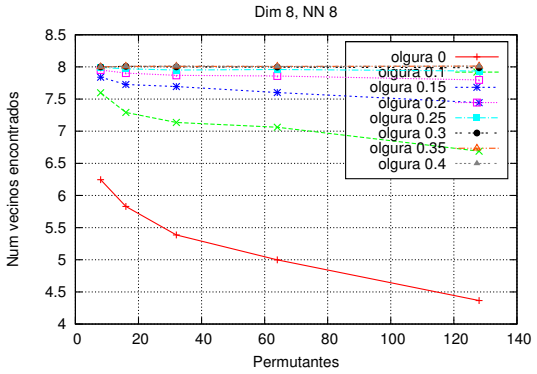

(d) Distancias calculadas.

Figura 4.9: Gráficas variando las dimensiones usando 128 permutantes para recuperar 4 y 8 vecinos más cercanos.

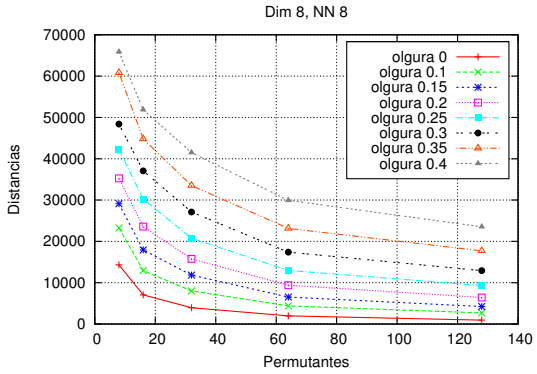
explicado anteriormente respecto a que, a mayor cantidad de permutantes menor es el porcentaje de recuperación y menor el número de cálculos de distancia. En las Figuras 4.14(a) y 4.14(b) se usaron 32 permutantes para precisión y su correspondiente costo en distancias. En las Figuras 4.14(c) y 4.14(d) se muestra para 64 permutantes.

Para el caso de tener dimensión 32 se usaron 32 y 64 permutantes. El desempeño se puede ver en la Figura 4.15. La gráficas 4.15(a) y 4.15(b) son de precisión y costo en distancias para la 32 permutantes, buscando recuperar 8 vecinos. Para el caso de 64 permutantes, obsérvese las Figuras 4.15(c) y 4.15(d). En estos experimentos se nota que para holguras pequeñas se preferirían pocos permutantes, para holguras más grandes más o menos se tienen los mismos resultados.

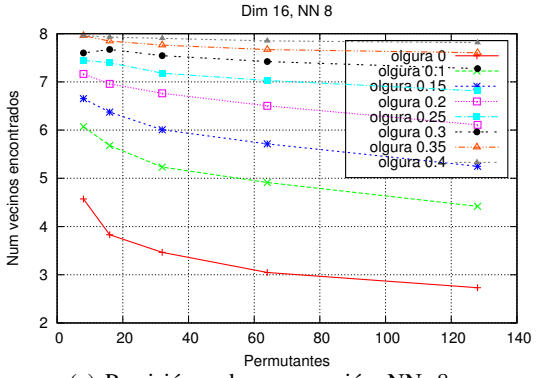
Para una dimensión muy alta se aprecia el efecto de la maldición de la dimensión. Esta técnica es propensa pues depende de un radio como lo es el parámetro de holgura. En la Figura 4.16 se muestra el desempeño de la técnica propuesta para dimensión 64, usando 32 y 64 permutantes. La primer fila es para una el caso de usar 32 permutantes (véase las Figuras 4.16(a) y 4.16(b)) en precisión y costo. Para 64 permutantes, en precisión y costo en distancias se puede apreciar en las Figuras 4.16(c) y 4.16(d). Note que para un parámetro de holgura de 0.15 se tiene mejor



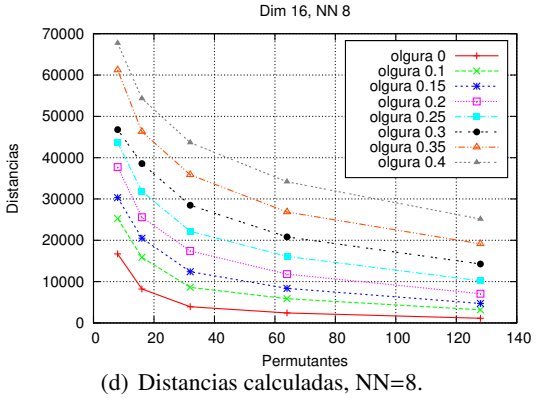

(a) Precisión en la recuperación, NN=8.




(b) Distancias calculadas, NN=8.



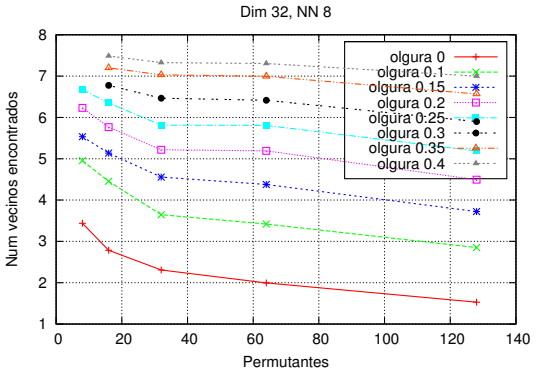

(c) Precisión en la recuperación, NN=8.



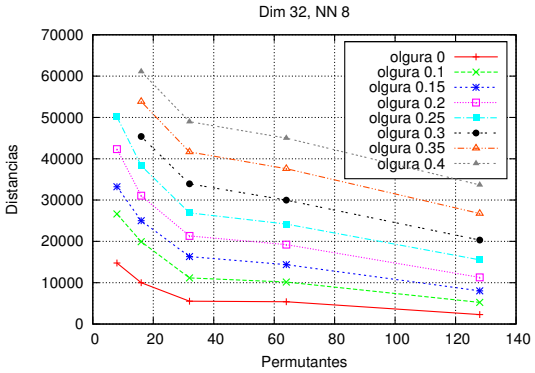

(d) Distancias calculadas, NN=8.

Figura 4.10: Gráficas variando la cantidad de permutantes para dimensiones 32 y 64 (primera y segunda fila respectivamente), para buscar 8 vecinos más cercanos

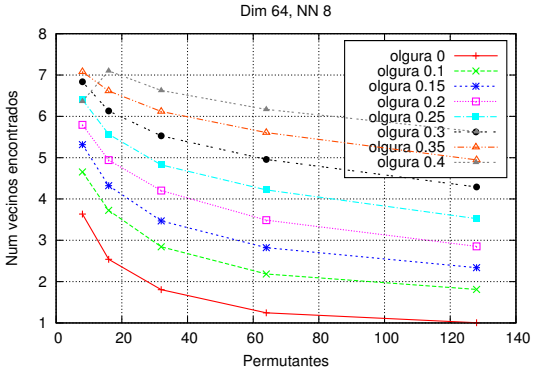
recuperación con 32 permutantes aunque usando casi 20,000 distancias. Para el caso de usar 64 permutantes, empleando 20,000 distancias (holgura 0.25) se pueden lograr encontrar un vecino más.



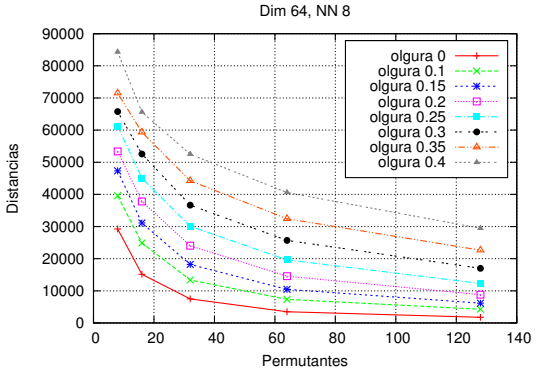

(a) Precisión en la recuperación, NN=8.




(b) Distancias calculadas, NN=8.

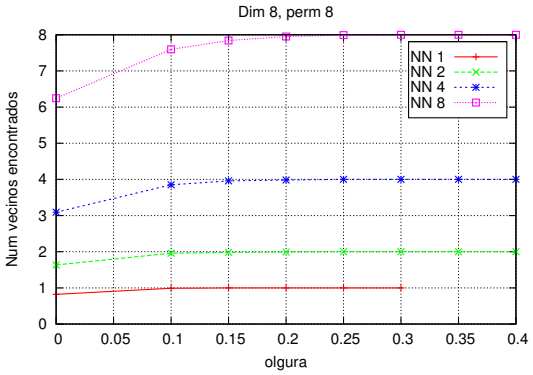



(c) Precisión en la recuperación, NN=8.

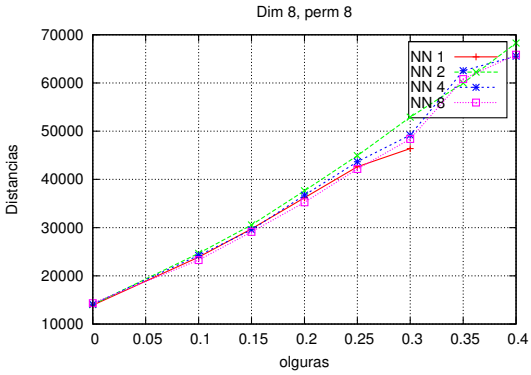



(d) Distancias calculadas, NN=8.

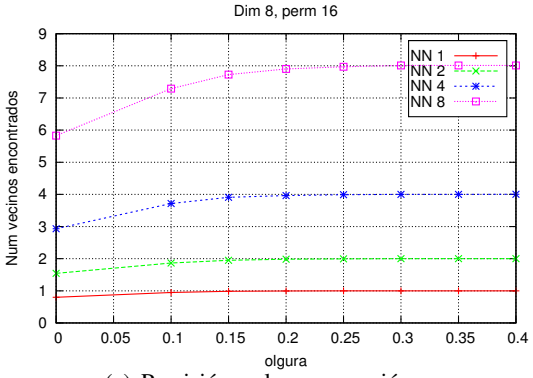
Figura 4.11: Gráficas variando la cantidad de permutantes para dimensiones 32 y 64 (primera y segunda fila respectivamente), para buscar 8 vecinos más cercanos



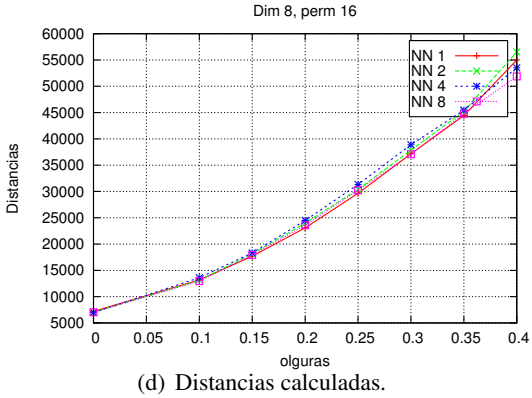

(a) Precisión en la recuperación.




(b) Distancias calculadas.

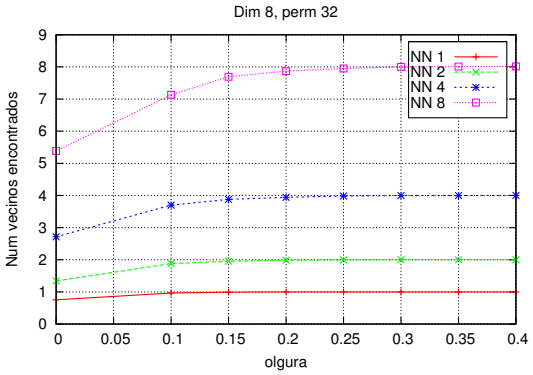



(c) Precisión en la recuperación.

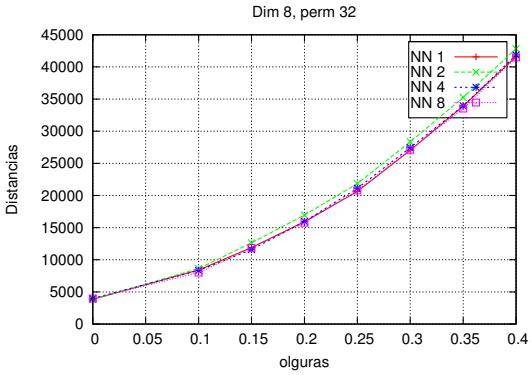



(d) Distancias calculadas.

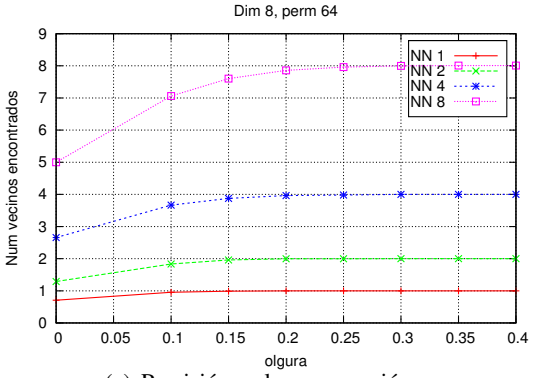
Figura 4.12: Gráficas variando el parámetro de holgura para 8 y 16 permutantes (primera y segunda fila respectivamente), para buscar 8 vecinos más cercanos, en dimensión 8.



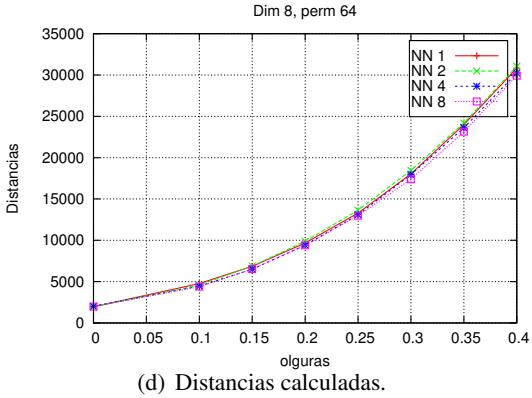

(a) Precisión en la recuperación.




(b) Distancias calculadas.

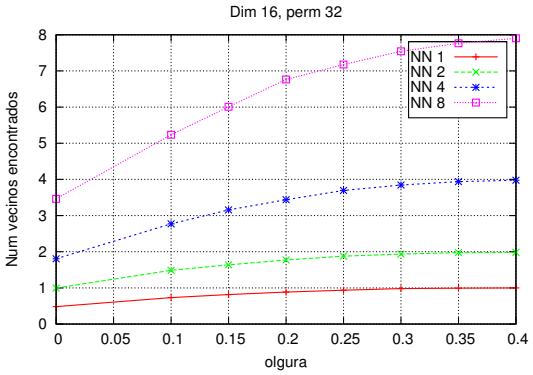



(c) Precisión en la recuperación.

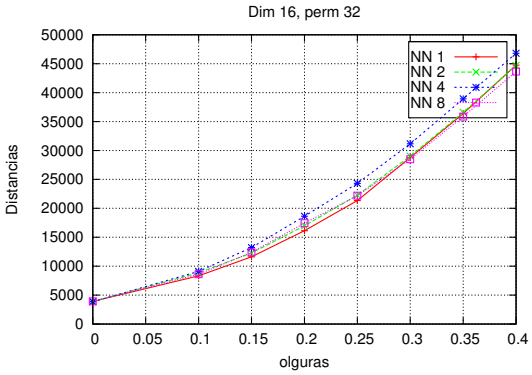



(d) Distancias calculadas.

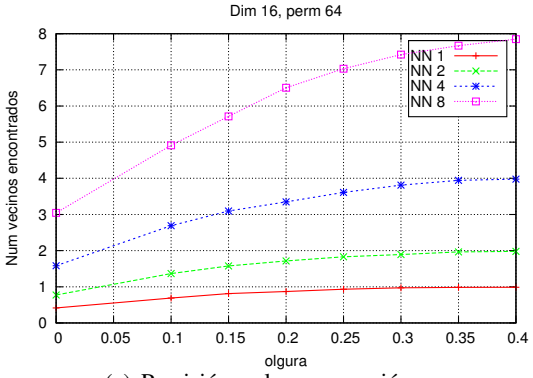
Figura 4.13: Gráficas variando el parámetro de holgura para 32 y 64 permutantes (primera y segunda fila respectivamente), para buscar 8 vecinos más cercanos, en dimensión 8.



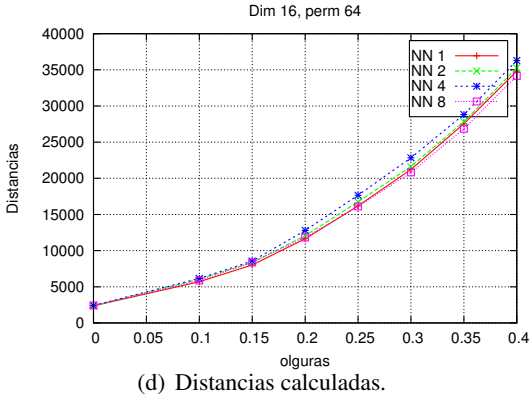

(a) Precisión en la recuperación.




(b) Distancias calculadas.

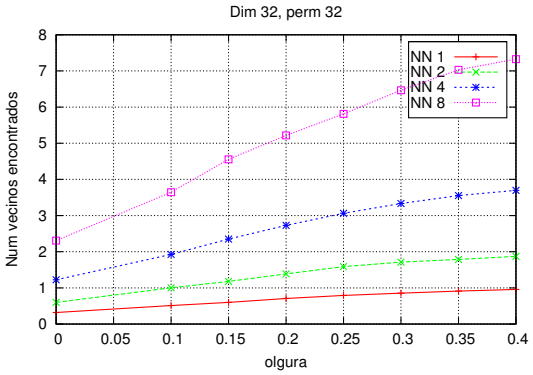



(c) Precisión en la recuperación.

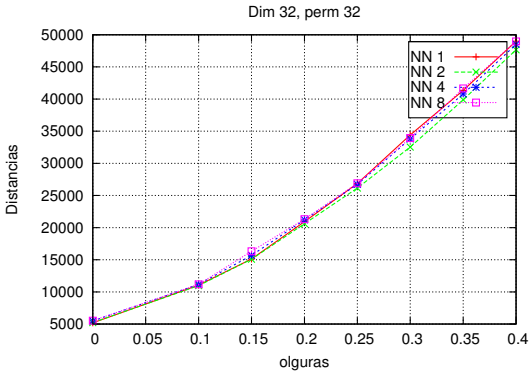



(d) Distancias calculadas.

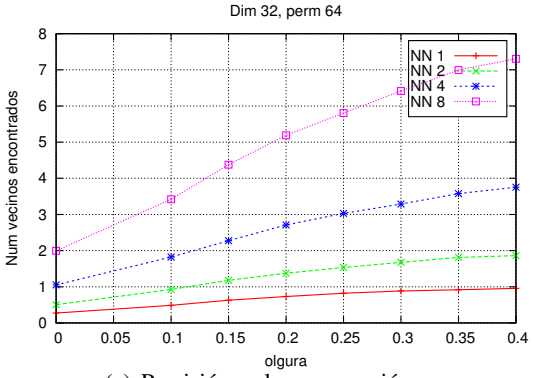
Figura 4.14: Gráficas variando el parámetro de holgura para 32 y 64 permutantes (primera y segunda fila respectivamente), para buscar 8 vecinos más cercanos, en dimensión 16.



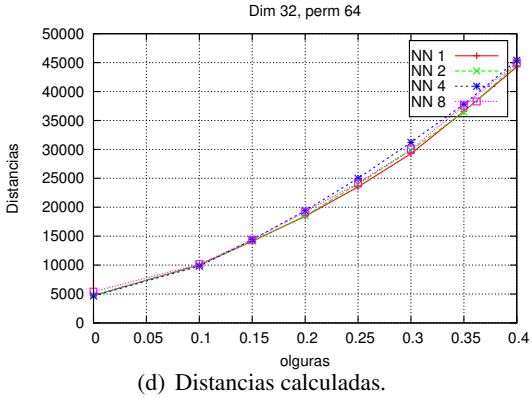

(a) Precisión en la recuperación.




(b) Distancias calculadas

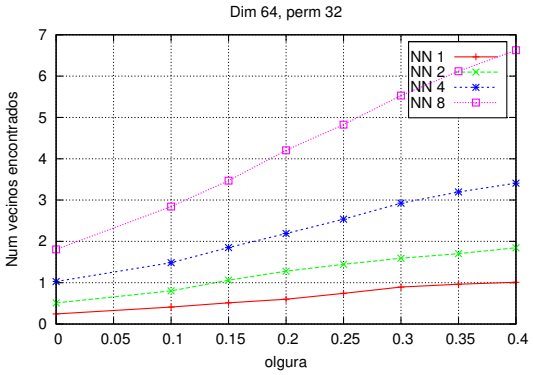



(c) Precisión en la recuperación.

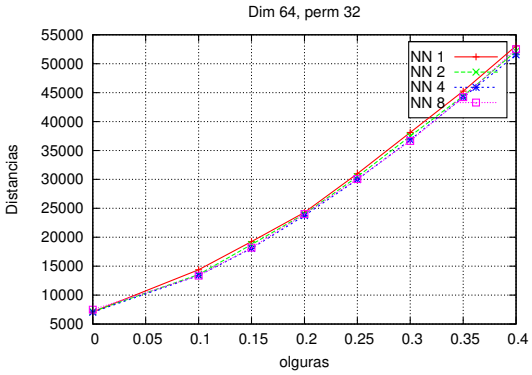



(d) Distancias calculadas.

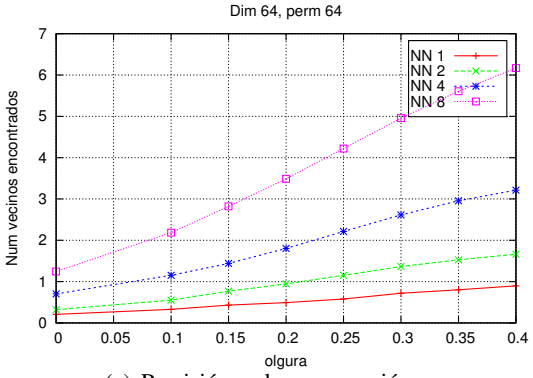
Figura 4.15: Gráficas variando el parámetro de holgura para 32 y 64 permutantes (primera y segunda fila respectivamente), para buscar 8 vecinos más cercanos, en dimensión 32.



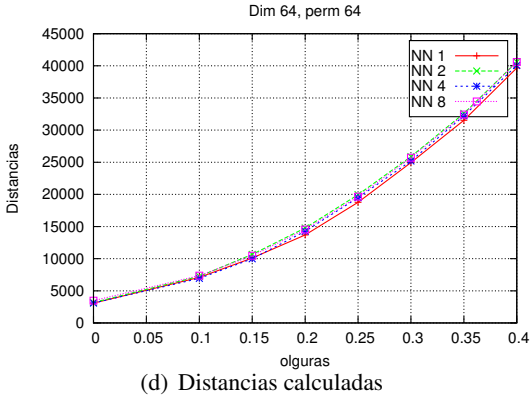

(a) Precisión en la recuperación.




(b) Distancias calculadas.




(c) Precisión en la recuperación.




(d) Distancias calculadas

Figura 4.16: Gráficas variando el parámetro de holgura para 32 y 64 permutantes (primera y segunda fila respectivamente), para buscar 8 vecinos más cercanos, en dimensión 64.

# Capítulo 5

# Conclusiones

Las búsquedas por proximidad o similaridad reflejan la importancia de adquirir los resultados más significativos para búsquedas de cualquier índole, ya que en la actualidad se está rodeado de la tecnología y se hace uso de ella para diversas tareas. Dada cualquier consulta de cualquier tipo se busca obtener el resultado más relevante para dicha consulta, y este resultado se logra mediante la comparación de la consulta con la base de datos.

En este trabajo se hizo uso de las búsquedas por similaridad en espacios métricos es decir se mapeó el problema a un espacio métrico para así poder hacer de la medida de distancia lo cual permite la creación de un índice a través del cual se realizan las comparaciones de distancias de la consulta con la base de datos. Este trabajo propone un parámetro denominado holgura $\varphi$ que nos permite saber que inicios de los permutantes son significativos y por tanto cuales se deben visitar y cuales se deben descartar, esto permite reducir el número de comparaciones a realizar.

Los resultados obtenidos en los experimentos con el uso de esta técnica muestran una mejora en comparación con las técnicas ya conocidas en este caso se puede comparar con el PPIndex y se observa la mejora ya que si se ve el descenso del árbol con esta técnica sólo desciende por cierto número de ramas.

Lo que se observó a través de todo el proceso es que si se quiere refinar el resultado de la búsqueda en algunas dimensiones se debe "pagar" un cálculo mayor de distancias, y por otro lado si se quiere obtener un resultado con menor número de distancias calculadas y aún así obtener un buen resultado, si bien es cierto que en algunos casos se puede perder algún resultado valioso.

# Bibliografía


[Ben79] J. Bentley. Multidimensional binary search trees in database applications. *IEEE Transactions on Software Engineering*, 5(4):333–340, 1979.

[BYCMW94] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proceedings 5th Combinatorial Pattern Matching (CPM'94)*, volume 807 of *Lecture Notes in Computer Science*, pages 198–212, 1994.

[CFN05] E. Chávez, K. Figueroa, and G. Navarro. Proximity searching in high dimensional spaces with a proximity preserving order. In *MICAI 2005: Advances in Artificial Intelligence*, volume 3789 of *Lecture Notes in Computer Science*, pages 405–414, 2005.

[CMN99] E. Chávez, J. Marroquín, and G. Navarro. Overcoming the curse of dimensionality. In *European Workshop on Content-Based Multimedia Indexing (CBMI'99)*, pages 57–64, 1999.

[CN00] E. Chávez and G. Navarro. An effective clustering algorithm to index high dimensional metric spaces. In IEEE CS Press, editor, *7th International Symposium on String Processing and Information Retrieval (SPIRE 2000)*, pages 75–86, 2000.

[CNBYM99] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. Technical Report TR/DCC-99-3, Dept. of Computer Science, Univ. of Chile, 1999. `ftp://ftp.dcc.uchile.cl/pub/users/gnavarro/-survmetric.ps.gz`.

[CNBYM01a] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Proximity searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[CNBYM01b] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Proximity searching in metric spaces. *AMC Computing Surveys*, pages 33(3):373–321, 2001.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: an efficient access method for similarity search in metric spaces. In *Proc. of the 23rd Conference on Very Large Databases (VLDB'97)*, pages 426–435, 1997.

[DG77] Persi Diaconis and R. L. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(2):262–268, 1977.

[DN87] F. Dehne and H. Noltemeier. Voronoi trees and clustering problems. *Information Systems*, 12(2):171–175, 1987.

[HS00] G. Hjaltason and H. Samet. Incremental similarity search in multimedia databases. *Technical Report TR 4199. Department of Computer Science, University of Maryland*, 2000.

[LI09] LSDS-IR@SIGIR, editor. *PP-Index: Using Permutation Prefixes for Efficient and Scalable Approximate Similarity Search*, 2009.

[PZB06] V. Dohnal P. Zezula, G. Amato and M. Batko. *Similarity Search: The Metric Space Approach, volume 32 of Advances in Database System*. Springer, 2006.

[Uhl91] J. Uhlmann. Implementing metric trees to satisfy general proximity/similarity queries. Manuscript, 1991.