UNIVERSIDAD NACIONAL DE SAN LUIS
FACULTAD DE CIENCIAS FÍSICO
MATEMÁTICAS Y NATURALES

Trabajo Final de la Licenciatura en Ciencias de la Computación

Una Aplicación de Búsquedas por Similitud en el Ámbito del Comercio Electrónico

Libertad Speranza
María de los Ángeles de la Torre

Directora: M.Cs. Norma Edith Herrera

San Luis - Argentina, Mayo 2020

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Introducción

Las bases de datos tradicionales son construidas basándose en el concepto de búsqueda exacta: la base de datos es dividida en registros y cada registro contiene campos completamente comparables. Las consultas a la base de datos retornan todos aquellos registros cuyos campos coinciden exactamente con los aportados en tiempo de búsqueda.

A través del tiempo las bases de datos han evolucionado para incluir la capacidad de almacenar nuevos tipos de datos tales como imágenes, sonido, video, etc. Estructurar este tipo de datos en registros para adecuarlos al concepto tradicional de búsqueda exacta es difícil en muchos casos y hasta imposible si la base de datos cambia más rápido de lo que se puede estructurar (como por ejemplo la web). Aún cuando pudiera hacerse, las consultas que se pueden satisfacer con la tecnología tradicional están limitadas a variaciones de la búsqueda exacta.

En el ámbito de éste trabajo, nos interesan las búsquedas en donde se puedan recuperar objetos similares a uno dado. Este tipo de búsqueda se conoce con el nombre de búsqueda por similitud, y surge en diversas áreas; tales como reconocimiento de voz, reconocimiento de imágenes, compresión de texto, biología computacional, entre otras.

Todas estas aplicaciones tienen algunas características comunes. Existe un universo X de objetos y una función de distancia $d : X \times X \to \Re$ que modela la similitud entre los objetos. El par (X, d) es llamado espacio métrico. La base de datos es un conjunto $U \subseteq X$, el cual se preprocesa a fin de resolver búsquedas por similitud eficientemente.

Una de las consultas típicas en este nuevo modelo que implica recuperar

objetos similares de una base de datos es la **búsqueda por rango**, que denotaremos con $(q, r)_d$. Dado un elemento de consulta q, al que llamaremos query y un radio de tolerancia r, una búsqueda por rango consiste en recuperar aquellos objetos de la base de datos cuya distancia a q no sea mayor que r.

Otra consulta que se utiliza para recuperar objetos similares es la **búsqueda de los *k*-vecinos más cercanos**, donde tenemos el elemento de consulta q y la cantidad de objetos de base de datos que se quieren recuperar, que llamaremos k. La cercanía de estos k objetos está dada por el valor de la función de distancia d hasta la query q.

Los cómputos realizados durante una búsqueda implican cálculos de la función de distancia d, operaciones adicionales (sumas, restas, comparaciones, etc.) y tiempo de I/O si el índice y/o los datos se encuentran en memoria secundaria. Las búsquedas por similitud pueden ser resueltas trivialmente por medio de una búsqueda exhaustiva, con una complejidad $O(n)$. Para evitar esta situación, se preprocesa la base de datos por medio de un algoritmo de indexación con el objetivo de construir una estructura de datos o índice, diseñada para ahorrar cálculos en el momento de resolver una búsqueda.

Básicamente existen dos enfoques para el diseño de algoritmos de indexación en espacios métricos: uno está basado en Diagramas de Voronoi [CNBYM01] y el otro está basado en pivotes [CNBYM01]. En este trabajo abordamos el estudio de algoritmos de indexación basados en pivotes, enfocándonos en una aplicación que utilice este tipo de algoritmos en el ámbito del comercio electrónico.

Los algoritmos basados en pivotes construyen el índice basándose en la distancia de los objetos de la base de datos a un conjunto de elementos preseleccionados llamados pivotes.

1.2. Objetivo

El comercio electrónico, también conocido como e-commerce consiste en la compra y venta de productos o de servicios a través de la web. El desarrollo de nuevas tecnologías ha permitido que la capacidad y volumen de las comunicaciones se expanda de una manera exponencial, esto ha facilitado que el comercio electrónico tenga también un crecimiento exponencial.

Para el desarrollo de un sitio de comercio electrónico hay varios problemas que deben resolverse tales como administración de categorías de productos, búsqueda de productos, encriptación de datos, registración de usuarios, administración de medios de pago, entre otros. En este trabajo nos hemos centrado específicamente

en el problema de búsqueda de productos.

Nuestra meta es utilizar búsquedas por similitud sobre los títulos asociados a los productos con el fin de estudiar el desempeño de los algoritmos basados en pivotes en un caso real.

1.3. Como se organiza la tesis

Hemos organizado este informe en dos partes: en la primera se introducen los conceptos necesarios para comprender el informe y en la segunda presentamos el trabajo realizado y los resultados que obtuvimos al aplicar los algoritmos de indexación.

Primera Parte

Capítulo 1: Definimos la motivación y los objetivos del trabajo.

Capítulo 2: Presentamos los conceptos básicos de las búsquedas en espacios métricos con el objetivo de dar el marco teórico necesario para comprender y fundamentar el desarrollo realizado en este trabajo final. También explicamos la situación actual del comercio electrónico, sobre el cual vamos a aplicar los conceptos teóricos.

Capítulo 3: Definimos completamente en qué consiste la aplicación de espacios métricos a esta base de datos real de comercio electrónico.

Segunda Parte

Capítulo 4: Describimos el trabajo realizado. Presentamos la implementación de dos algoritmos de indexación basados en pivotes: Selección de pivotes en forma aleatoria y selección de pivotes en forma incremental. También detallamos las diversas variaciones implementadas a fin de encontrar la que mejor se adapte al tipo de búsqueda que queremos realizar.

Capítulo 5: Mostramos la evaluación experimental de las variaciones implementadas.

Capítulo 6: Presentamos las conclusiones obtenidas como así también algunos puntos interesantes para abordar en futuros trabajos.

Capítulo 2

Conceptos Previos

Este capítulo introduce el marco teórico del presente trabajo a través de la definición de los siguientes conceptos: espacios métricos (ejemplo: diccionario de palabras), medidas de distancia; se incluyen algunas de las medidas más comunes utilizadas dentro de la temática a tratar, e introduciremos un modelo formal para las búsquedas por similitud.

2.1. Espacios Métricos

En este trabajo se aborda el tema de búsqueda no convencional, es decir, dentro de un universo de datos, nos interesa encontrar aquellos objetos que son "similares" a un objeto dado. El modelo formal que abarca este tipo de búsquedas se denomina Espacio Métrico.

Espacio Métrico: Sea X un universo de objetos válidos y sea $U \subseteq X$ un subconjunto finito de tamaño n sobre el que se realizarán búsquedas. Llamaremos U a las base de datos, diccionario o simplemente conjunto de elementos. Definimos una función $d : (X \times X) \rightarrow \Re$ que denotará una medida de distancia entre objetos de X, esto significa que a menor distancia más cercanos o similares son los objetos. Esta función d cumple con las propiedades características de una función de distancia:

(a) $\forall x, y \in X d(x, y) \geq 0$ (Positividad)

(b) $\forall x, y \in X d(x, y) = d(y, x)$ (Simetría)

(c) $\forall x \in X d(x, x) = 0$ (Reflexividad)

en la mayoría de los casos:

(d) $\forall x, y \in X x \neq y \Rightarrow > 0$ (Positividad estricta)

Estas propiedades sólo aseguran una definición consistente de la función, pero no pueden usarse para evitar comparaciones durante una búsqueda por similitud. Para que la función d sea realmente una métrica debe satisfacer la siguiente propiedad:

(e) $\forall x, y, z \in X d(x, y) \leq d(x, z) + d(y, z)$ (Desigualdad triangular)

El par X, d es llamado espacio métrico. Si la función d no satisface la propiedad de positividad estricta (d), entonces diremos que el espacio es pseudo métrico. Estos casos pueden ser fácilmente resueltos; basta con adaptar las técnicas de espacios métricos de manera tal que todos los objetos a distancia cero se identifiquen como un único objeto. Esto funciona dado que $d(x, x) = 0 \Rightarrow \forall z, d(x, z) = d(y, z)$ (puede demostrarse utilizando la desigualdad triangular).

En algunos casos podemos tener cuasi métricas donde no se cumpla la propiedad de simetría (b). En estos casos podemos definir a partir de la función d una nueva función d', que sí sea métrica $d'(x, y) = d(x, y) + d(y, x)$.

Finalmente podemos llegar a relajar la desigualdad triangular (e) permitiendo $d(x, y) \leq \alpha d(x, z) + \beta d(z, y) + \gamma$. En estos casos podemos seguir usando los mismos algoritmos de espacio métrico, previo escalamiento.

2.2. Búsquedas por similitud

Básicamente existen tres tipos de búsquedas de interés en espacios métricos [CNBYM01]:

Búsqueda por rango $(q, r)_d$**:** consiste en recuperar todos los elementos de $\mathcal{U}$ que están a distancia r de un elemento q dado (que llamaremos query). En símbolos:

$$(q, r)_d = \{u \in \mathcal{U} : d(q, u) \leq r\}$$

Son probablemente las más comunes dentro de los espacios métricos. Los objetos pertenecientes a la respuesta pueden ordenarse en base a su distancia de q, si es necesario. Nótese que el objeto q no necesita existir en la base de datos U, basta con que sea un elemento del universo X,

Búsqueda del vecino más cercano $NN(q)$**:** Cuando se quiere realizar búsquedas por similitud sobre objetos utilizando búsquedas por rango, se debe especificar la distancia máxima para que los objetos sean incluídos en la respuesta. Existen casos donde puede ser difícil especificar el radio

sin conocimiento sobre los datos y la función de distancia. Se nos presenta entonces la situación en donde, si especificamos un radio de búsqueda muy pequeño no obtendremos ningún resultado, y deberemos repetir la búsqueda con un radio mayor. Por otro lado, si especificamos un radio de búsqueda muy grande, corremos el riesgo de que el costo computacional de calcular la distancia sea muy alto y de que el conjunto de resultados contenga objetos no significativos. Una alternativa para solucionar este tipo de problemas en la búsqueda por similitud es realizar una búsqueda de el (o los) vecino(s) má(s) cercano(s) a q. Esta búsqueda se define de la siguiente manera:

$$NN(q) = \{u \in \mathcal{U} : \forall v \in \mathcal{U},\ d(q,u) \leq d(q,v)\}$$

Búsqueda de los k-vecinos más cercanos $NN_k(q)$: El concepto de vecino más cercano puede generalizarse al caso de búsqueda de los k vecinos más cercanos. Específicamente $kNN(q)$ retorna los k vecinos más cercanos del objeto q, Esto significa encontrar un conjunto $A \subseteq \mathcal{U}$ tal que:

$$|A| = k\ \ \wedge \forall u \in A,\ v \in (\mathcal{U} - A) :\ d(q,u) \leq d(q,v)$$

Cuando existen varios objetos a la misma distancia del k-ésimo vecino más cercano, la elección de uno se realiza arbitrariamente.

El tiempo total de resolución de una búsqueda puede ser calculado de la siguiente manera:

T= #evaluaciones de d× complejidad(d) + tiempo extra de CPU + tiempo de I/O

En muchas aplicaciones la evaluación de la función d es tan costosa que las demás componentes de la fórmula anterior pueden ser despreciadas. Este es el modelo de costos que usaremos en este trabajo.

Claramente una búsqueda por similitud puede resolverse de forma ineficiente en tiempo $O(n)$ examinando exhaustivamente la base de datos $\mathcal{U}$. Para evitar esto, se preprocesa $\mathcal{U}$ usando algún *algoritmo de indexación* que construye un índice, diseñado para ahorrar cómputos en el momento de resolver una búsqueda. El proceso de construcción del índice puede ser costoso, pero este costo se verá justificado por el gran número de consultas que posteriormente se realizarán en la base de datos.

Hay dos casos principales a considerar: cuando el índice y los datos pueden ser mantenidos en memoria principal, y cuando es necesario utilizar memoria secundaria para almacenar los datos y/o el índice.

Para los algoritmos de indexación en memoria principal, el objetivo principal es reducir el número de cálculos de distancia (se adecúa al modelo que mencionáramos

anteriormente).

Para los algoritmos en memoria secundaria, además de realizar pocos cálculos de distancia, se requiere que también realicen pocos accesos a disco.

2.3. Ejemplos de Espacios Métricos

Veremos a continuación algunos ejemplos de espacios métricos.

Diccionario de Palabras

En este caso, los objetos del espacio métrico son cadenas de caracteres de un determinado lenguaje. Para medir la distancia entre cadenas, una función posible es la conocida como distancia de edición o distancia de Levenshtein, que mide el mínimo número de operaciones de inserción, eliminación y reemplazo de caracteres necesarias para transformar una palabra x en una palabra y.

Claramente, esta es un función de distancia discreta y tiene una variedad de aplicaciones en recuperación de texto, procesamiento de señales y biología computacional. Como veremos mas adelante, en este trabajo hemos utilizado un espacio métrico similar al diccionario de palabras, ya que nuestro universo de objetos se compone de títulos de productos en venta publicados en un sitio de e-commerce.

Espacios vectoriales

Los espacios vectoriales k-dimensionales son un caso especial de espacios métricos, donde los objetos son vectores de k componentes. Hay varias funciones de distancia para espacios vectoriales, pero las más usadas son las pertenecientes a la familia L_p o familia de distancias de Minkowsky (veremos su definición en la próxima sección).

La Figura 2.1 ejemplifica búsquedas por rango $(q, r)_d$ sobre un conjunto de puntos en $\Re^2$, usando como función de distancia L_2 (izquierda) y L_∞ (derecha), dos casos particulares de la distancia L_p. Las líneas punteadas representan aquellos puntos que están exáctamente a distancia r de q, por lo tanto todos aquellos elementos que caen dentro de estas líneas forman parte del resultado de la búsqueda. La distancia L_2, más conocida como *Distancia Euclidiana*, se corresponde con nuestra noción de distancia espacial. Las búsquedas con distancia L_∞ se corresponde con la búsqueda por rango clásica, donde el rango es un hiper-rectángulo k dimensional.

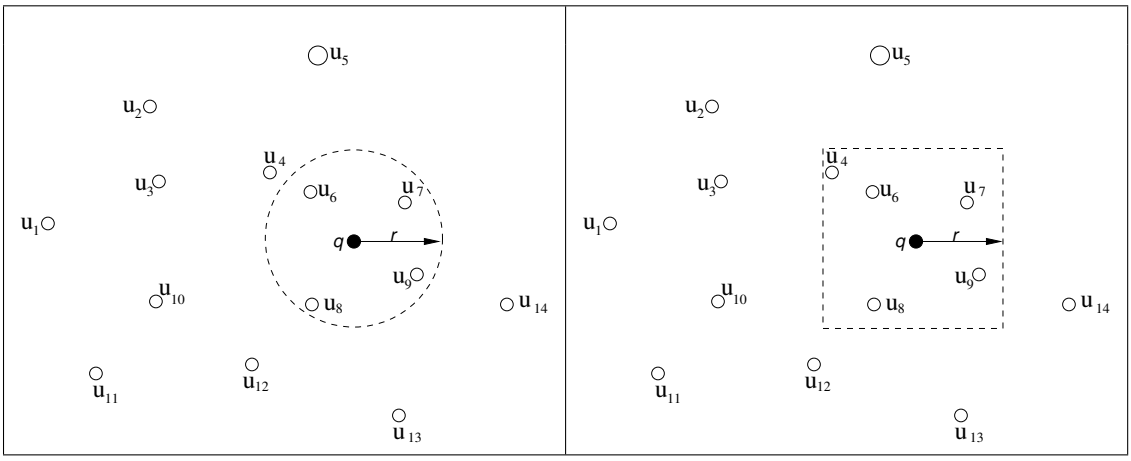


Figura 2.1: *Ejemplos de búsquedas por rango* $(q, r)_d$*, con* $d = L_2$ *(izquierda) y* $d = L_\infty$ *(derecha).*

En el ámbito de espacios vectoriales existen soluciones eficientes para búsquedas por similitud, como por ejemplo: $KD-tree$, $R-tree$, $Quadtree$ y $X-tree$ entre otras. Estas técnicas usan información sobre coordenadas para clasificar y agrupar puntos en el espacio. Desafortunadamente, las técnicas existentes son afectadas por la dimensión del espacio vectorial. Por lo tanto, la complejidad de búsqueda por rango depende exponencialmente de la dimensión del espacio.

Los espacios vectoriales pueden presentar grandes diferencias entre su dimensión representacional y su dimensión intrínseca (es decir la dimensión real en la que se pueden embeber los puntos manteniendo la distancia entre ellos). Por ejemplo, un plano embebido en un espacio de dimensión 50, tiene una dimensión representacional de 50 y una dimensión intrínseca de 2.

Por esta razón, muchas veces se recurre a espacios métricos generales, aun sabiendo que el problema de búsqueda es más difícil. No se debe descartar que también es posible tratar un espacio vectorial como un espacio métrico general usando solo la distancia entre los puntos. Una ventaja inmediata de esto es que toma peso la dimensión intrínseca del espacio, independientemente de cualquier dimensión representacional.

2.4. Funciones de Distancia

Cuando hablamos de funciones de distancia, podemos dividirlas en dos grupos, de acuerdo al tipo de valor que retornan:

- **Discretas**: son aquellas que retornan un valor discreto como valor de distancia, como por ejemplo la distancia de edición.
- **Continuas**: son las que retornan un número real como distancia entreobjetos, como por ejemplo la distancia L_2 o distancia Euclidiana.

Describimos a continuación algunos ejemplos de funciones de distancia para espacios métricos.

Distancia de Minkowski

Como ya lo mencionamos, estas funciones forman realmente una familia de funciones denominadas métricas L_p, ya que los casos individuales dependen del parámetro numérico p. Estas funciones se definen sobre vectores K-dimensionales de números reales de la siguiente manera:

$$L_p[(x_1, \ldots, x_K), (y_1, \ldots, y_K)] = \sqrt[p]{\sum_{i=1}^{k} |x_i - y_i|^p}$$

La métrica L_1 es conocida como la distancia de Manhattan y la métrica L_2 denota la distancia Euclidiana.

Para el caso de $p = \infty$, se toma el límite de la fórmula anterior para p tendiendo ∞. Se obtiene como resultado, que la distancia entre dos puntos del espacio vectorial es la máxima diferencia entre sus coordenadas:

$$L_\infty((x_1, ..., x_k), (y_1, ..., y_k)) = max_{1 \leq i \leq k} |x_i - y_i|$$

La función L_∞ se conoce con el nombre de distancia máxima, infinita o distancia de tablero de ajedrez.

Estas funciones son utilizadas en varios casos donde los vectores numéricos tienen condenadas independientes, por ejemplo, en mediciones de experimentos científicos, observaciones ambientales, o el estudio de diferentes aspectos de procesos de negocios.

Distancia de forma cuadrática

En muchas aplicaciones que utilizan vectores de datos, los valores de las distintas dimensiones no son totalmente independientes. En estos casos las distancias de Minskowski no son aplicables ya que no tienen en cuenta la existencia de esta relación.

La distancia de forma cuadrática es utilizada en ámbitos en los que existe una correlación entre las dimensiones que componen el vector, ya que tiene el poder de modelar este tipo de dependencias. La medida de distancia entre dos vectores $\bar{x}$ e $\bar{y}$ de k dimensiones está basada en una matriz positiva de pesos $M_{k\times k}$ donde $m_{i,j}$ denotan cuán fuerte es la conexión entre los componentes i y j de los vectores $\bar{x}$ e $\bar{y}$, respectivamente. Generalmente estos pesos son normalizados de manera que $0 \leq m_{i,j} \leq 1$ donde las diagonales $m_{i,i} = 1$. La distancia de forma cuadrática generalizada d_M se define como:

$$d_M(\bar{x}, \bar{y}) = \sqrt{(\bar{x} - \bar{y})^T \cdot M \cdot (\bar{x} - \bar{y})}$$

donde el superíndice T denota la transposición de vectores.

El cálculo de ésta distancia puede ser muy caro en términos computacionales, dependiendo de la dimensionalidad de los vectores.

Notar que M es la matríz identidad, esta distancia se transforma en la distancia euclidiana.

Distancia de Edición

La cercanía entre secuencias de símbolos (cadenas) puede medirse de manera efectiva a través de la distancia de Edición, también conocida como distancia de Levenshtein. La distancia entre dos cadenas $x = x_1...x_n$ e $y = y_1...y_n$ está definida como el número mínimo de operaciones atómicas de edición (inserción, eliminación y reemplazo) necesarias para transformar la cadena x en la cadena y. En términos formales, las operaciones de edición se definen como sigue:

- **insert:** inserción del caracter c en la posición i de la cadena x, en símbolos:

$$ins(x, i, c) = x_1 x_2 \dots x_i \ c \ x_{i+1} \dots x_n$$

- **delete**: eliminación del caracter c en la posición i de la cadena x, en símbolos:

$$del(x, i) = x_1 x_2 \dots x_{i-1} x_{i+1} \dots x_n$$

- **replace**: sustitución de un caracter c de la cadena x en la posición i por un nuevo caracter c', en símbolos:

$$repl(x, i, c) = x_1 x_2 \dots x_{i-1} \ c' \ x_{i+1} \dots x_n$$

La función de distancia de edición generalizada asigna pesos (números reales positivos) a cada operación atómica, por esto, la distancia entre las cadenas x e y es el mínimo valor de la suma de los pesos de las operaciones atómicas necesarias

para transformar x en y. Si los pesos asignados a cada operación difieren, la distancia de edición no es simétrica (violando la propiedad (b) del punto 2.1) y por lo tanto, no es una función métrica.

A los fines del presente trabajo, a todas las operaciones se le asigna un peso equivalente de uno.

Distancia de Edición de árbol

Esta distancia es una bien conocida medida de proximidad entre árboles, en la cual se define la distancia entre dos estructuras de árboles como el costo mínimo necesario para convertir el árbol fuente en el árbol destino utilizando un conjunto predefinido de operaciones de edición sobre árboles, tales como la inserción y la eliminación de un nodo. El costo individual de estas operaciones de edición puede ser constante para todo el árbol, o puede variar de acuerdo al nivel en el cual se lleva a cabo la operación. La razón de esta variación es que el costo de la inserción de un nodo cercano al nivel de la raíz puede ser más significativo que el costo de agregar un nodo hoja. Por supuesto, esto depende del dominio de aplicación.

Dado que los documentos XML generalmente se modelan como árboles etiquetados, esta distancia puede utilizarse también para medir las diferencias estructurales entre dos documentos XML.

Coeficiente de Jaccard

Si quisiéramos medir la distancia en un tipo diferente de datos, como lo son los conjuntos, podemos utilizar el denominado coeficiente de Jaccard. Asumiendo dos conjuntos A y B, el coeficiente de Jaccard se define como:

$$d(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

Esta función se basa simplemente en la razón entre las cardinalidades de la intersección y la unión de los conjuntos comparados. Como un ejemplo de una aplicación que trabaja con conjuntos, supongamos que tenemos un registro con usuarios y direcciones de sitios web que visitó cada uno. Para evaluar la similitud en el comportamiento de cada usuario, podemos utilizar el coeficiente de Jaccard.

2.5. Algoritmos de Indexación

Los algoritmos de indexación sirven para organizar la información o universo de datos en estructuras de datos o índices. Este pre proceso de la base de

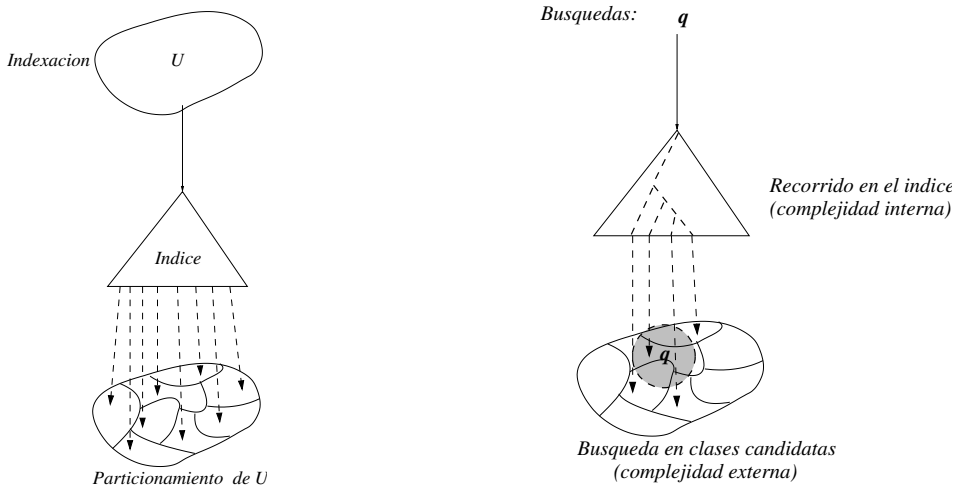


Figura 2.2: *Modelo general de algoritmos de indexación para Espacios Métricos.*

datos tiene como objetivo reducir el número de cálculos al momento de resolver una búsqueda. Un diseño eficiente de los *algoritmos de indexación*, junto con la desigualdad triangular, permite descartar elementos evitando comparar la query con cada uno de los elementos de la base o universo de datos. Así, dada una query, primero se recorre el índice para determinar un subconjunto de elementos candidatos a ser parte de la respuesta y luego se examinan esos conjuntos en forma exhaustiva para obtener la respuesta definitiva.

2.5.1. Un Modelo Unificado

Todos los algoritmos de indexación particionan la base de datos U en subconjuntos U_I. El índice permite identificar una lista de U_i que potencialmente contiene elementos significativos para la consulta. Por consiguiente, la búsqueda se reduce a:

- Recorrer el índice para obtener los candidatos. El costo de este proceso se denomina *complejidad interna*
- Examinar exhaustivamente estos candidatos a fin de encontrar los elementos que realmente forman el resultado de la búsqueda. El costo de este proceso se denomina *complejidad externa*.

La Figura 2.2 ejemplifica este modelo general de indexación. La figura de la izquierda muestra la división en clases de equivalencia del espacio métrico. La figura de la derecha muestra las clases de equivalencia que potencialmente

contienen elementos relevantes para una búsqueda $q(r, d)$.

Para comprender totalmente el proceso de indexación se necesita entender adecuadamente los conceptos de relación de equivalencia y partición de conjuntos. Aquí, la relevancia de dichos conceptos se debe a la posibilidad de particionar un espacio métrico en clases de equivalencias y obtener así un nuevo espacio métrico derivado del conjunto cociente.

Toda partición $\pi(X) = \{\pi_1, \pi_2, ..., \pi_n\}$ de un conjunto X induce una relación de equivalencia (que denotamos con $\tilde{}$). Inversamente toda relación de equivalencia induce una partición:

$$\forall x, y \in X : x\ y \Leftrightarrow x \text{ está en la misma parte que } y$$

Las clases de equivalencia $[x]$ de esta relación se corresponden con las partes π_i de la partición π, es decir,

$$\pi(X) = X/\tilde{}$$

Por consiguiente, los algoritmos de indexación existentes definen una relación de equivalencia sobre el espacio métrico X. Las clases de equivalencia de X en el conjunto cociente $\pi(X)$ pueden considerarse como elementos de un nuevo espacio métrico, bajo alguna función de distancia $D : \pi(X) \times \pi(X) \to \Re^+$.

Ahora definimos la función D_0 que será de utilidad para encontrar la función D.

$$D_0 : \pi(X)(X) \to \Re^+ D_0([x], [y]) : \min_{x \in [x], y \in [y]} \{d(x, y)\}$$

D_0 representa la menor distancia entre un elemento de la clase $[x]$ y un elemento de la clase $[y]$. Esta distancia es el máximo valor posible que mantiene el mapeo contractivo, es decir que $\forall x, y \in X : D_0([x], [y]) \leq d(x, y)$.

D_0 no puede ser utilizada con la finalidad de indexación porque no satisface la desigualdad triangular pero nos aproxima a una solución dado que sabemos que cualquier función de distancia D, que además cumple con las propiedades para ser una métrica, es una cota inferior de D_0 (manteniendo así el mapeo contractivo) que nos sirve para definir un nuevo espacio métrico. En otras palabras, esta nueva función D debe cumplir que $\forall [x], [y] en \pi(X), D([x], [y]) \leq D_0([x], [y])$.

Luego, podemos convertir un problema de búsqueda en otro, que esperamos sea más sencillo. Para una búsqueda $(q, r)_d$ primero, buscamos espacio $\pi(X, D)$ obteniendo como resultado $([q], r)_D = \{u \in U/D([u], [q]) \leq r\}$. Como D_0 es contractiva, podemos asegurar que $(q, r)_d \subseteq ([q], r)_D$. Luego realizamos una

búsqueda exhaustiva en $([q], r)_D$ a fin de determinar los elementos que forman parte de la respuesta a la consulta $(q, r)_d$.

Hay dos enfoques generales para crear estas relaciones de equivalencia: uno está basado en pivotes y el otro se basa en particiones compactas o tipo Voronoi. Ambos se describen a continuación.

2.5.2. Algoritmos Basados en Pivotes

Los algoritmos de pivotes definen una relación de equivalencia basados en la distancia de los elementos a un conjunto de elementos preseleccionados que llamaremos *pivotes*.

Sea $\{p_1, p_2, ..., p_k\}$ un conjunto de pivotes, dos elementos son equivalentes si y sólo si están a la misma distancia de todos los pivotes:

$$x\ p_i y \Leftrightarrow d(x, p_i) = d(y, p_i) \forall_{i=1....k}$$

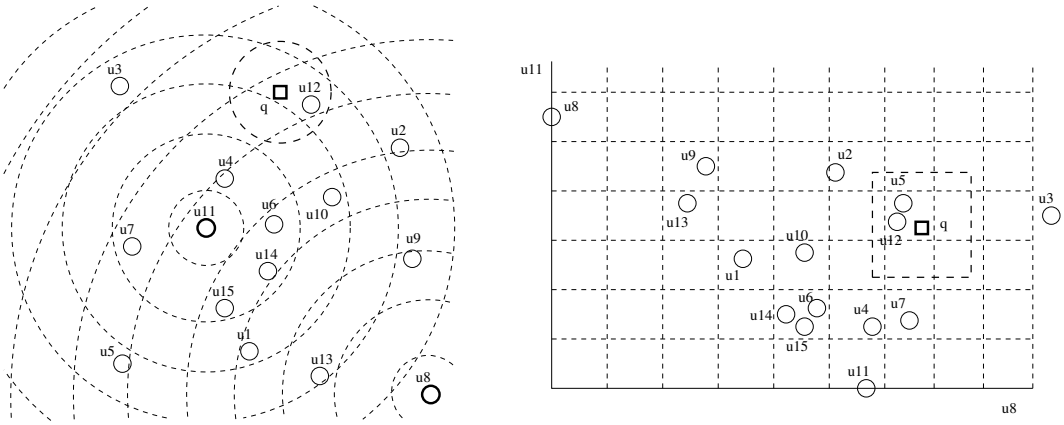


Figura 2.3: *Sobre la izquierda, un ejemplo de la relación de equivalencia inducida por la intersección de anillos centrados en dos pivotes, u_8 y u_{11}. Sobre la derecha se ilustra la transformación en un espacio vectorial de dimensión 2. También se grafica la transformación de una búsqueda $(q, r)_d$.*

Se puede ver que cada clase de equivalencia está definida por la intersección de varias capas de esferas centradas en los puntos p_i como se muestra en la Figura 2.3.

Veamos ahora cuál sería una función de distancia D para las clases de equivalencia. Por la desigualdad triangular, para cualquier $x \in X$, se cumple que:

- 1. $d(p, x) \leq d(p, y) + d(y, x) \Leftrightarrow d(p, x) - d(p, y) \leq d(y, x)$

- 2. $d(p,y) \leq d(p,x) + d(x,y) \Leftrightarrow d(p,y) - d(p,x) \leq d(x,y)$
- 3. $d(p,x) - d(p,y) \leq d(y,x) \wedge d(p,y) - d(p,x) \leq d(x,y) \Leftrightarrow |d(x,p) - d(y,p)| \leq d(x,y)$

Esto significa que, para cualquier elemento p la distancia $d(x,y)$ no puede ser menor que $|d(x,p) - d(y,p)|$. Luego $D([x],[y]) = |d(x,p) - d(y,p)|$ es un límite inferior seguro para D_0. Si extendemos D para k los pivotes:

$$D([x],[y]) = max_{1 \leq i \leq k}\{|d(x,p_i) - d(y,p_i)|\}$$

Esta función de distancia D limita inferiormente a d y en consecuencia puede usarse como distancia en el espacio cociente.

La relación de equivalencia definida por un conjunto de k pivotes también puede considerarse como una proyección al espacio vectorial $\Re^k$. La i-ésima coordenada de un elemento es la distancia al i-ésimo pivote. Luego, a cada elemento x del espacio le corresponde el vector $\delta(x) = (d(x,p_1), d(x,p_2), ..., d(x,p_k)) \in \Re^k$. Entonces, dada un consulta de búsqueda $(q,r)_d$ debemos encontrar un conjunto de elementos candidatos $([q],r)_D = x : D([q],[x]) \leq r$. En este caso particular significa encontrar los elementos tales que:

$$max_{1 \leq i \leq k}\{|d(x,p_i) - d(q,p_i)|\} = L_\infty(\delta(x), \delta(q)) \leq r$$

Esto significa que hemos proyectado el espacio métrico original (X,d) en el espacio vectorial $\Re^k$ con la función de distancia L_∞. La Figura 2.3 ilustra estas ideas para un conjunto de puntos usando sólo dos pivotes.

2.5.3. Algoritmos Basados en Particiones Compactas

La idea en este caso es dividir el espacio en particiones o zonas lo más compactas posibles, almacenando puntos representativos de dichas zonas, denominados centros, y algunos datos extra que permitan descartar zonas completas lo más rápidamente posible al momento de realizarse una consulta. Cada zona, a su vez, puede ser recursivamente particionada en más zonas, lo cual induce una jerarquía de búsqueda.

Criterio de partición de Voronoi

Los Diagramas de Voronoi [Aur91] han sido usados para búsquedas por proximidad en espacios vectoriales. Estos diagramas son una de las estructuras fundamentales dentro de la Geometría Computacional, de alguna forma ellos almacenan toda la información referente a la proximidad entre puntos.

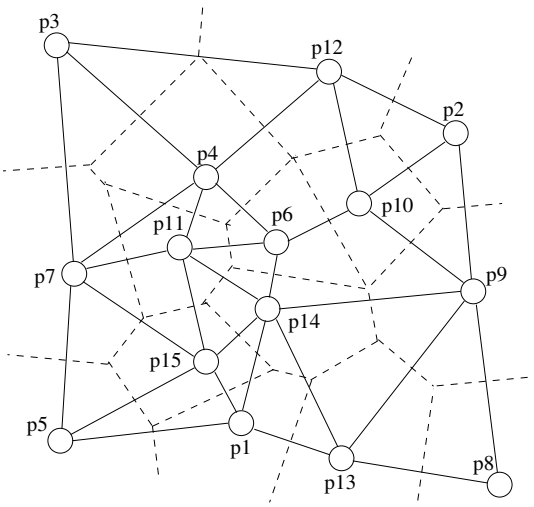


Figura 2.4: *Diagrama de Voronoi (líneas punteadas) para un conjunto de puntos en $\mathbb{R}^2$ con distancia L_2; y su concepto dual la Triangulación de Delanauy.*

Se elige un conjunto de m centros $c_1, c_2, ..., c_m$. El resto de los objetos se asignan a la zona de su centro más cercano. Cuando todos los objetos de la base de datos son centros, el concepto de partición de Voronoi descrito coincide con el concepto de dominio de Dirichlet [3]. La Figura 2.4 ejemplifica este concepto en un espacio vectorial de dimensión 2.

Al momento de realizar una consulta $(q, r)_d$, se evalúan las distancias entre q y los m centros, eligiendo el centro más cercano a q, el cual se denominará c.

Si la bola de la consulta $(q, r)_d$ intersecta la zona de algun centro distinto a c, entonces se debe revisar exhaustivamente esa zona para ver si hay objetos que caigan dentro de la bola de consulta. Sea $x \in X$ un objeto perteneciente a la zona cuyo centro es c_i y que se encuentra a distancia menor o igual que r de la consulta q. Por la desigualdad triangular se tiene que:

1. $d(c, x) \leq d(c, q) + d(q, x) \leq d(c, q) + r$
2. $d(c_i, q) \leq d(c_i, x) + d(x, q) \leq d(c_i, x) + r \Rightarrow d(c_i, q) - r \leq d(c_i, x)$

Como x pertenece a la zona de c_i se cumple que $d(c_i, x) \leq d(c, x)$. De esta condición más (1) y (2) se obtiene:

3. $d(c_i, q) - r \leq d(c_i, x) \leq d(c, x) \leq d(c, q) + r \Rightarrow d(c_i, q) - r \leq d(c, q) + r$

Dada la condición (3), se pueden descartar las zonas cuyo centro c_i satisfaga la condición $d(c_i, q) > d(c, q) + 2r$, dado que en ese caso dicha zona no puede tener intersección con la bola de consulta.

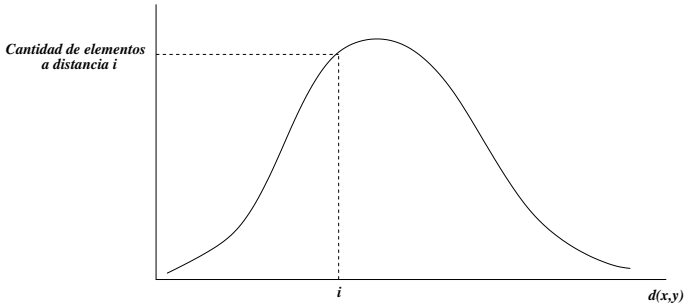


Figura 2.5: *Histograma de distancias de un espacio métrico*

2.6. Dimensión en Espacios Métricos

Uno de los grandes obstáculos en el diseño de algoritmos de búsqueda por similitud es la llamada maldición de la dimensionalidad [CNBYM01]. Si se pudiera representar el conjunto de elementos de un espacio métrico como puntos en un espacio vectorial, su dimensión correspondería al número de coordenadas que tienen los puntos que lo componen.

Todos los algoritmos de búsqueda por similitud disminuyen su rendimiento cuando aumenta la dimensión del conjunto, hecho conocido como maldición de la dimensionalidad.

En [CNBYM01] se muestra que el concepto de dimensión de un espacio vectorial se puede extender a espacios métricos generales utilizando el concepto de dimensión intrínseca, y que la maldición de la dimensionalidad tiene consecuencias similares en dichos espacios.

La distribución de distancias de un espacio métrico se define como la probabilidad de que dos elementos se encuentren a cierta distancia l. Una aproximación a esta distribución es el *histograma de distancias*, el cual se construye desde un subconjunto del espacio métrico, calculando las distancias entre los elementos del subconjunto (ver figura 2.5).

El histograma de distancias permite visualizar la distribución de los elementos del espacio métrico y se menciona en varios artículos como una medida fundamental relacionada a la dimensión intrínseca del espacio [Bri95, CM97, CN00, CN01, CPZ98]. A medida que la dimensión intrínseca de un espacio métrico aumenta la media μ del histograma crece y su varianza σ^2 se reduce.

En [CNBYM01] se define la dimensión intrínseca de un espacio métrico como:

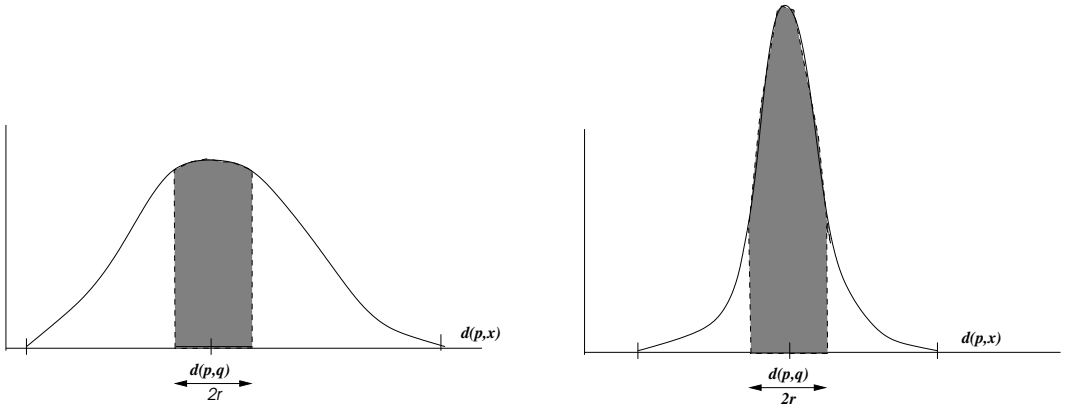


Figura 2.6: *Histogramas de distancias de baja dimensionalidad (izquierda), y de alta dimensionalidad (derecha)*

$$\gamma = \frac{\mu^2}{2\sigma^2}$$

Los parámetros μ y σ^2 son la media y la varianza, respectivamente, del histograma de distancias de los puntos que conforman dicho espacio métrico.

La figura 2.6 da una idea intuitiva de por qué el problema de búsqueda se torna más difícil cuando el histograma es más concentrado.

Consideremos una búsqueda $(q, r)_d$ y un índice basado en pivotes elegidos aleatoriamente. En la figura se ejemplifican dos posibles casos para el histograma local respecto del punto p. Si p es un pivote, estas gráficas representan dos posibles distribuciones para los valores de $d(q, p)$. La regla de eliminación dice que podemos descartar aquellos puntos y tales que $y \notin [d(p, q) - r, d(p, q) + r]$. Las áreas sombreadas muestran los puntos que no podrán descartarse. Esto significa que a medida que el histograma se concentra más alrededor de su media disminuye la cantidad de puntos que pueden descartarse usando como dato $d(p, q)$.

Este fenómeno es independiente de la naturaleza del espacio métrico, y nos brinda una forma de cuantificar cuán difícil es una búsqueda sobre el mismo. Tal como se muestra experimental y analíticamente en [CNBYM01], todos los algoritmos degradan sistemáticamente cuando la media ρ del espacio se incrementa.

Capítulo 3

Una Aplicación de Espacios Métricos a Comercio Electrónico

3.1. Introducción

El e-commerce (o comercio electrónico) es un modelo de negocio que gestiona compra y venta de productos y/o servicios, y utiliza Internet como principal medio de intercambio. Esto también incluye gestión de cobros, pagos y distribución de productos. El desarrollo tecnológico ha permitido que las comunicaciones crezcan de una manera exponencial, impactando directamente en el crecimiento del comercio electrónico.

Dentro de los desafíos que implica construir un sitio de e-commerce exitoso, sin duda uno de los más importantes es contar con un catálogo de productos bien organizado, donde el cliente no solo encuentre rápidamente lo que busca, sino que los resultados ofrecidos sean relevantes a lo que necesita.

En este trabajo nos focalizamos en dicho problema, utilizando la búsqueda por similitud para resolverlo.

3.2. Sistema de Recomendación

Con el crecimiento de la web y del negocio electrónico, el volumen de datos e información que se maneja ha crecido exponencialmente, con lo cual las búsquedas se enfrentan a desafíos de rendimiento y velocidad de respuesta, dejando en segundo plano la congruencia de datos que ofrecen los modelos relacionales (búsquedas exactas).

La forma más conocida y popular de almacenar información se basa en las

relaciones entre los distintos atributos (bases de datos relacionales). En éste tipo de universo, las consultas vienen dadas por una especificación de criterios que deben cumplirse para que un objeto forme parte del resultado; pero ¿qué pasa cuando en la consulta no podemos definir ese conjunto de criterios en forma determinística?

En el campo del ecommerce, a menudo se necesita seleccionar una de varias alternativas sin tener un conocimiento exacto de cada una de ellas. La decisión final, en estos casos, puede depender de las recomendaciones realizadas por el propio sistema. La recomendación de productos a clientes es un caso de recuperación de información que no se puede abordar con la visión clásica de bases de datos relacionales.

Los sistemas de recomendaciones son herramientas que generan sugerencias sobre un determinado objeto de estudio, a partir de las preferencias y opiniones dadas por los usuarios. El uso de estos sistemas va en aumento debido a que son muy útiles para evaluar y filtrar la gran cantidad de información disponible en la Web. El objetivo final es asistir a los usuarios en sus procesos de búsqueda y recuperación de información.

Un sistema de recomendación es un software que filtra información de interés para el usuario con el fin de proponerle aquel producto más adecuado a sus necesidades. Estos sistemas evalúan cuál es el grado de interés de un usuario por ciertos productos y buscan productos similares y con una alta probabilidad de atraer su atención.

Las recomendaciones pretenden ser una forma de ayudar al usuario a encontrar productos de su agrado realizando una preselección basándose, por ejemplo, en su historial de búsquedas. Esto proporciona un alivio a los consumidores, ya que les evita tener que recorrer una lista interminable de ofertas poco relevantes. Por otro lado, se espera que estos beneficios para los usuarios se terminen reflejando en aumentos de tráfico y ventas, ya que en el e-commerce, las buenas recomendaciones siempre conducen a incrementos en las compras y, por ende, en los márgenes de ganancia. Es por ello que la efectividad del sistema de recomendación elegido es crucial.

El funcionamiento de un sistema de recomendación está basado en cierta información que es procesada por los algoritmos de filtrado. Dependiendo de la naturaleza de dicha información, podemos clasificar estos algoritmos en: basados en contenido, colaborativos, sensibles al contexto, entre otros.

En este trabajo nos hemos centrado en los algoritmos basados en contenido. Estos algoritmos filtran **objetos o contenidos similares** a los que el usuario ya

ha buscado, comprado o calificado positivamente. Por ejemplo, en las plataformas de reproducción de música online, el software evalúa las piezas musicales analizando su estructura interna para encontrar piezas similares que podrían tener una línea de bajo parecida.

Los sistemas de recomendación tienen aplicación en varias áreas. Quizás los tres sectores más importantes son: los servicios de transmisión audiovisual en línea (como por ejmeplo Spotify o Netflix), las tiendas electrónicas (como Amazon) y los sistemas de publicidad basada en contenido.

3.3. Problema abordado

En la plataforma de e-commerce Mercado Libre los usuarios publican productos dentro de un árbol de categorías; dichos productos constan de un título principal, un título secundario y una descripción.

El problema que abordaremos en este trabajo es cómo encontrar los productos similares a un producto específico, obteniendo de esa manera recomendaciones para los usuarios. Aplicaremos la teoría de espacios métricos a este caso de e-commerce real, donde nuestro universo de datos será un conjunto de productos disponibles en la plataforma Mercado Libre, y la función que provee la medida de distancia, será la distancia de edición (o Levenshtein) aplicada al título principal de los productos.

El objetivo de este trabajo es, teniendo en mente un sistema de recomendación, encontrar una forma eficiente de resolver la búsqueda por similitud evaluando técnicas conocidas en este caso real de aplicación. Pretendemos que este trabajo sea el paso inicial para a futuro construir un sistema de recomendación basado en el modelo de espacios métricos.

Para lograr nuestro objetivo, primero organizamos los productos por categoría con el enfoque clásico de búsqueda exacta y luego dentro de cada categoría aplicamos el enfoque de espacios métricos.

En consecuencia, la búsqueda de un producto se divide en dos etapas:

1. Filtramos por categoría, realizando una búsqueda exacta.
2. Realizamos la búsqueda por similitud sobre los productos que pertenecen a la categoría elegida.

Para el filtrado por categoría armamos un hashing lineal en memoria principal. Para la búsqueda por similitud utilizamos el enfoque basado en pivotes: elegimos k

pivotes y para cada producto x almacenamos su firma $(d(x, p_i), \ldots, d(xp_k)$ donde p_i es el iésimo pivote y d es la función de distancia de edición. En el Capítulo 4 brindamos una descripción detallada de esta propuesta.

Un punto crucial es la técnica de selección de pivotes usados para construir la firma de los productos. La forma en que los pivotes son seleccionados puede afectar drásticamente la performance de un algoritmo. Una buena elección de pivotes puede en gran parte reducir los tiempos de búsqueda.

En nuestro caso elegimos la técnica de selección incremental de pivotes [BNC01] que ha demostrado ser eficiente en los espacios métricos normalmente usados por la comunidad científica que trabaja en búsqueda por similitud. Esta técnica la comparamos con la selección aleatoria de pivotes, para poder tener una base de comparación sobre su desempeño en este caso de estudio. Explicamos a continuación ambas técnicas.

3.4. Técnicas de Selección de Pivotes Utilizadas

Para lograr minimizar el número de evaluaciones de distancia que se realizan al momento de hacer una búsqueda por rango, los pivotes elegidos deben descartar la mayor cantidad de elementos posibles antes de realizar una búsqueda dentro de una lista de elementos candidatos (Ver sección 2.5.1). Esto significa que mientras más pequeña sea la lista de elementos candidatos generados mejor será el conjunto de pivotes utilizado.

Sea (X, d) un espacio métrico. Un conjunto de pivotes $\{p_1, p_2, ..., p_k\} \in X$ definen un espacio P de tuplas de distancias entre pivotes y elementos del conjunto. Un elemento $x \in P$ se denotará como $[x]$ que es igual a la siguiente ecuación:

$$[x] = (d(x, p_1), d(x, p_2), ..., d(x, p_k)), x \in X, [x] \in P \tag{3.1}$$

Definimos la métrica $D = D_{\{p_1,...,p_k\}}$ del espacio P como:

$$D([q], [x]) = max_{i=1}^{k} |d(q, p_i) - d(x, p_i)| \tag{3.2}$$

Luego, obtenemos el espacio métrico (X, D) que resulta ser $(\Re^k, L_\infty)$.

Para lograr que la cantidad de elementos elegidos sea pequeña se debe maximizar la probabilidad de que $D_{\{p_1,...,p_k\}}([q],[x]) > r$. Una forma de lograr esto es maximizar la media de la distribución de distancias en P, la cual la llamaremos μ_p.

La estimación del μ_p se realiza de la siguiente manera:

- Elegir A pares de elementos $\{(a_1, a'_1), (a_2, a'_2), ..., (a_A, a'_A)\}$ del conjunto E, distintos entre sí.
- Mapear los A pares de elementos al espacio P y calcular la distancia D entre cada par de elementos, esto produce como resultado el conjunto finito de distancias $\{D_1, D_2, ..., D_A\}$.
- Luego de obtener las A distancias se estima el valor de p de la siguiente forma:

$$\mu_p = \frac{\sum_{i=1}^{A} D_i}{A} \tag{3.3}$$

De las ecuaciones 3.1 y 3.2, se puede deducir que dados un par de elementos (a, a') el costo de calcular $D([a],[a'])$ es $2k$ evaluaciones de la función d.

Se se utilizan A pares de elementos para estimar el valor de μ_p , se puede ver que el costo total de la estimación es de $2kA$ evaluaciones de la función d.

Teniendo estas ideas presentes, veamos en qué consisten las dos técnicas de selección de pivotes utilizadas.

Selección Aleatoria

Esta técnica consiste en la elección al azar de los pivotes, no se usa ningún criterio de selección. En este trabajo mostraremos la diferencia o beneficios en las búsquedas al usar pivotes seleccionados usando técnicas incrementales versus la selección aleatoria de pivotes.

Claramente la principal ventaja de esta técnica es que el costo de optimizar la selección es 0.

Selección Incremental

El método consiste en elegir un pivote p_1 utilizando A pares de elementos del espacio E mapeados a P, tal que ése pivote maximice el μ_p. Luego elegir un segundo pivote p_2, tal que $\{p_1, p_2\}$ maximicen μ_p pero p_1 ya queda fijo. Luego elegir un tercer pivote p_3, tal que $\{p_1, p_2, p_3\}$ maximicen μ_p pero con p_1 y p_2 fijos. Repetir el proceso hasta elegir los k pivotes.

En cada iteración se elige un pivote de una muestra de tamaño X del espacio E, dado que buscar un elemento que maximice μ_p dentro del todo el conjunto E sería muy costoso.

Si bien en cada iteración se estima el μ_p con los i pivotes seleccionados hasta el momento, no es necesario rehacer el cálculo completo si se almacena $D_{\{p_1,...,p_{i-1}\}}([a_r],[a'_r])\forall r \in 1..A$, es decir, para cada valor de $r = 1..A$ el valor máximo de $|d(a'_r, p_j) - d(a_r, p_j)|, j = 1..i-1$. En este caso solo se calcula la distancia con respecto a los pivotes candidatos $|d(a'_r, p_{cand}) - d(a_r, p_{cand})|, r = 1..A$ y se toma el valor máximo entre las dos distancias para el calculo de $D\{p_1, ..., p_i\}$.

Notar que, si la muestra de donde se toman los pivotes candidatos es de tamaño X, en cada iteración se realizan $2AX$ evaluaciones de la función d. Luego, si se eligen k pivotes, el trabajo total realizado por el algoritmo tiene un costo de $2kAX$ evaluaciones de la función d.

Capítulo 4

Detalles de implementación

4.1. Introducción

En el siguiente capítulo describiremos los detalles del trabajo realizado, desde la estructuración de las fuentes de datos utilizados, hasta la implementación de los algoritmos de búsqueda. Dado que el objetivo principal de éste trabajo es determinar (a través de la aplicación a un caso de uso real) que estrategia de selección de pivotes es la mejor, nos restringimos a implementar el método básico de búsqueda que puede utilizarse desde cualquier tipo de sistema: sitio web, aplicación de teléfono celular, etc.

Para el desarrollo del software seleccionamos el framework Grails (versión 1.3.7), que contiene el lenguaje Groovy para la codificación y Java para la ejecución.

Todas las estructuras de datos fueron manejadas en memoria principal, tanto para la creación de índices como para las búsquedas.

4.2. Procesamiento inicial de los datos de entrada

Cuando hablamos de datos de entrada, nos referimos a los productos ofrecidos en el sitio Mercado Libre (a los que también llamaremos items). Estos productos están clasificados dentro de un árbol de categorías, donde cada categoría reúne productos relacionados. Remitiendonos a los números, obtuvimos alrededor de 2 millones de productos, distribuidos en 12.000 categorías (hojas).

La información relevante a éste trabajo obtenida de los productos fue la siguiente: categoría, identificador del producto, título, descripción principal y descripción secundaria.

Para optimizar el desarrollo de creación de índices, los datos de entrada

tuvieron que ser pre procesados de manera de crear los archivos necesarios para el correcto funcionamiento de dicho proceso. De este pre procesamiento obtuvimos:

1. Archivo de productos: texto plano conteniendo toda la información pertinente de los productos, separada por comas.

2. Archivo de categorías: texto plano conteniendo el nombre de la categoría.

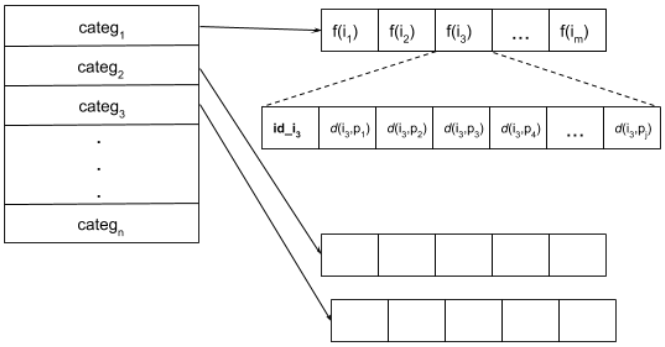
3. Archivo de productos divididos por categoría: archivo de datos serializados, conteniendo un mapa cuyas claves son las categorías y cuyo valor es la lista de productos correspondiente, conteniendo información mínima para la selección de pivotes: identificador del producto y codificación del título (la codificación consiste en eliminar caracteres especiales, reemplazar vocales acentuadas por no acentuadas, suprimir espacios superfluos y pasar a mayúsculas el título completo).

4.3. Estructuras de datos

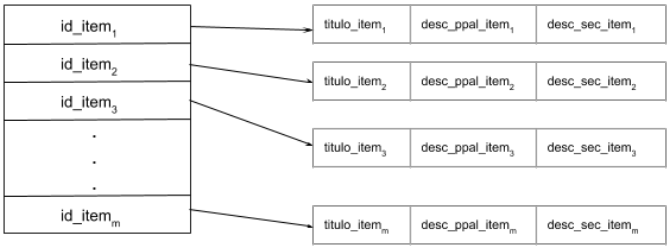
Para el almacenamiento de las categorías se implementó un rebalse abierto lineal con un factor de carga de *0,4*. Cada elemento del rebalse se compone de una estructura que contiene el nombre de la categoría y una lista de firmas. Cada firma representa la distancia de edición de cada título del ítem a cada título del elemento pivote, e incluye el identificador de ítem en cuestión.

Por otro lado, para el almacenamiento de los datos completos del producto se utilizó un mapa cuya clave es el identificador y el valor asociado es una estructura que contiene la totalidad de los datos del producto.

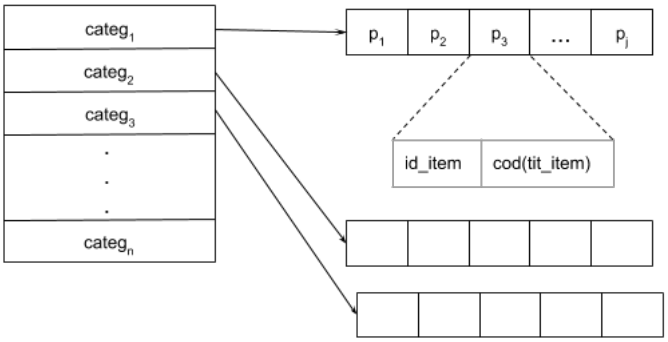
Como última estructura de datos auxiliar, se eligió un mapa cuya clave es la categoría y el valor asociado es la lista de pivotes correspondiente, conteniendo solo el identificador del producto y la codificación del título.



(a) Rebalse de categorías



(b) Mapa de productos



(c) Mapa de pivotes por categorías

Figura 4.1: Estructuras de datos utilizadas

4.4. Software desarrollado

El desarrollo realizado consta de dos partes complementarias: la funcionalidad de creación de índices y la funcionalidad de búsqueda propiamente dicha, que utiliza los índices creados por la primera funcionalidad.

4.4.1. Creación de índices para las búsquedas

La funcionalidad de creación de índices se desarrolló en forma genérica, preparada para recibir los siguientes parámetros: *tipo de pivotes* (aleatorio o incremental), *cantidad de pivotes* y *conjunto de pivotes* (mismo conjunto para todas las categorías o diferentes conjuntos para cada categoría).

La lógica básica del algoritmo de creación de índices carga los archivos descritos en la sección 4.2 en las estructuras de datos correspondientes, y procede a generar las firmas de los items; utilizando, en primera instancia, la estrategia de selección de pivotes en conjunción con el resto de los parámetros (cantidad y conjunto de pivotes).

Luego de completar la estructura con las firmas de los items, almacena cada estructura en un archivo de datos serializados; para que puedan ser utilizados en futuras búsquedas.

Algoritmo 4.1: Creación de índice

```
1  crearIndice(File fCategs, File fPivs, String estSel, int cantPivs)
2  begin
3   /* Se inicializan las estructuras */
4   Hash categsHash = inicializarHashCategorias(fCategs)
5   Map pivotesPorCateg
6   /* Se obtiene el conjunto de pivotes en base a la estrategia */
7   if(estSel == "aleatorio")
8   begin
9    pivotesPorCateg = obtConjPivsAleatorio(fPivs, cantPivs)
10  end
11
12  if(estSel == "incremental")
13  begin
14   pivotesPorCateg = obtConjPivsIncremental(fPivs, cantPivs)
15  end
16
17  Map items
18  /*Se calculan las firmas para todos los items*/
19  forall item in archivoItems
20  begin
21   Pivotes pivs = pivotesPorCateg.get(item.categ)
22   Dist[] d = calcularDistancias(item.cod_titulo, pivotes)
23   categsHash.get(item.categ).add(d)
24   items.put(item.id, item)
25  end
26  /* Se almacenan las estructuras que componen el indice en archivos
27   * para su utilizacion futura */
28  generarArchivoSerializado(categsHash)
29  generarArchivoSerializado(pivotesPorCateg)
30  generarArchivoSerializado(items)
31 end
```

A continuación se presenta el pseudo-código de los algoritmos de selección

de pivotes. Para facilitar la lectura de los mismos solo se incluyó la variante de distinto conjunto de pivotes para cada categoría.

Algoritmo 4.2: Selección de pivotes aleatoria

```
1  obtConjPivsAleatorio(File fPivs, int cantPivs)
2  begin
3   Map conjuntoFinal
4   Map pivotesPorCateg = obtenerPivotesPorCateg(fPivs)
5   for(categ, pivotes in pivotesPorCateg)
6   begin
7    /* Se eliminan elementos aleatoriamente hasta
8     * alcanzar la cantidad de pivotes deseada */
9    while(pivotes.size > cantPivs)
10   begin
11    eliminarElementoAleatorio(pivotes)
12   end
13   conjuntoFinal.put(categ, pivotes)
14  end
15  return conjuntoFinal
16 end
```

Algoritmo 4.3: Selección de pivotes incremental

```
1  obtConjPivsIncremental(File fPivs, int cantPivs)
2  begin
3   Map conjuntoFinal
4   Map pivotesPorCateg = obtenerPivotesPorCateg(fPivs)
5   for(categ, pivotes in pivotesPorCateg)
6   begin
7    conjuntoPivotes
8    int candidato, selec
9    float dim, max
10   for(i = 1 to cantPivs)
11   begin
12    /* Se elige un candidato inicial */
13    selec = random(n)
14    while(selec ya haya sido elegido antes)
15     selec = random(n)
16    conjuntoPivotes->puntos[i] = pivotes->puntos[selec]
17    /* 10 es la cantidad de pares de elementos para el calculo
18     * de la media D */
19    max = mediaD(conjuntoPivotes, 10)
20    candidato = selec
21    for j = 2 to 10
22     begin
23     /* Se busca nuevo cantidato */
24     selec = random(n)
25     while(selec ya haya sido escogido antes)
26      selec=random(n)
27     conjuntoPivotes->puntos[i] = pivotes->puntos[selec]
28     /* 10 es la cantidad de pares de elementos para el calculo
29      * de la media D */
30     dim = mediaD(conjuntoPivotes, 10)
31     if(dim > max)
32     begin
33      max = dim
34      candidato = selec
35     end
```

```
36      end
37      conjuntoPivotes->puntos[i] = pivotes->puntos[candidato]
38     end
39     conjuntoFinal.put(categ, pivotes)
40    end
41    return conjuntoFinal
42  end
```

Como se puede inferir entonces, el algoritmo de creación de indices particiona el universo de datos U (los productos) a través de las categorías hojas, resultando en multiples U_i listas que potencialmente contienen elementos relevantes para una consulta.

4.4.2. Busqueda por similitud

Se implementaron dos funcionalidades de búsqueda: búsqueda por rango, que recibe por parámetro el radio de búsqueda, la categoría del producto y el título de un producto existente; y búsqueda de los k-vecinos, que recibe por parámetro la cantidad de elementos a retornar (k), la categoría del producto y el título de un producto existente.

La búsqueda por rango calcula la distancia del título del producto dado por parámetro a cada uno de los pivotes, y luego compara esa firma contra las firmas de los productos de la categoría seleccionada. Este paso retorna los productos candidatos a formar parte de la respuesta final, los cuales son utilizados para calcular la distancia real de edición contra el producto buscado, descartando aquellos que estén fuera del rango especificado.

Algoritmo 4.4: Búsqueda por rango

```
1  busquedaPorRango(String q, String categ, int radio)
2  begin
3   Pivotes pivotes = pivotesPorCateg.get(categ)
4   /* Se calcula la firma para la query */
5   Dist[] d = calcularDistancia(q, pivotes)
6   List candidatos = []
7   /* Se obtienen todas las firmas para la categoria */
8   List[Dist[]] firmas = categsHash.get(categ)
9   /* Se compara la firma de la query con las firmas de la categoria,
10   * si el valor es mayor que el radio, se descarta el item */
11  for(j = 0 to firmas.size)
12  begin
13   int[] dists = firmas[j].dists
14   boolean agregar = true
15   for (i = 0 to dists.size)
16   begin
17    int value = valorAbsoluto(sig.dists[i] - dists[i])
18    if (value > radio)
19    begin
```

```
20        i = dists.size
21        agregar = false
22       end
23      end
24      if(agregar)
25       candidatos.add(firmas[j])
26    end
27    List itemsEncontrados = []
28    for(i = 0 to candidatos.size)
29    begin
30     Item item = items.get(candidatos[i].id)
31     int dist = distanciaEdicion(q, item.cod_titulo)
32     if((radio - dist) > 0)
33      itemsEncontrados.add(item)
34    end
35    return itemsEncontrados
36  end
```

La búsqueda de los k-vecinos utiliza una variación de la búsqueda por rango, comenzando con un rango de 5 e incrementando ese valor hasta llegar a la cantidad deseada de resultados.

Algoritmo 4.5: Búsqueda de los k-vecinos

```
1   busquedaKVecinos(String q, String categ, int radio, int k)
2   begin
3    int radio = 0
4    int i = 1
5    List resultadoFinal
6    List itemsTemp = []
7    while(itemsTemp.size < k)
8    begin
9     radio = potencia(5,i)
10    itemsTemp = busquedaPorRango(q, categ, radio)
11    /* Si se obtuvieron mas elementos que el k deseado, se procede
12     * a realizar una biseccion del radio */
13    if(itemsTemp.size > k)
14    begin
15     int li = potencia(5,i - 1)
16     int ls = radio
17     /* Se realiza una biseccion del radio */
18     while(li <= ls)
19     begin
20      radio = ((ls + li)/2)
21      itemsTemp = busquedaPorRango(q, categ, radio)
22      /* Se alcanzo el numero deseado de elementos, se finaliza la biseccion */
23      if(itemsTemp.size == k)
24      begin
25       li = ls + 1
26       resultadoFinal = itemsTemp
27      end
28      /* Se continua la biseccion para uno u otro lado, dependiendo
29       * de si se supero o no la cantidad de elementos */
30      else
31      begin
32       if(itemsTemp.size < k)
33       begin
34        li = radio + 1
35        radio = radio + 1
```

```
36        end
37        else
38          ls = radio - 1
39        end
40      end
41      if(itemsTemp.size != k)
42      begin
43        /* Si el ultimo resultado de la biseccion obtuvo menos elementos,
44         * se debe hacer la b\'usqueda con el valor final del radio */
45        if(itemsTemp.size < k)
46          itemsTemp = busquedaPorRango(q, categ, radio)
47        /* Se ordenan los elementos por su distancia y se seleccionan los k primeros */
48        ordenarPorMenorDistancia(itemsTemp)
49        resultadoFinal = itemsTemp.subList(0,k)
50      end
51    end
52    else
53      resultadoFinal = itemsTemp
54    i = i + 1
55  end
56  return resultadoFinal
57 end
```

Como funcionalidad auxiliar, se implementó un proceso de carga que puede ser utilizado al iniciar el programa, para cargar los archivos de índices previamente generados.

4.5. Re-particionado del universo

Luego de planificar, diseñar e implementar todas las estructuras de datos, realizamos pruebas manuales para asegurarnos del correcto funcionamiento del software completo.

Ante estas pruebas, detectamos que el particionado no era semánticamente correcto, ya que al elegir la categoría hoja del árbol, estabamos restringiendo demasiado el universo de búsqueda.

Representando la situación con un ejemplo, supongamos que deseamos recomendar productos similares al producto *"Samsung Galaxy A30 32 GB Blanco 3 GB RAM"*, la categoría hoja de dicho producto es *"A30"*, dentro del árbol: *"Celulares y Teléfonos >Celulares y Smartphones >Samsung >A30"*, si solo tenemos en cuenta los productos que pertenecen a la categoría *"A30"*, es probable que no encontremos, por ejemplo, celulares blancos de 32 GB de la marca Motorola. Por éste motivo, definimos volver a particionar el universo de productos, ésta vez utilizando la categoría inicial del árbol (Celulares y Teléfonos en el ejemplo anterior).

Con ésta nueva estrategia, obtuvimos 30 particiones distintas de nuestro universo, para las cuales realizamos los experimentos descritos en el próximo

capítulo.

Capítulo 5

Evaluación experimental

En este capítulo presentamos los resultados de ejecutar el algoritmo de búsqueda por rango usando dos técnicas de selección de pivotes: aleatoria y incremental.

Comenzaremos describiendo el seteo experimental, los experimentos realizados y luego daremos el análisis de los resultados obtenidos.

5.1. Seteo Experimental

Como mencionamos anteriormente, para los experimentos presentados en éste capítulo utilizamos los productos (*items*) publicados en la plataforma de e-commerce Mercado Libre, divididos en 30 categorías.

5.1.1. Elección de la cantidad de pivotes y agrupación de categorías

El hecho de que la cantidad de elementos de las categorías estuviera distribuído en un rango muy amplio (de 1034 elementos para la categoría más pequeña a 213578 elementos para la categoía más grande), nos obligó a segmentar dichas categorías en 4 grupos de acuerdo a su tamaño.

En base a éste mismo criterio, realizamos la elección de la cantidad de pivotes, teniendo en cuenta el porcentaje que los mismos representaban sobre el total de los items de cada categoría. La distribución resultante se presenta en la siguiente tabla:

Grupo	Cantidad de Categs	Rango del tamaño	Número de Pivotes
1	6	[1.034 - 15.964]	16, 32, 64, 128, 256
2	12	[19.032 - 46.530]	64, 128, 256, 512, 1024
3	8	[57.198 - 13.6323]	256, 512, 1024, 2048
4	4	[16.7995- 21.3578]	512, 1024, 2048, 4096

Cuadro 5.1: Tabla de grupos.

5.1.2. Selección del rango de búsqueda

La mayoría de los trabajos de investigación existentes sobre búsqueda por similitud en textos, utilizan como universo de datos diccionarios de palabras como por ejemplo inglés o español. Esto genera una base común que puede utilizarse a la hora de generar variaciones sobre algoritmos de búsqueda en nuevos trabajos, es decir, se comparte la base de datos y los parámetros de los experimentos, lo cual permite comparar fácilmente los resultados de los mismos entre sí.

Es común encontrar trabajos de búsqueda por rango donde la variación del radio es siempre entre 1 y 5 y la elección de ese valor no requiere más que una simple decisión azarosa por parte del autor.

En nuestro trabajo, el universo de datos es bastante más singular, ya que se trata de títulos de productos reales, cuya redacción está a cargo del usuario que publica el producto para su venta y donde la única limitante es el tamaño de ese título (60 caracteres).

Esta particularidad tiene como consecuencia un universo de datos variado y heterogéneo, donde cada elemento de dicho universo es una combinación de palabras, abreviaciones, números y caracteres especiales. Ante ésta combinación de características, el primer obstáculo que debimos sortear para comenzar con la evaluación experimental fue, precisamente, la selección del radio de búsqueda.

Para realizar una primera aproximación al radio de búsqueda ideal, seleccionamos 4 títulos de productos (cuadro 5.3) de la categoría con mayor cantidad de elementos (cuadro 5.2) y luego procedimos a realizar una búsqueda de los k-vecinos con las siguientes variaciones:

- $k = 0{,}001 \times cantidad\_de\_elementos\_de\_la\_categoria$ (0,1 %)
- $k = 0{,}01 \times cantidad\_de\_elementos\_de\_la\_categoria$ (1 %)
- $k = 0{,}1 \times cantidad\_de\_elementos\_de\_la\_categoria$ (10 %)

Las búsquedas se realizaron utilizando la estrategia de selección de pivotes incremental, y la cantidad de pivotes elegida fue de 128.

Tamaño de base de datos	Número de k-vecinos		
	0.001 %	0.01 %	0.1 %
31.632	32	316	3163

Cuadro 5.2: Número de k-vecinos utilizados.

Títulos de búsqueda	Radio promedio		
	0.001 %	0.01 %	0.1 %
Libro Te Amo Pero Soy Feliz Sin Ti. Jaime Jaramillo	32	35	37
El Secreto Rhonda Byrne Lvbp13	21	24	26
Libro De Italiano Forza 2	14	17	23
Libro La Magia Rhonda Byrne El Secreto Lvbp13	28	32	35

Cuadro 5.3: Muestra para determinar el radio de búsqueda.

Títulos de búsqueda	Radio Promedio
Libro Te Amo Pero Soy Feliz Sin Ti. Jaime Jaramillo	32
El Secreto Rhonda Byrne Lvbp13	21
Libro De Italiano Forza 2	14
Libro La Magia Rhonda Byrne El Secreto Lvbp13	28
PROMEDIO GENERAL	**27**

Cuadro 5.4: Tabla de radio promedio.

Como podemos visualizar en los resultados de la tabla 5.4, el promedio del radio de búsqueda arroja un valor de 27, el cual utilizamos para realizar algunas pocas búsquedas sobre el resto de las categorías. Al analizar los resultados obtenidos, llegamos a la conclusión de que debíamos disminuír el valor del r, ya que en muchos casos la búsqueda retornaba una cantidad de elementos cercana a la

totalidad de la base de datos, y en otros casos los elementos retornados no eran similares al título de búsqueda. Ante éstos hallazgos, tomamos una segunda aproximación: primero obtuvimos el promedio de la longitud de los títulos para cada categoría, luego promediamos esos valores para obtener un único resultado y lo dividimos por 2. De esa forma llegamos al rango elegido para todos los experimentos: $r = 23$.

5.1.3. Criterio de eficiencia

Como se mencionó en el capítulo 2, el criterio de eficiencia del algoritmo de búsqueda, para cuando las estructuras se manejan en memoria principal, es la cantidad de evaluaciones de la función de distancia. Este criterio es suficiente para cuando se evalúan los experimentos realizados en una sola base de datos, ya que puede compararse directamente en función de la cantidad total de elementos.

En el ámbito de este trabajo, cada categoría representa una base de datos por sí sola, ya que la búsqueda utilizando los pivotes se realiza sólo dentro de los elementos de cada categoría.

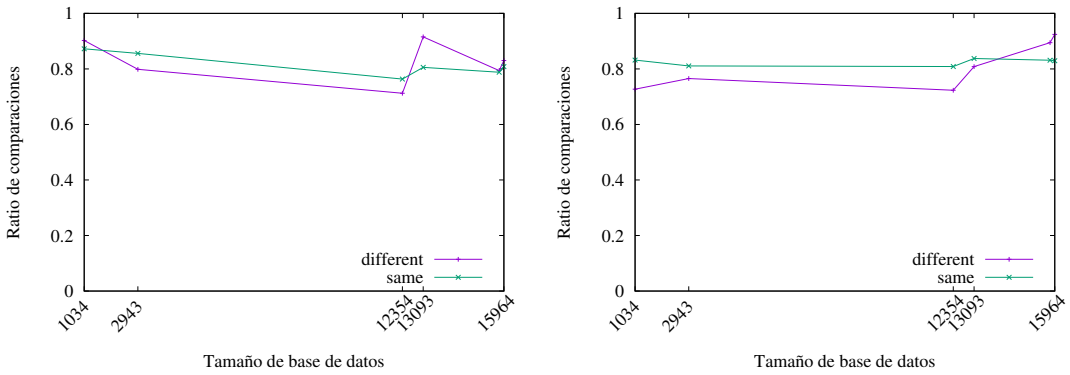
Por esto consideramos pertinente agregar otro criterio de eficiencia, que nos permite comparar los resultados de los experimentos realizados entre todas las categorías: el ratio de comparaciones. Definimos este criterio como la cantidad de evaluaciones de la función de distancia sobre la cantidad de elementos de la categoría, y de esta forma podemos obtener valores comparables independientemente de la cantidad de elementos que tenga la base de datos que estamos analizando.

5.1.4. Selección de los conjuntos de pivotes

En este trabajo, además de las técnicas de selección de pivotes, aleatoria e incremental, consideramos dos políticas para la elección de los grupos de pivotes:

Mismo grupo de pivotes para todas las categorías: Esta estrategia contempla un solo grupo de pivotes seleccionado al azar considerando todos los elementos del universo, sin importar la categoría a la que pertenecen. Luego, ese único grupo de pivotes es utilizado en cada uno de los experimentos sobre las distintas bases de datos.

Diferentes grupos de pivotes por categoría: En esta estrategia los conjuntos de pivotes se seleccionan en forma aleatoria sobre cada una de las bases de datos , es decir, cada categoría cuenta con su propio grupo de pivotes seleccionados dentro de los elementos de dicha categoría.



(a) Técnica de selección incremental (b) Técnica de selección aleatoria

Figura 5.1: Estrategia de selección para 16 pivotes: mismos pivotes vs diferentes

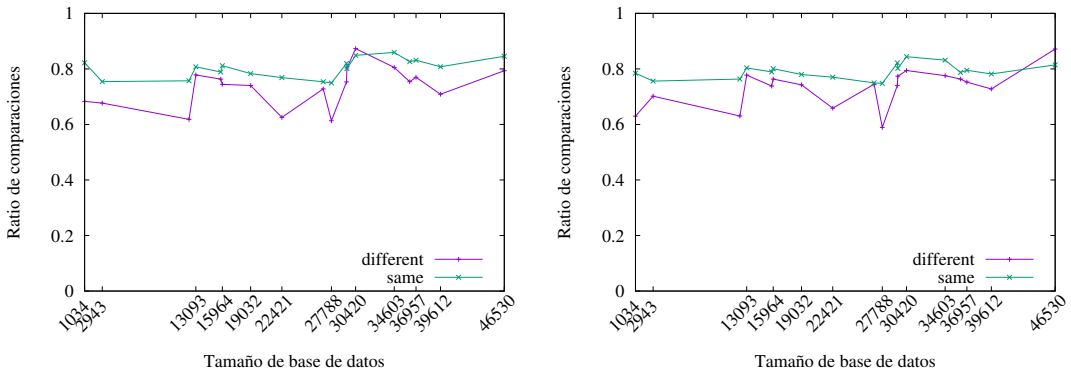
Para determinar cuál estrategia era la más adecuada, se seleccionaron conjuntos de pivotes, aleatoria e incremental, según las políticas arriba mencionadas para los grupos de 16 y 64 pivotes.

Por cada base de datos se eligió al azar el 10 % de los títulos como elementos de búsqueda. Para las búsquedas con 16 pivotes, se usaron 6 bases de datos (tamaño total: 61.184) y se ejecutaron 12.232 búsquedas en total. Para las búsquedas con 64 pivotes, se usaron 18 bases de datos (tamaño total: 440.631) y se ejecutaron un total de 88.114 búsquedas.

En la figura 5.1, se muestra el ratio de comparaciones para las búsquedas con 16 pivotes. Para la técnica de selección incremental (a) se puede observar que se comporto mejor la política *Mismo grupo de pivotes para todas las categorías* y para la técnica de selección aleatoria (b) fue mejor la política *Diferentes grupos de pivotes por categoría*.

En la figura 5.2, se visualiza el ratio de comparaciones para las búsquedas con 64 pivotes. Para ambas técnicas de selección, incremental (a) y aleatoria (b), se puede observar que se comporto mejor la política *Diferentes grupos de pivotes por categoría*. Esto es, el ratio de comparaciones realizadas directamente con los objetos de la base de datos es menor al usar diferentes pivotes por categoría.

En base a estos resultados, seleccionamos la política *Diferentes grupos de pivotes por categoría* para realizar la ejecución de la totalidad de los experimentos presentados en la siguiente sección.



(a) Técnica de selección incremental

(b) Técnica de selección aleatoria

Figura 5.2: Estrategia de selección para 64 pivotes: mismos pivotes vs diferentes

5.2. Ejecución experimental

Se seleccionó al azar el 10 % de los elementos de cada una de las bases de datos para realizar las búsquedas. La cantidad total de búsquedas por rango realizadas fue: 339.899. Este valor incluye las búsquedas usadas para descartar la política de selección *Mismo grupo de pivotes para todas las categorías* mencionada anteriormente.

En total se ejecutaron 276 experimentos. Esto es, 138 experimentos usando técnica de selección de pivotes aleatoria y 138 experimentos usando técnica de selección de pivotes incremental.

Los experimentos fueron ejecutados en una computadora portátil MacBook Pro, con un procesador 2,5 GHz Intel Core i7 y 16GB de memoria RAM.

5.3. Análisis de resultados

Vamos a presentar los resultados segmentados en cuatro grupos y analizaremos tres enfoques diferentes.

5.3.1. Efecto de la cantidad de pivotes

A continuación vamos a analizar el efecto de la cantidad de pivotes sobre el ratio de comparaciones para cada base de datos. Cada línea representa una base de datos y su descripción está determinada por la cantidad de elementos que contiene, como puede observarse en las gráficas.

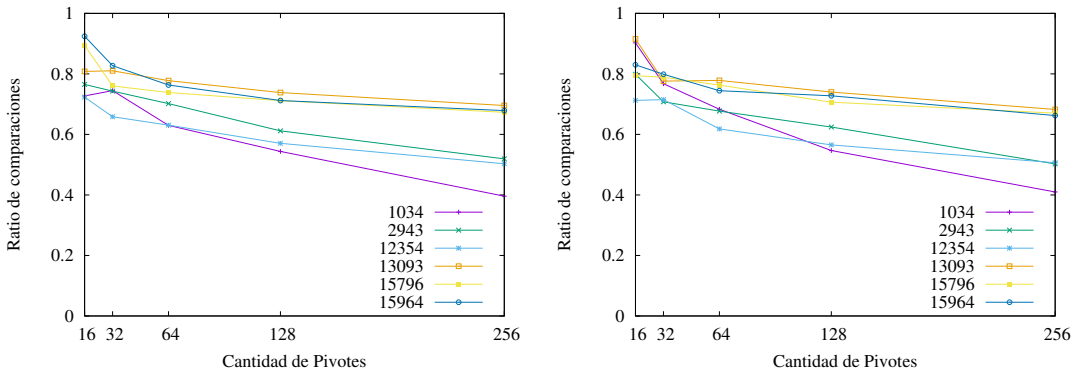
En la figura 5.3, observamos que tanto para selección de pivotes aleatoria (a) como para selección de pivotes incremental (b), para $16, 32, 64, 128$ y 256 pivotes, a medida que aumenta el número de pivotes, el ratio de comparaciones disminuye. También vemos que las 3 bases de datos mas grandes tienen un comportamiento similar, es decir el ratio de comparaciones oscila entre $0,7$ y $0,93$ aproximadamente y para las 3 bases de datos mas chicas, oscila entre $0,4$ y $0,8$ aproximadamente.

En la figura 5.4, observamos que para los pivotes $64, 128, 256, 512$ y 1024 ambas técnicas de selección de pivotes se comportan de manera similar, a mayor número de pivotes, menor ratio de comparaciones. El ratio de comparaciones está entre $0,6$ y $0,8$ para todas las bases de datos excepto la 27788 que se encuentra entre $0,4$ y $0,6$.

En la figura 5.5, observamos el mismo comportamiento que para los grupos 1 y 2. Para la mayoría de las bases de datos y para ambas técnicas de selección, el ratio de comparaciones está entre $0,6$ y $0,8$. Para la cantidad de pivotes de este grupo, $256, 512, 1024$ y 2048, también se cumple la tendencia de que a mayor número de pivotes menor ratio de comparaciones.

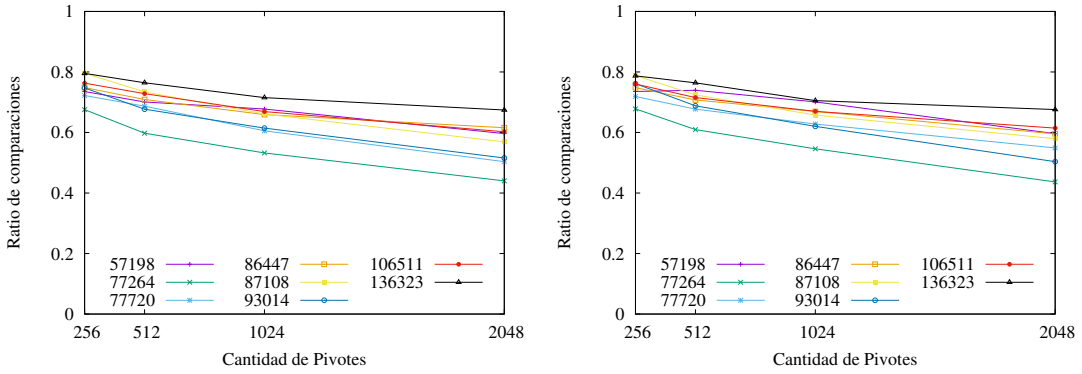
En la figura 5.6, observamos que a medida que aumenta el número de pivotes el ratio de comparaciones disminuye, de manera similar al resto de los grupos. También vemos que 3 de las 4 bases de datos que estamos analizando, para los pivotes 512, 1024 y 2048; el ratio de comparaciones está entre $0,6$ y $0,75$ aproximadamente y para 4096 pivotes, entre $0,5$ $0,6$.

A modo de resumen, analizando el efecto de la cantidad de pivotes en general para los cuatro grupos, vemos que cuando aumenta el número de pivotes decrementa el ratio de comparaciones pero las técnicas de selección aleatoria e incremental no muestran diferencias significativas entre sí.



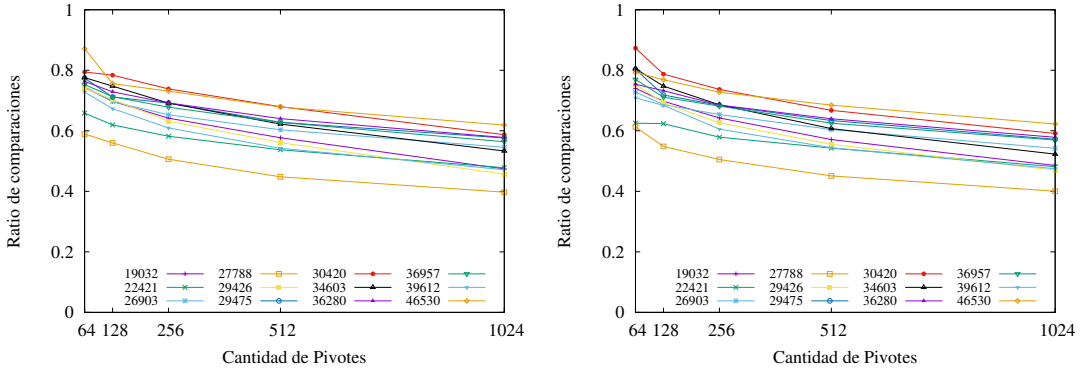
(a) Técnica de selección de pivotes aleatoria (b) Técnica de selección de pivotes incremental

Figura 5.3: Grupo 1 - Efecto de la cantidad de pivotes sobre el ratio de comparaciones.



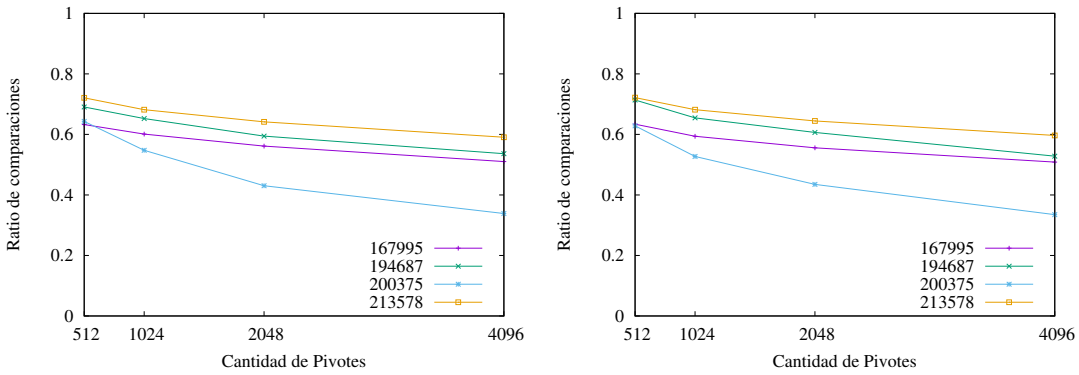
(a) Técnica de selección de pivotes aleatoria (b) Técnica de selección de pivotes incremental

Figura 5.5: Grupo 3 - Efecto de la cantidad de pivotes sobre el ratio de comparaciones.



(a) Técnica de selección de pivotes aleatoria (b) Técnica de selección de pivotes incremental

Figura 5.4: Grupo 2 - Efecto de la cantidad de pivotes sobre el ratio de comparaciones.



(a) Técnica de selección de pivotes aleatoria (b) Técnica de selección de pivotes incremental

Figura 5.6: Grupo 4 - Efecto de la cantidad de pivotes sobre el ratio de comparaciones.

5.3.2. Efecto del tamaño de la base de datos

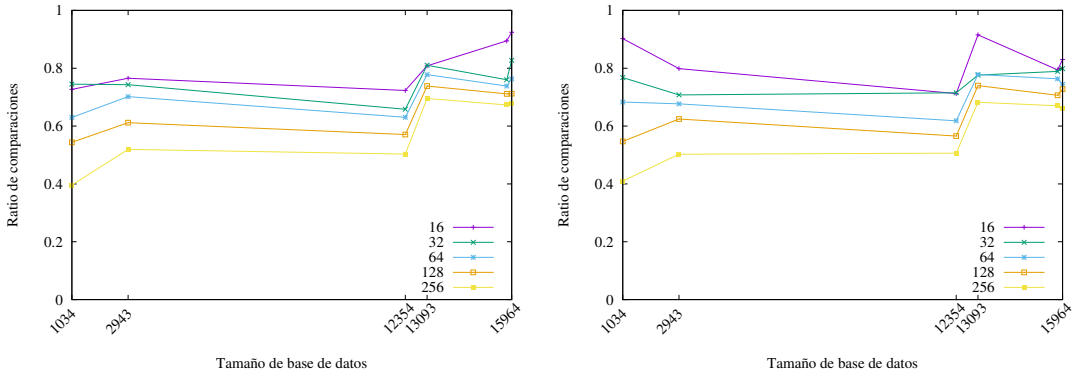
A continuación vamos a analizar cual es el efecto del tamaño de las bases de datos sobre el ratio de comparaciones por pivotes. Las líneas de las graficas representan el número de pivotes con los que se ejecutaron las búsquedas.

En la figura 5.7 presentamos los resultados para el grupo 1; donde el rango del tamaño de las bases de datos varía de 1034 a 15964 objetos y la cantidad de pivotes utilizados es $16, 32, 64, 128$ y 256.

Para la técnica de selección de pivotes aleatoria (a), en la base de datos mas chica, el ratio de comparaciones esta entre $0,4$ y $0,8$ y para las bases de datos más grandes, entre $0,7$ y $0,9$. Por otro lado, si miramos la gráfica para 256 pivotes podemos observar que para la base de datos mas chica el ratio de comparaciones es $0,4$ y a medida que crece en tamaño de la base de datos el ratio de comparaciones tambien aumenta ($0,7$ en las bases de datos mas grandes).

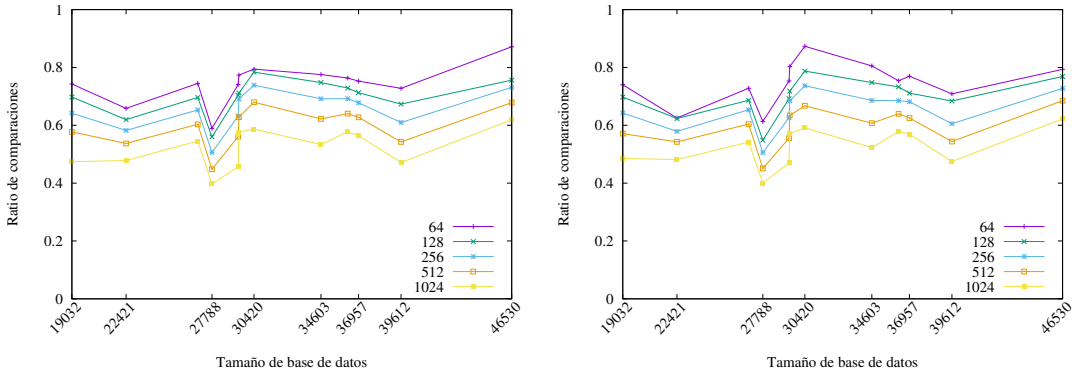
Para la técnica de selección de pivotes incremental (b), el comportamiento es similar, a menor cantidad de pivotes, mayor ratio de comparaciones.

En la figura 5.8, donde el tamaño de las bases de datos van desde 19032 a 46530 elementos y los pivotes usados son $64, 128, 256, 512$ y 1024, el ratio de comparaciones no tiene mucha variabilidad desde la base de datos mas chica a las mas grande. Por ejemplo: para técnica de selección de pivotes aleatoria, 512 pivotes y una base de datos de 19.000 elementos el ratio de comparaciones es $0,58$ aproximadamente y para una base de datos de 46.000 elementos, el ratio es $0,67$. Estos valores son similares para el caso de técnica de selección de pivotes incremental.



(a) Técnica de selección de pivotes aleatoria (b) Técnica de selección de pivotes incremental

Figura 5.7: Grupo 1 - Efecto del tamaño de la base de datos sobre el ratio de comparaciones por pivotes.



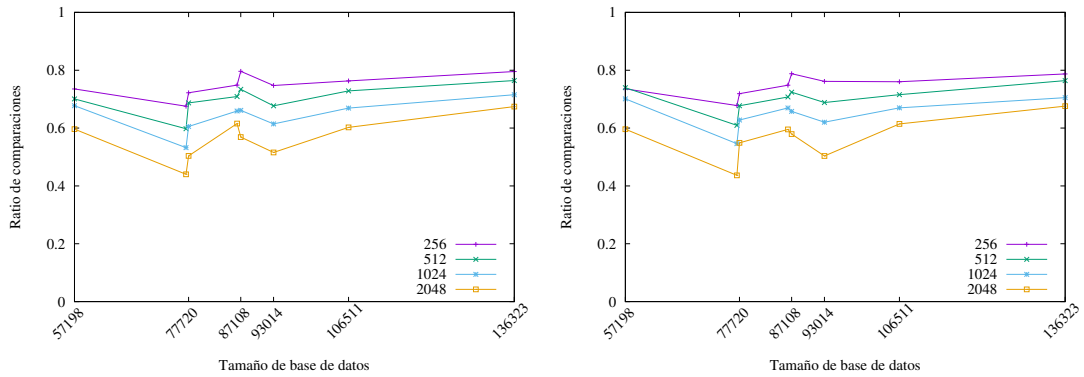
(a) Técnica de selección de pivotes aleatoria (b) Técnica de selección de pivotes incremental

Figura 5.8: Grupo 2 - Efecto del tamaño de la base de datos sobre el ratio de comparaciones por pivotes.

A nivel general, para técnica de selección de pivotes aleatoria, podemos ver que para la base de datos mas chica, el ratio de comparaciones varia entre $0,49$ y $0,79$; y para las bases de datos mas grande, entre $0,6$ y $0,84$. Para la técnica de selección de pivotes incremental los rangos de ratios de comparaciones estan entre $[0,5 - 0,77]$ y $[0,61 - 0,79]$.

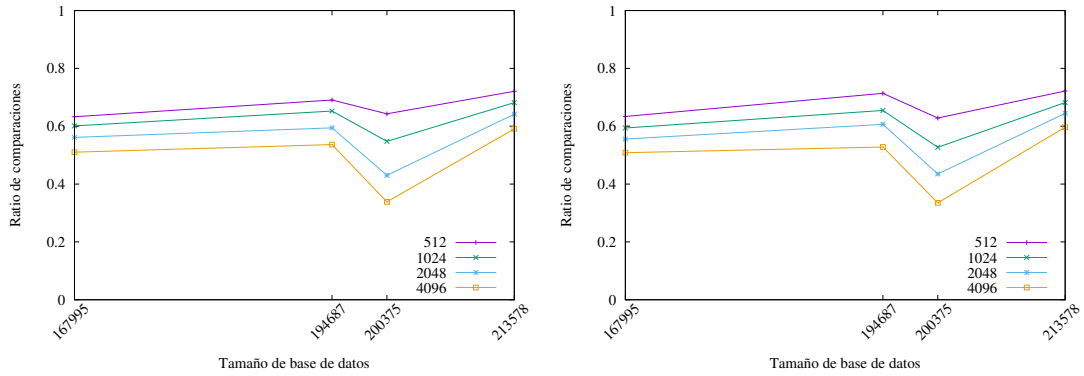
Para las figuras 5.9 y 5.10, donde se muestran los grupos 3 y 4 respectivamente, el comportamiento es similar al de los grupos antes mencionados.

A medida que el tamaño de las bases de datos aumenta, el ratio de comparaciones también crece, aunque la variabilidad de este último es muy baja. Esto es similar para ambas técnicas de selección de pivotes.



(a) Técnica de selección de pivotes aleatoria (b) Técnica de selección de pivotes incremental

Figura 5.9: Grupo 3 - Efecto del tamaño de la base de datos sobre el ratio de comparaciones por pivotes.



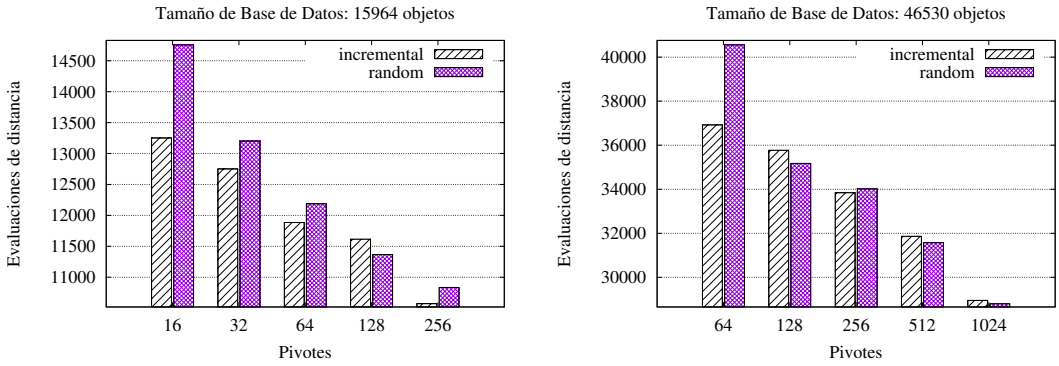
(a) Técnica de selección de pivotes aleatoria (b) Técnica de selección de pivotes incremental

Figura 5.10: Grupo 4 - Efecto del tamaño de la base de datos sobre el ratio de comparaciones por pivotes.

5.3.3. Comparación de las técnicas de selección de pivotes

En esta sección vamos a comparar las técnicas de selección de pivotes aleatoria e incremental, usando como métrica el número de evaluaciones de distancia para cada unos de los grupos de pivotes elegidos. Se realizaron las comparaciones para todas las bases de datos con las que trabajamos (30) pero a continuación sólo analizaremos la base de datos de mayor tamaño de cada uno de los grupos que mencionamos anteriormente, el resto se adjuntan en el Anexo A.

Para el grupo 1, figura 5.11(a), observamos que para una base de datos de aproximadamente 16.000 elementos, usando la técnica de selección incremental se realizan menos evaluaciones de distancia que la técnica de selección aleatoria para todos los grupos de pivotes, excepto para 128 pivotes. También se puede



(a) Grupo 1 (b) Grupo 2

Figura 5.11: Comparación de las técnicas de selección de pivotes random vs. incremental.

observar que la diferencia entre ambas técnicas es amplia.

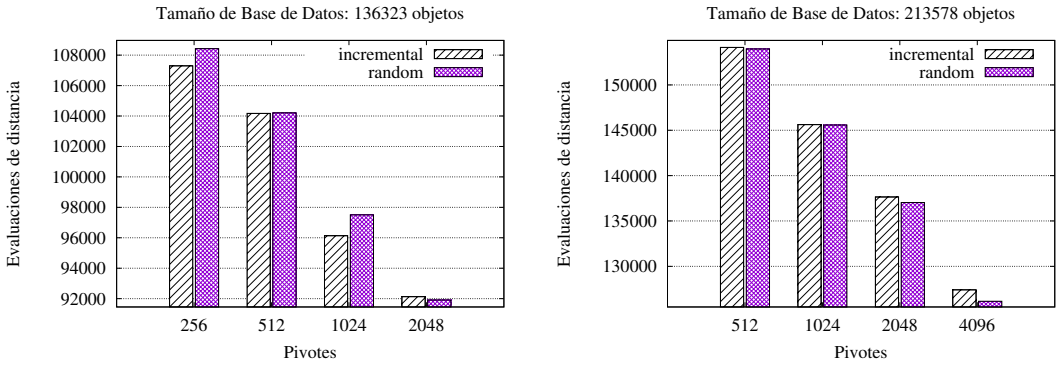
Para el grupo 2, figura 5.11(b), para una base de datos de aproximadamente 45.000 elementos, la técnica de selección incremental es mejor solo en dos grupos de pivotes, 64 y 256. Para los grupos de pivotes de 128, 512 y 1024, la técnica de selección aleatoria realiza menos evaluaciones de distancia que la incremental. También se puede observar que a partir del grupo de pivotes 256, la diferencia en la cantidad de evaluaciones de distancia no es tan grande entre ambas técnicas.

Para el grupo 3, figura 5.12(a), con una base de datos de aproximadamente 136.000 elementos, la técnica de selección incremental en general realiza menor cantidad de evaluaciones de distancia, salvo para 2048 pivotes. Para este ultimo grupo de pivotes, la técnica de selección aleatoria esta por debajo de incremental pero solo por una mínima diferencia.

Para el grupo 4, figura 5.12(b), con una base de datos de aproximadamente 213.000 elementos, en general, la técnica de selección aleatoria realiza menos evaluaciones de distancia que la técnica de selección incremental, aunque dicha diferencia es casi imperceptible (excepto para 4096, donde la diferencia es mas marcada).

Como conclusión general, se puede observar que la técnica de selección aleatoria ejecuta menos evaluaciones de distancia, aunque la diferencia con la selección incremental no es altamente significativa.

En la gran variedad de los trabajos existentes sobre el tema se ha demostrado que la técnica de selección incremental funciona mejor que la aleatoria, pero



(a) Grupo 3 (b) Grupo 4

Figura 5.12: Comparación de las técnicas de selección de pivotes random vs. incremental.

como podemos observar de los resultados de los experimentos realizados, ésta afirmación no se cumple para las bases de datos utilizadas. Para encontrar la razón del comportamiento observado, procedimos a realizar un análisis empírico de la distribución de los elementos de las diferentes categorías. Dicho análisis se presenta en la siguiente sección.

5.4. Histogramas de distancia

Los resultados obtenidos de los experimentos, en conjunción con el excesivo tiempo de ejecución de los algoritmos de búsqueda fueron un indicativo de que podíamos estar ante un espacio de alta dimensionalidad.

Dado que cada base de datos es un universo completamente distinto y que en todos los casos vimos el mismo comportamiento, decidimos realizar el cálculo de los histogramas de distancia para todas las bases de datos. En esta sección solo analizaremos los histogramas de las categorías con mayor cantidad de elementos de cada grupo, pero un listado completo puede encontrarse en el Anexo B.

En la figuras 5.13 y 5.14 se puede observar como se concentran los elementos, asemejandose a un histograma de alta dimensionalidad como vimos en la figura 2.6 (página 19).

Esto indica que a medida que los elementos se encuentran mas concentrados alrededor de su media, disminuye la cantidad de elementos que pueden ser descartados por el proceso de búsqueda. Esto es, aumenta la cantidad de comparaciones con elementos de base de datos.

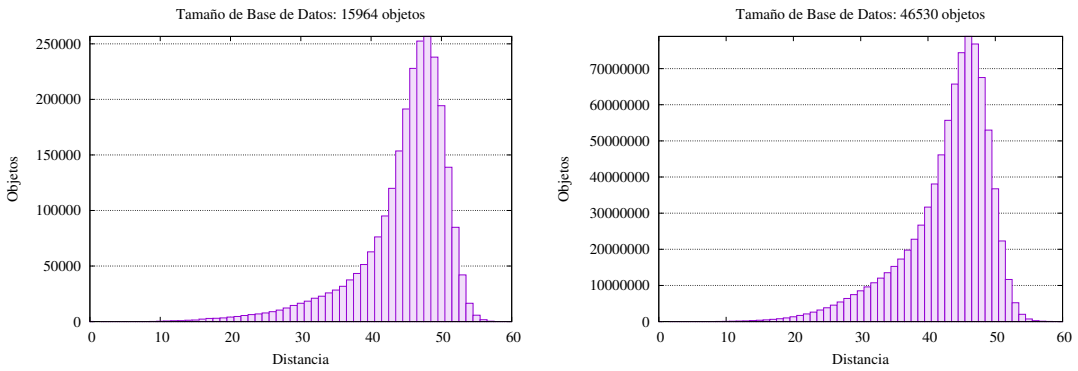


Figura 5.13: Histogramas de frecuencia de la base de datos más grandes del grupo 1 (izquierda) y del grupo 2 (derecha).

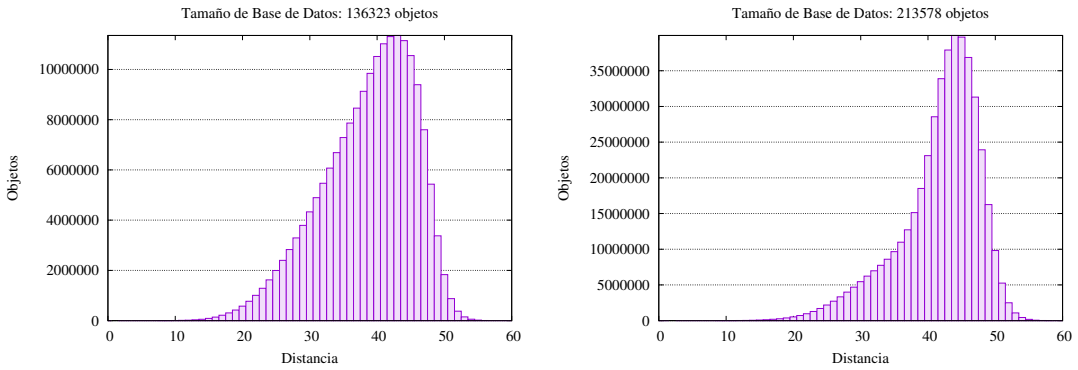


Figura 5.14: Histogramas de frecuencia de la base de datos más grandes del grupo 3 (izquierda) y del grupo 4 (derecha).

Si analizamos nuevamente las figuras 5.6 y 5.10 que muestran el efecto de la cantidad de pivotes y el efecto del tamaño de la base de datos respectivamente, para ambos casos, independientemente de la técnica de selección, vemos que el ratio de comparaciones para 4096 pivotes o para la base de datos más grande ronda el 0,6 lo cual es equivalente al 60 %. Esto quiere decir que ante una búsqueda, el algoritmo descarta menos de la mitad de los elementos de la base de datos.

Como síntensis, vemos en general que ninguna de las técnicas de selección ofrece una opción eficiente, ya que el porcentaje de comparaciones con los elementos de la base de datos es elevado para ambas técnicas.

5.5. Búsqueda de los k-vecinos

Como último paso en la ejecución de experimentos, definimos realizar la búsqueda de los *k-vecinos* con $k = 5$, para la base de datos con mayor cantidad de elementos de cada grupo, utilizando la mayor cantidad de pivotes del mismo y la estrategia de selección aleatoria.

Esta clase de experimento nos daría un indicio del comportamiento en general de nuestro sistema de recomendación.

Para las búsquedas de los *k-vecinos* se utilizó el mismo conjunto de items que para el resto de los experimentos, es decir, 10 % de los elementos pertenecientes a cada categoría, seleccionados al azar.

Tamaño de base de datos	Ratio de comparaciones		
	Prom.	Max.	Min.
15964	0.83845	1	0.00031
46530	0.78821	1	0.00011
136323	0.84512	1	0.00012
213578	0.76582	1	0.00004

Cuadro 5.5: Resultados de la búsqueda de los *k-vecinos* con $k = 5$.

En la tabla 5.5 presentamos los resultados obtenidos de dichos experimentos. Como podemos observar, el promedio del ratio de comparaciones fue superior a 0,7, es decir que para obtener los resultados de la búsqueda el algoritmo realizó más del 70 % de evaluaciones de distancia respecto del total de elementos de la base de datos.

Otro dato de interés a tener en cuenta es la gran variabilidad entre el máximo y el mínimo del ratio de comparaciones, el cual es un indicador de que para algunas pocas búsquedas el algoritmo funcionó con óptima eficiencia, pero para otras se comportó de la misma manera que una búsqueda secuencial.

Claramente, el comportamiento subóptimo del algoritmo de búsqueda se debe a la alta dimensionalidad del espacio métrico representado por cada una de las categorías.

Capítulo 6

Conclusiones

En este trabajo hemos abordado la aplicación del algoritmo de búsqueda por rango (y su variación de la búsqueda de k-vecinos) para obtener los productos similares a otro específico en un caso real de e-commerce, con el fin de estudiar el desempeño de dicho algoritmo comparando las técnicas de selección de pivotes aleatoria e incremental.

Para poder estudiar el desempeño del algoritmo, se realizaron los siguientes pasos:

1. Se hizo un análisis y particionado por categorías de los productos existentes en el sitio de e-commerce de Mercado Libre.

2. Se realizó una aproximación experimental para determinar el radio de búsqueda ideal.

3. Se construyeron índices utilizando las técnicas de selección aleatoria e incremental.

4. Se implementó el algoritmo de búsqueda por rango y se seleccionó como criterio para analizar su desempeño, la cantidad de evaluaciones de la función de distancia.

Se ha demostrado experimentalmente en el presente trabajo, que a medida que aumenta el número de pivotes que compone el índice utilizado para las búsquedas, la cantidad de evaluaciones de la función de distancia disminuye. Esto es independiente de la técnica de selección de pivotes utilizada para la construcción de dicho índice y del tamaño de la base de datos analizada.

Por otro lado, si analizamos los resultados experimentales de las técnicas de selección, no es posible observar una diferencia significativa en el comportamiento de ambas, por lo cual no podemos afirmar que una técnica es mejor que la otra. Esto se contrapone con las conclusiones expuestas en [BNC01], donde queda

demostrado que la técnica de selección incremental es mejor que la técnica de selección aleatoria.

Como último punto, relevando los mejores resultados obtenidos por cada técnica de selección en cada una de las categorías, podemos determinar que la elevada cantidad de evaluaciones de la función de distancia hace imposible la aplicación de estas técnicas a un sistema de recomendaciones eficiente.

Las conclusiones mencionadas pueden ser explicadas por la llamada maldición de la dimensionalidad. La aproximación de los histogramas de distancia de los espacios conformados por cada una de las categorías dieron indicios de que las mismas constituían espacios de alta dimensionalidad, donde está demostrado que los algoritmos basados en pivotes no tienen buen desempeño [BNC01]

6.1. Trabajos futuros

El trabajo realizado deja la puerta abierta para avanzar en distintos sentidos.

Por un lado, se podría ampliar el software desarrollado para que pueda manejar estructuras en memoria secundaria; esto sería posible a través de un cambio de lenguaje que permita tener mayor control sobre instrucciones de bajo nivel y que tenga un menor uso de recursos.

Por otro lado, sería interesante evaluar el comportamiento de los algoritmos basados en particiones compactas en este tipo de aplicaciones de la vida real.

Anexos

Anexos A

Técnicas de selección de pivotes

A continuación incluimos las graficas correspondientes al efecto de las técnicas de selección para cada una de las bases de datos que utilizamos.

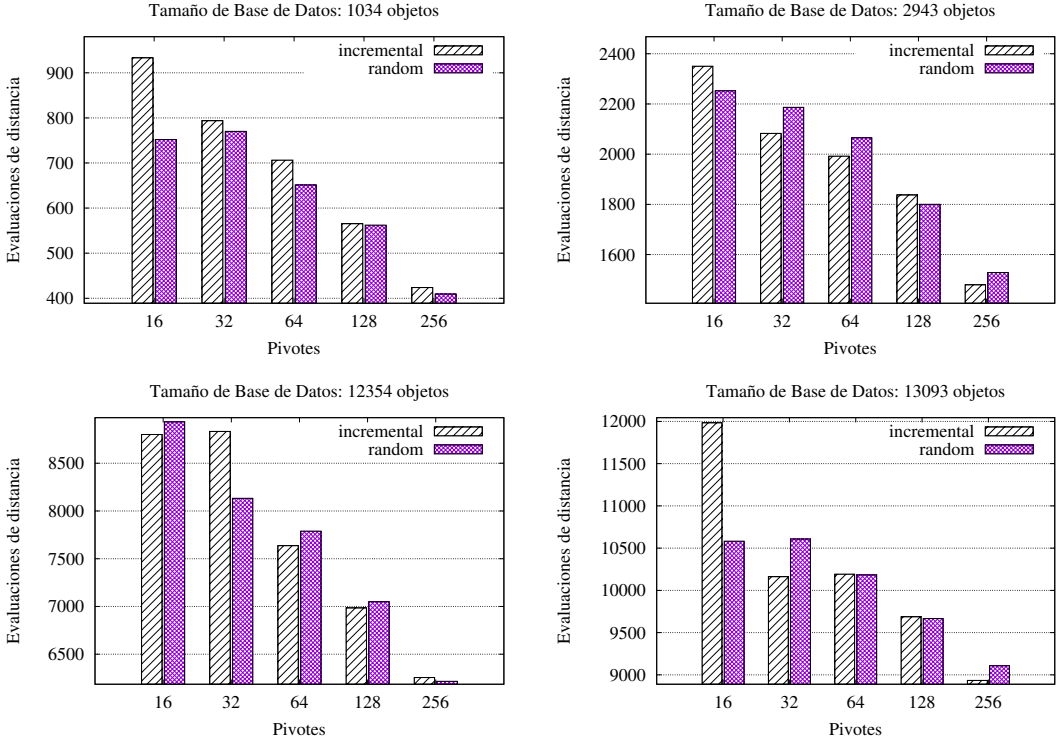


Figura A.1: Grupo 1 - Efecto de las técnicas de selección de pivotes random vs incremental respecto de evaluaciones de distancia.

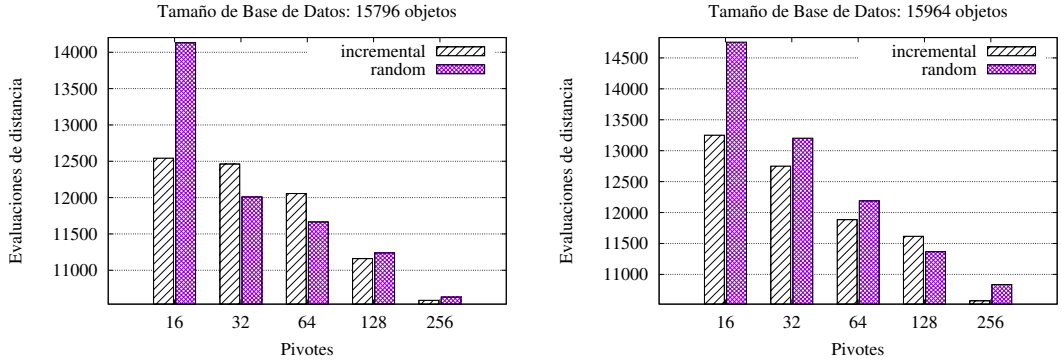


Figura A.2: Grupo 1 - Efecto de las técnicas de selección de pivotes random vs incremental respecto de evaluaciones de distancia.

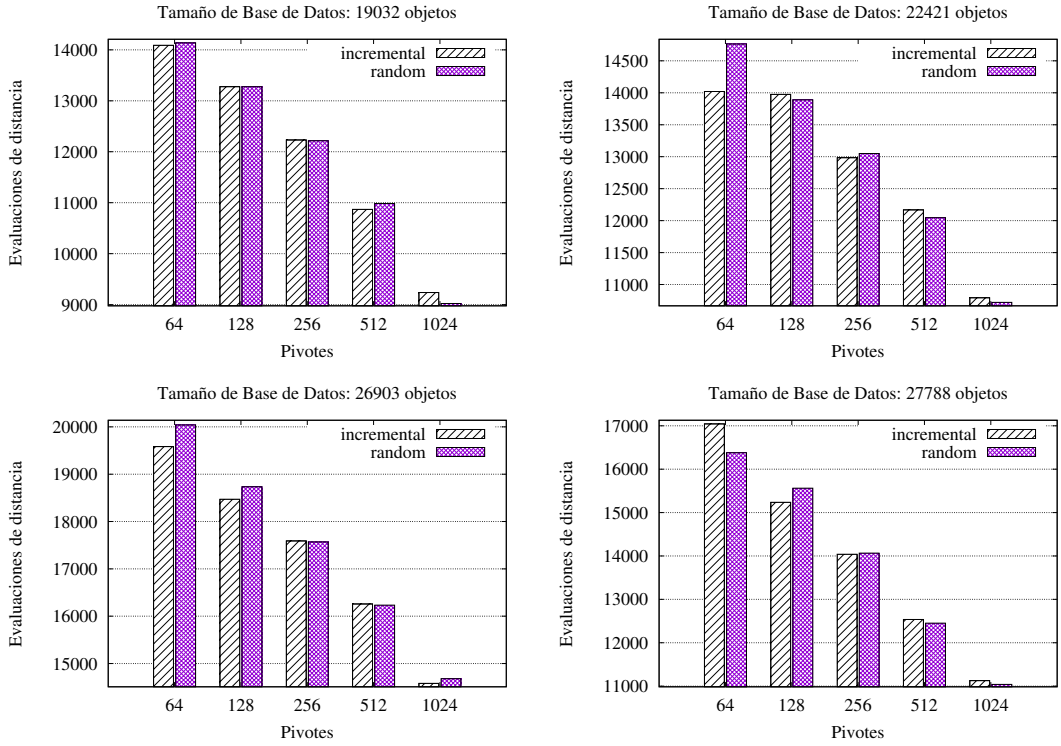


Figura A.3: Grupo 2 - Efecto de las técnicas de selección de pivotes random vs incremental respecto de evaluaciones de distancia.

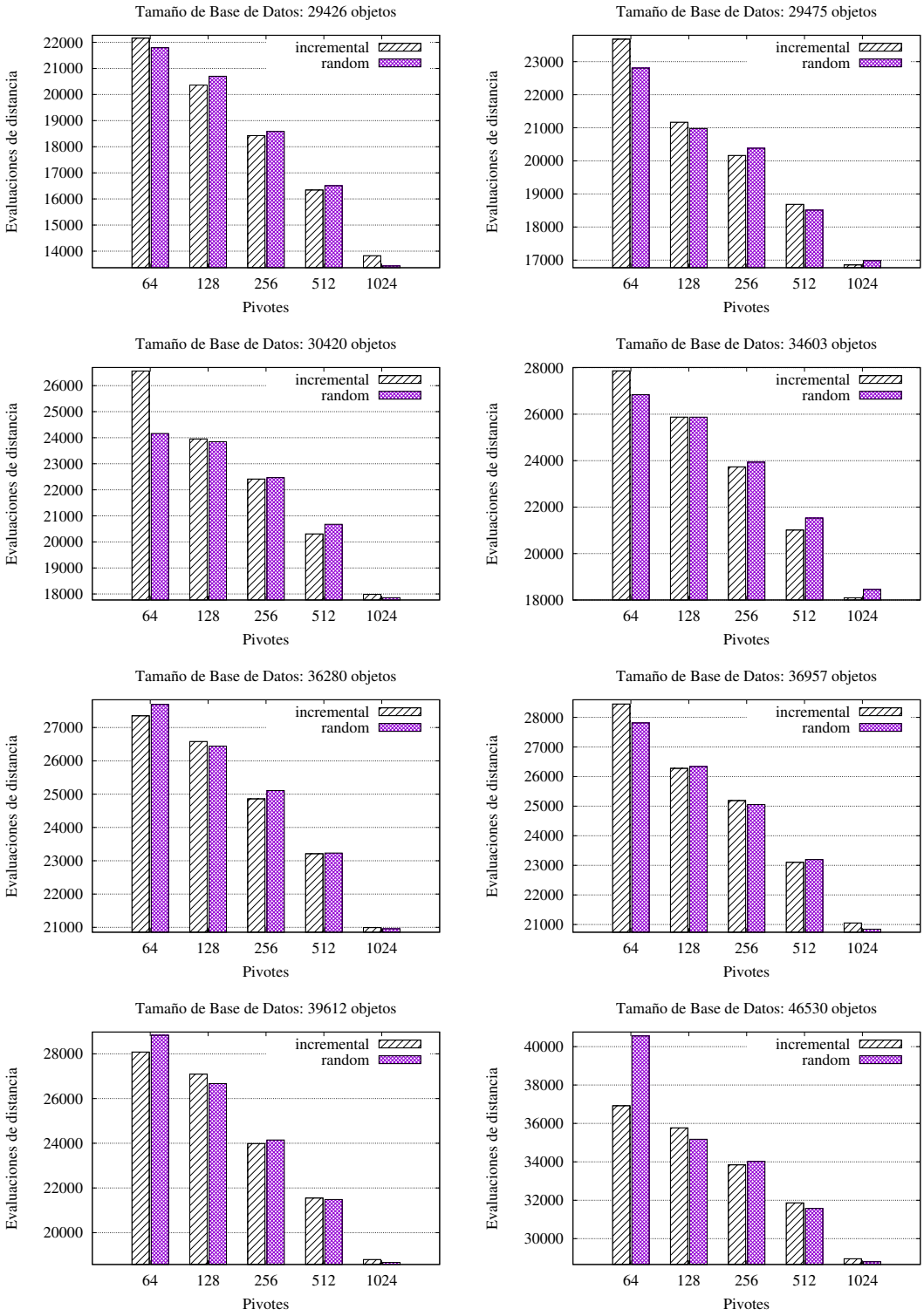


Figura A.4: Grupo 2 - Efecto de las técnicas de selección de pivotes random vs incremental respecto de evaluaciones de distancia.

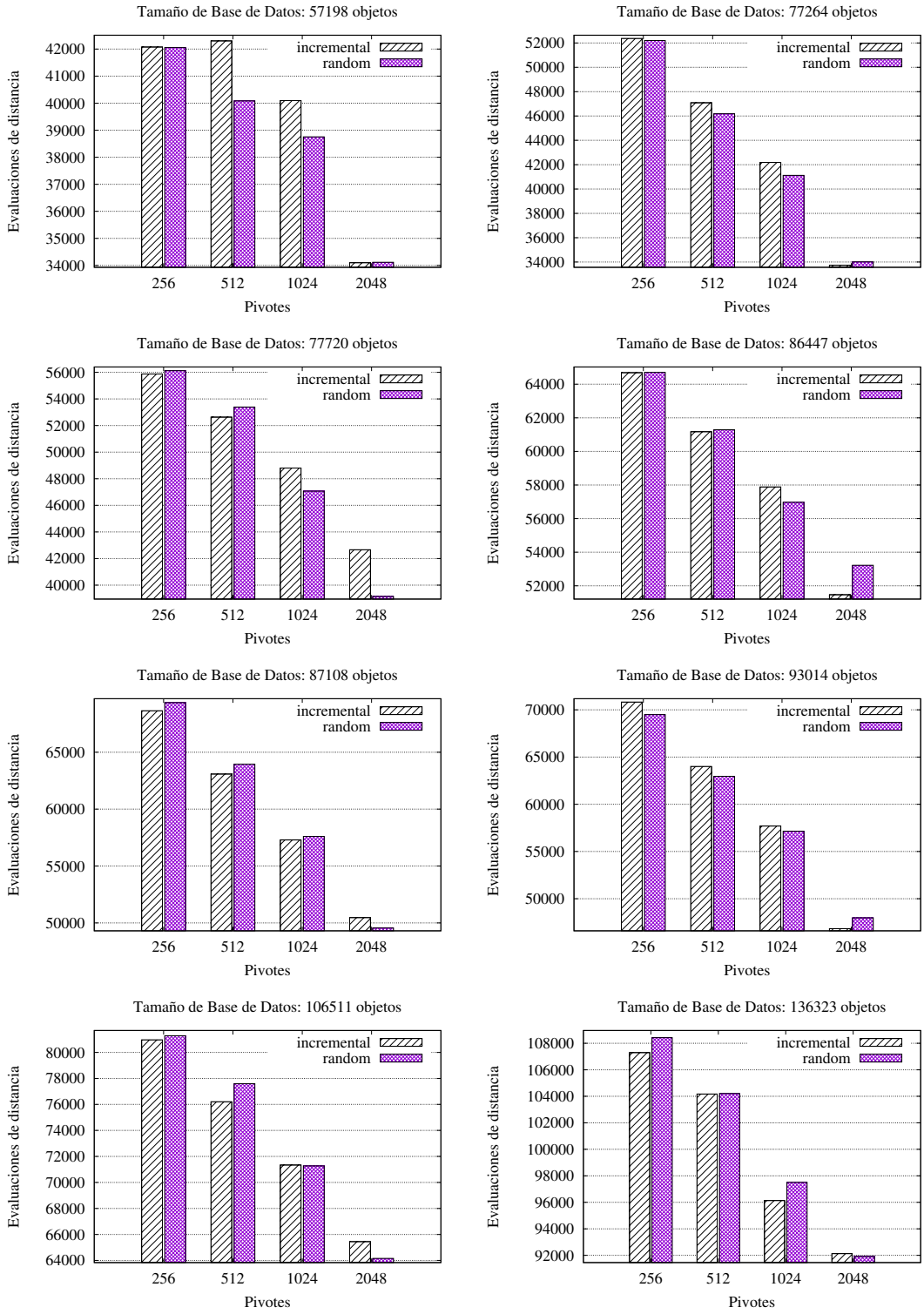


Figura A.5: Grupo 3 - Efecto de las técnicas de selección de pivotes random vs incremental respecto de evaluaciones de distancia.

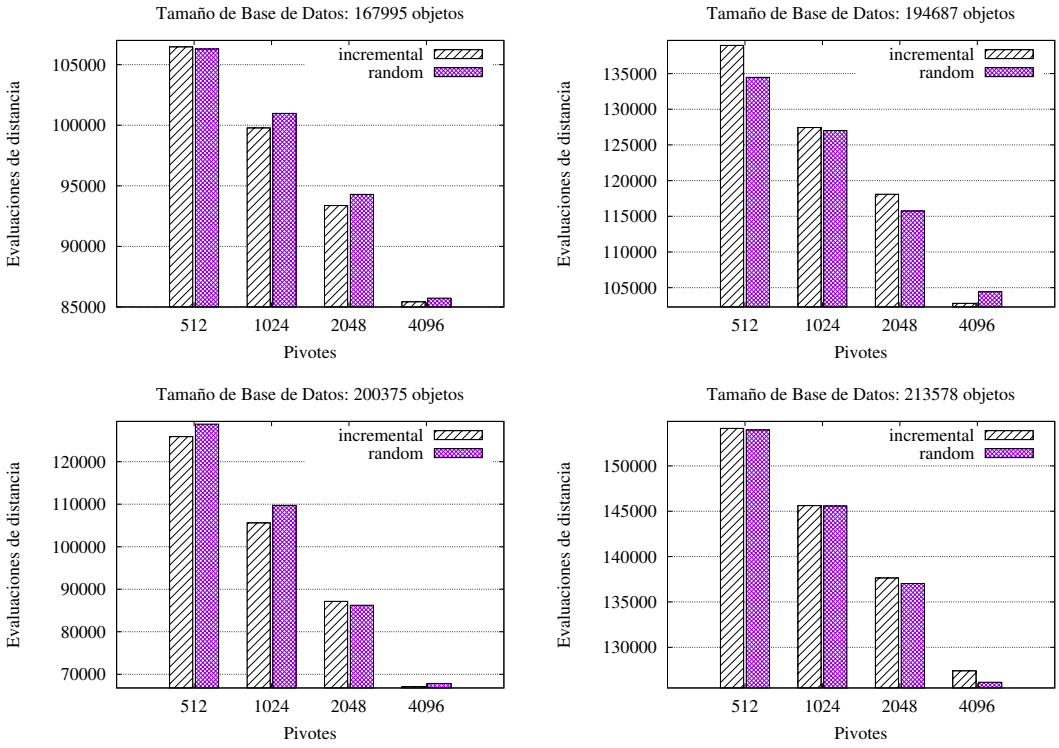


Figura A.6: Grupo 4 - Efecto de las técnicas de selección de pivotes random vs incremental respecto de evaluaciones de distancia.

Anexos B

Histogramas de frecuencia

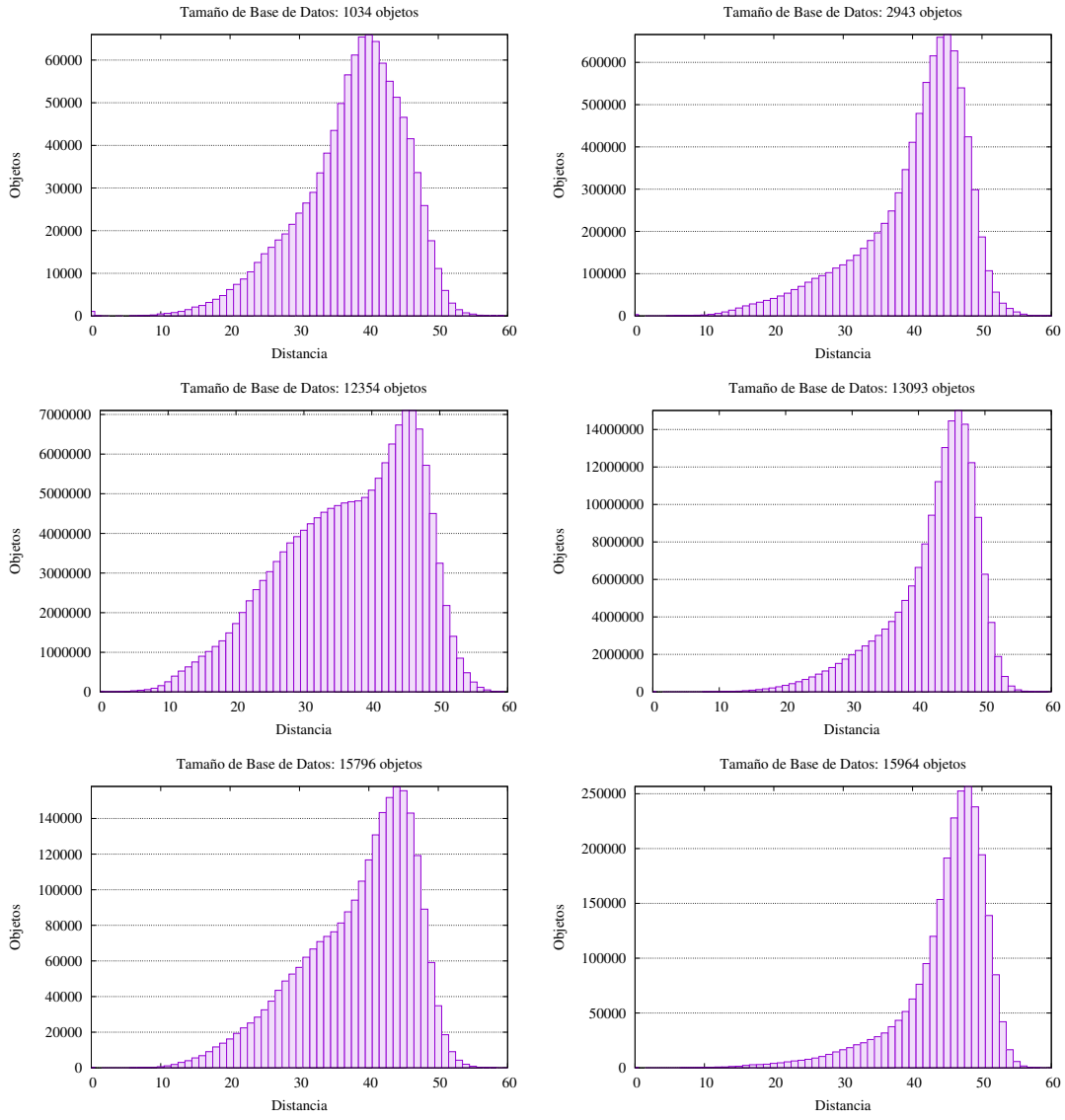


Figura B.1: Grupo 1 - Histogramas de frecuencia.

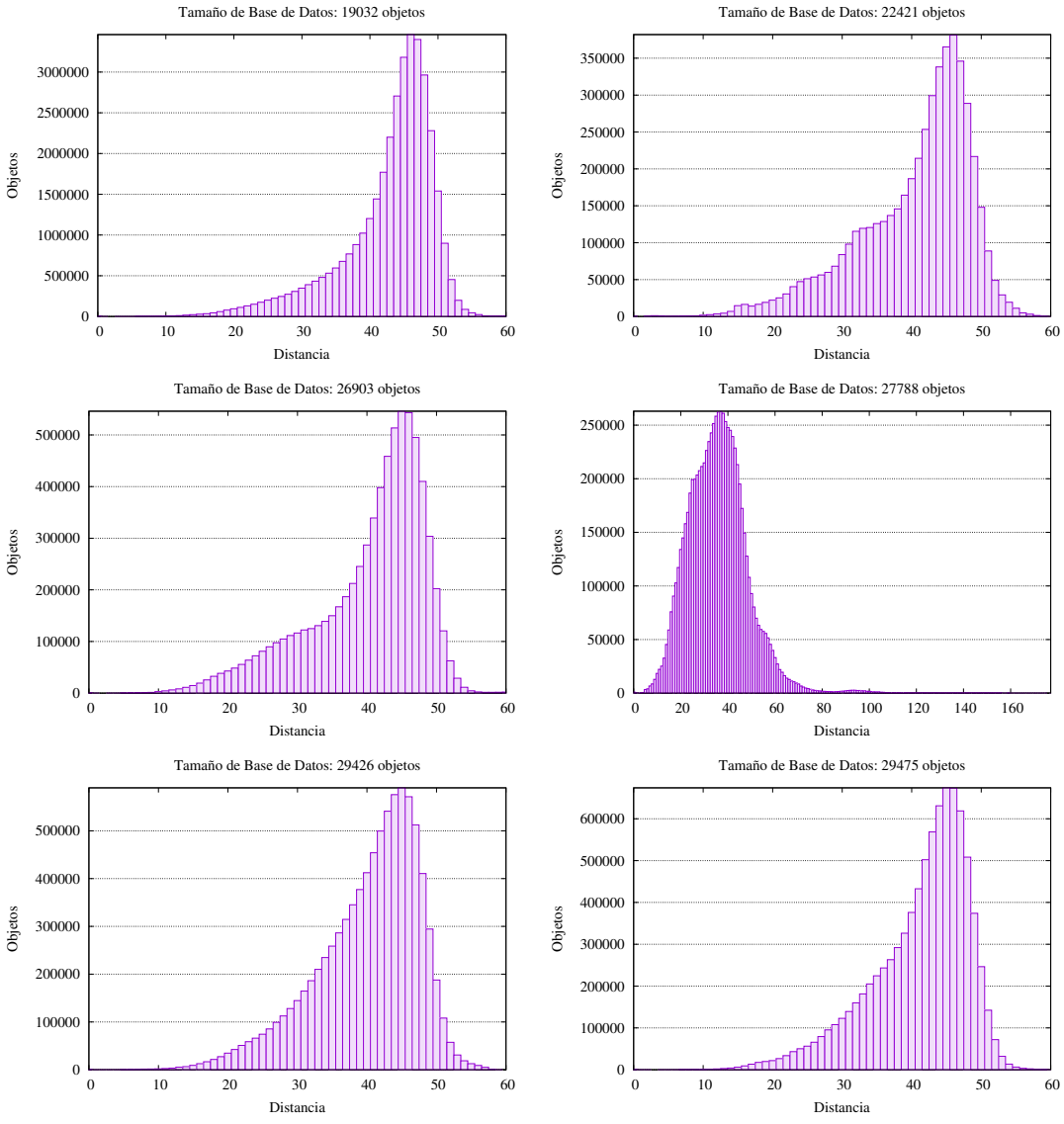


Figura B.2: Grupo 2 - Histogramas de frecuencia.

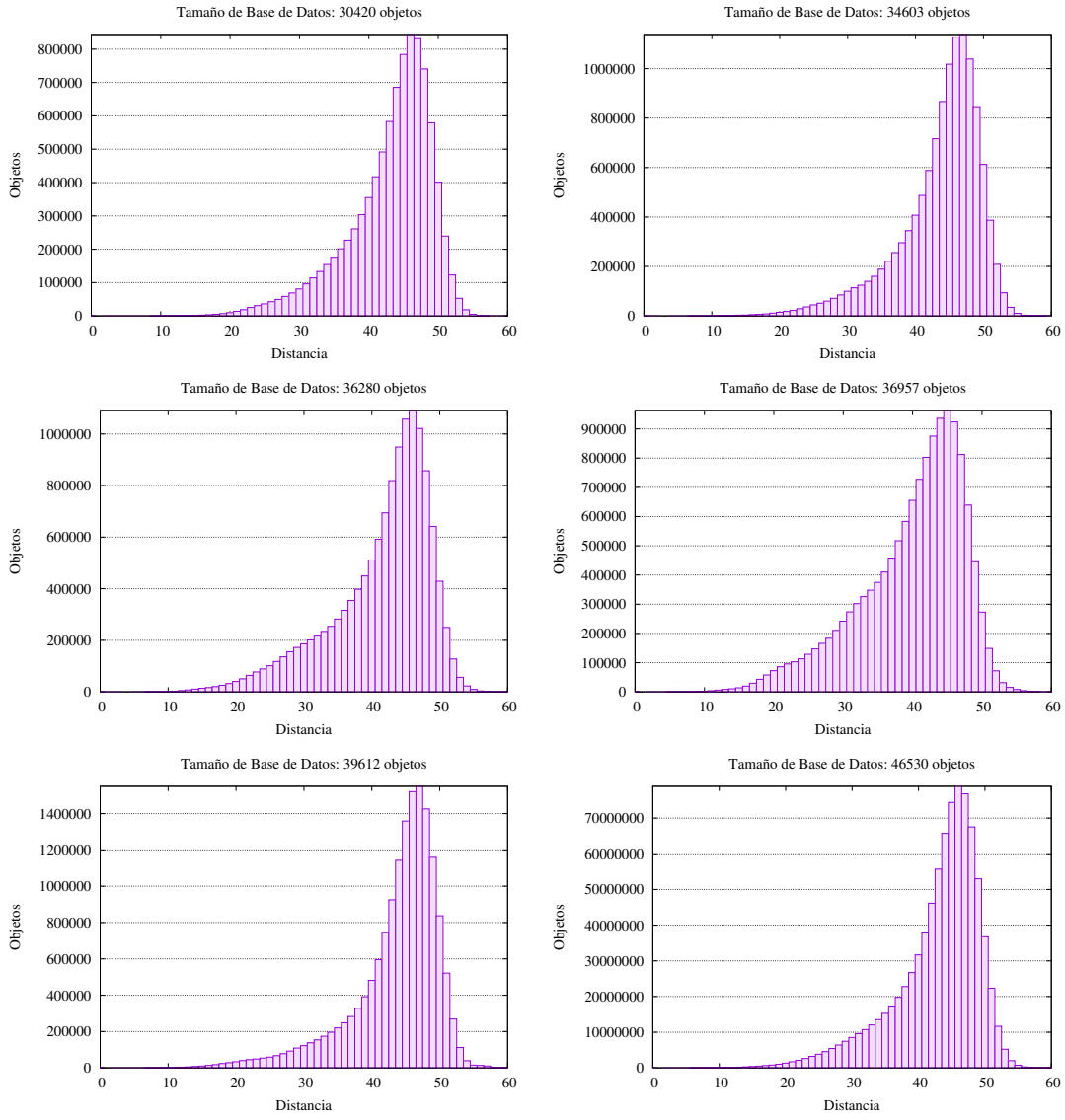


Figura B.3: Grupo 2 - Histogramas de frecuencia.

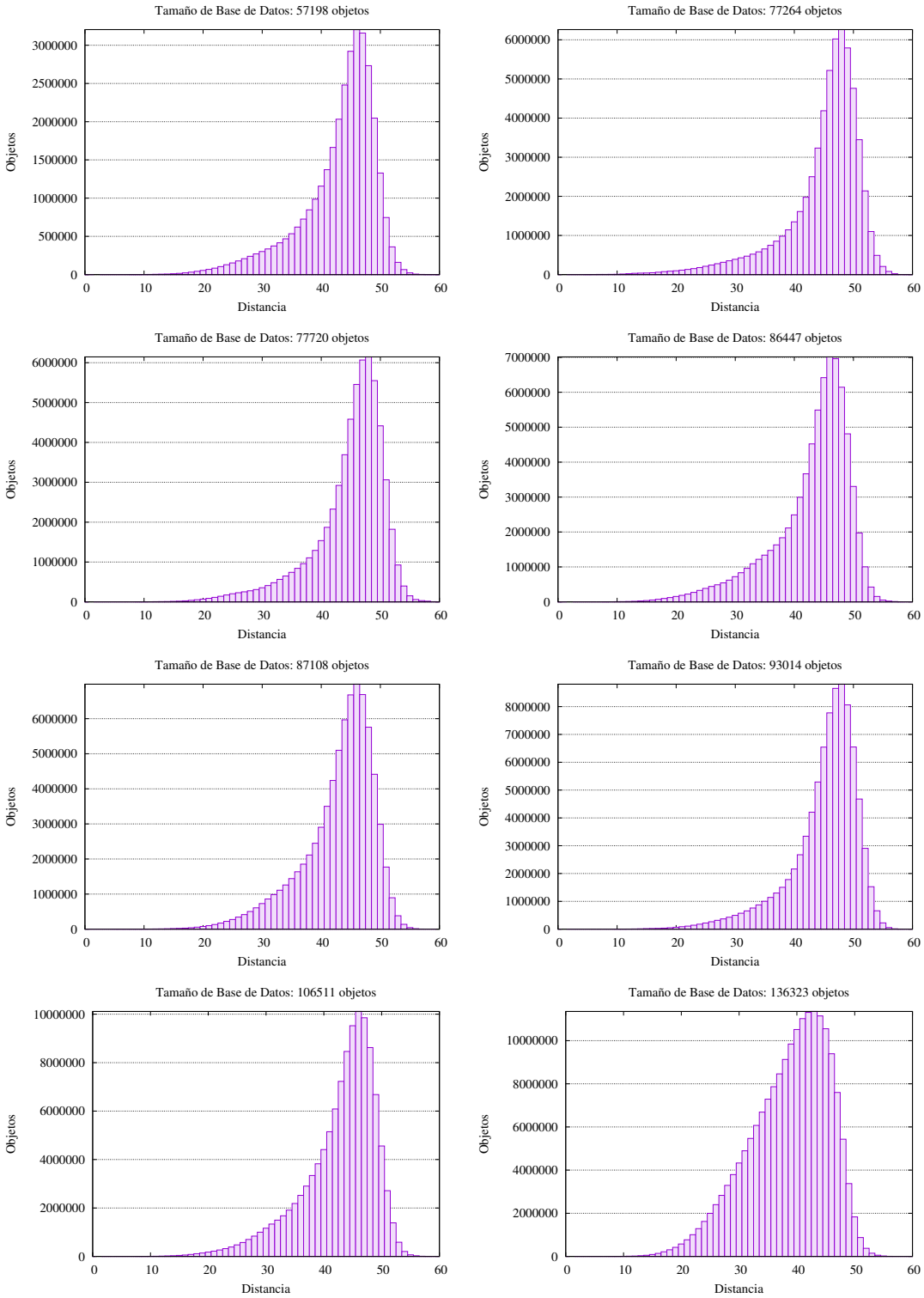


Figura B.4: Grupo 3 - Histogramas de frecuencia.

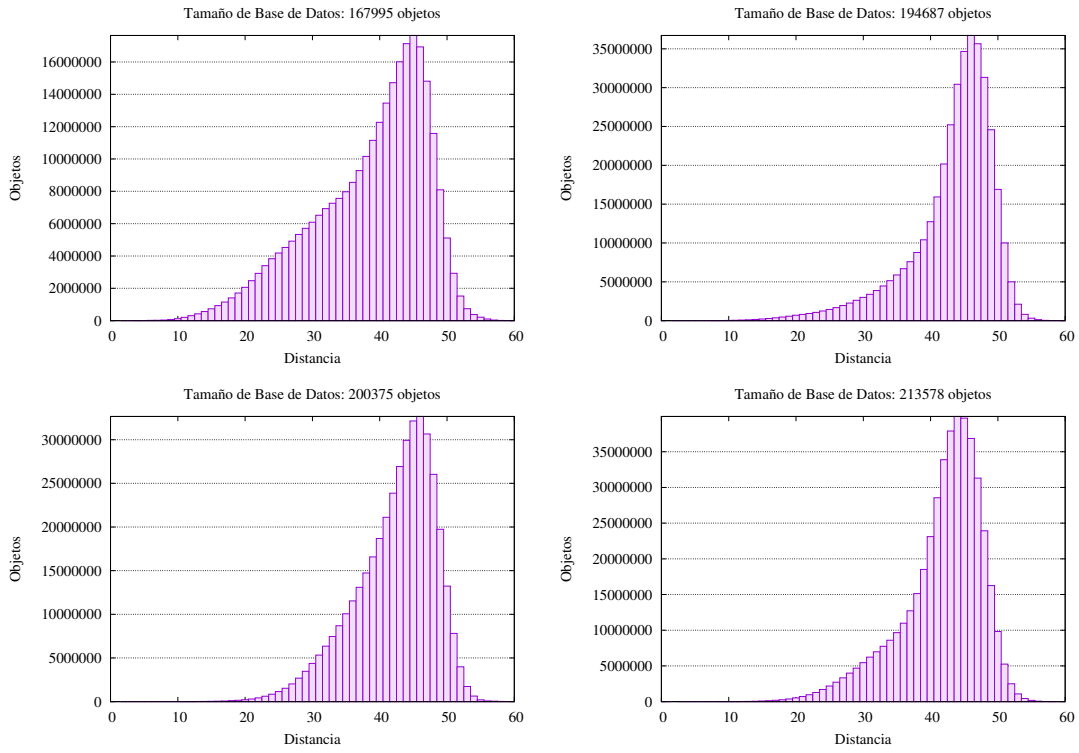


Figura B.5: Grupo 4 - Histogramas de frecuencia.

Bibliografía

[Aur91] Aurenhammer, F. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys 23*, 3 (1991).

[BNC01] Bustos, B., Navarro, G., and Chávez, E. Pivot selection techniques for proximity searching in metric spaces. In *Proc. of the XXI Conference of the Chilean Computer Science Society (SCCC'01)* (2001), IEEE CS Press, pp. 33–40.

[Bri95] Brin, S. Near neighbor search in large metric spaces. In *Proc. 21st Conference on Very Large Databases (VLDB'95)* (1995), pp. 574–584.

[CM97] Chávez, E., and Marroquín, J. Proximity queries in metric spaces. In *Proc. 4th South American Workshop on String Processing (WSP'97)* (1997), R. Baeza-Yates, Ed., Carleton University Press, pp. 21–36.

[CN00] Chávez, E., and Navarro, G. An effective clustering algorithm to index high dimensional metric spaces. In *Proc. 7th International Symposium on String Processing and Information Retrieval (SPIRE'00)* (2000), IEEE CS Press, pp. 75–86.

[CN01] Chávez, E., and Navarro, G. A probabilistic spell for the curse of dimensionality. In *Proc. 3rd Workshop on Algorithm Engineering and Experiments (ALENEX'01)* (2001), LNCS.

[CNBYM01] Chávez, E., Navarro, G., Baeza-Yates, R., and Marroquín, J. Searching in metric spaces. *ACM Computing Surveys 33*, 3 (September 2001), 273–321.

[CPZ98] Ciaccia, P., Patella, M., and Zezula, P. A cost model for similarity queries in metric spaces. In *Proc. 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98)* (1998).