UNIVERISDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

ALGORITMOS PROBABILÍSTICOS BASADOS EN PARTICIONES COMPACTAS PARA BÚSQUEDAS EN PROXIMIDAD

BENJAMIN EUGENIO BUSTOS CÁRDENAS

2002

Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Algoritmos Probabilísticos Basados en Particiones Compactas para Búsquedas en Proximidad

Benjamin Eugenio Bustos Cárdenas

Profesor Guía	:	Sr. Gonzalo Navarro B.
Profesores de Comisión	:	Sr. Ricardo Baeza Y.
	:	Sr. Patricio Poblete O.
Profesor Invitado	:	Sra. Andrea Rodríguez T.

Tesis para optar al Grado de
Magíster en Ciencias, Mención Computación

Agosto, 2002

Algoritmos Probabilísticos Basados en Particiones Compactas para Búsquedas en Proximidad

La búsqueda en proximidad en espacios métricos consiste en encontrar objetos de la base de datos que sean "parecidos" o "relevantes" a una consulta dada. Este concepto tiene una amplia gama de aplicaciones prácticas en minería de datos, bases de datos multimediales, recuperación de información, etc.

El mayor obstáculo en la búsqueda de objetos en un espacio métrico es la denominada "maldición de la dimensionalidad", la cual hace que dicha búsqueda sea intrínsecamente difícil en los espacios de dimensión alta, independientemente del algoritmo utilizado para realizarla. Sin embargo, éste es un caso en donde un algoritmo probabilístico puede ser de gran utilidad, dado que en general el concepto de relevancia que se intenta modelar posee un cierto grado de inexactitud. En estas aplicaciones, encontrar la mayoría de los objetos relevantes es más que suficiente, sobre todo si el costo de la búsqueda se reduce drásticamente.

En esta tesis se proponen dos técnicas probabilísticas basadas en particiones compactas (es decir, en particionar el conjunto en zonas espacialmente compactas) para realizar búsquedas en proximidad. La idea de estas técnicas es fijar de antemano la cantidad de trabajo a realizar durante la búsqueda y recorrer los objetos de la base de datos en un orden promisorio, de modo de encontrar rápidamente objetos cercanos a la consulta. Para esto, primero se modificó un algoritmo incremental que recupera los objetos en orden de cercanía a la consulta, revisándolos en el orden de promisoriedad dado por la cota inferior de la distancia a su zona. Luego, se observó que basta con ordenar las zonas según algún criterio de promisoriedad, para posteriormente recorrerlas en dicho orden. Si bien el método de cota inferior es óptimo para buscar los k vecinos más cercanos, otros criterios de ordenación resultaron ser más efectivos para la búsqueda probabilística. Las ventajas de las técnicas propuestas con respecto a las ya existentes son: requerimientos de memoria significativamente menores, tiempo de búsqueda acotado, obtención incremental de la respuesta (pudiéndose interrumpir en cualquier momento), y mejor escalabilidad al aumentar la dimensión del espacio.

Se implementaron ambas técnicas y se realizaron experimentos tanto en espacios sintéticos como en ejemplos reales. Se observa que se recupera la mayoría de los objetos relevantes recorriendo sólo una fracción de la base de datos, incluso en dimensiones tan altas como 128 en el caso de los espacios vectoriales. Como ejemplo de un caso real, en una base de documentos se recuperó más del 99% de los textos relevantes recorriendo apenas un 17% de la base de datos. Una consecuencia práctica de este estudio es que es posible aplicar el enfoque de búsqueda en espacios métricos a aplicaciones reales en donde antes no era posible, como lo es la búsqueda de documentos similares, un problema clásico en Recuperación de la Información.

A mis Maestros, que con su luz me guían por el camino de la vida.

Índice General

1 Introducción 1

2 Conceptos Básicos 6

2.1 Espacios métricos 6

2.2 Ejemplos de espacios métricos 7

2.2.1 Espacios vectoriales 7

2.2.2 Modelo vectorial de documentos 8

2.3 Búsqueda en proximidad en espacios métricos 10

3 Algoritmos de Búsqueda en Proximidad en Espacios Métricos 12

3.1 Algoritmos basados en pivotes 12

3.1.1 Ejemplos de algoritmos basados en pivotes 14

3.2 Algoritmos basados en particiones compactas 17

3.2.1 Criterios de exclusión 18

3.2.2 Spatial Approximation Tree 21

3.2.3 List of Clusters 29

3.2.4 Otros algoritmos basados en particiones compactas 34

3.3 Algoritmos probabilísticos de búsqueda en proximidad 36

3.3.1 Algoritmo probabilístico para búsqueda del vecino más cercano 36

3.3.2 Algoritmo probabilístico basado en pivotes 38

3.3.3 Otros trabajos relacionados para espacios vectoriales 41

4 Algoritmos Probabilísticos Basados en Particiones Compactas 44

4.1 Búsqueda incremental probabilística 44

4.2 Ranking de zonas 48

5 Resultados Experimentales 51

5.1 Experimentos en espacios vectoriales 52

5.2 Experimentos en base de datos de documentos 63

5.3 Ranking de zonas versus ranking de objetos 73

5.4 Modelo de comparación de criterios de ranking 75

6 Conclusiones **79**

A Artículo Publicado **86**

Índice de Figuras

2.1 Distancias de Minkowski para distintos valores del parámetro s. 8

2.2 Ejemplo de una consulta por rango en un espacio vectorial de dimensión 2. 10

3.1 Partición de Voronoi de un espacio vectorial de dimensión 2. 18

3.2 Condición de exclusión con criterio de partición de Voronoi. 19

3.3 Condición de exclusión con criterio de radio cobertor. 20

3.4 Ejemplo de búsqueda por aproximación espacial. 22

3.5 Algoritmo de construcción del SAT. 25

3.6 Ejemplo del proceso de búsqueda por rango en SAT. 26

3.7 Algoritmo de búsqueda por rango en SAT. 27

3.8 Algoritmo de búsqueda de k vecinos más cercanos en SAT. 29

3.9 Algoritmo de construcción del List of Clusters. 30

3.10 Ejemplo de List of Clusters. 31

3.11 Algoritmo de búsqueda en List of Clusters. 32

3.12 Algoritmo de búsqueda de k vecinos más cercanos en List of Clusters. 33

3.13 Histograma de distancias para un espacio métrico de dimensión baja (izquierda) y alta (derecha). 39

3.14 Funciones de densidad de las variables aleatorias X y Z. 39

3.15 Cómo funciona el algoritmo probabilístico basado en pivotes. 41

3.16 Rendimiento del algoritmo probabilístico basado en pivotes. 42

4.1 Búsqueda incremental de vecinos más cercanos. 45

4.2 Búsqueda incremental probabilística. 47

4.3 Criterio de ordenación de zonas $d(q,c)$. 49

4.4 Criterio de ordenación de zonas $cr(c)$. 49

4.5 Criterio de ordenación de zonas $d(q,c) + cr(c)$. 49

4.6 Criterio de ordenación de zonas $d(q,c) - cr(c)$. 50

5.1 SAT y List of Clusters probabilístico, dimensión 64, 0,01% recuperado. 53

5.2 SAT y List of Clusters probabilístico, dimensión 64, 0,10% recuperado. 53

5.3 SAT y List of Clusters probabilístico, dimensión 64, 1,00% recuperado. 54

5.4 Comparación con algoritmo basado en pivotes, dimensión 64, 0,01% recuperado. . . 55

5.5 Comparación con algoritmo basado en pivotes, dimensión 64, 0,10% recuperado. . . 55

5.6 Comparación con algoritmo basado en pivotes, dimensión 64, 1,00% recuperado. . . 56

5.7 SAT y List of Clusters probabilístico, dimensión 128, 0,01% recuperado. 56

5.8 SAT y List of Clusters probabilístico, dimensión 128, 0,10% recuperado. 57

5.9 SAT y List of Clusters probabilístico, dimensión 128, 1,00% recuperado. 57

5.10 Comparación con algoritmo basado en pivotes, dimensión 128, 0,01% recuperado. . . 58

5.11 Comparación con algoritmo basado en pivotes, dimensión 128, 0,10% recuperado. . . 58

5.12 Comparación con algoritmo basado en pivotes, dimensión 128, 1,00% recuperado. . . 59

5.13 SAT y List of Clusters probabilístico, 100.000 elementos, dimensión 128, 0,01% recuperado. 60

5.14 SAT y List of Clusters probabilístico, 100.000 elementos, dimensión 128, 0,10% recuperado. 60

5.15 SAT y List of Clusters probabilístico, 100.000 elementos, dimensión 128, 1,00% recuperado. 61

5.16 Comparación con algoritmo basado en pivotes, 100.000 elementos, dimensión 128, 0,01% recuperado. 61

5.17 Comparación con algoritmo basado en pivotes, 100.000 elementos, dimensión 128, 0,10% recuperado. 62

5.18 Comparación con algoritmo basado en pivotes, 100.000, dimensión 128, 1,00% recuperado. 62

5.19 Comparación de criterios en espacio de documentos, 0,035% recuperado. 64

5.20 Comparación de criterios en espacio de documentos, 0,048% recuperado. 65

5.21 Comparación de criterios en espacio de documentos, 0,064% recuperado. 65

5.22 Comparación del criterio beta dinámico con distintos tamaños de zona, 0,035% recuperado. 66

5.23 Comparación del criterio beta dinámico con distintos tamaños de zona, 0,048% recuperado. 67

5.24 Comparación del criterio beta dinámico con distintos tamaños de zona, 0,064% recuperado. 67

5.25 Fracción recuperada en función de la cuota de trabajo para distintos tamaños de zona, 0,035% recuperado. 68

5.26 Fracción recuperada en función de la cuota de trabajo para distintos tamaños de zona, 0,048% recuperado. 68

5.27 Fracción recuperada en función de la cuota de trabajo para distintos tamaños de zona, 0,064% recuperado. 69

5.28 Comparación de criterio beta dinámico con método basado en pivotes, 0,035% recuperado. 70

5.29 Comparación de criterio beta dinámico con método basado en pivotes, 0,048% recuperado. 70

5.30 Comparación de criterio beta dinámico con método basado en pivotes, 0,064% recuperado. 71

5.31 Fracción de objetos recuperados en función de la fracción de la base de datos recorrida, 0,035% recuperado. 71

5.32 Fracción de objetos recuperados en función de la fracción de la base de datos recorrida, 0,048% recuperado. 72

5.33 Fracción de objetos recuperados en función de la fracción de la base de datos recorrida, 0,064% recuperado. 72

5.34 Comparación con variantes de criterios de ordenación, 0,064% recuperado. 74

5.35 Comparación con variantes de criterios de ordenación, 0,064% recuperado. 74

5.36 Comparación con variantes de criterios de ordenación, 0,064% recuperado. 75

5.37 Ejemplo de nube de puntos para un criterio dado. 76

5.38 Fracción de objetos recuperados en función de la fracción de la base de datos recorrida para los distintos criterios de *ranking*, 0,064% recuperado. 77

5.39 Resultado utilizando el modelo de comparación, 0,064% recuperado. 77

Índice de Tablas

5.1 Tamaños de los índices para un espacio vectorial (10.000 elementos), utilizando distintas estructuras de datos. 63

5.2 Porcentaje de documentos recuperados según radio de consulta. 64

5.3 Tamaños de los índices de la base de datos de documentos para distintas estructuras de datos. 73

Capítulo 1

Introducción

Muchas aplicaciones computacionales necesitan buscar información dentro de una base de datos. En las bases de datos tradicionales dicha búsqueda corresponde a una búsqueda exacta. En este enfoque, los elementos de la base de datos se definen como registros o datos estructurados, los cuales poseen campos con valores escalares como por ejemplo enteros, reales o secuencias de caracteres (*strings*). Al realizar una búsqueda en la base de datos se compara el valor requerido, que se denominará *llave de búsqueda* o simplemente *llave*, con un determinado campo de cada registro, reportando todos aquellos elementos que coincidan exactamente con la llave de búsqueda. Por ejemplo, suponga que se desea buscar en una base de datos bancaria la información personal de un cliente. Para recuperar dicha información se puede utilizar el RUT del cliente como llave de búsqueda, ya que sólo puede haber un cliente por cada número de RUT, y por lo tanto la información recuperada, en caso que exista un registro en la base de datos con el RUT especificado, tiene que ser la información del cliente requerido.

El concepto de búsqueda exacta no se limita solamente a calces exactos con la llave de búsqueda, sino que también permite buscar elementos en donde un determinado campo posea un valor que se encuentre dentro de un cierto rango definido por la llave. Por ejemplo, si se desean recuperar todos los movimientos bancarios realizados por un cliente entre el 10 y el 15 de febrero del año en curso, entonces además del RUT se necesitará comparar las fechas de las transacciones y verificar si corresponden o no al periodo requerido, reportando sólo aquellas transacciones que pertenezcan al cliente en particular y que hayan sido realizadas dentro del rango de fechas definido.

No obstante lo anterior, en la actualidad existen muchas bases de datos que contienen información *no estructurada*, como por ejemplo imágenes, sonidos, texto libre y otros, en donde no siempre es claro cómo definir un registro por cada elemento perteneciente a la base de datos. Aunque existan formas de estructurar dicha información de la manera tradicional, es posible que la llave de búsqueda de una consulta sea un elemento que no pertenezca a la base de datos, en donde se requiere recuperar algún elemento de la base de datos que sea "parecido" o "relevante" con respecto a la llave de búsqueda. Por ejemplo, suponga que se desea encontrar en una base de datos de documentos todos los textos similares a un documento q. Se podría pensar que basta con buscar documentos que compartan algunas palabras con las que aparecen en q, pero primero habría que definir algún mecanismo que permita detectar las palabras "relevantes" de q para luego poder realizar dicha búsqueda en la base de datos. En la práctica, los mejores resultados, en términos de relevancia de las respuestas, se obtienen definiendo una medida de similaridad entre documentos y recuperando los más similares a q. Este tipo de consultas corresponden al concepto denominado como *búsqueda en proximidad.*

La búsqueda en proximidad tiene una amplia gama de aplicaciones computacionales prácticas en distintas áreas, tales como:

- *Bases de datos multimediales*, donde el concepto de búsqueda exacta no es de utilidad, sino que se realizan búsquedas de objetos similares. Por ejemplo, los dispositivos biométricos realizan una lectura de alguna característica física de un individuo, su huella digital por ejemplo, para luego buscar dentro de una base de datos y verificar si el individuo está registrado o no. Es muy improbable que dos objetos multimediales sean idénticos a no ser que sean copias digitales del mismo objeto.

- *Clasificación y aprendizaje*, donde se tiene un sistema con elementos clasificados según algunas características definidas, y se desea clasificar un nuevo elemento de acuerdo al elemento más "cercano" a éste que ya pertenezca al sistema.

- *Compresión y transmisión de video*, donde se transmite una señal de video dividiendo la imagen en varias "subimágenes", transmitiendo la primera imagen completa y luego retransmitiendo sólo las subimágenes que varíen con respecto a aquellas previamente transmitidas por sobre un cierto nivel de tolerancia, lo cual se determina mediante una búsqueda en proximidad.

- *Recuperación de texto*, donde es necesario buscar palabras en una base de datos de documentos

permitiendo un pequeño número de errores en caso que, por ejemplo, la palabra haya sido mal tipeada al momento de realizar la consulta.

- *Recuperación de Información*, donde es necesario buscar documentos que sean relevantes a una consulta dada.
- *Biología computacional*, donde se necesita buscar secuencias de proteínas o de ADN en una base de datos, permitiendo algunos errores debido a variaciones típicas. Dichas secuencias pueden ser modeladas como secuencias de caracteres.

Todas estas aplicaciones prácticas tienen en común que los elementos pertenecientes a la base de datos forman un espacio métrico [15], esto es, es posible definir una función d real no negativa entre los elementos, llamada *distancia* o *métrica*, que satisface las propiedades de *positividad estricta*, *simetría* y *desigualdad triangular*. Por ejemplo, un *espacio vectorial normado* es un caso particular de espacio métrico, donde los elementos son tuplas de números reales. En los espacios vectoriales es posible definir distintas funciones de distancia, pero la más ampliamente utilizada es la familia de *distancias de Minkowski* [26].

En general, la función de distancia es considerada computacionalmente costosa de calcular. Por ejemplo, calcular la distancia entre dos huellas digitales puede resultar bastante costoso comparado con otras operaciones realizadas durante la búsqueda. Es por esto que es usual definir la complejidad de la búsqueda en proximidad como el número de evaluaciones de distancia realizadas para responder una consulta, obviando otros costos tales como el tiempo de CPU utilizado en operaciones adicionales o el costo de Entrada/Salida (E/S).

Existen dos tipos de consultas típicas en búsqueda en proximidad, las cuales son:

- *Consulta por rango*: se desea recuperar los elementos de la base de datos que se encuentren dentro de un cierto radio de tolerancia a un elemento de consulta dado.
- *k vecinos más cercanos*: se desea recuperar los k elementos de la base de datos más cercanos a un elemento de consulta dado.

Una manera trivial de responder ambos tipos de consulta es realizando una búsqueda exhaustiva en la base de datos, esto es, comparando todos los elementos de la base de datos con el elemento

de consulta y retornando aquellos que se encuentren suficientemente cerca de éste, pero por lo general esto resulta ser demasiado costoso para las aplicaciones de la vida real. Los algoritmos de búsqueda en proximidad construyen un *índice* de la base de datos que, en conjunto con la desigualdad triangular, permite filtrar elementos sin medir su distancia real al elemento de consulta, con lo cual se evita realizar la búsqueda exhaustiva. Existen muchos algoritmos de búsqueda en proximidad, los cuales se dividen en dos grandes áreas: algoritmos basados en *pivotes* [8, 4, 12, 31, 6, 32, 29, 22, 11, 28] y algoritmos basados en *particiones compactas* [25, 18, 24, 16, 13, 23, 7].

Un problema inherente a la búsqueda en proximidad en espacios métricos es que dicha búsqueda se torna más difícil mientras mayor sea la dimensión intrínseca del espacio, hecho conocido como la *maldición de la dimensionalidad*. La dimensión intrínseca de un espacio métrico se define en [15] como $\mu^2/2\sigma^2$, donde μ y σ^2 son la media y la varianza del histograma de distancias del espacio métrico. Esta definición es coherente con la noción de dimensión en un espacio vectorial con coordenadas uniformemente distribuidas. En [15] se muestra analítica y experimentalmente que las búsquedas se degradan sistemáticamente a medida que la dimensión intrínseca del espacio métrico aumenta. Esto ocurre debido a dos causas: en primer lugar, se observa en los espacios métricos de dimensión intrínseca alta que el histograma de distancias es muy concentrado (varianza pequeña), lo cual entrega poca información a la hora de filtrar elementos (en el caso extremo, se tiene un espacio donde $d(x,x) = 0$ y $\forall y \neq x, d(x,y) = 1$, donde es imposible evitar realizar una búsqueda exhaustiva); en segundo lugar, para recuperar una fracción fija de la base de datos, es decir, recuperar un número constante de vecinos cercanos, es necesario utilizar un radio de búsqueda mayor, ya que los elementos se encuentran más "alejados" entre sí a medida que la dimensión del espacio crece. En general, ambas causas ocurren simultáneamente.

Se observa que los algoritmos basados en particiones compactas tienen un mejor desempeño cuando la dimensión intrínseca del espacio es alta en comparación con los algoritmos basados en pivotes [15, 13], y que éstos necesitan elevados requerimientos de memoria para poder superar a su contraparte. Existen muchas aplicaciones prácticas donde la dimensión del espacio es alta, por lo que es interesante estudiar y mejorar mecanismos que tienen un buen rendimiento en este tipo de espacios.

En [14] se presenta un algoritmo probabilístico de búsqueda en proximidad basado en pivotes que aprovecha la alta dimensionalidad del espacio para realizar la búsqueda de manera más eficiente, pudiendo perder algunos elementos relevantes. El algoritmo se basa en una reducción del radio de

tolerancia utilizado al realizar una consulta. Como el costo de búsqueda crece exponencialmente con el radio de tolerancia, una pequeña reducción redunda en una búsqueda mucho más rápida. En dicho trabajo se muestra que incluso cuando la reducción del radio es pequeña y "pierde" muy pocos elementos, se reduce drásticamente el tiempo de búsqueda. Sin embargo, como se había observado con anterioridad, los algoritmos basados en pivotes requieren de cantidades imprácticas de memoria para almacenar el índice en dimensiones altas, por lo que resultaría muy interesante aplicar el método probabilístico a un algoritmo basado en particiones compactas, sabiendo que éstos son más eficientes en espacios con dimensión alta.

En esta tesis se pretende realizar un estudio similar al de [14], pero ahora enfocándose en un algoritmo de búsqueda basado en particiones compactas. El estudio comprenderá una revisión de los algoritmos exactos de búsqueda en proximidad que existen en la actualidad, la forma en la cual se puede aplicar un enfoque probabilístico a un algoritmo basado en particiones compactas para mejorar su rendimiento, y un análisis de cómo afecta dicho enfoque en la calidad de los resultados obtenidos, es decir, cuánta información relevante se pierde en comparación a la mejora en el rendimiento del algoritmo de búsqueda.

Capítulo 2

Conceptos Básicos

Todas las aplicaciones prácticas que realizan búsquedas en proximidad comparten el hecho que necesitan encontrar objetos que estén cercanos, donde la cercanía se define en términos de una función de distancia. En este capítulo se define formalmente el concepto de espacio métrico, se muestran algunos ejemplos particulares y se definen las consultas típicas de búsqueda en proximidad.

2.1 Espacios métricos

Sea $\mathbb{X}$ el universo de los elementos válidos, también denominados *objetos*, que pueden pertenecer a la base de datos. El par $(\mathbb{X}, d)$ representa un *espacio métrico* si y sólo si d es una función $d : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{R}^+$ que cumple con las siguientes propiedades:

- *Simetría*: $\forall x, y \in \mathbb{X}, d(x, y) = d(y, x)$.
- *Reflexividad*: $\forall x \in \mathbb{X}, d(x, x) = 0$.
- *Positividad estricta*: $\forall x, y \in \mathbb{X}, x \neq y \Rightarrow d(x, y) > 0$.
- *Desigualdad triangular*: $\forall x, y, z \in \mathbb{X}, d(x, z) \leq d(x, y) + d(y, z)$.

La función d se denomina *distancia* o *métrica* del espacio $\mathbb{X}$.

2.2 Ejemplos de espacios métricos

A continuación se presentarán dos ejemplos particulares de espacios métricos: espacios vectoriales y documentos representados como vectores.

2.2.1 Espacios vectoriales

Un *espacio vectorial* $\mathbb{R}^k$ es un conjunto de k-tuplas de números reales. Los espacios vectoriales son un caso particular de espacio métrico, donde es posible definir distintas funciones de distancia. Dentro de las más conocidas está la familia de *distancias de Minkowski*, que se define como:

$$L_s\left((x_1,\ldots,x_k),(y_1,\ldots,y_k)\right) = \left(\sum_{1\leq i\leq k} |x_i - y_i|^s\right)^{1/s} \tag{2.1}$$

Algunos ejemplos de métricas pertenecientes a esta familia de distancias son:

- L_1, conocida como *distancia Manhattan*:

$$L_1\left((x_1,\ldots,x_k),(y_1,\ldots,y_k)\right) = \sum_{i=1}^{k} |x_i - y_i| \tag{2.2}$$

- L_2, conocida como *distancia Euclidiana*:

$$L_2\left((x_1,\ldots,x_k),(y_1,\ldots,y_k)\right) = \sqrt{\sum_{i=1}^{k} |x_i - y_i|^2} \tag{2.3}$$

- L_∞, conocida como *distancia del máximo*, que corresponde a tomar el límite cuando s tiende a infinito:

$$L_\infty\left((x_1,\ldots,x_k),(y_1,\ldots,y_k)\right) = \max_{i=1}^{k} |x_i - y_i| \tag{2.4}$$

La Figura 2.1 muestra una comparación de las distancias L_s para distintos valores de s. Todos los puntos que se ubican en el perímetro de las figuras se encuentran a la misma distancia de su centro respectivo.

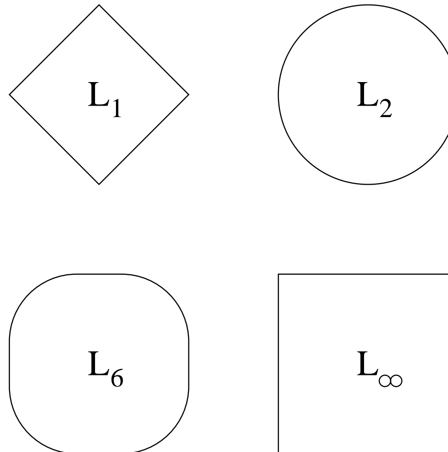

Figura 2.1: Distancias de Minkowski para distintos valores del parámetro s.

Si sólo se utiliza la función de distancia para realizar búsquedas en un espacio vectorial, es decir, no se utiliza la información de las coordenadas de los objetos del espacio, entonces es fácil simular búsquedas en un espacio métrico general teniéndose una ventaja adicional: es posible controlar exactamente la dimensión del espacio métrico en estudio.

2.2.2 Modelo vectorial de documentos

En recuperación de la información [5] se define un *documento* como una "unidad de recuperación", la cual puede ser un párrafo, una sección, un capítulo, una página Web, un artículo o un libro completo. Los modelos clásicos en recuperación de la información consideran que cada documento está descrito por un conjunto representativo de palabras claves llamadas *términos*, que son palabras cuya semántica ayuda a definir los temas principales del documento.

Uno de estos modelos, el *modelo vectorial*, considera un documento como un vector t-dimensional,

donde t es el número total de términos del sistema. Cada coordenada i del vector está asociada a un término del documento, cuyo valor corresponde a un "peso" positivo w_{ij} si es que dicho término pertenece al documento ó 0 en caso contrario. Si $\mathbb{D}$ es el conjunto de documentos y d_j es el j-ésimo documento perteneciente a $\mathbb{D}$, entonces $d_j = (w_{1j}, w_{2j}, \ldots, w_{tj})$.

En el modelo vectorial se calcula el *grado de similitud* entre un documento d y una consulta q, la cual puede ser vista como un conjunto de términos o como un documento completo, como el grado de similitud entre los vectores $\vec{d_j}$ y $\vec{q}$. Esta correlación puede ser cuantificada, por ejemplo, como el coseno del ángulo formado entre ambos vectores:

$$sim(d_j, q) = \frac{\vec{d_j} \cdot \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \times \sum_{i=1}^{t} w_{iq}^2}} \tag{2.5}$$

donde w_{iq} es el peso del i-ésimo término en la consulta q.

Los pesos de los términos pueden calcularse de varias formas. Una de las más importantes es utilizando *esquemas tf-idf*, en donde los pesos están dados por:

$$w_{ij} = f_{i,j} \times \log\left(\frac{N}{n_i}\right)$$

donde N es el número total de documentos, n_i es el número de documentos en donde el i-ésimo término aparece, y $f_{i,j}$ es la *frecuencia normalizada* del i-ésimo término, dada por:

$$f_{i,j} = \frac{freq_{i,j}}{\max_{\ell=1\ldots t}(freq_{\ell,j})}$$

donde $freq_{i,j}$ es la frecuencia del i-ésimo término en el documento d_j, y $\max_{\ell=1\ldots t}(freq_{\ell,j})$ es el máximo valor sobre todos los términos contenidos en d_j.

Si se considera que los documentos son puntos en un espacio métrico, el problema de la búsqueda de documentos similares a una consulta dada se reduce a una búsqueda en proximidad en el espacio métrico. Dado que $sim(d_j, q)$ sólo es una medida de similaridad, que en particular no cumple

con la desigualdad triangular, se utiliza el ángulo formado entre los vectores $\vec{d_j}$ y $\vec{q}$, $d(d_j, q) = \arccos(sim(d_j, q))$, como función de distancia. Por lo tanto, $(\mathbb{D}, d)$ es un espacio métrico.

2.3 Búsqueda en proximidad en espacios métricos

Sea $\mathbb{U} \subseteq \mathbb{X}$ el conjunto de objetos que conforma la base de datos, con $|\mathbb{U}| = n$ (note que $(\mathbb{U}, d)$ también es un espacio métrico). El concepto de *búsqueda en proximidad* consiste en recuperar todos los elementos pertenecientes a $\mathbb{U}$ que sean similares a un elemento de consulta $q \in \mathbb{X}$.

Existen dos consultas típicas de búsqueda en proximidad:

- *Consulta por rango*: una consulta por rango $(q, r)_d$ consiste en recuperar todos los elementos $u \in \mathbb{U}$ que se encuentren a lo más a distancia $r \in \mathbb{R}^+$ del objeto de consulta q:

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

  La Figura 2.2 muestra un ejemplo de una consulta por rango en un espacio vectorial de dimensión 2, donde $(q, r)_{L_2} = \{u_6, u_{10}, u_{14}\}$. El área del espacio definida por la consulta $(q, r)_d$ se conoce como la *bola de consulta*, y todos los objetos que caen dentro de ella son los *relevantes* a la consulta.

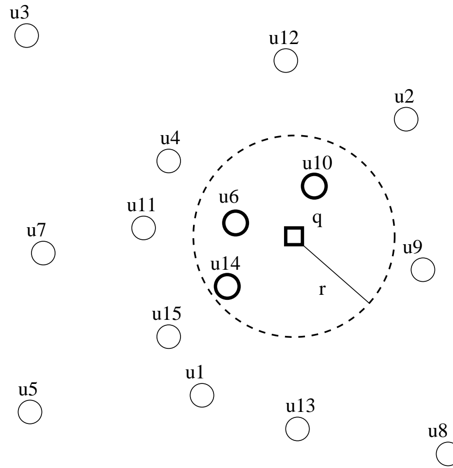

Figura 2.2: Ejemplo de una consulta por rango en un espacio vectorial de dimensión 2.

- *Consulta por los k vecinos más cercanos (k-NN)*: consiste en recuperar los k elementos más cercanos a q en $\mathbb{U}$. Esto es, recuperar un conjunto $C \subseteq \mathbb{U}$ tal que $|C| = k$ y $\forall x \in C, y \in \mathbb{U} - C, d(x, q) \leq d(y, q)$.

La consulta más básica es la consulta por rango, pudiéndose implementar la búsqueda de los k vecinos más cercanos a partir de consultas por rango.

Una manera trivial de responder ambos tipos de consulta es realizando una búsqueda exhaustiva, esto es, examinando completamente la base de datos. De hecho, esta es la única manera de realizar búsquedas en proximidad si no es posible preprocesar la información para construir un índice que ayude a responder las consultas. Sin embargo, en términos generales la función de distancia se considera *computacionalmente costosa de calcular*. Piense, por ejemplo, en un dispositivo biométrico que necesita comparar huellas digitales para identificar a una persona en particular. El proceso de comparar dos huellas digitales, esto es, medir a qué distancia se encuentran, es un proceso complicado y costoso desde el punto de vista computacional. Por este motivo, no es práctico realizar una búsqueda exhaustiva para realizar búsquedas en proximidad en aplicaciones reales, ya que tomaría mucho tiempo responder cada consulta realizada. En este estudio se utilizará el *número de cómputos de distancia* como la medida de complejidad de los algoritmos de búsqueda, obviando otros costos tales como E/S o tiempo extra de CPU.

Un *algoritmo de indexamiento* es un procedimiento que construye una estructura de datos denominada *índice*, el cual permite realizar búsquedas en proximidad evitando, en lo posible, el tener que revisar toda la base de datos, ahorrándose evaluaciones de distancia al momento de realizar las consultas. Todos los algoritmos de búsqueda en espacios métricos se basan en el descarte de objetos utilizando la desigualdad triangular, lo cual evita el tener que medir su distancia real con el objeto de consulta.

Capítulo 3

Algoritmos de Búsqueda en Proximidad en Espacios Métricos

El objetivo de estos algoritmos es reducir al mínimo el número de evaluaciones de distancia necesarios para responder las consultas, ya sean por rango o los k vecinos más cercanos.

Los algoritmos de búsqueda en espacios métricos se dividen en dos grandes áreas: algoritmos basados en pivotes y algoritmos basados en particiones compactas (ver [15] para un estudio completo).

3.1 Algoritmos basados en pivotes

Un *pivote* es un objeto distinguido dentro de la base de datos. Los pivotes sirven para filtrar objetos en una consulta utilizando la desigualdad triangular, sin medir directamente la distancia entre el objeto de consulta y los objetos descartados. A continuación se describirá el algoritmo canónico de búsqueda en proximidad basado en pivotes.

Sea $(\mathbb{X}, d)$ un espacio métrico y $\mathbb{U} \subseteq \mathbb{X}$ un conjunto de objetos. Dada una consulta por rango $(q, r)_d$ y un conjunto de k pivotes $\{p_1, \ldots, p_k\}$, $p_i \in \mathbb{U}$, por la desigualdad triangular, para cualquier

$x \in \mathbb{X}$, se cumple que $d(p_i, x) \leq d(p_i, q) + d(q, x)$, y además que $d(p_i, q) \leq d(p_i, x) + d(x, q)$. De ambas ecuaciones se sigue que una cota inferior para $d(q, x)$ es $d(q, x) \geq |d(p_i, x) - d(p_i, q)|$. Los objetos $u \in \mathbb{U}$ de interés son aquellos que satisfacen $d(q, u) \leq r$, por lo que pueden ser excluidos todos aquellos objetos que satisfagan la siguiente *condición de exclusión*:

$$|d(p_i, u) - d(p_i, q)| > r \text{ para algún } p_i \tag{3.1}$$

sin necesidad de evaluar directamente $d(q, u)$.

Si el conjunto $\mathbb{U}$ contiene n objetos, se construye un índice consistente en las kn distancias $d(u, p_i)$ entre cada objeto perteneciente a $\mathbb{U}$ y cada pivote. Al momento de realizar una consulta por rango sólo es necesario calcular las k distancias entre los pivotes y q para poder aplicar la condición de exclusión (3.1). Dichas evaluaciones de distancia se conocen como la *complejidad interna* del algoritmo de búsqueda, y dicha complejidad es fija si se utiliza un número fijo de pivotes.

La lista de objetos $\{u_1, \ldots, u_m\} \subseteq \mathbb{U}$ que no fueron excluidos, conocida como la *lista de objetos candidatos*, debe ser comparada directamente con q. Dichas evaluaciones de distancia son conocidas como la *complejidad externa* del algoritmo de búsqueda.

La complejidad total del algoritmo es la suma de las complejidades interna y externa, $k + m$. Dado que una se incrementa y la otra disminuye con k, existe un número óptimo k^* de pivotes que depende del radio de tolerancia de la consulta. Sin embargo, en la práctica k^* resulta ser muy grande, por lo que no es posible almacenar las k^*n distancias, y el índice se compone del número máximo de pivotes que la memoria disponible permita.

En general, el conjunto de pivotes se elige en forma aleatoria, aunque se han propuesto algunas técnicas de selección de pivotes que mejoran significativamente el rendimiento [10].

3.1.1 Ejemplos de algoritmos basados en pivotes

Estos algoritmos difieren de la versión canónica en la forma como se utilizan los pivotes, que puede ser un poco más sutil. Algunos utilizan un árbol como estructura de datos para indexar la base de datos, en donde las raíces de los subárboles corresponden a los pivotes. Otros utilizan tiempo de CPU adicional para ahorrar espacio en memoria y así poder almacenar más pivotes.

Burkhard-Keller Tree

Este algoritmo, presentado en [8], fue una de las primeras soluciones para la búsqueda en espacios métricos donde la función de distancia es discreta. Consiste en construir un árbol (*Burkhard-Keller Tree* o *BKT*) a partir de un nodo p elegido arbitrariamente del conjunto de objetos. Para cada distancia $i > 0$ se define el subconjunto de los objetos que se encuentran a distancia i de la raíz p. Para cada uno de los subconjuntos no vacíos se crea un hijo de p, en donde recursivamente se construye un BKT. El proceso se repite hasta que quede un solo objeto en el subconjunto o hasta que queden menos de b objetos, los cuales se almacenan en un *bucket*.

Dada una consulta por rango $(q, r)_d$, sólo se examinan aquellas ramas en donde se cumpla que $d(p, q) - r \leq i \leq d(p, q) + r$, y se procede recursivamente en cada subárbol, añadiéndose a los resultados todos aquellos objetos encontrados durante el recorrido del árbol que se encuentren a distancia menor o igual que r de q. Al llegar a una hoja del BKT, se realiza una búsqueda exhaustiva sobre todos los objetos contenidos en ella. La desigualdad triangular asegura que el algoritmo responde la consulta de manera correcta.

Las raíces p de todos los subárboles del BKT corresponden a los pivotes que utiliza el algoritmo para indexar el espacio métrico, aunque los pivotes son sólo útiles en el subárbol del que son raíces.

Fixed-Queries Tree

En [4] se presenta el algoritmo *Fixed-Queries Tree* o *FQT*, que es una variante del algoritmo BKT donde se ocupa un solo pivote para cada nivel del árbol, es decir, un solo pivote p es la raíz de todos

los subárboles de un mismo nivel.

El algoritmo para responder consultas es exactamente el mismo que el del BKT, pero en este caso sólo se calcula la distancia entre el pivote y la consulta una vez por nivel. Por otro lado, los árboles FQT tienden a ser más altos que los BKT.

Fixed-Height FQT

En [4] también se presenta el *Fixed-Height FQT* o *FHQT*, que es una variante del FQT en donde todas las hojas se encuentran a la misma altura h, independientemente del tamaño de los buckets. El FHQT equivale exactamente a indexar el espacio con h pivotes fijos, pero el árbol permite encontrar los objetos candidatos con un costo de CPU menor.

Fixed Queries Array

En [12] se presenta el *Fixed Queries Array* o *FQA*. Este algoritmo representa en forma compacta un FHQT, utilizando un arreglo, lo que, a cambio de un factor de CPU extra de $O(\log(n))$, permite ahorrar memoria y por ende utilizar una mayor cantidad de pivotes.

Vantage Point Tree

Este algoritmo [31] permite realizar consultas cuando la función de distancia es continua. El *Vantage Point Tree* o *VPT* consiste en construir un árbol binario, en donde la raíz es un objeto p elegido aleatoriamente del conjunto de objetos. Luego, se calcula la mediana del conjunto de todas las distancias, como $M = mediana\{d(p,u), u \in \mathbb{U}\}$. Aquellos objetos $u \in \mathbb{U}$ tal que $d(p,u) \leq M$ son insertados en el subárbol izquierdo, y los restantes son insertados en el subárbol derecho. Para responder una consulta $(q,r)_d$ en el VPT se calcula $dist = d(q,p)$, donde p es la raíz del VPT. Si $dist - r \leq M$ se busca recursivamente en el subárbol izquierdo, y si $dist + r > M$ se busca recursivamente en el subárbol derecho. Nótese que es posible entrar en ambos subárboles a la vez. A medida que se recorre el árbol se añaden a los resultados todos aquellos objetos p que se encuentren dentro de la bola de consulta.

Multi Vantage Point Tree

El VPT puede ser extendido a árboles t-arios usando $t-1$ percentiles uniformes en vez de la mediana. Este algoritmo recibe el nombre de *Multi Vantage Point Tree* o *MVPT* [6]. A su vez el MVPT permite tener varios pivotes por nodo, lo cual lo convierte en un híbrido con el FHQT.

Excluded Middle Vantage Point Forest

Otra extensión al VPT es el *Excluded Middle Vantage Point Forest* o *VPF* [32]. Este algoritmo está diseñado para responder consultas por el objeto más cercano con radio de búsqueda limitado, esto es, encontrar el vecino más cercano a q si es que éste está a una distancia de a lo más r. El índice excluye en cada nivel del VPT los objetos que se encuentren a distancias intermedias del pivote, y con esta parte excluida se construye un segundo árbol, con lo que se obtiene un bosque. Con esto es posible eliminar el *backtracking* al realizar búsquedas con un radio pequeño, pero es necesario realizar la búsqueda en todos los árboles del bosque. El algoritmo se puede adaptar para realizar consultas por rango.

Approximating Eliminating Search Algorithm

Un enfoque completamente distinto es el que utiliza el algoritmo *Approximating Eliminating Search Algorithm* o *AESA* [29]. El índice construido es simplemente una matriz en donde se precalculan las $\frac{n(n-1)}{2}$ distancias entre los objetos del espacio $\mathbb{U}$. Dada una consulta por rango $(q, r)_d$, se selecciona aleatoriamente un objeto $p \in \mathbb{U}$ y se calcula $dist = d(p, q)$. Se excluyen todos los objetos $u \in \mathbb{U}$ que no cumplan con la condición de exclusión (3.1) para ese pivote p. Como todas las distancias $d(p, u)$ están precalculadas, sólo $d(p, q)$ debe ser calculada al momento de realizar la búsqueda. El proceso de elección de pivotes y exclusión de objetos se repite hasta haber descartado o retornado todos los objetos.

Este algoritmo es uno de los que tienen mejor rendimiento en la práctica, pero tiene costo en espacio y tiempo de construcción $O(n^2)$, lo cual es demasiado costoso incluso para bases de datos pequeñas.

Linear AESA

Una nueva versión del algoritmo AESA, llamada *Linear AESA* o *LAESA* [22], propone utilizar una cantidad fija de pivotes, k . En este caso, el costo en espacio y tiempo de construcción es $O(kn)$. Aquellos objetos que no pueden ser excluidos después de considerar los k pivotes son comparados directamente con la consulta q.

Este algoritmo es prácticamente una implementación directa del algoritmo canónico de búsqueda en proximidad utilizando pivotes.

Spaghettis

En [11] se presenta este algoritmo de búsqueda en espacios métricos basado en arreglos y que es una variante de LAESA. Propone reducir el tiempo de CPU extra necesario al realizar una consulta utilizando una estructura de datos en donde las distancias de los objetos del conjunto a cada pivote están ordenadas, pero necesita espacio extra para almacenar la información que permita seguir el recorrido de un objeto en particular a través de los distintos arreglos.

3.2 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*.

A continuación se describen los criterios de exclusión para delimitar las zonas, se explican en detalle dos de las estructuras de datos más eficientes para búsquedas en proximidad y se describen otros algoritmos basados en particiones compactas.

3.2.1 Criterios de exclusión

Existen dos criterios generales para delimitar una zona: *partición de Voronoi* y *radio cobertor*.

Criterio de partición de Voronoi

Se elige un conjunto de m centros $\{c_1, c_2, \ldots, c_m\}$. 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 3.1 ilustra este concepto en un espacio vectorial de dimensión 2.

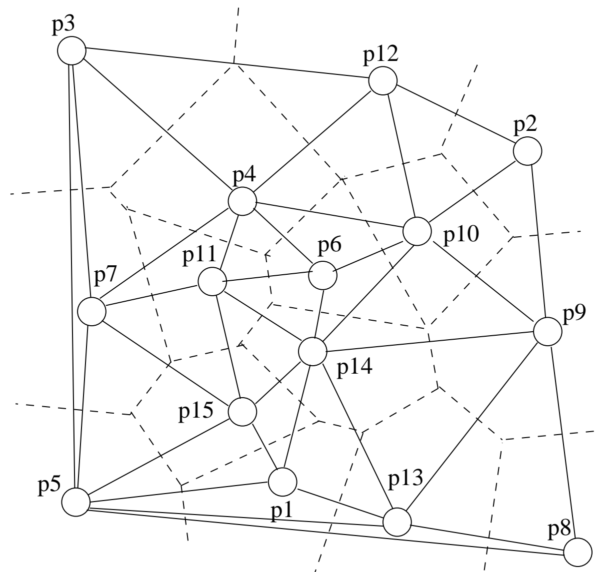

Figura 3.1: Partición de Voronoi de un espacio vectorial de dimensión 2.

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

Si la bola de consulta $(q, r)_d$ intersecta la zona de algún centro distinto a c, entonces se debe revisar exhaustivamente esa zona por si hay objetos que caigan dentro de la bola de consulta. La Figura 3.2 ilustra esta situación. Sea $x \in \mathbb{X}$ 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:

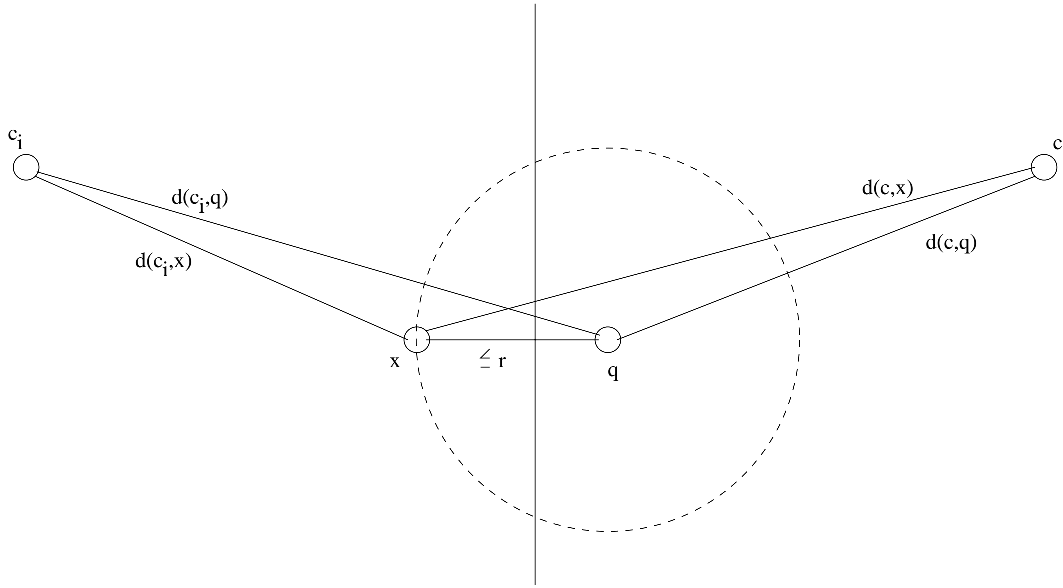

Figura 3.2: Condición de exclusión con criterio de partición de Voronoi.

$$d(c,x) \leq d(c,q) + d(q,x) \leq d(c,q) + r \tag{3.2}$$

Y también se tiene que:

$$d(c_i,q) \leq d(c_i,x) + d(x,q) \leq d(c_i,x) + r \quad \Longrightarrow \quad d(c_i,q) - r \leq d(c_i,x) \tag{3.3}$$

Como x pertenece a la zona de c_i, se cumple que $d(c_i,x) \leq d(c,x)$. De esta condición más (3.2) y (3.3) se obtiene:

$$d(c_i,q) - r \leq d(c_i,x) \leq d(c,x) \leq d(c,q) + r \quad \Longrightarrow \quad d(c_i,q) - r \leq d(c,q) + r \tag{3.4}$$

Dada la condición (3.4), se pueden descartar las zonas cuyo centro c_i satisfaga la condición $d(c_i,q) > d(c,q) + 2r$, puesto que en ese caso dicha zona no puede tener intersección con la bola de consulta.

Criterio de radio cobertor

El radio cobertor es la máxima distancia entre un centro c y algún objeto perteneciente a su zona, y se denotará como $cr(c)$.

La Figura 3.3 muestra cuál es el criterio de exclusión utilizando el radio cobertor. Dada una consulta por rango $(q,r)_d$, se tiene que si $d(q,c) - r > cr(c)$, entonces la bola de consulta no puede tener intersección con la zona de centro c, y por lo tanto se pueden excluir de la respuesta todos los objetos pertenecientes a la zona de centro c sin necesidad de chequearlos directamente. En caso contrario, es necesario revisar exhaustivamente dicha zona por si hay objetos que se encuentren dentro de la bola de consulta. En la figura, la bola de consulta (q_1, r) no tiene intersección con la zona de centro c, por lo que no es necesario revisar los objetos que pertenezcan a dicha zona. Por el contrario, en las consultas q_2 y q_3 sí hay intersección, lo que hace necesario revisar exhaustivamente dicha zona en busca de objetos que estén a distancia menor que r de la consulta.

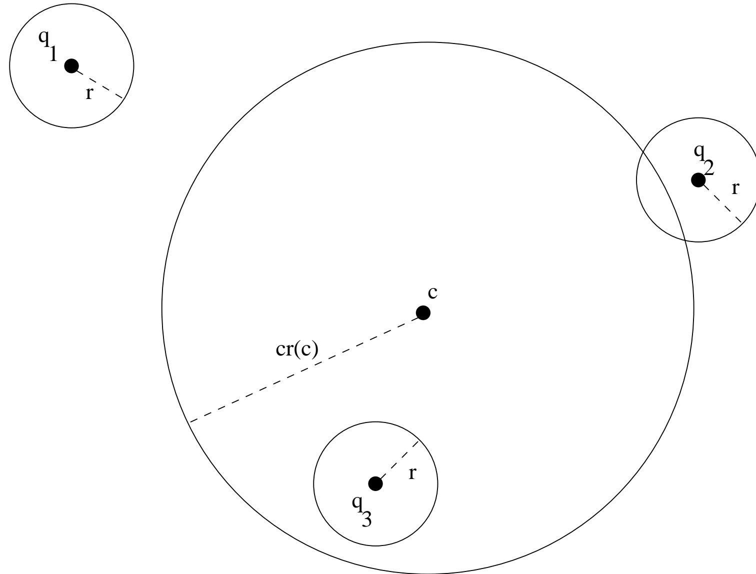

Figura 3.3: Condición de exclusión con criterio de radio cobertor.

3.2.2 Spatial Approximation Tree

El *Spatial Approximation Tree* o *SAT* [23] es una estructura de datos para búsqueda en proximidad que utiliza tanto el criterio del radio cobertor como el criterio de partición de Voronoi. Utiliza una cantidad de espacio lineal $O(n)$ con respecto al tamaño de la base de datos, tiene un tiempo promedio de construcción de $O(n \log^2(n)/\log(\log(n)))$ y tiempo promedio de búsqueda sublineal $O(n^{1-\Theta(1/\log(\log(n)))})$ en espacios métricos de dimensión alta y $O(n^\alpha)$ $(0 < \alpha < 1)$ en espacios métricos de dimensión baja.

Método de aproximación espacial

La idea de esta estructura es acercarse espacialmente a la consulta, aproximándose a través de los *vecinos* de un objeto. Sea $\mathbb{U}$ el conjunto de objetos. Cada objeto $u \in \mathbb{U}$ posee un conjunto de vecinos $N(u)$, y sólo se permite trasladarse directamente a objetos vecinos.

El proceso de búsqueda para responder consultas del vecino más cercano de un objeto $q \in \mathbb{X}$ es el siguiente: se comienza la búsqueda desde algún objeto $a \in \mathbb{U}$ cualquiera y se consideran todos sus vecinos. Si ningún vecino es más cercano a q que a a, entonces a es el vecino más cercano a q. En caso contrario, se elige algún vecino $b \in N(a)$ que esté más cercano a q que a, y se continúa la búsqueda por ese vecino. La Figura 3.4 muestra un ejemplo de una búsqueda por aproximación espacial. El algoritmo puede verse como un recorrido en un grafo, donde los arcos del grafo conectan a un objeto con sus vecinos.

Para que el algoritmo descrito funcione, es necesario que el grafo contenga suficientes arcos. Una solución es utilizar el grafo completo, pero esto no tiene sentido pues sólo chequear los vecinos tendría costo $O(n)$. La estructura ideal sería un grafo que tuviera el mínimo número de arcos y contestara de manera correcta todas las posibles consultas.

En [23] se muestra que el grafo ideal es el obtenido a través de una triangulación de Delaunay, donde los vecinos del nodo u corresponden a los objetos cuya área de Voronoi comparte un borde con u. Desafortunadamente, no es posible obtener el grafo de Delaunay de un espacio métrico general utilizando sólo las distancias entre los objetos del espacio, y se puede demostrar que el único grafo que funciona en un espacio métrico arbitrario es el grafo completo [23], lo cual ya se vió que no es

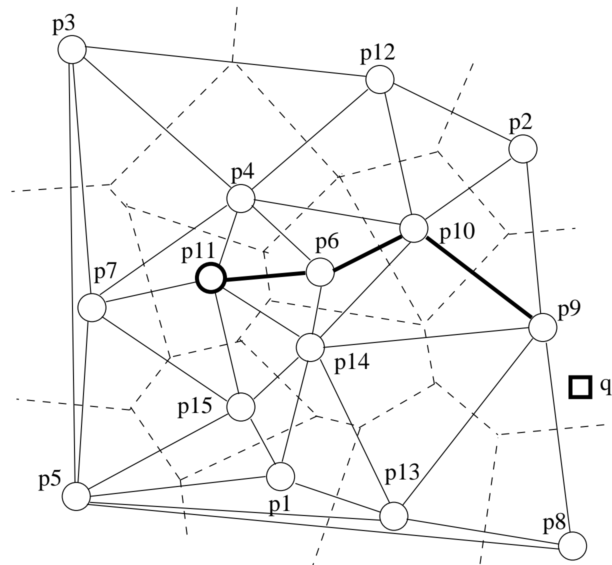

Figura 3.4: Ejemplo de búsqueda por aproximación espacial.

útil en la práctica.

No obstante lo anterior, es posible hacer dos simplificaciones a la idea descrita y obtener una solución factible. La primera simplificación es restringir el inicio de la búsqueda a un nodo fijo (la raíz del árbol), y la segunda restricción es que la estructura sólo pueda responder de manera correcta consultas por objetos que pertenezcan a la base de datos. Más adelante se verá que combinando el método de aproximación espacial con *backtracking* es posible responder cualquier búsqueda en proximidad utilizando esta estructura de datos.

Construcción del SAT

Se elige aleatoriamente un objeto $a \in \mathbb{U}$ como raíz del árbol. Luego se elige un conjunto de *vecinos* $N(a)$ que satisfaga la siguiente condición:

$$\forall x \in \mathbb{U}, x \in N(a) \Leftrightarrow \forall y \in N(a) - \{x\}, d(x, y) > d(x, a) \quad (3.5)$$

Los vecinos de la raíz a forman un conjunto tal que cualquier vecino se encuentra más cerca de a

que de cualquier otro vecino. El "sólo si" ($\Leftarrow$) de la definición garantiza que si es posible acercarse a cualquier $b \in \mathbb{U}$, entonces algún objeto en $N(a)$ es más cercano a b que a a, porque durante la construcción del árbol se colocó como vecinos directos todos aquellos objetos que no estaban más cerca a algún otro vecino. El "si" ($\Rightarrow$) de la definición apunta a que la cantidad de vecinos sea lo más pequeña posible.

Note que el conjunto $N(a)$ está definido en términos de sí mismo en una manera no trivial, y que pueden haber múltiples soluciones que calcen con la definición. Por ejemplo, dados $b, c \in \mathbb{U}$, si a se encuentra lejos de b y c y éstos se encuentran cercanos entre sí , entonces tanto $N(a) = \{b\}$ como $N(a) = \{c\}$ satisfacen la definición.

Encontrar el conjunto $N(a)$ más pequeño posible pareciera ser un problema de optimización combinatorial difícil de resolver [23], puesto que al incluir un objeto puede ser necesario sacar otros previamente elegidos (lo que ocurre entre b y c en el ejemplo anterior). Sin embargo, existe una heurística simple que añade más vecinos que los necesarios, pero que funciona bien en la práctica.

Inicialmente, se comienza con el nodo a y su "bolsa" que contiene los objetos restantes de $\mathbb{U}$. Primero se ordena la bolsa ascendentemente según la distancia a a. Luego, se añaden nodos a $N(a)$, que inicialmente se encuentra vacío. En cada oportunidad se considera un nuevo nodo b y se verifica si se encuentra más cerca a algún objeto de $N(a)$ que a a. Si ese no es el caso, se añade b a $N(a)$.

Una vez finalizado este proceso se tiene un conjunto apropiado de vecinos. Note que la condición 3.5 se satisface gracias al hecho que se consideraron los objetos en orden de distancia creciente a a. El "sólo si" de la condición se satisface claramente, puesto que sólo aquellos objetos que la cumplían fueron agregados a $N(a)$. El "si" es un poco más delicado. Sea $x \neq y \in N(a)$, si y está más cerca de a que x, entonces y fue considerado primero. El algoritmo de construcción garantiza que si se insertó x en $N(a)$, entonces $d(x, a) < d(x, y)$. En cambio si x es más cercano a a que y, entonces $d(y, x) > d(y, a) \geq d(x, a)$, lo que significa que un vecino no puede ser removido por un nuevo vecino insertado posteriormente.

Cada nodo $x \in \mathbb{U} - N(a)$ se inserta en la bolsa del objeto perteneciente a $N(a)$ más cercano a x. Esto se conoce como estrategia de *best-fit*. Note que esto requiere una segunda pasada sobre $\mathbb{U}$ una vez que $N(a)$ ha sido determinado.

Una vez terminado con a se procede recursivamente sobre todos sus vecinos, considerando los objetos que quedaron en su bolsa respectiva. La estructura resultante es un árbol que puede ser utilizado para realizar búsquedas para cualquier $q \in \mathbb{U}$ por aproximación espacial, para consultas de vecino más cercano. La razón por la cual funciona es que, al momento de realizar la búsqueda, se repite exactamente el proceso realizado con q durante el proceso de construcción, esto es, se entra en el subárbol del vecino más cercano a q hasta encontrarlo. Lo anterior funciona puesto que q se encuentra en el árbol, es decir, se está realizando una búsqueda exacta.

Finalmente, se almacena en cada nodo a su radio cobertor, esto, es la máxima distancia $cr(a)$ entre a y algún objeto en alguno de sus subárboles. Esto servirá posteriormente para ahorrar algunas comparaciones al momento de realizar una búsqueda.

La Figura 3.5 muestra en seudocódigo el proceso de construcción del SAT. El método se invoca inicialmente como `ConstruirArbol(`a`, `$\mathbb{U}$`-{`a`})`, donde a es un objeto elegido aleatoriamente de $\mathbb{U}$. Note que, excepto en el primer nivel de recursión, ya se conocen las distancias $d(x, a)$ para todo $x \in \mathbb{U}$ y por lo tanto no es necesario volver a calcularlas. De la misma forma, algunas de las distancias $d(x, c)$ en la línea 10 ya fueron calculadas en la línea 6. La información almacenada en la estructura de datos es la raíz a y los valores de $N()$ y $cr()$ para todos los nodos.

Algoritmos de búsqueda en SAT

Consulta por rango $(q, r)_d$. Si bien, en una consulta general, $q \notin \mathbb{U}$, las respuestas a la consulta son objetos $q' \in \mathbb{U}$, por lo que el árbol se utiliza para simular la búsqueda de un objeto perteneciente a $\mathbb{U}$. Cuando el objeto q es conocido y se comienza la búsqueda en la raíz a, la aproximación espacial indica que se debe buscar directamente en el vecino de a más cercano a q. Como la búsqueda es sobre un objeto q' desconocido y no se tiene la seguridad de cuál es el vecino de a más cercano a q', se deben explorar varios vecinos. Algunos de ellos pueden ser descartados utilizando que $d(q, q') \leq r$ (ya que los objetos q' se encuentran dentro de la bola de consulta).

Primero se determina cuál es el objeto c más cercano a q, con $c \in \{a\} \cup N(a)$. Luego, para todo $b \in \{a\} \cup N(a)$ se sabe que $d(c, q) \leq d(b, q)$, pero es posible que $d(c, q') \geq d(b, q')$ por lo explicado anteriormente, y no será posible encontrar q' entrando sólo en el subárbol de raíz c. En lugar de esto, se deben visitar *todos* los vecinos $b \in N(a)$ tal que $d(q, b) \leq d(q, c) + 2r$ (criterio

```
ConstruirArbol (Nodo a, Conjunto de nodos S)

1.  N(a) ← ∅          /* vecinos de a */
2.  cr(a) ← 0         /* radio cobertor */
3.  Ordenar S por distancias a a (más cercanos primero)
4.  for v ∈ S do
5.      cr(a) ← max(cr(a), d(v,a))
6.      if ∀b ∈ N(a), d(v,a) < d(v,b) then N(a) ← N(a) ∪ {v}
7.  enddo
8.  for b ∈ N(a) do S(b) ← ∅          /* subárboles */
9.  for v ∈ S − N(a) do
10.     c ← arg min(d(v,b)),  b ∈ N(a)
11.     S(c) ← S(c) ∪ {v}
12. enddo
13. for b ∈ N(a) do ConstruirArbol (b, S(b))     /* construye subárboles */
```

Figura 3.5: Algoritmo de construcción del SAT.

de partición de Voronoi). Esto viene de lo siguiente: por la desigualdad triangular para cualquier $u \in \mathbb{U}$ se cumple que $d(u,q) - r \leq d(u,q') \leq d(u,q) + r$; si para algún vecino $b \in N(a)$ se cumple que $d(q,b) > d(q,c) + 2r$, también satisface que $d(q',b) \geq d(q,b) - r > d(q,c) + r \geq d(q',c)$, y por lo tanto q' no pudo haber sido insertado en el subárbol de b. En cualquier otro caso no existe seguridad, por lo que debe entrarse en dicho subárbol.

El proceso garantiza que q será comparado contra todo nodo que no pueda probarse que está suficientemente lejos de q. Debe añadirse al resultado todo aquel objeto q' comparado con q y que cumpla $d(q,q') \leq r$. Como puede observarse, lo que inicialmente se concibió como una búsqueda por aproximación espacial a través de un camino simple se combina ahora con *backtracking*, por lo que se debe buscar por varios caminos. La Figura 3.6 muestra cómo se realiza el proceso de búsqueda, donde la raíz del árbol es p_{11}. Todas las ramas marcadas fueron recorridas, pero sólo p_9 es retornado.

El algoritmo de búsqueda puede ser mejorado un poco considerando lo siguiente: cuando se realiza una búsqueda de un objeto $u \in \mathbb{U}$ (búsqueda exacta) se sigue un único camino desde la raíz

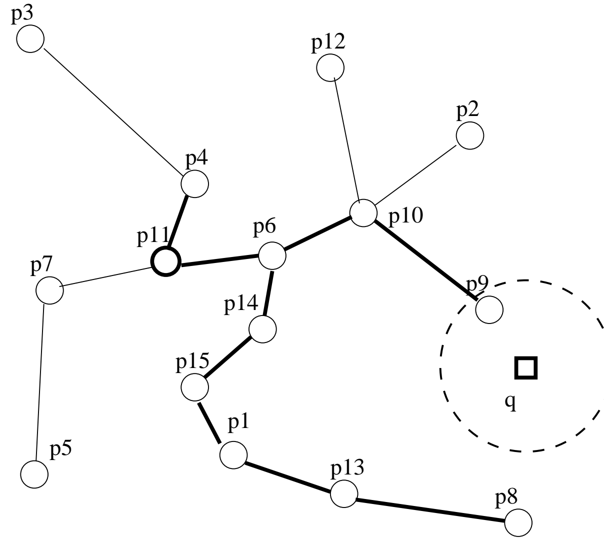

Figura 3.6: Ejemplo del proceso de búsqueda por rango en SAT.

hasta u. Para cualquier nodo a' en este camino, se elige aquél más cercano a q entre $\{a'\} \cup N(a')$. Por lo tanto, si la búsqueda se encuentra en el nodo a se tiene que q es más cercano a a que a cualquier ancestro a' de a y que a cualquier vecino de a'. Si se define $A(a)$ como el conjunto de los ancestros de a (incluyendo a), se tiene que, al momento de realizar la búsqueda, se pueden descartar aquellos subárboles cuya raíz $x \in N(a)$ cumpla que:

$$d(q,x) > 2r + \min\{d(q,c), c \in \{a'\} \cup N(a'), a' \in A(a)\} \tag{3.6}$$

dado que se puede demostrar, usando la desigualdad triangular, que ningún q' puede estar almacenado en el subárbol de raíz x. La condición 3.6 es una versión más estricta que la condición original $d(q,x) > 2r + \min\{d(a,c), c \in \{a\} \cup N(a)\}$.

Esta observación se puede utilizar de la siguiente forma: para cada nodo b visitado durante la búsqueda se recuerda la mínima distancia d_{min} a q hasta este momento a través de este camino, incluyendo a los vecinos. Sólo se revisan recursivamente aquellos vecinos que no estén más lejos que $d_{min} + 2r$ de q.

Por último, se utiliza la información del radio cobertor $cr(a)$ para reducir aún más el costo de búsqueda. Se puede detener la búsqueda en aquellos subárboles con raíz a donde $d(q,a) > cr(a) + r$

(criterio de radio cobertor), dado que esto implica que $d(q', a) > cr(a)$ para cualquier q' tal que $d(q, q') \leq r$, y por lo tanto q' no puede pertenecer al subárbol de a.

La Figura 3.7 muestra el seudocódigo del algoritmo de búsqueda por rango utilizando el SAT. La función se invoca inicialmente como *BusquedaRangoSAT(a, q, r, d(q,a))*, donde a es la raíz del árbol. Note que en los llamados recursivos $d(a, q)$ ya ha sido previamente calculado.

```
BusquedaRangoSAT(Nodo a, Consulta q, Radio r, Distancia d_min)

1.  if d(a,q) <= cr(a) + r then
2.      if d(a,q) <= r then añadir a al resultado
3.      d_min <- min{d_min} ∪ {d(c,q), c ∈ N(a)}
4.      for b ∈ N(a) do
5.          if d(b,q) <= d_min + 2r then BusquedaRangoSat(b,q,r,d_min)
```

Figura 3.7: Algoritmo de búsqueda por rango en SAT.

Consulta por k vecinos más cercanos a q. Este tipo de consultas se puede responder simulando una consulta por rango, reduciendo el radio de búsqueda a medida que se dispone de más información. Para resolver consultas del tipo 1-NN (vecino más cercano) se comienza buscando con una tolerancia $r = \infty$, reduciendo ésta cada vez que se realice una comparación que entregue una distancia menor que r. Al final, se retorna el objeto más cercano a la consulta encontrado durante la búsqueda.

Para responder consultas del tipo k-NN se utiliza una cola de prioridad con los k objetos más cercanos a q conocidos hasta el momento. El radio de tolerancia r corresponde a la distancia entre q y el candidato más lejano a q en la cola de prioridad ($r = \infty$ si se tienen menos de k candidatos). Cada vez que se encuentra un nuevo candidato se inserta en la cola, pudiendo desplazar a algún candidato previo y por ende reducir r. Al final, la cola contiene los k objetos más cercanos a q.

Un aspecto importante al resolver este tipo de consultas es encontrar rápidamente objetos cercanos a q y por ende reducir r lo más temprano posible. Esto implica que el orden en el cual se recorre el árbol de aproximación espacial cobra mucha importancia, lo cual no sucede en el caso de la búsqueda por rango. En [27] se propone una idea general que puede ser adaptada al SAT y que consiste en lo siguiente [23]: se tiene una cola de prioridad de subárboles, los más prometedores

primero. Inicialmente, se inserta la raíz del SAT en la cola. Luego, iterativamente se extrae el subárbol más promisorio de la cola, se procesa, y se insertan todas las raíces de sus subárboles en la cola. Este proceso se repite hasta que la cola se vacíe o la raíz del subárbol más prometedor pueda ser descartada, lo cual implica que ninguno de los subárboles no procesados puede contener un objeto suficientemente "bueno".

Para medir cuán prometedor es un subárbol se utiliza una cota inferior de la distancia entre q y cualquier objeto perteneciente al subárbol. Una vez que esta cota inferior exceda r se puede detener la búsqueda. En el caso particular de la búsqueda en el SAT se tienen dos cotas inferiores posibles:

1. Dado que se conoce el vecino más cercano c y se revisan los vecinos b que cumplan $d(q,b) - d(q,c) \leq 2r$, se tiene que una cota inferior para r es $r \geq \frac{d(q,b)-d(q,c)}{2}$. De hecho, este nodo c se toma de entre los vecinos de cualquier ancestro del nodo actual.
2. La cota que entrega el criterio de radio cobertor, esto es $d(q,b) - cr(b) \leq r$.

Como r se reduce durante la búsqueda, una raíz b que pareciera ser prometedora al momento de insertarla en la cola de prioridad puede no serlo al momento de extraerla. Las raíces se insertan con el máximo valor entre las dos cotas inferiores, y se utiliza este valor para ordenar los subárboles en la cola, donde el subárbol más prometedor es aquel con cota de valor mínimo. A medida que se extraen los subárboles de la cola, se verifica si su cota excede r, en cuyo caso se detiene el proceso de búsqueda dado que no pueden haber objetos relevantes en los subárboles restantes. Note que los nodos vecinos heredan la cota inferior de su nodo padre, y que es necesario mantener el valor de d_{min} por separado.

El método descrito es *rango-óptimo*, esto es, encuentra el k-ésimo vecino más cercano de q, o_k, después de visitar los mismos nodos del SAT que al realizar una búsqueda por rango con radio $d(q,o_k)$.

La Figura 3.8 muestra el seudocódigo del algoritmo de consulta por k vecinos más cercanos utilizando el SAT.

```
BusquedaVecinosSAT (Arbol a, Consulta q, Cantidad de vecinos k)

1.  Q ← {(a, max(0, d(q,a) − cr(a)), d(q,a))}     /* subárboles prometedores */
2.  r ← ∞,  A ← ∅     /* mejor respuesta hasta el momento */
3.  while Q no está vacía do
4.      (b, t, d_min) ← objeto en Q con menor t,  Q ← Q − {(b, t, d_min)}
5.      if t > r then return A    /* criterio de detención global */
6.      A ← A ∪ {(b, d(q,b))}
7.      if |A| = k + 1 then
8.          (c, d_max) ← objeto en A con mayor d_max,  A ← A − {(c, d_max)}
9.      if |A| = k then
10.         (c, d_max) ← objeto en A con mayor d_max,  r ← d_max
11.     d_min ← min{d_min} ∪ {d(c,q), c ∈ N(b)}
12.     for v ∈ N(b) do
13.         Q ← Q ∪ (v, max(t, d_min/2, d(q,v) − cr(v)), d_min)
14. return A
```

Figura 3.8: Algoritmo de búsqueda de k vecinos más cercanos en SAT.

3.2.3 List of Clusters

El *List of Clusters* [13] es una lista de "zonas" o "particiones compactas" que contienen una cierta cantidad de objetos. Cada zona posee un centro y un radio cobertor (máxima distancia entre el centro y algún objeto perteneciente a la zona).

Construcción de la lista

Se elige un centro $c \in \mathbb{U}$ aleatoriamente y un radio rp, cuyo valor se discutirá posteriormente. Se define la *bola* (c, rp) como el subconjunto de objetos de $\mathbb{X}$ que se encuentran a lo más a distancia rp de c. A partir de lo anterior se define $I_{\mathbb{U},c,rp} = \{u \in \mathbb{U}, d(c,u) \leq rp\}$ como la partición compacta de objetos "internos", es decir, los que se encuentran dentro de la bola (c, rp), y $E_{\mathbb{U},c,rp} = \{u \in \mathbb{U}, d(c,u) > rp\}$ como el conjunto de los objetos restantes (los objetos "externos"). El proceso se repite recursivamente dentro de E. El procedimiento de construcción retorna una lista de tuplas

(c_i, rp_i, I_i) (centro, radio, elementos de la zona), y su seudocódigo se muestra en la Figura 3.9. El operador ":" enlaza dos listas. Es fácil eliminar la recursividad del algoritmo para hacerlo iterativo.

```
ConstruirLista (Conjunto de objetos 𝕌)

1.  if 𝕌 = ∅ then return lista vacia
2.  Elegir c ∈ 𝕌
3.  Elegir radio rp
4.  I ← {u ∈ 𝕌, d(c,u) ≤ rp}
5.  E ← 𝕌 − I
6.  return (c,rp,I):ConstruirLista(E)
```

Figura 3.9: Algoritmo de construcción del List of Clusters.

Esta estructura de datos puede parecer simétrica, pero no lo es. El primer centro escogido tiene preferencia sobre los siguientes centros en caso que las zonas se traslapen, lo cual se ilustra en la Figura 3.10. Todos los objetos que estén dentro de la zona del primer centro (c_1 en la figura) se almacenan en su partición compacta I respectiva, aunque puedan también estar dentro de las zonas de los centros siguientes (c_2 y c_3 en la figura). La figura también muestra cómo la estructura de datos puede ser vista como una lista.

Con respecto a la elección del radio de cada partición compacta y de los centros, existen varias alternativas. En [13] se muestra experimentalmente que los mejores resultados se obtienen:

- Escogiendo particiones compactas con una cantidad fija de objetos, con lo que el radio cobertor corresponde a la máxima distancia entre el centro y algún objeto perteneciente a la partición compacta, esto es, $rp_c = cr(c)$ para todo centro c.
- Escogiendo el siguiente centro como aquel objeto que maximice la suma de distancias a los centros previamente escogidos.

Algoritmo de búsqueda en List of Clusters

Consulta por rango $(q, r)_d$. Para una consulta $(q, r)_d$, se mide la distancia entre q y el primer centro, $d(c, q)$, añadiendo al resultado el centro si se encuentra a distancia menor o igual que r de

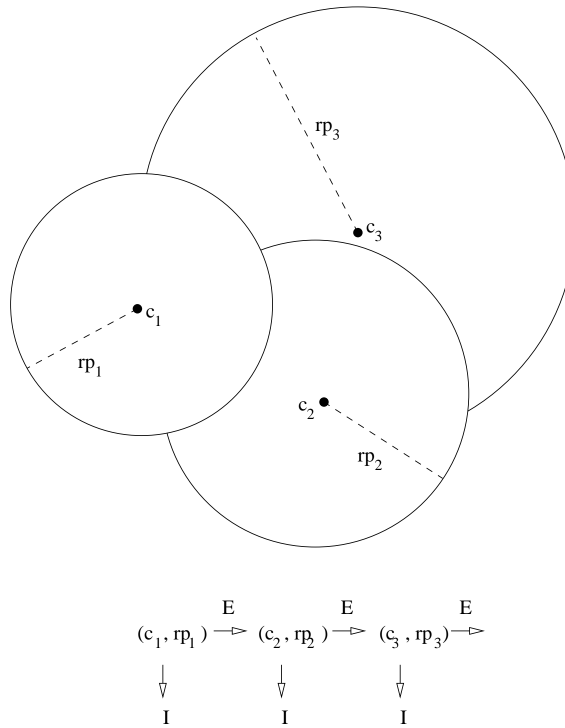

Figura 3.10: Ejemplo de List of Clusters.

q. Luego, se busca exhaustivamente en la partición compacta I si y sólo si la bola de consulta tiene alguna intersección con la zona de centro c. Por último, dada la asimetría de la estructura, se puede podar la búsqueda en el otro sentido: si la bola de consulta está totalmente contenida en la zona de centro c, entonces no es necesario continuar la búsqueda en E, puesto que por construcción se sabe que no puede haber intersección entre la bola de consulta y otras particiones compactas. La Figura 3.11 describe en seudocódigo el algoritmo de consulta por rango utilizando el List of Clusters.

Consulta por k vecinos más cercanos. En este caso se utiliza un esquema similar al descrito en 3.2.2 para consultas k-NN. Se tiene una cola de prioridad, A, donde se almacenan los k objetos candidatos más cercanos al objeto de consulta q conocidos hasta el momento, siendo r la distancia entre q y el candidato más lejano a q en A ($r = \infty$ si se tienen menos de k candidatos). Cada vez que se encuentra un nuevo candidato se inserta en A, pudiendo desplazar a alguno previo y por ende reducir r. Al final, la cola A contiene los k objetos más cercanos a q.

En otra cola de prioridad, Q, se almacenan las particiones compactas y objetos individuales aún

```
BusquedaRangoLista (Lista L, Consulta q, Radio r)

1.  if L = ∅ then return /* lista vacía */
2.  Sea L ← (c,rp,I) : E
3.  Calcular d(c,q)
4.  if d(c,q) ≤ r then añadir c al resultado
5.  if d(c,q) ≤ rp + r then buscar en I − {c} exhaustivamente
6.  if d(c,q) > rp − r then BusquedaRangoLista(E,q,r)
```

Figura 3.11: Algoritmo de búsqueda en List of Clusters.

no procesados, los más prometedores primero. Para medir cuán prometedor es un objeto o una partición compacta, se utiliza una cota inferior de la distancia entre q y dicho elemento (objeto o partición compacta). Al principio, se inserta en la cola la primera partición compacta con una cota $t = 0$. Luego, se extrae un elemento de Q, junto con su cota t, y se procesa. Si la cota del elemento extraído es mayor que r, entonces se detiene el proceso, puesto que no pueden haber objetos relevantes dentro de las particiones compactas y objetos aún no revisados.

Si el elemento extraído es una partición compacta, se compara su centro c directamente con q y se inserta en A. Luego, se calcula una cota inferior para la distancia entre q y los objetos pertenecientes a dicha zona, $d(q,c) - cr(c)$. Finalmente, se insertan en Q:

1. Los objetos de la zona con el máximo valor entre la cota del padre y la nueva cota calculada.
2. La partición compacta que sigue a c en la lista, b, con cota $\max(t, cr(c) - d(q,c))$.

Si el elemento extraído es un objeto, entonces se compara directamente con q y se inserta en A.

Cada vez que se inserta un objeto en A, éste puede desplazar a algún candidato previo, en cuyo caso el valor de r debe ser actualizado. La Figura 3.12 muestra el seudocódigo del algoritmo de consulta por k vecinos más cercanos utilizando el List of Clusters.

```
BusquedaVecinosLista (Lista L, Consulta q, Cantidad de vecinos k)

1.  Q ← {(L,0)}     /* elementos prometedores */
2.  r ← ∞,  A ← ∅     /* mejor respuesta hasta el momento */
3.  while Q no está vacía do
4.    (b,t) ← elemento en Q con menor t,  Q ← Q − {(b,t)}
5.    if t > r then return A    /* criterio de detención global */
6.    if b es una lista then
7.      Sea b = (c, rp_c, I_c) : E_c
8.      A ← A ∪ {(c, d(q,c))}
9.      if |A| = k + 1 then
10.       (e, d_max) ← objeto en A con mayor d_max,  A ← A − {(e, d_max)}
11.     if |A| = k then
12.       (e, d_max) ← objeto en A con mayor d_max,  r ← d_max
13.     for each v ∈ I_c − {c} do Q ← Q ∪ (v, max(t, d(q,c) − rp_c))
14.     if E_c ≠ ∅ then Q ← Q ∪ (E_c, max(t, rp_c − d(q,c)))
15.   else /* b es un objeto */
16.     A ← A ∪ {(b, d(q,b))}
17.     if |A| = k + 1 then
18.       (e, d_max) ← objeto en A con mayor d_max,  A ← A − {(e, d_max)}
19.     if |A| = k then
20.       (e, d_max) ← objeto en A con mayor d_max,  r ← d_max
21.   endif
22. enddo
23. return A
```

Figura 3.12: Algoritmo de búsqueda de k vecinos más cercanos en List of Clusters.

3.2.4 Otros algoritmos basados en particiones compactas

Bisector Trees

El *Bisector Tree* o *BST* [21] es un árbol binario que se construye recursivamente de la siguiente forma: para cada nodo del árbol se eligen dos centros, c_1 y c_2. Los objetos más cercanos a c_1 que a c_2 se insertan en el subárbol izquierdo, y los objetos restantes se insertan en el subárbol derecho. Para ambos centros se almacena la información de su radio cobertor respectivo, $cr(c_i)$. Al realizar una búsqueda, se pueden excluir todos aquellos subárboles donde se cumpla que $d(q, c_i) - cr(c_i) > r$ (criterio de radio cobertor). Note que es posible entrar en ambos subárboles a la vez al realizar una búsqueda.

Monotonous BST

El *Monotonous BST* [25] es un BST donde uno de los centros del nodo es el centro correspondiente al nodo padre. Esto permite que el radio cobertor disminuya a medida que se baja en el árbol.

Voronoi Tree

El *Voronoi Tree* o *VT* [18] es una mejora del BST, donde el árbol puede almacenar 2 ó 3 objetos por nodo. Cuando se crea un nuevo nodo al insertar un objeto, el objeto más cercano del nodo padre también es insertado en el nuevo nodo. El VT tiene la propiedad que el radio cobertor se reduce a medida que se baja en el árbol, y posee un mejor balance que el BST.

Generalized-Hyperplane Tree

El *Generalized-Hyperplane Tree* o *GHT* [28] es idéntico en construcción a un BST. Sin embargo, el algoritmo utiliza la condición de exclusión con criterio de partición de Voronoi para descartar ramas del árbol al momento de realizar una búsqueda, esto es, se puede descartar el subárbol izquierdo si se

cumple que $d(q,c_1) \geq d(q,c_2)+2r$ y se puede descartar el subárbol derecho si $d(q,c_2) > d(q,c_1)+2r$. Note que también es posible entrar en ambos subárboles a la vez al realizar una búsqueda.

Geometric Near-neighbor Access Tree

El *Geometric Near-neighbor Access Tree* o *GNAT* [7] es un GHT extendido a un árbol m-ario, manteniendo la misma idea original. Para el primer nivel se eligen m centros $c_1, \ldots, c_m$, y se define $\mathbb{U}_i = \{u \in \mathbb{U}, d(c_i,u) < d(c_j,u)\ \forall i \neq j\}$, esto es, $\mathbb{U}_i$ es el conjunto de objetos que están más cercanos a c_i que a cualquier otro centro, y se construye recursivamente un GNAT en cada uno de estos conjuntos. Durante la construcción del índice, se almacena para cada nodo la tabla $rango_{ij} = [\min_{u\in\mathbb{U}_j}(c_i,u), \max_{u\in\mathbb{U}_j}(c_i,u)]$, que contiene las distancias mínimas y máximas desde cada centro c_i a algún objeto de cada $\mathbb{U}_j$.

Para una consulta $(q,r)_d$, se compara q con algún centro c_i y se descartan todas aquellas zonas con centro c_j $(i \neq j)$ tal que no haya intersección entre el rango $[d(q,c_i)-r, d(q,c_i)+r]$ y $rango_{ij}$. El proceso se repite eligiendo otros centros aleatoriamente hasta que ya no se puedan descartar más zonas. Recursivamente, se continúa la búsqueda en todas aquellas zonas no descartadas. Durante este proceso se añaden al resultado todos los centros que estén suficientemente cerca de q.

M-tree

El *M-tree* [16] es una estructura de datos dinámica que provee un buen rendimiento en memoria secundaria. La estructura posee cierta similitud con el GNAT, dado que es un árbol donde se elige un conjunto de centros y los objetos restantes se almacenan en la zona de su centro más cercano, construyendo recursivamente un M-tree en cada una de las zonas. Sin embargo, el algoritmo de búsqueda es más cercano al de un BST: cada centro almacena su radio cobertor, y al realizar una búsqueda utiliza dicha información para descartar zonas completas.

Un nuevo objeto e se inserta en el “mejor” subárbol, definido como aquel en donde crece menos el radio cobertor, y en el caso de un empate se escoge el centro más cercano a e. Se procede recursivamente hasta llegar a una hoja, en donde se inserta el nuevo objeto. Si el nodo rebalsa, es decir, si el nodo queda con $m+1$ objetos, se divide en dos y un objeto del nodo se sube un nivel en

el árbol, al estilo de una inserción en un B-tree. Esto implica que el M-tree es un árbol balanceado.

3.3 Algoritmos probabilísticos de búsqueda en proximidad

Todos los algoritmos revisados hasta el momento son *algoritmos exactos*, que en el caso de una consulta por rango recuperan exactamente los objetos pertenecientes a $\mathbb{U}$ que caen dentro de la bola de consulta $(q,r)_d$. Este trabajo esta enfocado a investigar *algoritmos probabilísticos*, en los cuales se relaja la condición de entregar la solución exacta. Los algoritmos probabilísticos son aceptables en la mayoría de las aplicaciones que requieren buscar objetos en espacios métricos, puesto que en general el modelamiento de la base de datos como un espacio métrico conlleva algún tipo de relajación. En muchos casos, obtener casi todos los objetos cercanos es suficientemente satisfactorio, sobre todo si el costo de búsqueda se reduce drásticamente.

A continuación se presentan dos algoritmos probabilísticos de búsqueda existentes, uno diseñado para responder consultas de vecinos más cercanos y otro basado en el uso de pivotes.

3.3.1 Algoritmo probabilístico para búsqueda del vecino más cercano

En [17] se propone una estructura de datos denominada $M(\mathbb{U},Q)$ para responder consultas del vecino más cercano. Esta estructura requiere de un conjunto de datos de entrenamiento Q, con $|Q| = m$. Los objetos pertenecientes a Q deben ser representativos de objetos típicos de consulta.

La estructura $M(\mathbb{U},Q)$ puede fallar en retornar la respuesta correcta a la consulta. Sin embargo, la probabilidad de falla se puede hacer arbitrariamente pequeña a costa de incrementar el tiempo necesario para responder una consulta y el espacio requerido para almacenar el índice.

Construcción de $M(\mathbb{U},Q)$

Sea un espacio métrico $(\mathbb{U},d)$ y un conjunto de objetos $R \subset \mathbb{U}$. Se define la *bola de vecinos γ-cercanos* de q con respecto a R como:

$$B_\gamma(q, R) = \{x \in R, d(q, x) \leq \gamma d(q, y)\ \forall y \in R - \{q\}\}$$

Además, se define la *distancia del vecino más cercano* de q con respecto a R como:

$$d(q, R) \equiv \min_{p \in R - \{q\}} \{d(q, p)\}$$

Un punto $p \in \mathbb{U}$ es un γ-vecino más cercano de $q \in \mathbb{U}$ con respecto a R si $d(q, p) \leq \gamma d(q, R)$, esto es, p es γ-vecino más cercano de q si y sólo si $p \in B_\gamma(q, R)$. Esta propiedad también se denotará como $q \xrightarrow{\gamma} p$ y como $p \xleftarrow{\gamma} q$.

La estructura de datos $M(\mathbb{U}, Q)$ consiste en una lista de objetos A_j por cada objeto $p_j \in \mathbb{U}$. La estructura de datos posee un parámetro real γ y un parámetro $K \in \mathbb{N}, K > 1$, elegidos previamente a la construcción.

Sea $(p_1, p_2, \ldots, p_n)$ una permutación aleatoria de $\mathbb{U}$, y sea $R_i = \{p_1, p_2, \ldots, p_i\}$, con lo que R_i es un subconjunto aleatorio de $\mathbb{U}$. De la misma forma, sea Q_j un subconjunto aleatorio de Q de tamaño j, para $j = 1, \ldots, m = |Q|$, y considere $Q_j = Q$ para todo $j \geq m$.

La lista de objetos A_j se define como:

$$A_j = \{p_i, i > j, \exists q \in Q_{Ki} \text{ con } p_j \xleftarrow{1} q \xrightarrow{\gamma} p_i \text{ con respecto a } R_{i-1}\}$$

Los objetos en A_j se encuentran en orden creciente del índice i. Considere los subconjuntos R_i, que se construyen en forma incremental tomando p_i y agregándolo a $R_i - 1$. Cada lista A_j comienza estando vacía. Cuando se añade el objeto p_i al subconjunto R_{i-1}, se anexa a A_j si es que existe algún $q \in Q_{Ki}$ con p_j más cercano a q en R_{i-1} y si p_i es un γ-vecino más cercano de q.

Este método de construcción calcula una aproximación del conjunto de objetos "vecinos" a p_i. Si se considera el caso particular donde $\gamma = 1$, $\mathbb{U} \subset \mathbb{R}^k$, $Q = \mathbb{R}^k$, $K \longrightarrow \infty$ y las distribuciones de

$\mathbb{U}$ y Q son uniformes, entonces el método de construcción agrega p_i en A_j justo cuando p_i se sale de la región de Voronoi de p_j, atestiguado por algún objeto perteneciente a Q_{Ki}. Para valores de K suficientemente grandes, p_i es el vecino de Delaunay de p_j en R_i.

Algoritmo de búsqueda en $M(\mathbb{U}, Q)$

El procedimiento de búsqueda se asemeja mucho a la aproximación espacial descrita en la Sección 3.2.2. Dado un objeto de consulta $q \in \mathbb{U}$, se comienza con p_1 como el candidato más cercano a q. Se recorre la lista A_1, hasta se encuentre algún objeto p_j que esté más cercano a q que p_1. Ahora p_j es el candidato más cercano a q, y se prosigue la búsqueda recursivamente en A_j hasta que para alguna lista A_k no posea ningún objeto más cercano a q que p_k. Se retorna p_k como el objeto más cercano a q. Este método de búsqueda retorna siempre la respuesta correcta si $Q_{Ki} = Q$ para todo i y $q \in Q$.

Cuando el espacio métrico indexado por $M(\mathbb{U}, Q)$ satisface ciertas condiciones de "sphere-packing bound" [17], se puede demostrar que esta estructura de datos responde consultas del vecino más cercano en tiempo $O(K \log(n)) \log \Upsilon(\mathbb{U} \cup Q)$, con probabilidad de falla $O(\log^2(n)/K)$ y requiere espacio $O(Kn \log(\Upsilon(\mathbb{U} \cup Q)))$, donde K es el parámetro que permite controlar la probabilidad de fallo y $\Upsilon(T)$ es la razón entre las distancias del par de objetos más lejano y el par de objetos más cercano pertenecientes a T.

3.3.2 Algoritmo probabilístico basado en pivotes

En [14] se presenta un algoritmo probabilístico basado en "ensanchar" la desigualdad triangular. La idea es general, pero en [14] la aplican a un algoritmo de búsqueda basado en pivotes para realizar consultas por rango $(q, r)_d$.

La Figura 3.13 muestra la maldición de la dimensionalidad para los algoritmos basados en pivotes. Por cada pivote p_i, $1 \leq i \leq k$, se calcula $d(q, p_i)$ y se descartan todos aquellos objetos que caigan fuera del rango $d(q, p_i) \pm r$, que en la figura son aquellos objetos que caen fuera de la zona gris. Cuando el histograma de distancias es muy concentrado, el caso de los espacios métricos de dimensión intrínseca alta, son pocos los objetos que pueden ser descartados.

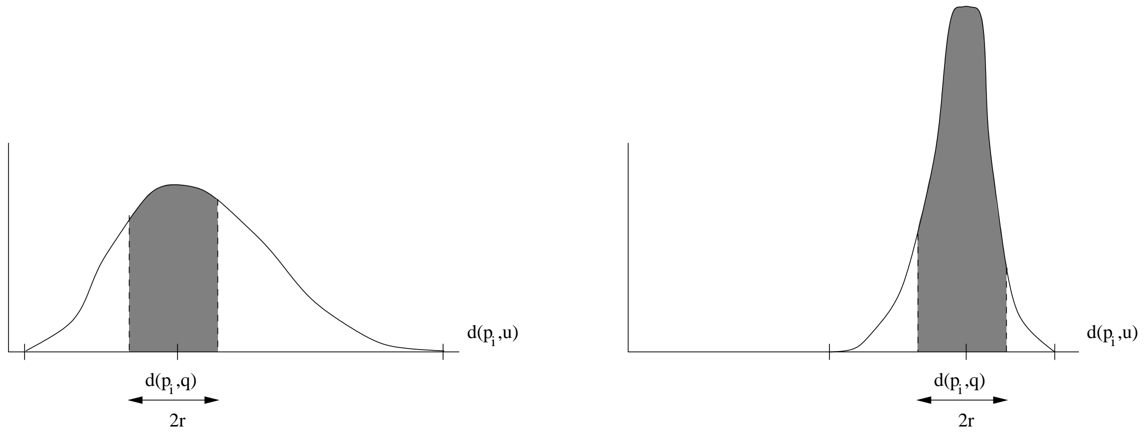

Figura 3.13: Histograma de distancias para un espacio métrico de dimensión baja (izquierda) y alta (derecha).

La diferencia entre dos distancias aleatorias $d(u, p_i)$ y $d(q, p_i)$ puede ser vista como una variable aleatoria con una distribución similar a la del histograma, con media 0 y varianza igual al doble que la varianza del histograma. Esto implica que, a medida que la dimensión del espacio $\mu^2/2\sigma^2$ crece, la diferencia $d(u, p_i) - d(q, p_i)$ tiene un valor absoluto pequeño.

La Figura 3.14 muestra esta situación. La variable aleatoria X representa una distancia aleatoria, y la variable aleatoria Z representa el valor absoluto de la diferencia entre dos distancias aleatorias. Si se desea recuperar una fracción f del conjunto de objetos, se debe utilizar un radio de búsqueda lo suficientemente grande para que el área bajo $f_x(0 \ldots r)$ sea igual a f.

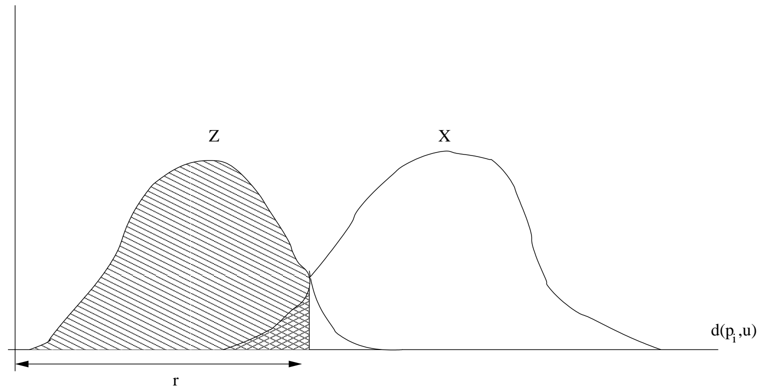

Figura 3.14: Funciones de densidad de las variables aleatorias X y Z.

Si Z representa $|d(u, p_i) - d(q, p_i)|$ para algún objeto u, la probabilidad de descartar u es pro-

porcional al área bajo $f_z(r \ldots \infty)$. Para poder recuperar el area doblemente achurada en la Figura 3.14 es necesario recorrer la fracción achurada del conjunto de objetos.

En espacios métricos de dimensión intrínseca alta f_x se desplaza hacia la derecha (media proporcional a μ), mientras que f_z se concentra alrededor del origen (media proporcional a σ). Por lo tanto, para poder recuperar la misma fracción f de objetos es necesario recorrer una mayor fracción del conjunto. Si ambos histogramas no se intersectan, para recuperar *cualquier fracción* del conjunto de objetos es necesario visitarlos todos.

Para evitar la maldición de la dimensionalidad es necesario correr f_z hacia la derecha. Una forma de lograr esto es agregando más pivotes, porque la distribución real será el máximo valor entre k variables con distribución como la de Z, pero el número máximo de pivotes está limitado por la cantidad de memoria disponible para el índice.

En [14] se propone correr f_z hacia la derecha multiplicando Z por un factor $\beta \geq 1$, lo cual implica *relajar* la condición de exclusión, de modo que ahora es posible descartar más objetos que con la condición original, incluso algunos que puedan estar dentro de la bola de consulta. La nueva condición de exclusión del algoritmo basado en pivotes es:

$$\beta \, |d(p_i, u) - d(p_i, q)| > r \text{ para algún } p_i \tag{3.7}$$

La condición 3.7 es equivalente a reducir el radio de búsqueda a r/β para descartar objetos, pero a la hora de verificar la lista de objetos candidatos se utiliza el radio r original.

La Figura 3.15 ilustra cómo opera la idea de este algoritmo probabilístico. El algoritmo exacto garantiza que ningún objeto relevante es descartado, mientras que el algoritmo probabilístico estrecha ambos lados del anillo y puede perder algunos objetos relevantes.

El factor β puede ser elegido al momento de realizar la búsqueda, por lo que el índice puede ser construido de antemano y luego se puede escoger el nivel deseado de exactitud y velocidad de la búsqueda. Como el factor β es usado solamente para descartar objetos, ningún objeto más cercano a q que r/β puede ser descartado durante la búsqueda.

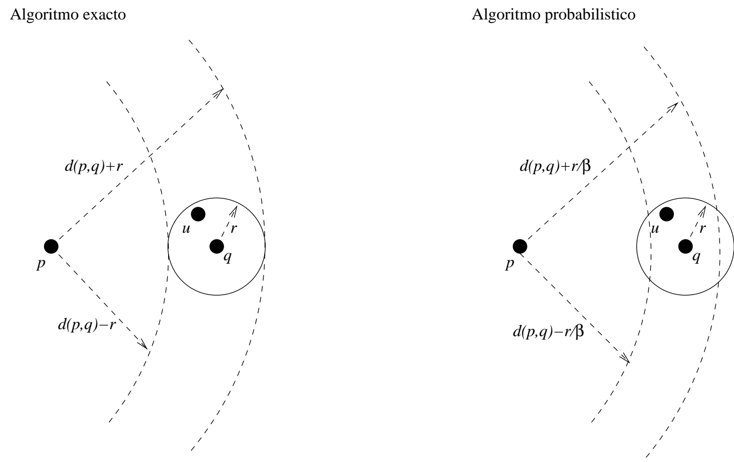

Figura 3.15: Cómo funciona el algoritmo probabilístico basado en pivotes.

La complejidad externa de este algoritmo no es monótonamente decreciente con k, puesto que a medida que k crece la probabilidad de perder un objeto relevante aumenta [14]. Esto implica que, al igual que en el caso del algoritmo exacto, existe un número óptimo de pivotes para el cual el costo de búsqueda es mínimo, pero este óptimo se alcanza antes que en el algoritmo exacto.

Es interesante notar que, a medida que la dimensión aumenta, se puede incrementar β sin perder más objetos, puesto que se hace más improbable descartar objetos con la desigualdad triangular. Aun así la eficiencia empeora, pero no tan rápidamente como en el algoritmo exacto.

En la práctica el método funciona bastante bien. La figura 3.16 muestra los resultados de la utilización del algoritmo probabilístico basado en pivotes en espacios vectoriales aleatorios, para distintas dimensiones, utilizando 16 pivotes. Los resultados muestran que en dimensiones bajas se puede recuperar mas del 90% de los objetos relevantes utilizando una pequeña fracción de los cálculos de distancia que realiza el algoritmo exacto, pero que la situación se torna cada vez más difícil a medida que la dimensión del espacio aumenta.

3.3.3 Otros trabajos relacionados para espacios vectoriales

En [30] se estudian en profundidad algoritmos aproximados de búsqueda en proximidad. Un ejemplo es [1], donde se propone un marco general para realizar búsquedas dentro de una región arbitraria

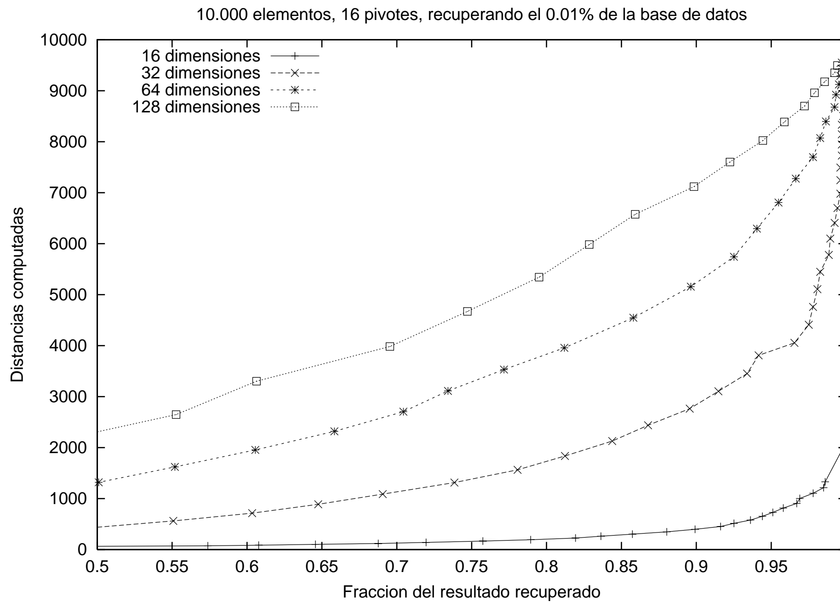

Figura 3.16: Rendimiento del algoritmo probabilístico basado en pivotes.

Q perteneciente a un espacio vectorial $(\mathbb{R}^k, L_2)$. La idea es definir áreas Q^- y Q^+ tal que $Q^- \subseteq Q \subseteq Q^+$. Está garantizado que los puntos dentro de Q^- serán retornados y que los puntos fuera de Q^+ no lo serán. En el área entre medio el algoritmo no garantiza nada. La máxima distancia entre el área real y la acotada es ϵ. Se utilizan árboles para particionar el espacio vectorial y guiar la búsqueda, incluyendo o excluyendo áreas completas del espacio. Cada decisión sobre inclusión/exclusión de áreas es tomada usando Q^+/Q^- para incrementar la probabilidad de podar la búsqueda en ambos sentidos. Aquellas áreas que no pueden ser completamente incluídas o excluídas son analizadas en mayor detalle, siguiendo la búsqueda en el subárbol apropiado. Se muestra en [1] que el tiempo de la búsqueda es $O(2^k \log(n) + (3\sqrt{k}/\epsilon)^k)$.

Se han propuesto algoritmos probabilísticos para búsqueda de vecinos más cercanos en [2, 33, 30]. En [2] se propone una estructura de datos denominada *BBD-tree*, que permite realizar búsquedas en espacios vectoriales $\mathbb{R}^k$ bajo cualquier métrica L_s. Esta estructura está inspirada en el *kd-tree* y puede ser usada para encontrar el $(1+\epsilon)$-vecino más cercano, esto es, un objeto u^* tal que $\forall u \in \mathbb{U}, d(u^*, q) \leq (1+\epsilon)d(u,q)$. La idea del algoritmo es situar el objeto de consulta q en una celda (cada hoja del árbol está asociada con una celda en la descomposición del espacio). Cada punto dentro de la celda se procesa, recordando el punto p más cercano encontrado hasta el momento. La búsqueda se detiene cuando ya se han visitado todas las celdas prometedoras, esto es, cuando cualquier bola centrada en q que intersecte una celda no vacía aún no visitada tenga un radio

superior a $d(q,p)/(1+\epsilon)$. El tiempo de búsqueda para este algoritmo es $O(\lceil 1+6k/\epsilon \rceil^k k \log(n))$.

En [33] se propone una estrategia denominada "poda agresiva" para "vecinos cercanos con radio limitado". Esta consulta busca vecinos más cercanos que se encuentren dentro de un cierto radio. La idea puede ser vista como un caso particular de [1], donde el área de búsqueda es una bola y la estructura de datos es un *kd-tree*. Se pueden perder objetos relevantes, pero nunca se retornan objetos irrelevantes, esto es, $Q^+ = Q$. La bola Q, de radio r y centrada en $q = (q_1, q_2, \ldots, q_k)$, es podada intersectándola con el área definida por los hiperplanos $q_i - r + \epsilon$ y $q_i + r - \epsilon$. En [33] se presenta un análisis probabilístico que supone distancias normalmente distribuidas, lo cual casi siempre se cumple cuando los puntos están uniformemente distribuidos en el espacio. El costo de búsqueda es de tiempo $O(n^\lambda)$, donde λ disminuye a medida que la probabilidad de fallo ϵ permitida aumenta.

Capítulo 4

Algoritmos Probabilísticos Basados en Particiones Compactas

Es bien sabido que los algoritmos de búsqueda basados en particiones compactas tienen mejor rendimiento que los algoritmos de búsqueda basados en pivotes en espacios métricos de dimensión intrínseca alta [15], y que estos últimos necesitan más espacio en memoria para alcanzar el rendimiento de los primeros. Es por esta razón que es interesante desarrollar algoritmos probabilísticos basados en particiones compactas, ya que eventualmente podrían tener al menos el mismo rendimiento que el algoritmo probabilístico basado en pivotes y con requerimientos de espacio menores.

En este capítulo se presentarán dos técnicas probabilísticas generales basadas en particiones compactas: una inspirada en una búsqueda incremental y la otra basada en el *ranking* de zonas.

4.1 Búsqueda incremental probabilística

Esta técnica es una adaptación de la *búsqueda incremental de vecinos más cercanos* [20]. La búsqueda incremental recorre la jerarquía de búsqueda definida por el índice (cualquiera sea éste) visitando primero los elementos más prometedores (*best-first*). En todo paso del algoritmo, la búsqueda incremental visita el *elemento*, que puede ser una *zona* o un *objeto*, que posea la menor

distancia a la consulta entre todos los elementos aún no visitados en la jerarquía de búsqueda. Esto puede realizarse utilizando una cola de prioridad de elementos, organizada por la máxima cota inferior de distancia conocida al objeto de consulta hasta ese momento.

La Figura 4.1 muestra en seudocódigo el algoritmo de búsqueda incremental de vecinos más cercanos. $Indice$ es la estructura de datos que indexa $\mathbb{U}$, q es el objeto de consulta, e es un elemento perteneciente al índice y $d_{LB}(q,e)$ es una cota inferior de la distancia real entre q y todos los elementos cuya raíz en la jerarquía de búsqueda es e. En particular, $d_{LB}(q,e) = d(q,e)$ si e es un objeto perteneciente a $\mathbb{U}$. Además, si e' es un ancestro de e se cumple que $d_{LB}(q,e) \geq d_{LB}(q,e')$. Por ejemplo, en List of Clusters, si e es hijo de a y pertenece a la zona de centro c se tiene que $d_{LB}(q,e) = d(q,c) - cr(c)$; en SAT, si e es hijo de a entonces $d_{LB}(q,e) = \max(d(q,e) - cr(e), (d(q,e) - \min(d(q,c), c \in \{a'\} \cup N(a'), a' \in A(a)))/2)$.

```
BusquedaIncrementalVecinos(q, Indice)

 1.   e ← raíz de Indice
 2.   Q ← {(e,0)} /* Cola de prioridad */
 3.   while Q no está vacía do
 4.     (e, d_LB(q,e)) ← elemento en Q con menor d_LB(q,e)
 5.     Q ← Q − {(e, d_LB(q,e))}
 6.     if e es un objeto then
 7.       Informar e como el siguiente objeto más cercano
 8.     else
 9.       for each elemento e' hijo de e do
10.         Calcular d_LB(q,e') según el tipo de índice usado
11.         Q ← Q ∪ {(e', max(d_LB(q,e), d_LB(q,e')))}
12.       enddo
13.     endif
14.   enddo
```

Figura 4.1: Búsqueda incremental de vecinos más cercanos.

En [20] se demuestra que la búsqueda incremental de vecinos más cercanos es *rango-óptima*, esto es, obtiene el k-ésimo vecino más cercano de q, o_k, después de realizar el mismo recorrido en la jerarquía de búsqueda que una consulta por rango de radio $d(q,o_k)$ implementada con un recorrido

top-down de la jerarquía.

La búsqueda incremental puede ser adaptada para responder consultas por rango $(q, r)_d$. Se reportan todos los objetos $u \in \mathbb{U}$ que satisfagan $d(q, u) \leq r$, y la búsqueda se detiene cuando se saque de la cola un elemento con cota inferior $\ell > r$ (criterio de detención global), ya que no es posible encontrar otro objeto entre los elementos no visitados que se encuentre dentro de la bola de consulta, puesto que fueron recuperados ordenados por cota inferior de distancia a q. Un método equivalente es encolar solamente los elementos cuya cota inferior de distancia sea $\ell \leq r$, en cuyo caso la cola de prioridad debe procesarse hasta que quede vacía.

La idea propuesta en [27] para implementar la búsqueda de k vecinos más cercanos es similar a la búsqueda incremental, salvo que en esta última no se requieren de dos colas de prioridad para almacenar los objetos prometedores y las zonas aún no visitados, sino que toda la información se almacena en una única cola.

La idea de la técnica probabilística basada en la búsqueda incremental es *fijar de antemano el número de evaluaciones de distancia permitidas para responder la consulta por rango*. Utilizando el método incremental adaptado que responde consultas por rango, si la búsqueda es recortada después de utilizar el máximo número de evaluaciones de distancia permitido se obtiene un algoritmo probabilístico, en el sentido que algunos elementos relevantes pueden no haber sido reportados aún. Sin embargo, dado que la búsqueda se realiza en un orden promisiorio dentro del índice se espera que las evaluaciones de distancias permitidas sean usadas en forma eficiente.

La Figura 4.2 muestra en seudocódigo la forma general del algoritmo de búsqueda incremental probabilístico. La variable *cuota* indica el máximo número de evaluaciones de distancia permitido para realizar la búsqueda. Una vez que *cuota* se ha alcanzado, ningún otro elemento es encolado en la cola de prioridad. Note que el único criterio de detención del algoritmo es que la cola se vacíe, incluso si la cuota de trabajo ha sido alcanzada, porque para todos los objetos encolados su distancia a q ya es conocida. La variable *costo* indica el número de evaluaciones de distancia necesarias para calcular la cota de distancia a un hijo e' de e. En SAT, el costo de procesar todos los hijos de e es igual al número de vecinos de éste, $|N(e)|$; en List of Clusters, el costo de procesar todos los hijos de e es igual al tamaño de la zona compacta, m.

Resulta interesante discutir el significado de las líneas 11 y 12 del seudocódigo. Si el costo de

```
BusquedaIncrementalProbabilistica(q, Indice, cuota)

 1.  e ← raíz de Indice
 2.  contador ← 0 /* Número de cálculos de distancia */
 3.  Q ← {(e,0)} /* Cola de prioridad */
 4.  while Q no está vacía do
 5.    (e, d_LB(q,e)) ← elemento en Q con menor d_LB(q,e)
 6.    Q ← Q − {(e, d_LB(q,e))}
 7.    if e es un objeto then
 8.      Añadir e al resultado
 9.    else
10.      for each elemento e' hijo de e do
11.        costo ← calcular d_LB(q,e')
12.        if contador + costo ≤ cuota
13.          Calcular d_LB(q,e')
14.          if d_LB(q,e') ≤ r then Q ← Q ∪ {(e', max(d_LB(q,e), d_LB(q,e')))}
15.          contador ← contador + costo
16.        endif
17.      enddo
18.    endif
19.  enddo
```

Figura 4.2: Búsqueda incremental probabilística.

calcular $d_{LB}(q, e')$ más las distancias ya calculadas durante la búsqueda es mayor que la cuota de trabajo, entonces no se procesa e', con lo cual se asegura que nunca se sobrepasará la cuota de trabajo permitida.

Esta técnica se enmarca dentro de los algoritmos probabilísticos del tipo *Monte Carlo*, puesto que a medida que se aumenta la cantidad de trabajo disminuye la probabilidad de error, esto es, disminuye la probabilidad de perder objetos relevantes a la consulta. Además, este método es *1-sesgado*, puesto que, si bien puede fallar en reportar objetos relevantes, nunca reporta como relevantes objetos que no lo son. Esto se puede observar en la línea 14 del seudocódigo: no se inserta ningún objeto en la cola de prioridad si su distancia a la consulta es mayor que r.

4.2 Ranking de zonas

El propósito de la búsqueda incremental probabilística es encontrar rápidamente objetos que se encuentren dentro de la bola de consulta, antes que la cuota de trabajo se termine. Como el máximo número de cálculos de distancia permitidas es fijo, el tiempo total de búsqueda también está acotado. Esta técnica puede ser generalizada a la denominada *técnica de ranking de zonas*, que consiste en ordenar las particiones compactas de manera de favorecer a las más prometedoras, para luego realizar la búsqueda en ese orden hasta que se acabe la cuota de trabajo. La búsqueda incremental probabilística puede ser vista como un método de *ranking*, donde se ordenan las zonas utilizando $d_{LB}(q, e)$ y se busca en dicho orden hasta ocupar el número de cálculos de distancia permitido. No obstante, este método para ordenar zonas no necesariamente tiene que ser el mejor.

El criterio de ordenamiento debe apuntar a encontrar rápidamente posibles elementos que se encuentren cerca del objeto de consulta. Como el espacio métrico está particionado en zonas, dichas zonas deben ordenarse en un orden promisorio de búsqueda. Para esto se utiliza la información entregada por el índice, esto es, primero deben calcularse las distancias entre la consulta y cada centro de zona (esto hace impráctico utilizar el SAT con ranking de zonas, puesto que en esta estructura de datos todo nodo es un centro de zona, por lo que toma tiempo $O(n)$ calcular las distancias entre la consulta y cada centro). La búsqueda debiera privilegiar tanto a aquellas zonas "cercanas" al objeto de consulta como a aquellas zonas que posean una mayor "densidad" de objetos. A pesar del hecho que es muy difícil definir el volumen de una zona en un espacio métrico general, se puede suponer que si todas las zonas poseen el mismo número de elementos, como es el caso de la mejor implementación de List of Clusters, entonces aquellas zonas que poseen un menor radio cobertor tienen una mayor densidad de objetos que aquellas zonas con radios cobertores mayores.

Algunos criterios específicos para ordenar las zonas son los siguientes (siempre de menor a mayor):

1. La distancia entre q y cada centro de zona, $d(q, c)$ (Figura 4.3).
2. El radio cobertor de cada zona, $cr(c)$ (Figura 4.4).
3. $d(q, c) + cr(c)$, esto es, una cota superior a la distancia de q al elemento más lejano en la zona de centro c (Figura 4.5).

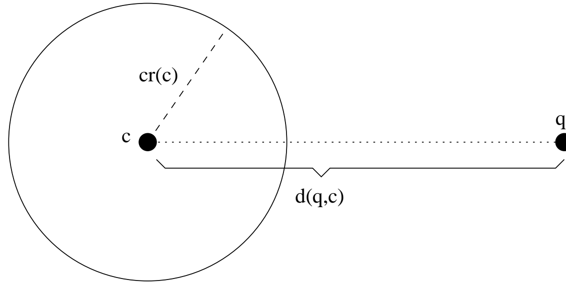

Figura 4.3: Criterio de ordenación de zonas $d(q, c)$.

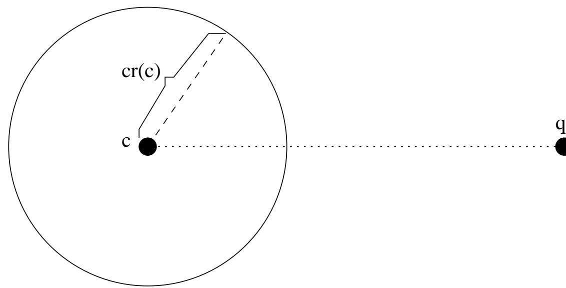

Figura 4.4: Criterio de ordenación de zonas $cr(c)$.

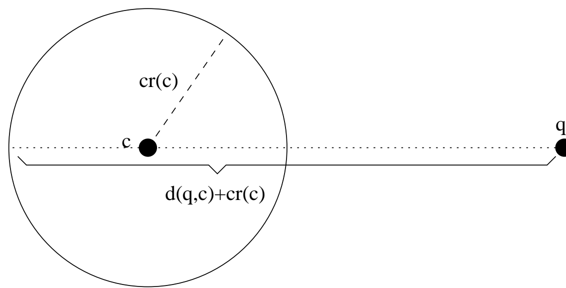

Figura 4.5: Criterio de ordenación de zonas $d(q, c) + cr(c)$.

4. $d(q,c) - cr(c)$, esto es, una cota inferior a la distancia de q al elemento más cercano en la zona de centro c (Figura 4.6).

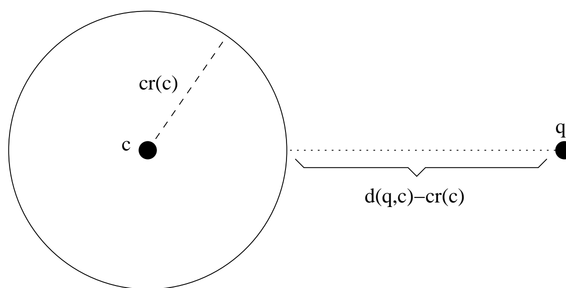

Figura 4.6: Criterio de ordenación de zonas $d(q,c) - cr(c)$.

5. $\beta(d(q,c) - cr(c))$, criterio denominado *beta dinámico*.

Los dos primeros criterios son los más simples que se pueden utilizar. El tercer criterio apunta a buscar primero en las zonas que se encuentren más cercanas a q y que además sean "densas". El cuarto criterio es parecido a la búsqueda incremental probabilística, pero esta última no parte calculando las distancias a los centros de zona y puede podar hacia el otro lado la búsqueda.

El último criterio es equivalente a reducir el radio de búsqueda por un factor β como en [14], donde $\beta \geq 1$. Si el factor β es fijo, entonces el último criterio para ordenar zonas es equivalente a la búsqueda incremental probabilística, puesto que el orden de los elementos en ambos casos es el mismo. Sin embargo, en vez de utilizar un factor β fijo se puede utilizar un *factor dinámico* de la forma $\beta = 1/(1.0 - \frac{cr(c)}{mcr})$, donde mcr es el máximo tamaño de los radios cobertores entre todas las zonas. Al utilizar el β dinámico, se reduce más el radio de búsqueda mientras mayor sea el radio cobertor de una zona en particular. Un caso especial es cuando una zona posee radio cobertor $cr(c) = mcr$, en cuyo caso se define $\beta = \infty$ para todos los objetos e que pertenezcan a la zona de centro c. La definición implica que estos objetos serán los últimos en ser revisados por el algoritmo de búsqueda.

Capítulo 5

Resultados Experimentales

A continuación se presentan los resultados experimentales obtenidos con las técnicas probabilísticas de búsqueda en proximidad descritas en el Capítulo 4. Los índices utilizados para implementar las técnicas fueron el SAT y el List of Clusters. Sin embargo, con SAT sólo se implementó la búsqueda incremental probabilística.

Las técnicas fueron comparadas con el algoritmo probabilístico basado en pivotes, implementado en su forma canónica (ver Sección 3.1 y Sección 3.3.2). Los pivotes fueron escogidos en dos formas: aleatoriamente y escogiendo “buenos” pivotes, utilizando el método de selección incremental descrito en [10], con parámetros $A = 10.000$ y $N = 40$.

Los gráficos presentados en este capítulo muestran el número de cálculos de distancia como función de la fracción de objetos relevantes recuperados. Se dirá que un algoritmo es más eficiente que otro si para recuperar la misma fracción de objetos relevantes necesita realizar un menor número de cálculos de distancia.

5.1 Experimentos en espacios vectoriales

Las técnicas probabilísticas fueron probadas utilizando un conjunto sintético de puntos aleatorios en un espacio vectorial k-dimensional. Este espacio fue tratado como un espacio métrico general, esto es, no se utilizó el hecho que los puntos del espacio poseen coordenadas, sino que los puntos fueron tratados como objetos abstractos de un espacio métrico desconocido, utilizando sólo la función de distancia para implementar las búsquedas. La ventaja de utilizar este esquema es que permite controlar la dimensión exacta del espacio métrico, lo cual es muy difícil de realizar con espacios métricos generales.

Los puntos del espacio vectorial generado están uniformemente distribuidos en el cubo unitario, esto es, las coordenadas de cada vector tienen un valor real, uniforme e independiente de los demás, perteneciente al rango [0..1). La función de distancia utilizada es la distancia Euclidiana (L_2). Se utilizaron dos bases de datos, una con $n = 10.000$ y otra con $n = 100.000$ elementos. Se realizaron consultas por rango que retornan en promedio una fracción equivalente al 0,01%, 0,10% y 1,00% del total de la base de datos, tomando un promedio sobre 1.000 consultas. Los espacios vectoriales utilizados en los experimentos son de dimensión 64 y 128, para los cuales no se conoce ningún algoritmo exacto que pueda evitar la búsqueda exhaustiva al responder consultas por rango útiles. En el caso de List of Clusters, el número de elementos por zona escogido fue de $m = 5$, ya que en espacios de dimensión alta este índice tiene un mejor rendimiento cuando las zonas poseen pocos elementos (esto se comprobó experimentalmente, probando con distintos tamaños de zona). En la elección del siguiente del centro de la lista, se tomó una muestra de 100 objetos entre aquellos aún no asignados a una zona, y se escogió aquel que maximizara la suma de distancias a los centros previamente escogidos.

Las Figuras 5.1, 5.2 y 5.3 muestran una comparación de las técnicas probabilísticas en un espacio de dimensión 64, con 10.000 elementos, variando el radio de búsqueda entre 0,01% y 1,00%. Los resultados muestran que la mejor técnica en este caso es el criterio de *ranking* de zona $d(q,c)$. La búsqueda probabilística incremental con SAT y el criterio de beta dinámico tuvieron un mal rendimiento en este experimento. Note que en el caso de List of Clusters se omitió el criterio $d(q,c) - cr(c)$ puesto que los resultados eran muy parecidos a los de la búsqueda incremental probabilística (esto es válido para todos los gráficos de esta sección).

Las Figuras 5.4, 5.5 y 5.6 muestran la comparación entre la mejor técnica de *ranking* para

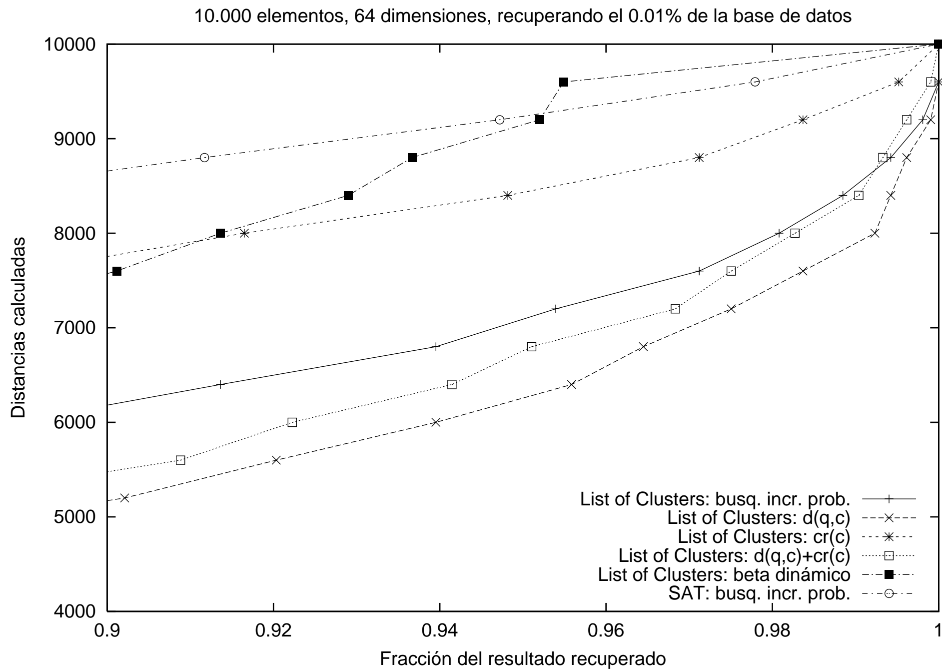

Figura 5.1: SAT y List of Clusters probabilístico, dimensión 64, 0,01% recuperado.

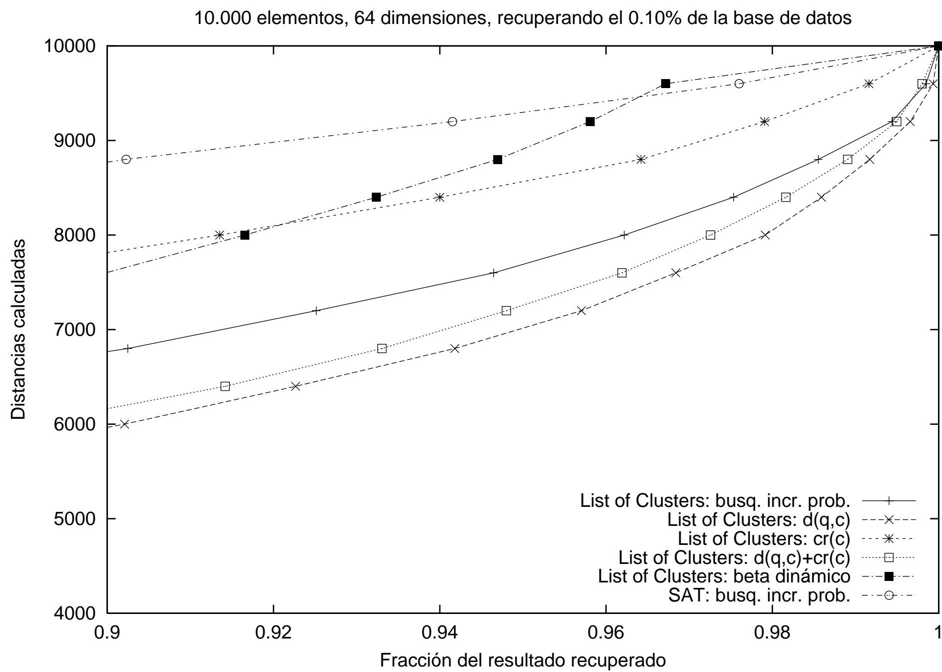

Figura 5.2: SAT y List of Clusters probabilístico, dimensión 64, 0,10% recuperado.

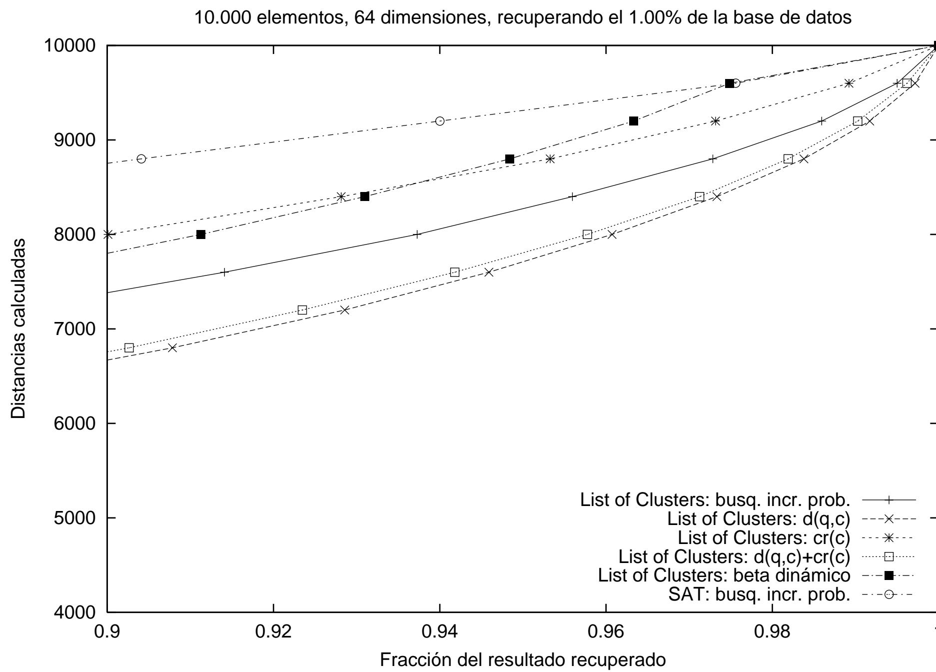

Figura 5.3: SAT y List of Clusters probabilístico, dimensión 64, 1,00% recuperado.

64 dimensiones y el algoritmo probabilístico basado en pivotes, variando el radio de búsqueda y utilizando 16 y 256 pivotes. Los gráficos muestran que la técnica de *ranking* es ligeramente mejor que el índice con 16 pivotes, pero el índice con 256 pivotes tiene un rendimiento superior.

Las Figuras 5.7, 5.8 y 5.9 muestran una comparación de las técnicas probabilísticas en un espacio de dimensión 128, con 10.000 elementos, variando el radio de búsqueda entre 0,01% y 1,00%. Los resultados muestran que la mejor técnica en este caso es el criterio de *ranking* de zonas $d(q,c)+cr(c)$. Nuevamente, la búsqueda probabilística incremental con SAT tiene un rendimiento inferior respecto a las otras técnicas utilizadas.

Las Figuras 5.10, 5.11 y 5.12 muestran la comparación entre el criterio de *ranking* $d(q,c)+cr(c)$ y el algoritmo probabilístico basado en pivotes, utilizando 16 y 256 pivotes, para distintos radios de búsqueda. En este caso, la técnica de *ranking* de zonas tiene un rendimiento intermedio entre los índices con 16 y 256 pivotes, pero cabe destacar que la diferencia de rendimiento por sobre el 98% de la fracción total recuperada es pequeña entre todos los algoritmos presentados. Los gráficos también muestran que, a medida que el radio de la consulta por rango aumenta, los algoritmos basados en pivotes empeoran con mayor rapidez su rendimiento que el método de *ranking*.

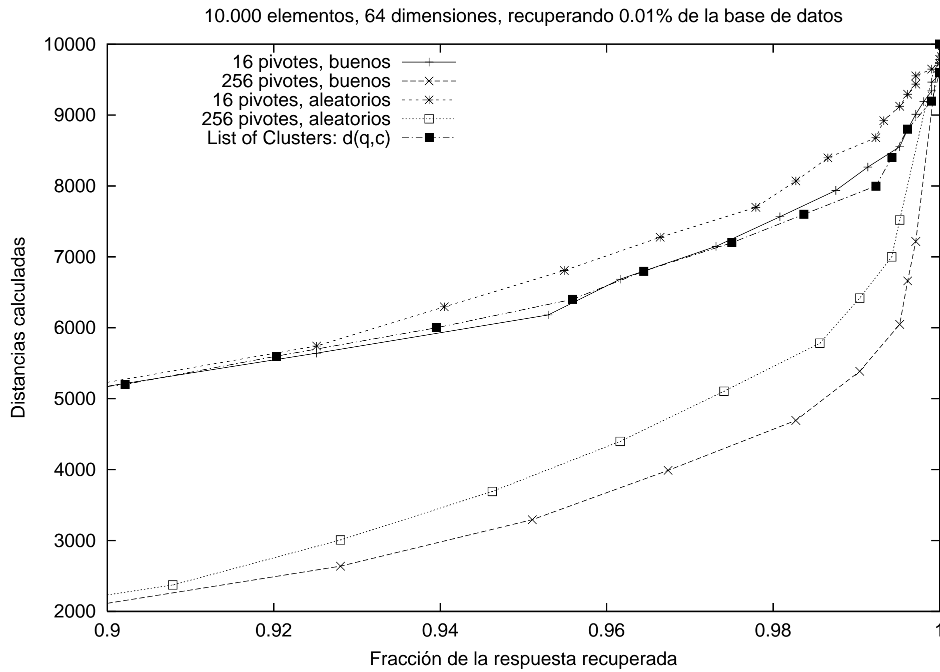

Figura 5.4: Comparación con algoritmo basado en pivotes, dimensión 64, 0,01% recuperado.

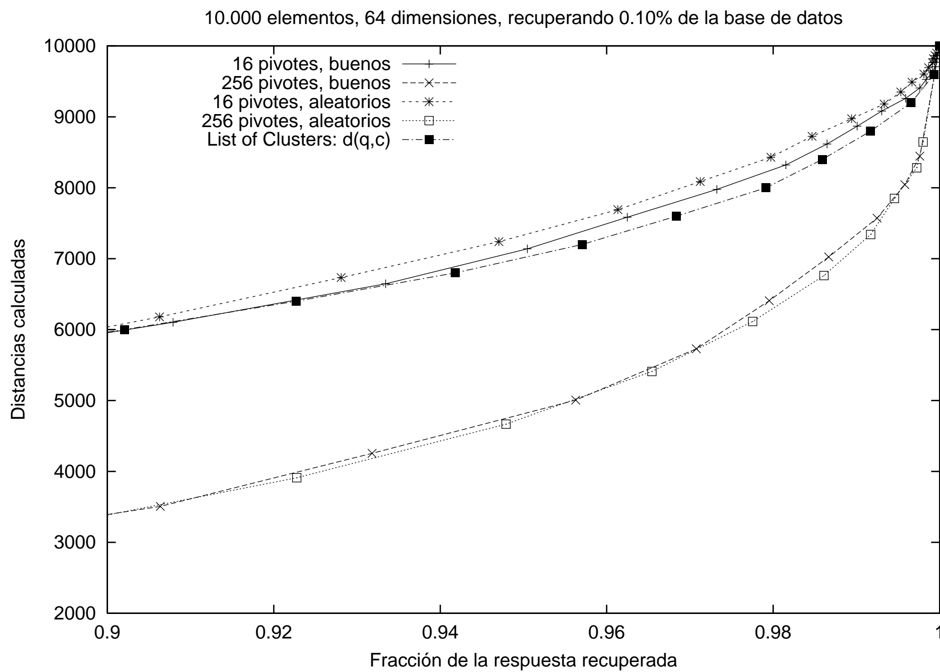

Figura 5.5: Comparación con algoritmo basado en pivotes, dimensión 64, 0,10% recuperado.

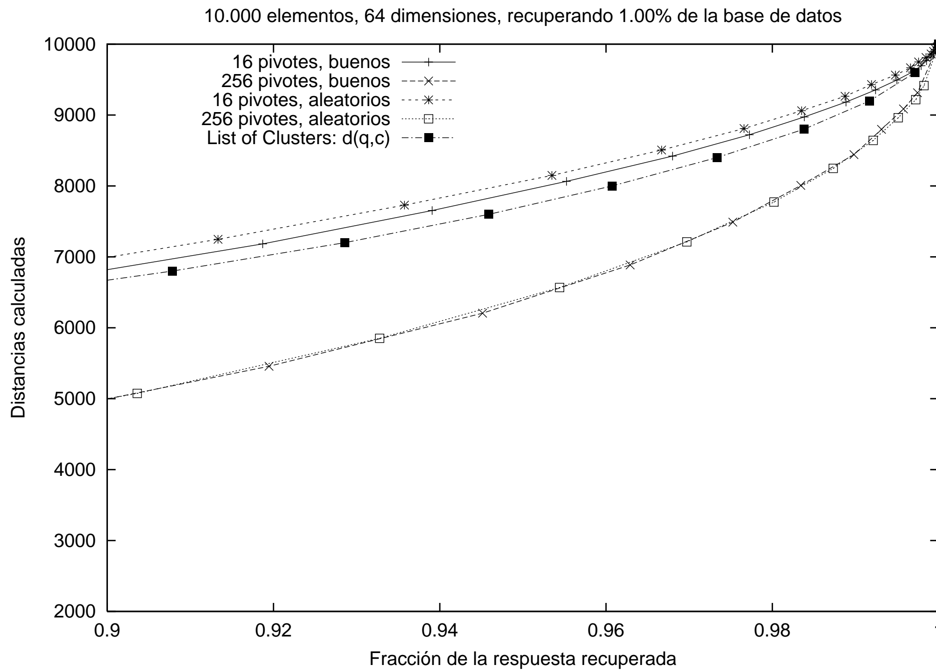

Figura 5.6: Comparación con algoritmo basado en pivotes, dimensión 64, 1,00% recuperado.

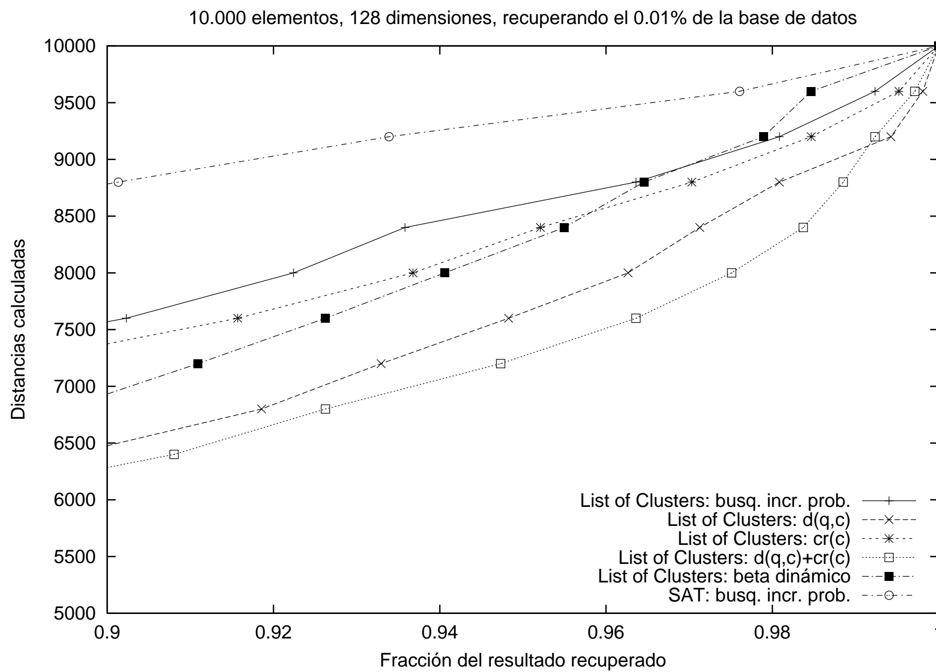

Figura 5.7: SAT y List of Clusters probabilístico, dimensión 128, 0,01% recuperado.

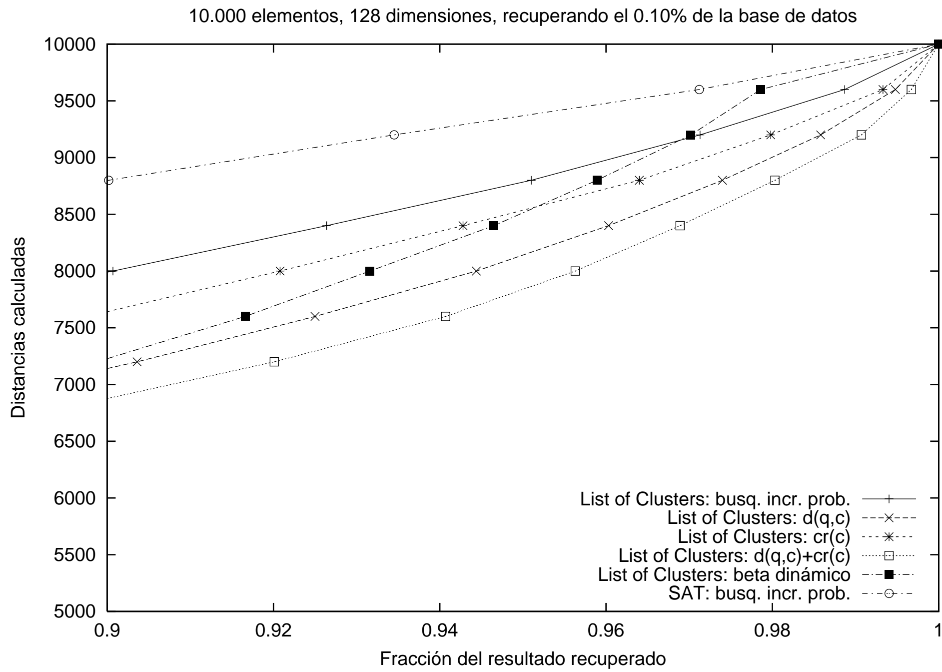

Figura 5.8: SAT y List of Clusters probabilístico, dimensión 128, 0,10% recuperado.

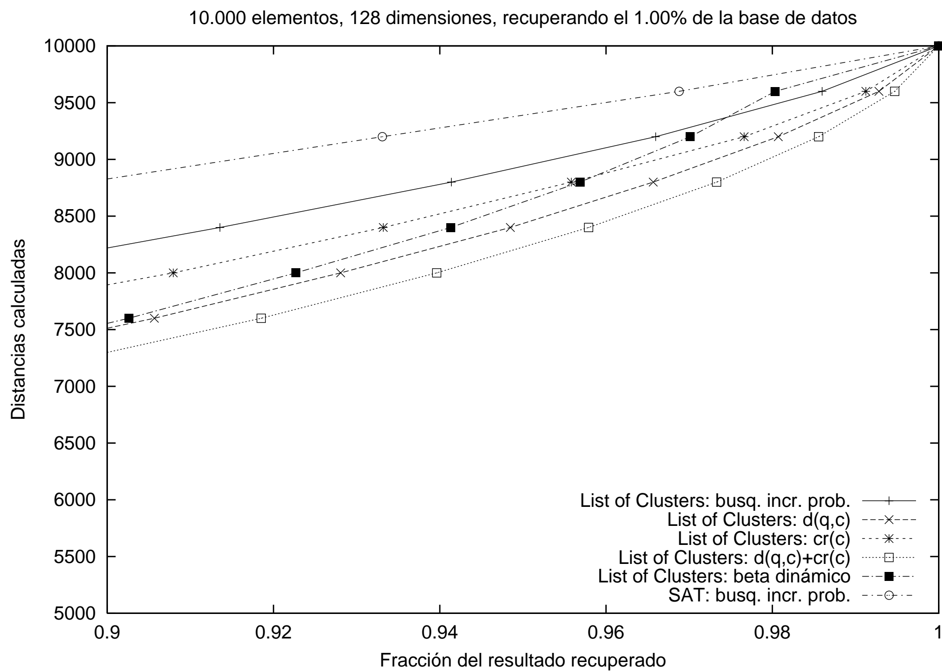

Figura 5.9: SAT y List of Clusters probabilístico, dimensión 128, 1,00% recuperado.

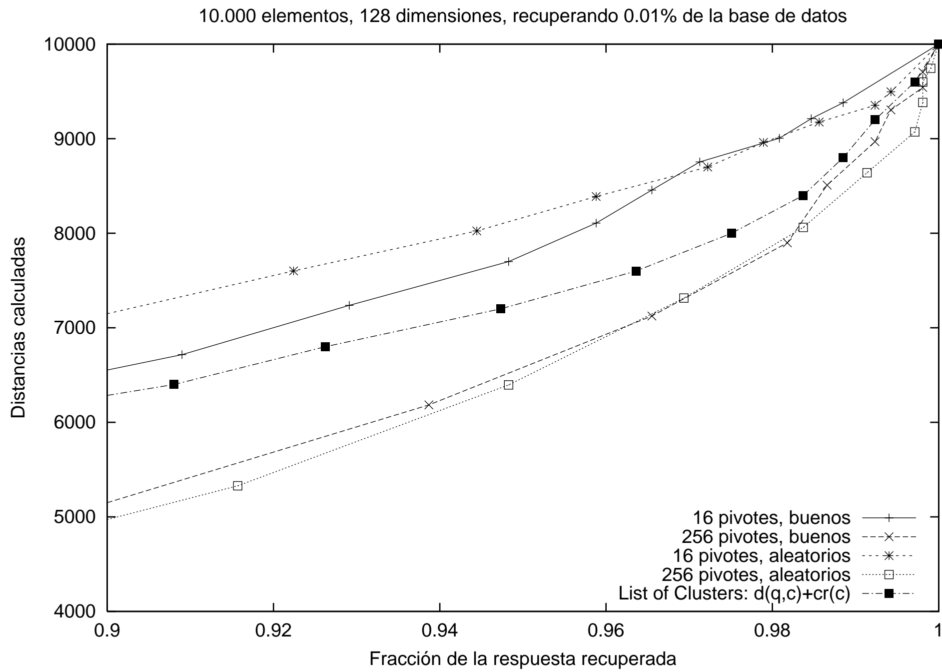

Figura 5.10: Comparación con algoritmo basado en pivotes, dimensión 128, 0,01% recuperado.

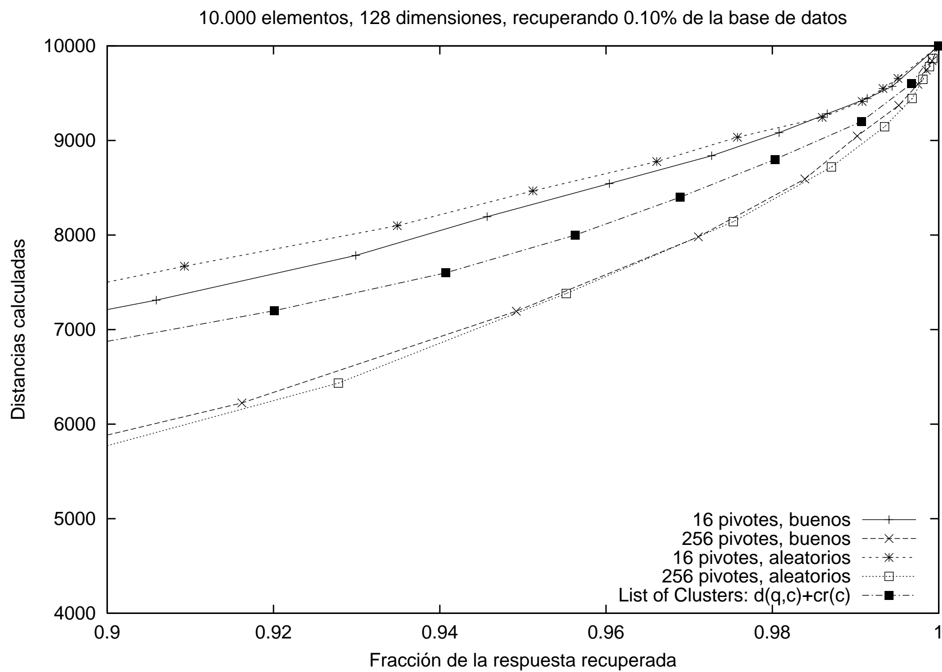

Figura 5.11: Comparación con algoritmo basado en pivotes, dimensión 128, 0,10% recuperado.

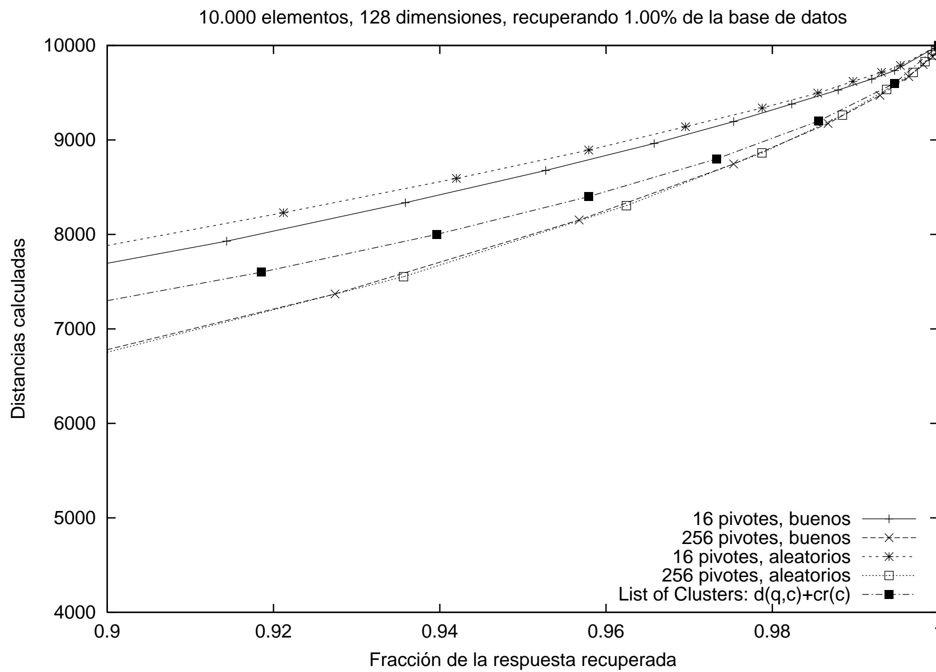

Figura 5.12: Comparación con algoritmo basado en pivotes, dimensión 128, 1,00% recuperado.

Las Figuras 5.13, 5.14 y 5.15 muestran una comparación de las técnicas probabilísticas en un espacio de dimensión 128, pero ahora utilizando una base de datos con 100.000 elementos, variando el radio de búsqueda entre 0,01% y 1,00%. Los mejores criterios de *ranking* son $cr(c)$ y $d(q,c)+cr(c)$ por sobre el 90% de fracción recuperada. La búsqueda incremental probabilística con SAT sigue teniendo un mal rendimiento, pero en estos experimentos en particular se obtienen los peores resultados con los criterios de beta dinámico y $d(q,c)$.

Las Figuras 5.16, 5.17 y 5.18 muestran la comparación entre el criterio de *ranking* $d(q,c)+cr(c)$ y el algoritmo probabilístico basado en pivotes, utilizando 16 y 64 pivotes (la memoria disponible no permitió usar 256 pivotes). Para un radio de búsqueda que retorna el 0,01% de la base de datos, el método de *ranking* funciona mejor que el índice con 64 pivotes al recuperar por sobre el 98% del total de objetos relevantes. Para un radio de búsqueda que retorna el 0,10% y el 1,00% de la base de datos, el índice con 64 pivotes tiene un mejor rendimiento que el método de *ranking* al recuperar entre el 90% y el 98% del total de los objetos relevantes, pero entre el 98% y el 100% los tres algoritmos ocupan casi la misma cantidad de cálculos de distancia en el rango estudiado.

Es interesante notar que los métodos basados en particiones compactas ocupan mucho menos

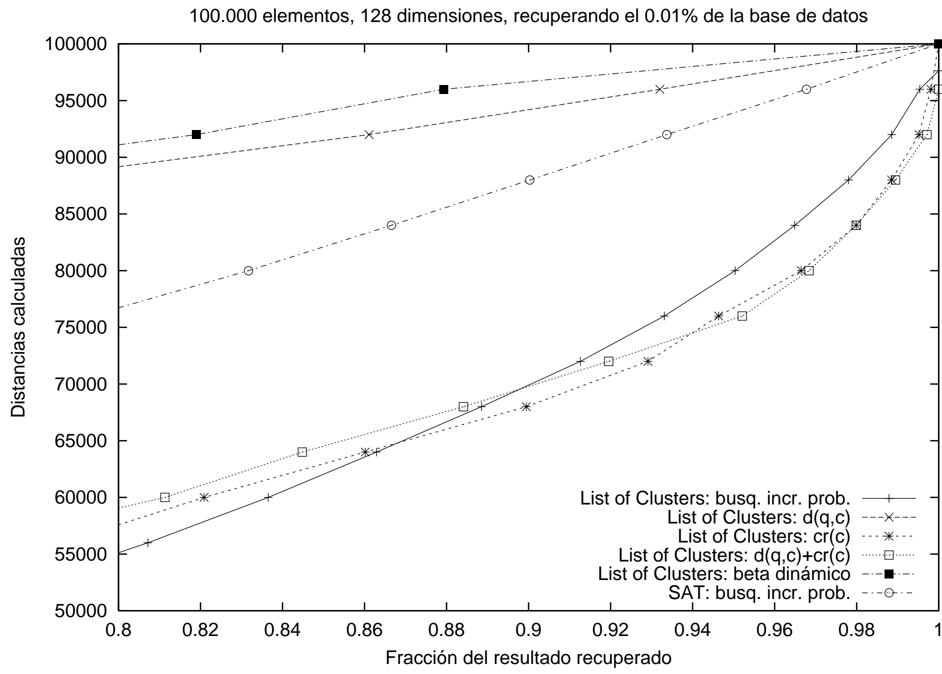

Figura 5.13: SAT y List of Clusters probabilístico, 100.000 elementos, dimensión 128, 0,01% recuperado.

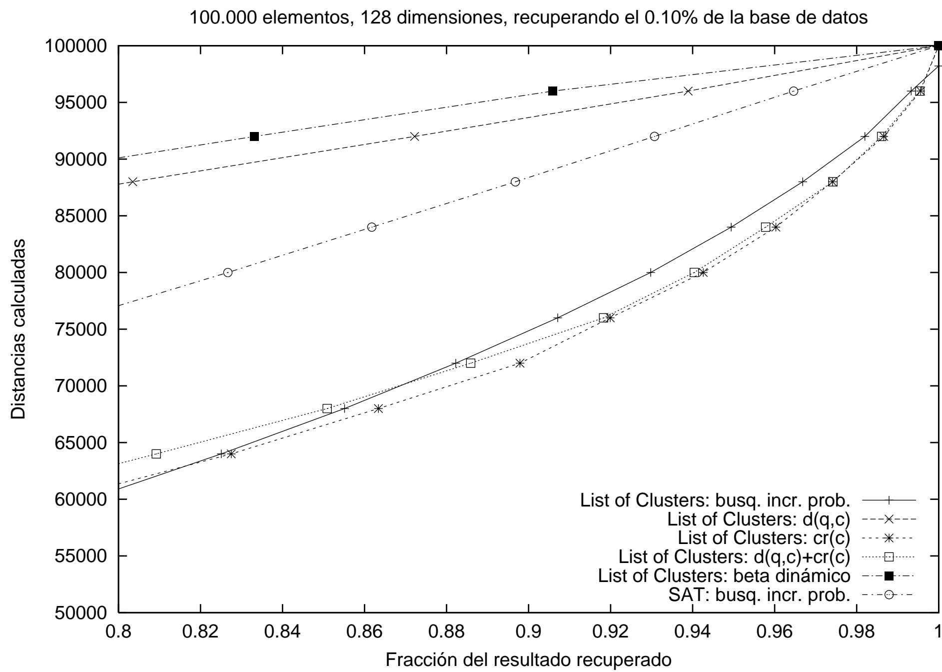

Figura 5.14: SAT y List of Clusters probabilístico, 100.000 elementos, dimensión 128, 0,10% recuperado.

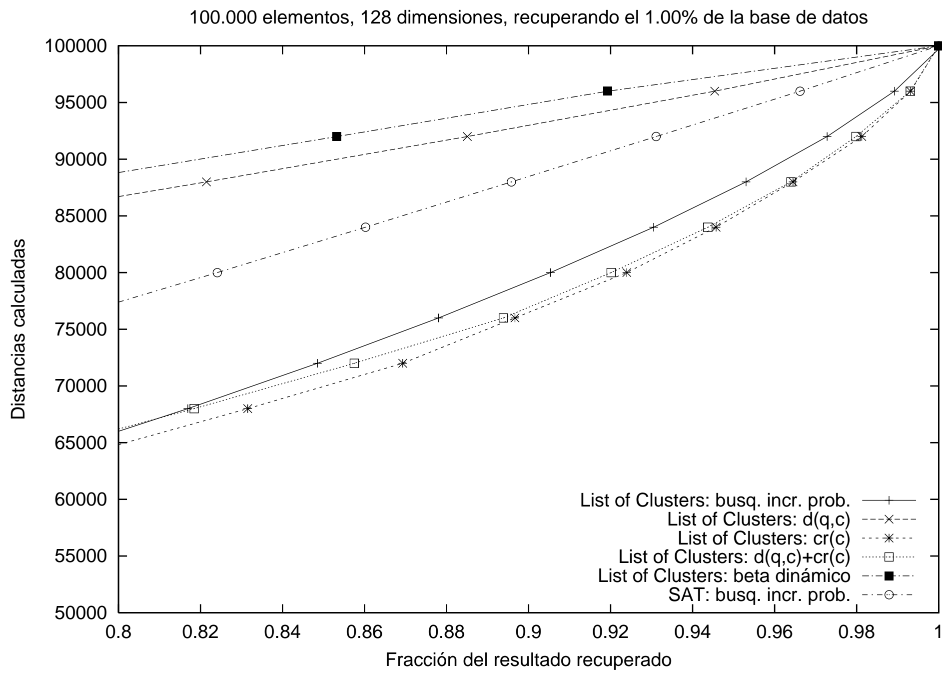

Figura 5.15: SAT y List of Clusters probabilístico, 100.000 elementos, dimensión 128, 1,00% recuperado.

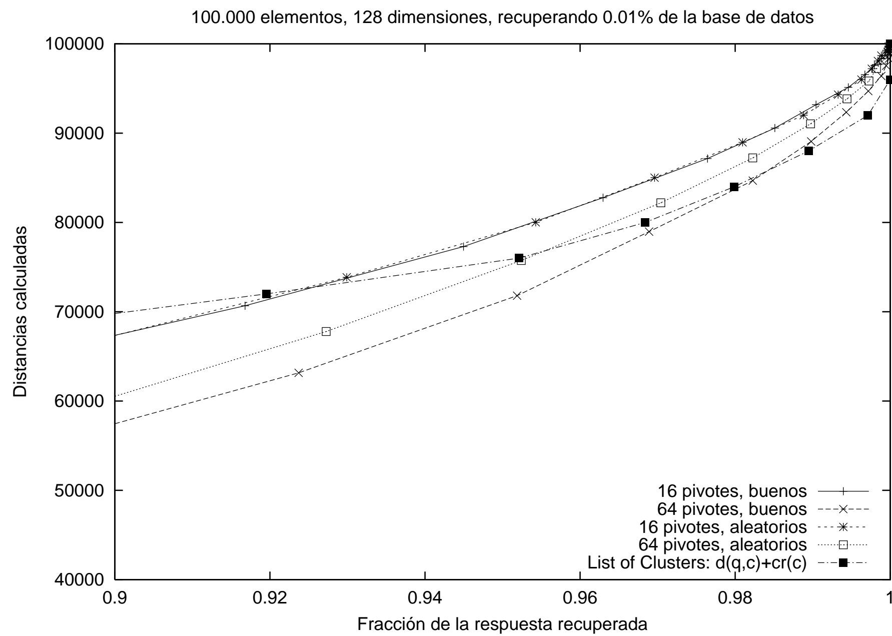

Figura 5.16: Comparación con algoritmo basado en pivotes, 100.000 elementos, dimensión 128, 0,01% recuperado.

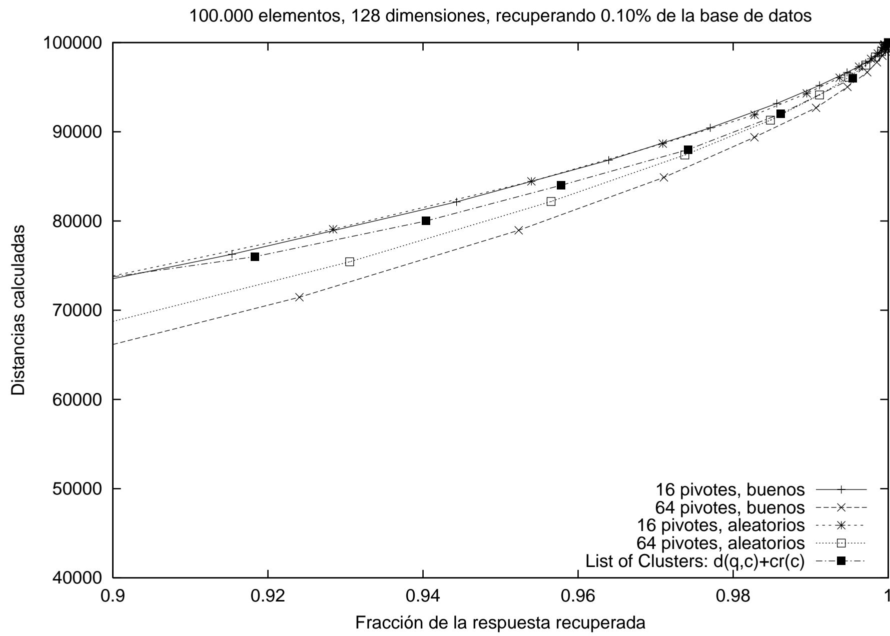

Figura 5.17: Comparación con algoritmo basado en pivotes, 100.000 elementos, dimensión 128, 0,10% recuperado.

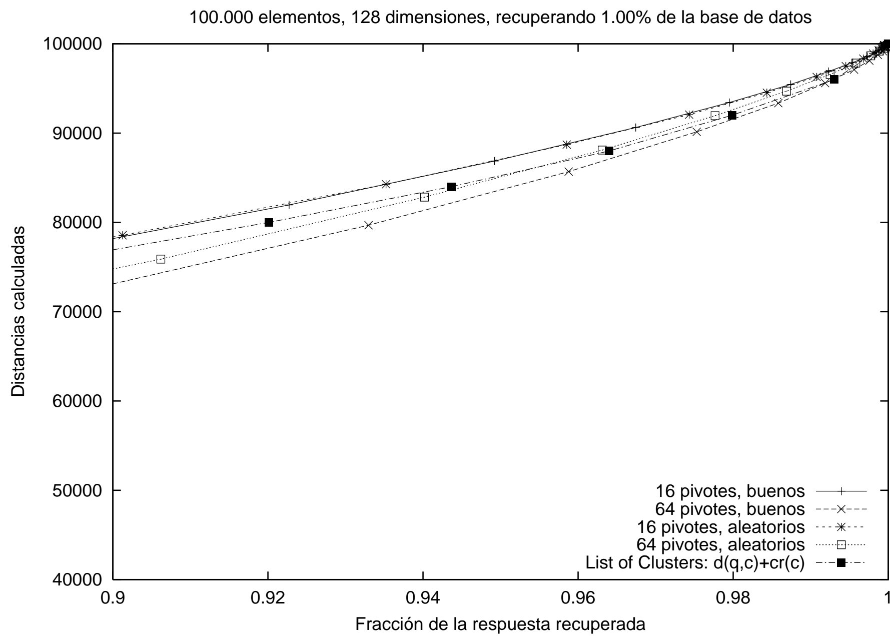

Figura 5.18: Comparación con algoritmo basado en pivotes, 100.000, dimensión 128, 1,00% recuperado.

espacio en memoria que su contraparte basada en pivotes. La Tabla 5.1 muestra el tamaño del índice para distintas cantidades de pivotes y para el List of Clusters, utilizando la base de datos de 10.000 elementos (el tamaño del índice es independiente de la dimensión del espacio). El índice del List of Clusters ocupa aproximadamente 5, 20 y 82 veces menos memoria que el índice con 16, 64 y 256 pivotes, respectivamente. Esto hace a la técnica probabilística basada en particiones compactas muy atractiva cuando la dimensión del espacio es alta, puesto que el rendimiento de esta técnica nunca fue inferior al método probabilístico con 16 pivotes y el índice basado en particiones compactas ocupa un quinto de la memoria en comparación con el índice de 16 pivotes.

Índice	Tamaño (Mb)
16 pivotes	0,62 Mb
64 pivotes	2,45 Mb
256 pivotes	9,78 Mb
List of Clusters	0,12 Mb

Tabla 5.1: Tamaños de los índices para un espacio vectorial (10.000 elementos), utilizando distintas estructuras de datos.

El algoritmo probabilístico basado en SAT no obtuvo un buen rendimiento con la búsqueda incremental probabilística, y en general ningún intento de utilizar esta estructura de datos con algún método probabilístico tuvo éxito. Es por esto que el SAT fue descartado como índice de prueba para el resto de los experimentos.

5.2 Experimentos en base de datos de documentos

En esta sección se presentan los resultados obtenidos al aplicar las técnicas probabilísticas a una base de datos de documentos. Los experimentos fueron realizados sobre el conjunto *Wall Street Journal 1987-89* de la colección de referencia TREC-3 [19]. Del total de 25.960 documentos de la colección, 24.960 de ellos fueron ocupados como la base de datos y 1.000 de ellos, escogidos aleatoriamente, fueron ocupados como objetos de consulta (note que los objetos de consulta no pertenecen a la base de datos). La función de distancia utilizada es el ángulo entre los vectores representantes de los documentos (vea la Sección 2.2.2 para una completa descripción de este espacio métrico).

En la Tabla 5.2 están señalados los radios utilizados en las consultas y el porcentaje de objetos

de la base de datos que se recuperan, en promedio, en cada consulta.

Radio	Nº documentos (promedio)	% base de datos
0,90	9	0,035%
0,95	12	0,048%
1,00	16	0,064%

Tabla 5.2: Porcentaje de documentos recuperados según radio de consulta.

Las técnicas probabilísticas fueron implementadas utilizando como índice el List of Clusters, con un tamaño de zona $m = 10$ (excepto en los casos donde se indique lo contrario).

Las Figuras 5.19, 5.20 y 5.21 muestran los resultados obtenidos con los distintos criterios de *ranking*. Dos de ellos sobresalen por sobre el resto: beta dinámico y $d(q, c)$. El criterio de beta dinámico recupera en promedio más de un 99% de los objetos relevantes recorriendo aproximadamente un 17% de la base de datos, lo cual muestra lo exitosa que resulta ser la técnica probabilística en este espacio métrico en particular.

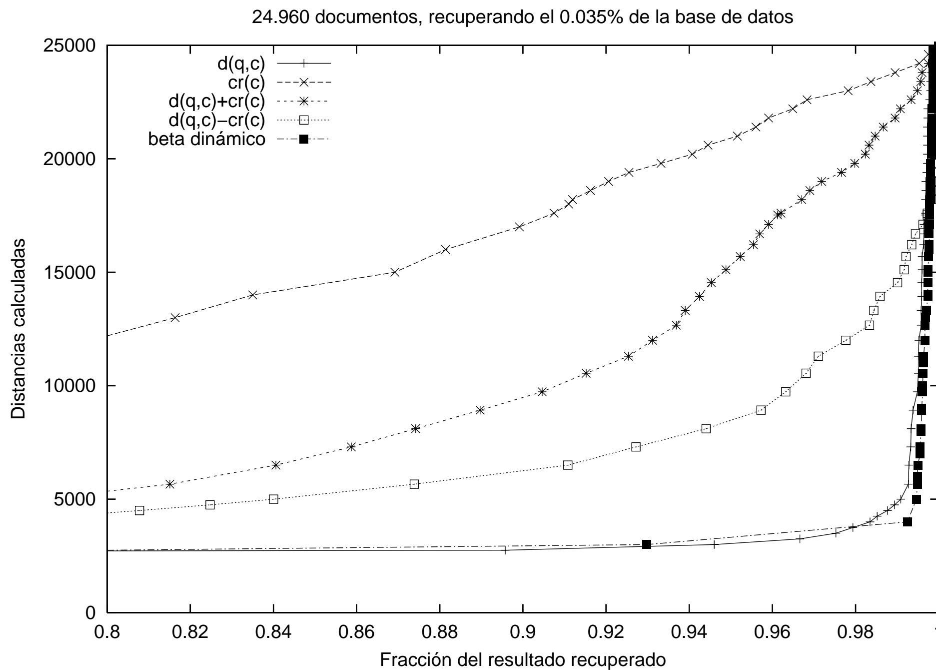

Figura 5.19: Comparación de criterios en espacio de documentos, 0,035% recuperado.

Las Figuras 5.22, 5.23 y 5.24 muestran cómo varía el rendimiento del criterio beta dinámico al variar el tamaño de zona entre $m = 5$ y $m = 30$. De los resultados se observa que para recuperar

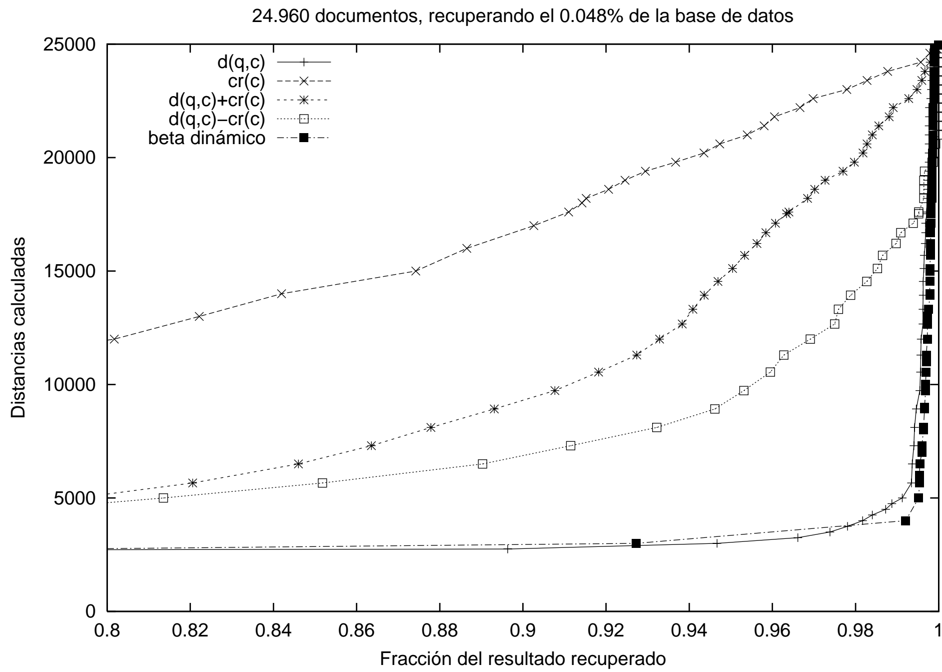

Figura 5.20: Comparación de criterios en espacio de documentos, 0,048% recuperado.

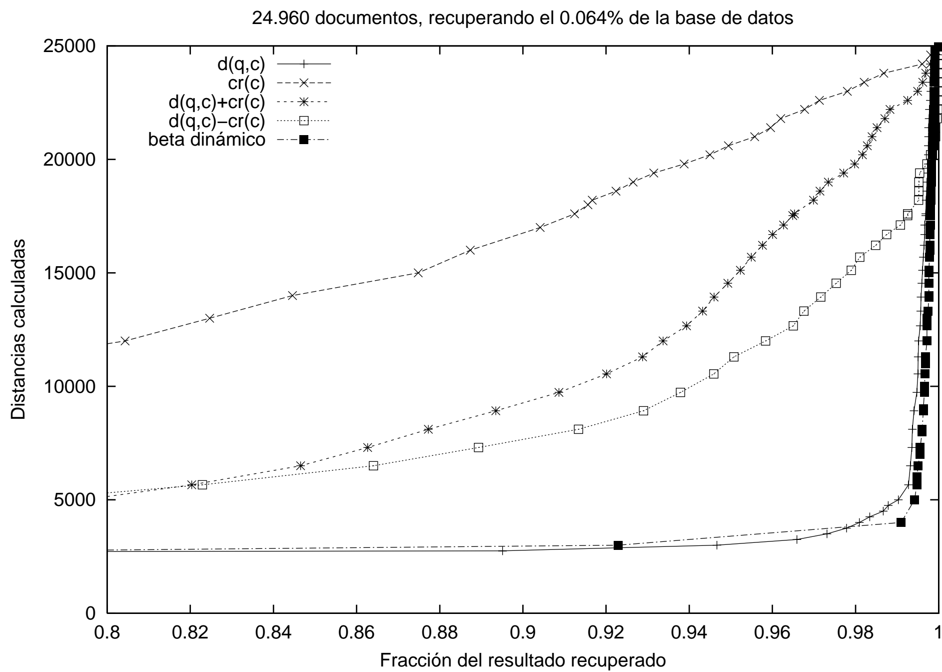

Figura 5.21: Comparación de criterios en espacio de documentos, 0,064% recuperado.

una fracción de los objetos relevantes mayor al 99,5% conviene utilizar un tamaño $m = 5$, y que para recuperar una fracción de los objetos relevantes entre 99% y 99,5% conviene utilizar $m = 10$ objetos por zona.

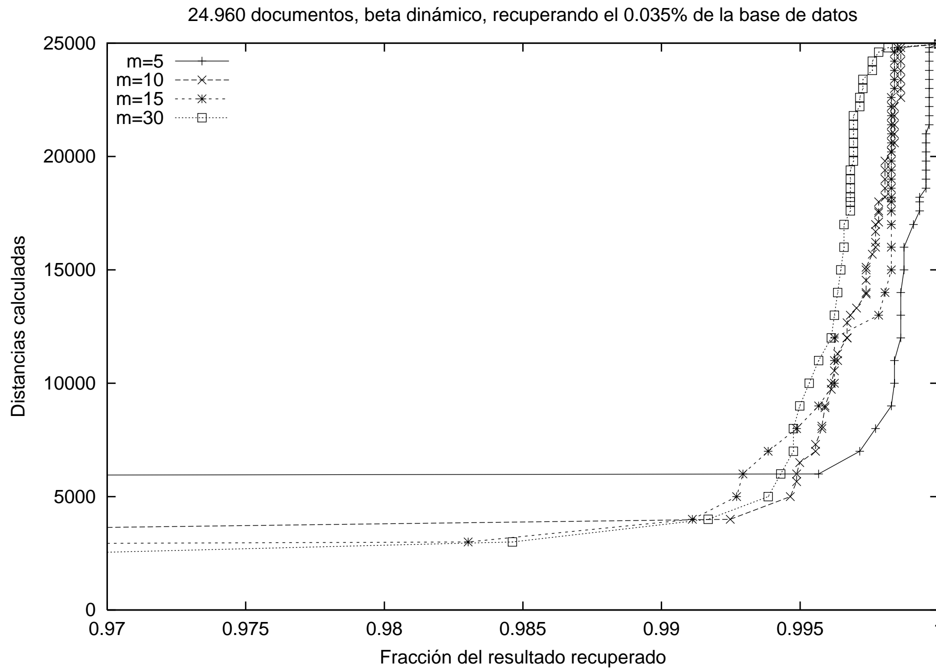

Figura 5.22: Comparación del criterio beta dinámico con distintos tamaños de zona, 0,035% recuperado.

A medida que la fracción de los objetos relevantes recuperados disminuye, se observa que conviene utilizar tamaños de zonas cada vez más grandes. Esto se debe a que la complejidad interna del algoritmo es menor al utilizar zonas con más elementos, puesto que disminuye el número total de zonas y por lo tanto se calculan menos distancias entre el objeto de consulta y los centros de zona. Esto permite ahorrar algunos cálculos de distancia al principio de la búsqueda, a costa de aumentar la complejidad externa de la búsqueda para recuperar fracciones mayores de objetos relevantes. Las Figuras 5.25, 5.26 y 5.27 muestran un gráfico de la fracción recuperada en función de la cuota de trabajo, utilizando el criterio de beta dinámico, para los distintos radios de búsqueda. Por ejemplo, para la Figura 5.27, si se desea recuperar aproximadamente un 94% de la fracción de objetos relevantes, el tamaño óptimo de zona es de 40 objetos, recorriendo en promedio un 8,01% de la base de datos. En cambio, si basta con recuperar poco más del 80% de los objetos relevantes en promedio, el tamaño óptimo de zona es de 160 objetos, recorriendo un 4% de la base de datos.

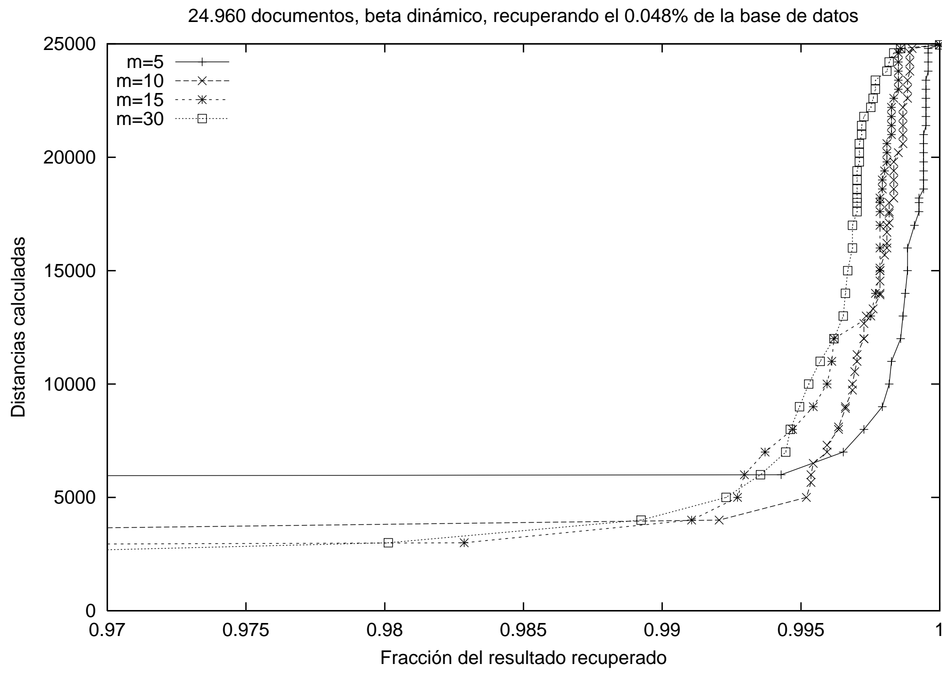

Figura 5.23: Comparación del criterio beta dinámico con distintos tamaños de zona, 0,048% recuperado.

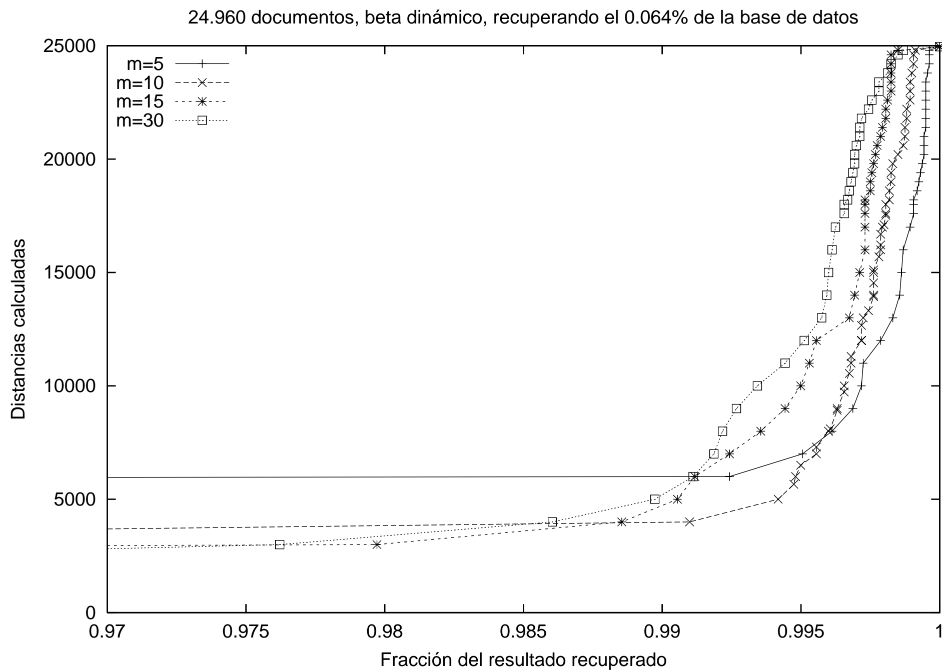

Figura 5.24: Comparación del criterio beta dinámico con distintos tamaños de zona, 0,064% recuperado.

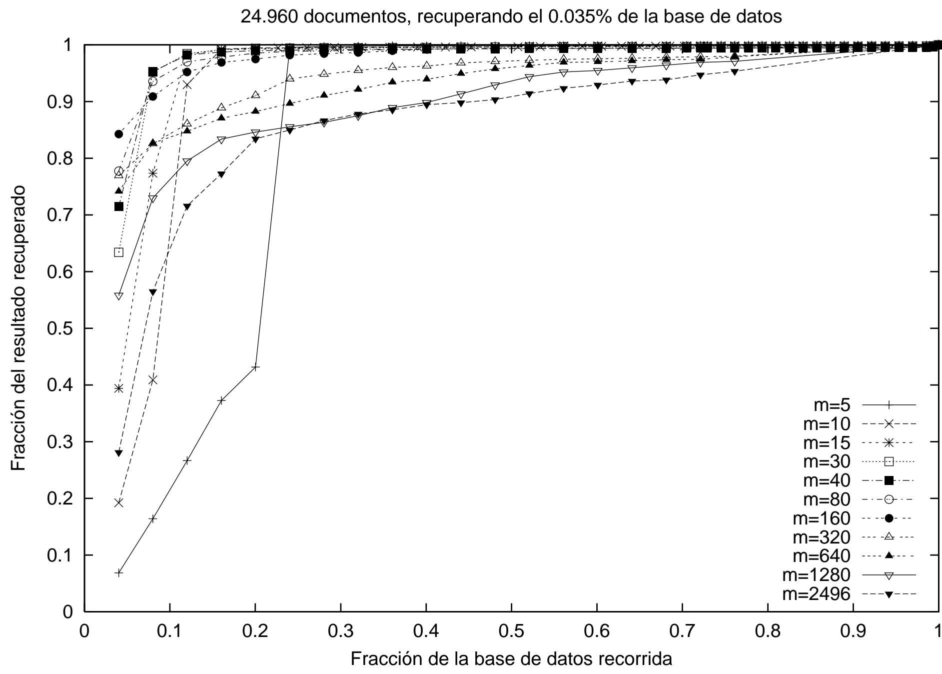

Figura 5.25: Fracción recuperada en función de la cuota de trabajo para distintos tamaños de zona, 0,035% recuperado.

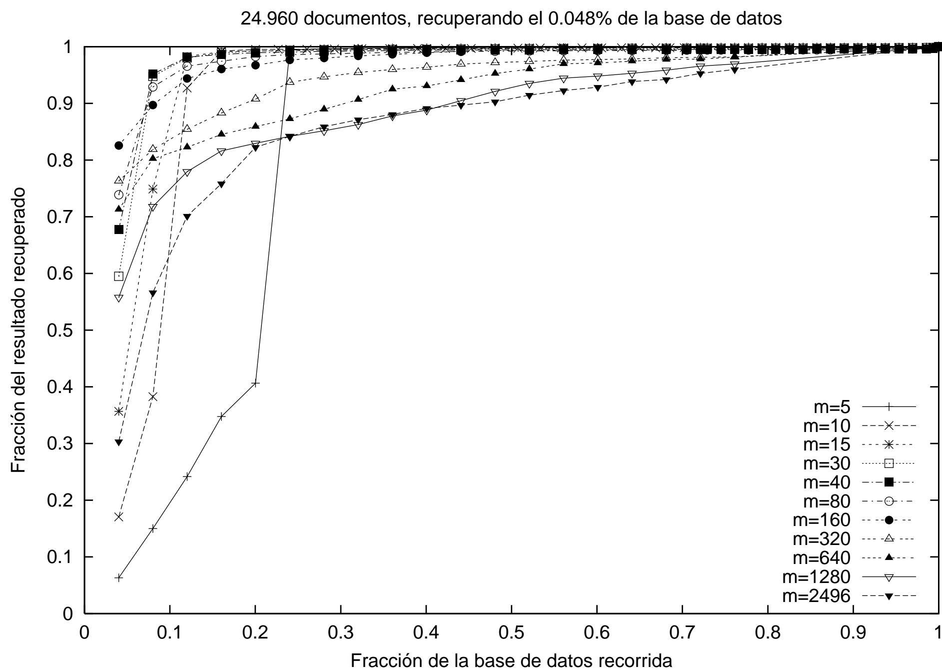

Figura 5.26: Fracción recuperada en función de la cuota de trabajo para distintos tamaños de zona, 0,048% recuperado.

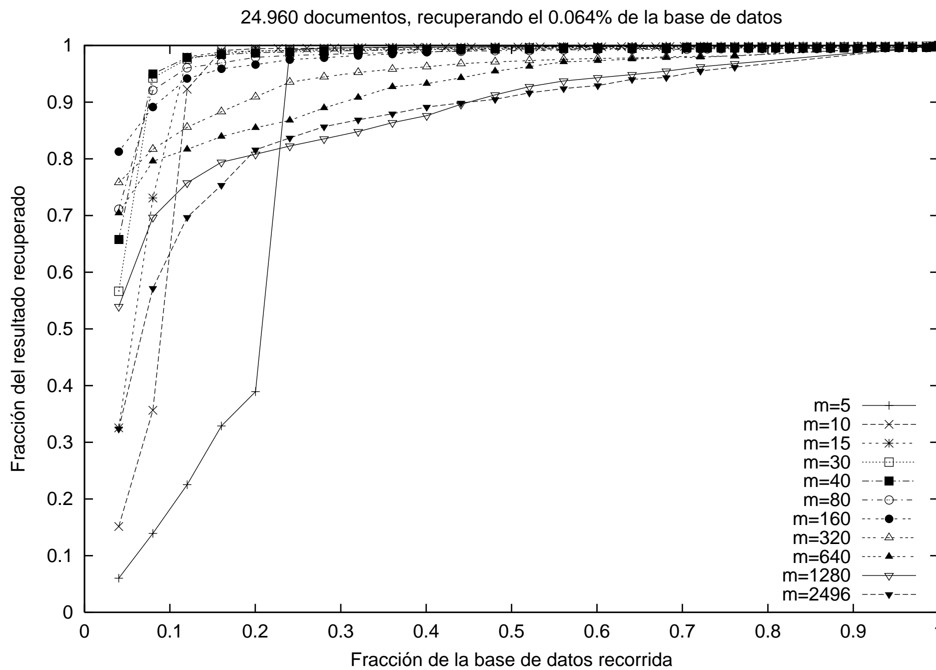

Figura 5.27: Fracción recuperada en función de la cuota de trabajo para distintos tamaños de zona, 0,064% recuperado.

Las Figuras 5.28, 5.29 y 5.30 muestran una comparación entre el criterio de *ranking* de zonas beta dinámico y el método probabilístico basado en pivotes. Se observa que el mejor método de pivotes es aquel que elige 64 pivotes aleatoriamente, pero es ampliamente superado por el criterio beta dinámico. Es interesante notar que, a pesar de agregar más pivotes al índice, éste disminuye su rendimiento al utilizar 128 pivotes. Esto implica que el método basado en pivotes ni siquiera puede superar a la técnica basada en particiones compactas utilizando más memoria para el índice. Además, se observa que es mejor usar pivotes aleatorios que "buenos" pivotes.

Las Figuras 5.31, 5.32 y 5.33 muestran los mismos resultados obtenidos en los gráficos anteriores, pero en esta ocasión se muestra la fracción de objetos relevantes recuperada en función de la fracción de distancias calculadas con respecto al tamaño de la base de datos. El gráfico muestra que para recuperar sobre el 80% de los objetos relevantes, en promedio, la técnica que realiza menos cálculos de distancia es utilizando el criterio de beta dinámico para ordenar las zonas del índice. Bajo el 80% de recuperación conviene utilizar el método basado en pivotes.

Al igual que en el caso de espacios vectoriales, el espacio en memoria requerido para almacenar

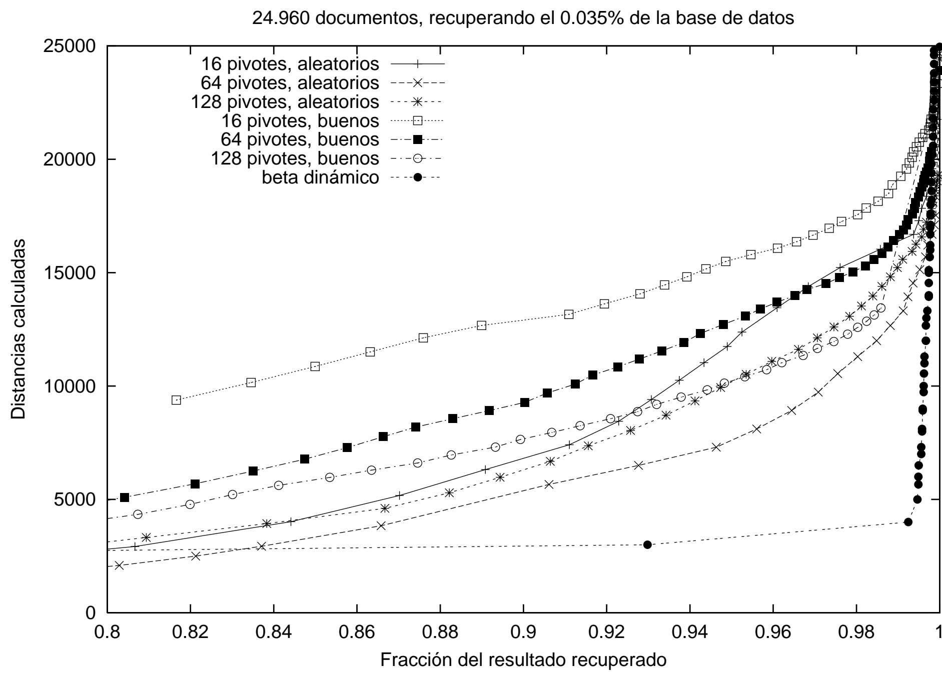

Figura 5.28: Comparación de criterio beta dinámico con método basado en pivotes, 0,035% recuperado.

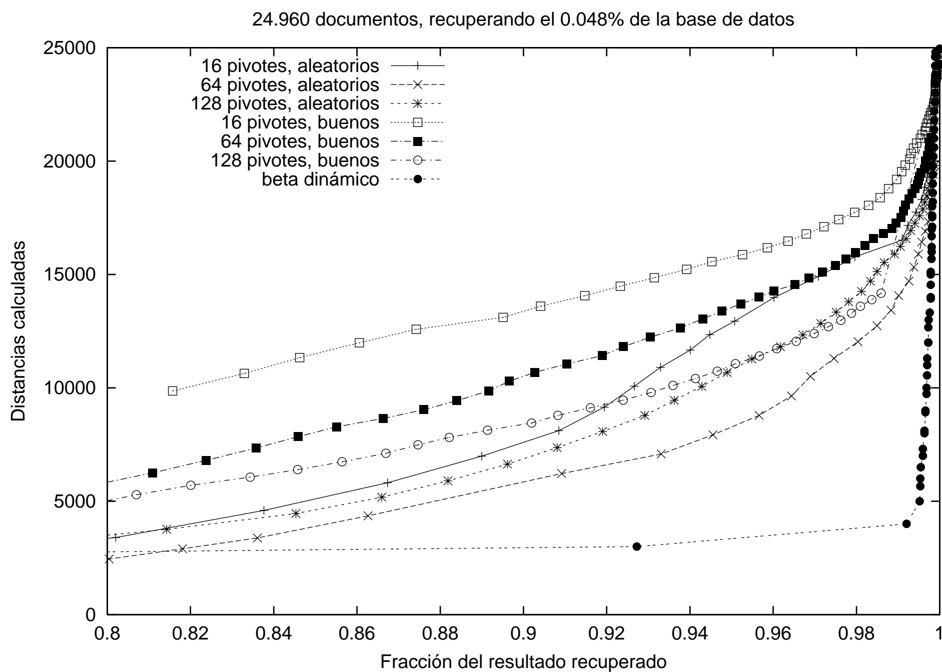

Figura 5.29: Comparación de criterio beta dinámico con método basado en pivotes, 0,048% recuperado.

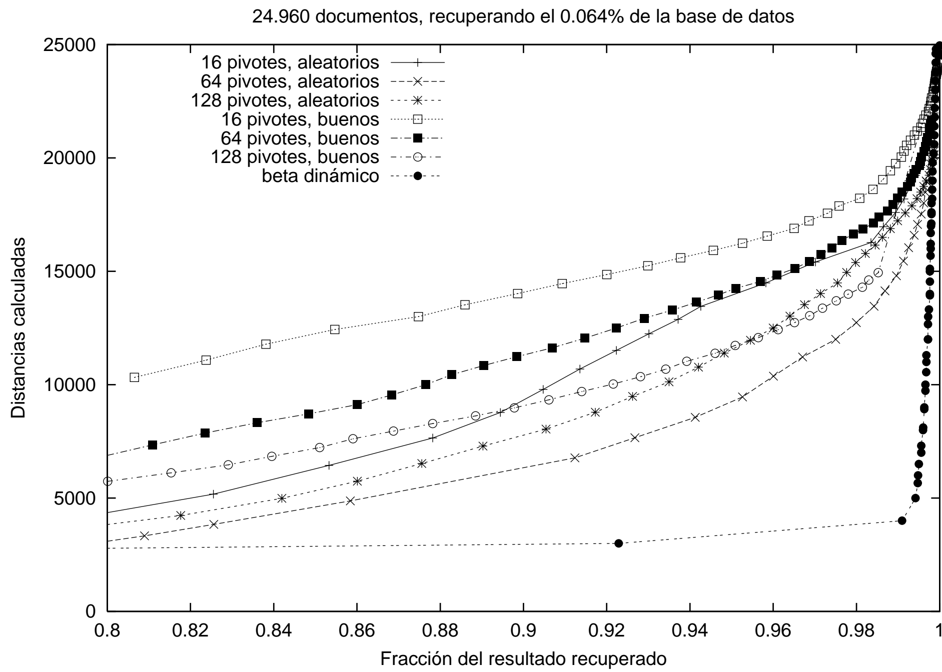

Figura 5.30: Comparación de criterio beta dinámico con método basado en pivotes, 0,064% recuperado.

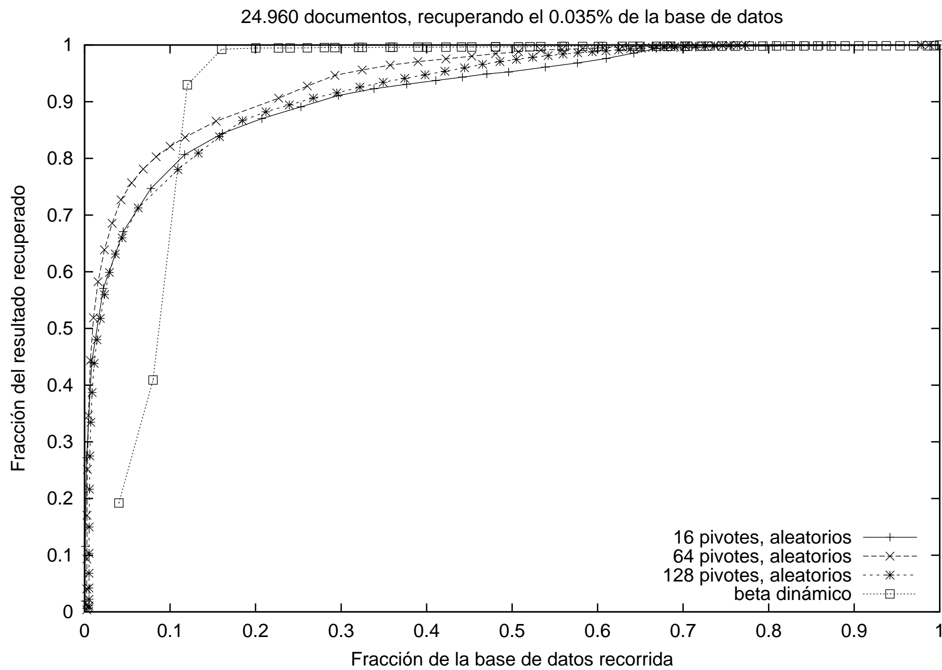

Figura 5.31: Fracción de objetos recuperados en función de la fracción de la base de datos recorrida, 0,035% recuperado.

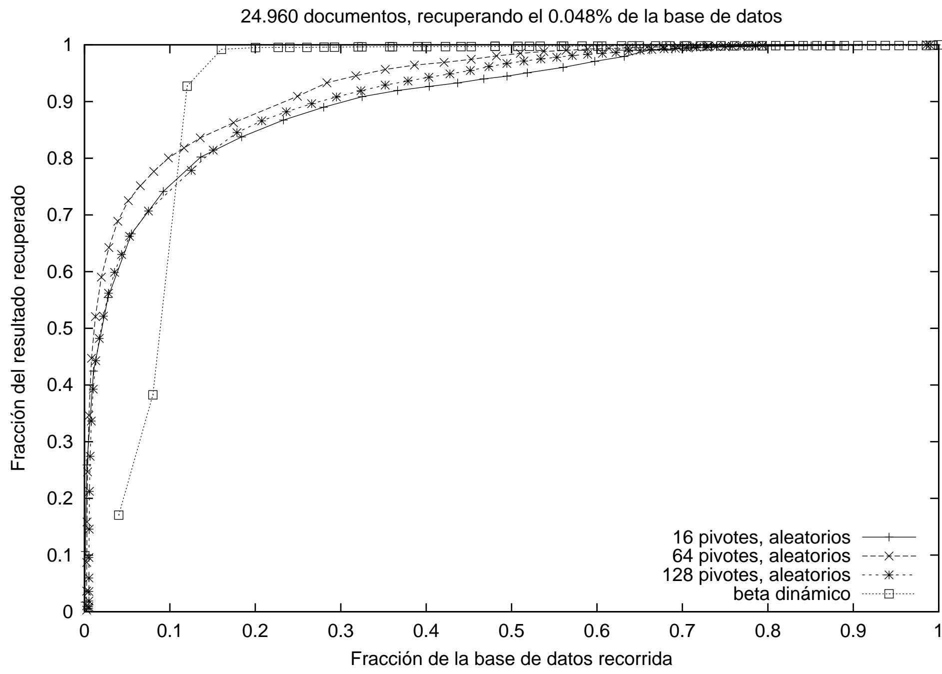

Figura 5.32: Fracción de objetos recuperados en función de la fracción de la base de datos recorrida, 0,048% recuperado.

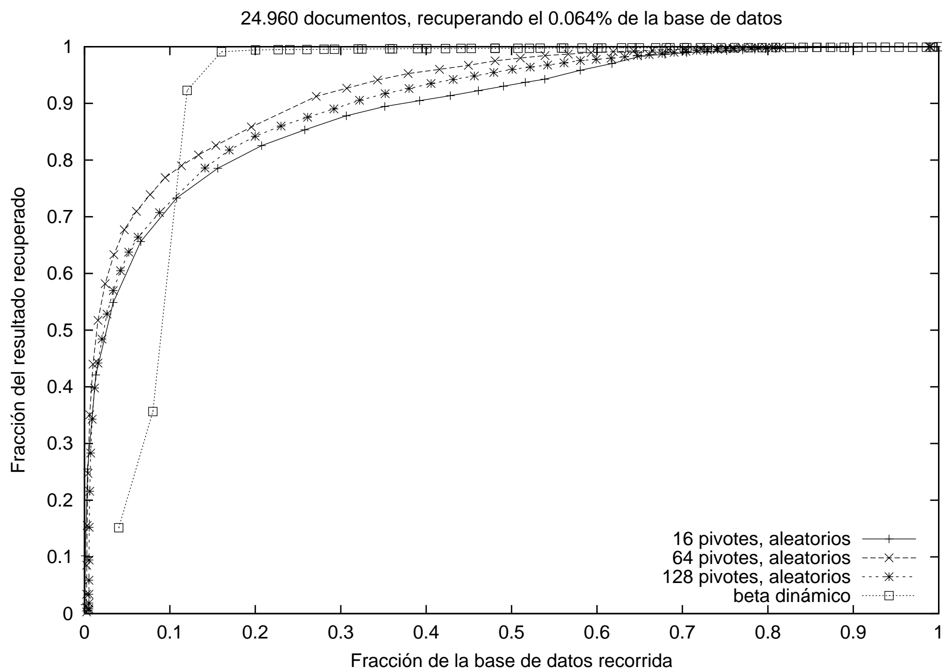

Figura 5.33: Fracción de objetos recuperados en función de la fracción de la base de datos recorrida, 0,064% recuperado.

el índice del List of Clusters es mucho menor que el utilizado por los índices basados en pivotes. La Tabla 5.3 muestra que el índice del List of Clusters ocupa aproximadamente 4, 16 y 31 veces menos memoria que los índices con 16, 64 y 128 pivotes respectivamente.

Índice	Tamaño (Mb)
16 pivotes	1,52 Mb
64 pivotes	6,09 Mb
128 pivotes	12,19 Mb
List of Clusters ($m = 10$)	0,39 Mb

Tabla 5.3: Tamaños de los índices de la base de datos de documentos para distintas estructuras de datos.

5.3 Ranking de zonas versus ranking de objetos

El criterio de ordenación $d(q,c) - cr(c)$ puede ser modificado levemente para aprovechar la información provista por la estructura de datos del List of Clusters. Si por cada zona de elementos, además de almacenar su radio cobertor, se almacenan las distancias del centro de la zona a cada objeto u_i perteneciente a ésta, se tiene que una mejor cota de la distancia entre q y u_i es $d(q,c) - d(c,u_i)$. Por ende, una variante al criterio original es utilizar $d(q,c) - d(c,u_i)$ para ordenar los objetos en la cola de prioridad. En forma similar, una variante al criterio $d(q,c) + cr(c)$ es utilizar $d(q,c) + d(c,u_i)$ como criterio de ordenación. Note que en estas variantes ya no se ordenan las zonas, sino que a cada objeto del espacio métrico se le asigna su prioridad y luego se ordenan los objetos ascendemente según dicha prioridad.

Sin embargo, en la práctica ambas variantes resultan menos efectivas que los criterios originales. Las figuras 5.34, 5.35 y 5.36 muestran los resultados de experimentos realizados en la base de datos de documentos, donde se comparan ambas variantes con los criterios originales y el criterio de beta dinámico. Se observa de los resultados que ambas variantes tienen un peor rendimiento que los criterios originales, excepto para el caso de la variante del criterio $d(q,c) - cr(c)$, en donde se observa una modesta mejora en un pequeño rango de la función (cerca del 98% de la fracción del resultado recuperado).

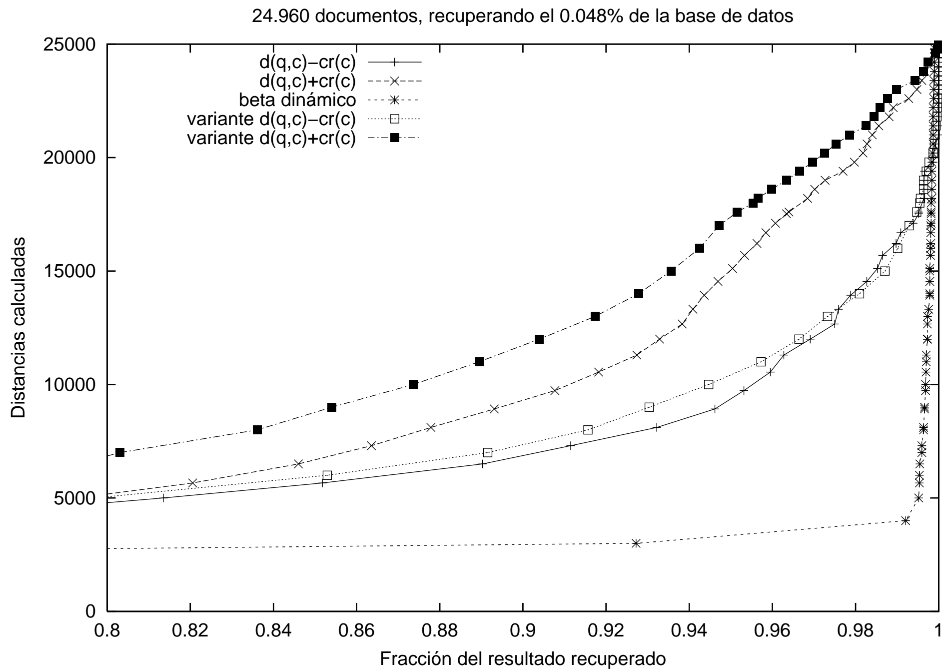

Figura 5.34: Comparación con variantes de criterios de ordenación, 0,064% recuperado.

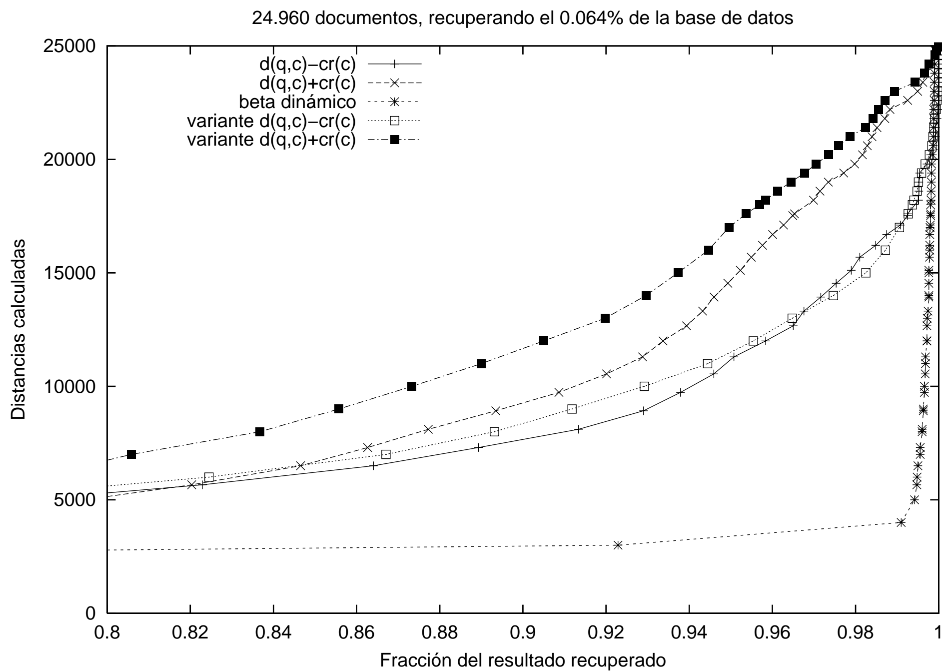

Figura 5.35: Comparación con variantes de criterios de ordenación, 0,064% recuperado.

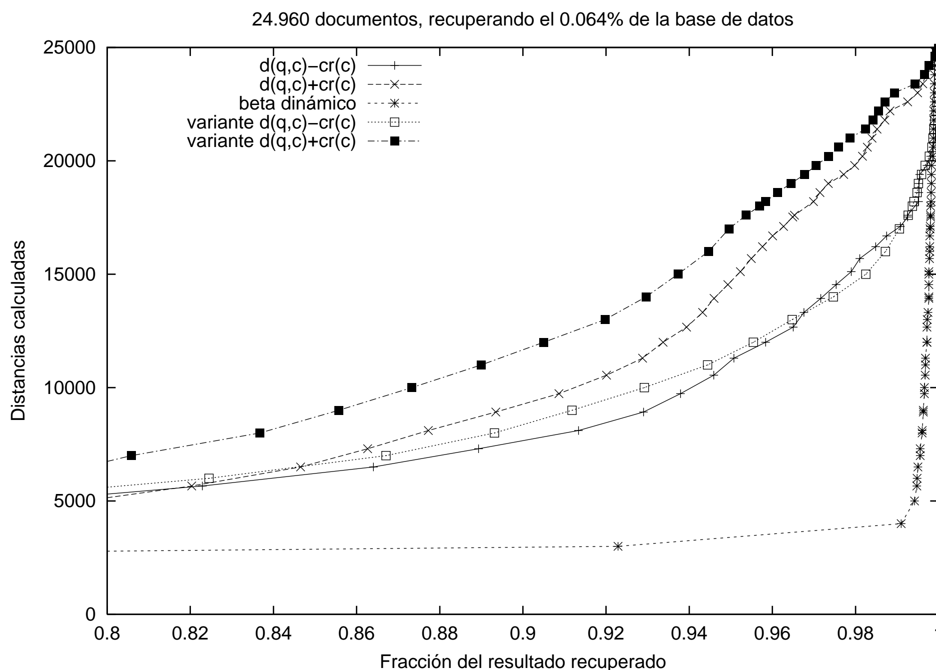

Figura 5.36: Comparación con variantes de criterios de ordenación, 0,064% recuperado.

5.4 Modelo de comparación de criterios de ranking

Una ventaja adicional que poseen los métodos de *ranking* de zonas es que permiten realizar comparaciones entre los distintos criterios de *ranking* sin tener que repetir experimentos para distintas cantidades de trabajo o distintos radios de búsqueda.

El modelo de comparación propuesto es el siguiente: para un conjunto de k consultas dado, se realiza cada consulta sin límite de trabajo, sino hasta recuperar toda la base de datos. Se almacenan los objetos en el orden en que son comparados contra la consulta, junto con su distancia al objeto de consulta. Es importante notar que, para un criterio de *ranking* y una consulta fija, el orden en el cual se obtienen los objetos de la cola de prioridad es siempre el mismo.

Con la información obtenida, se genera una nube de puntos que se representa en un gráfico *distancia a la consulta en función del número de distancias calculadas*. El rango del eje X va desde 0 hasta el total de elementos de la base de datos, y el eje Y tiene rango real positivo. Si el objeto u se extrajo luego de realizar i cálculos de distancia, se agrega el punto $(i, d(q,u))$ a la nube. Esto se hace para todos los puntos extraídos en todas las consultas, totalizando kn puntos. La figura 5.37

muestra un ejemplo de una nube puntos.

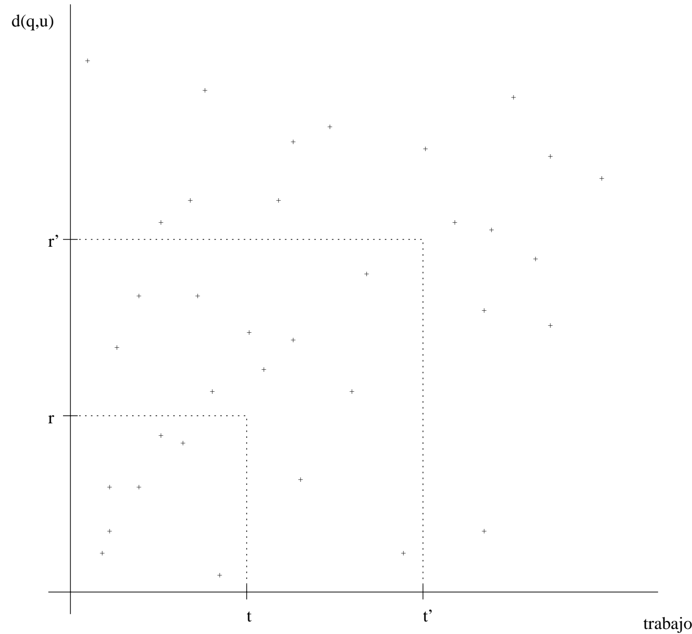

Figura 5.37: Ejemplo de nube de puntos para un criterio dado.

Esta nube de puntos permite simular cualquier experimento que se pudiera realizar en el conjunto de consulta procesado, en donde se varíe la cantidad de trabajo permitida y/o el radio de búsqueda. Por ejemplo, si se desea saber cuántos objetos relevantes se hubieran recuperado en promedio con un radio de búsqueda r y permitiendo t cálculos de distancia, basta con contar los puntos (x, y) de la nube con $x \leq t$ e $y \leq r$, dividiendo esa cantidad por el número total de consultas, k. Llamemos $A(t, r)$ a este valor. Dado que se conocen todas las distancias entre objetos y consultas, se puede saber exactamente cuántos objetos caen dentro de la bola de consulta para un radio de búsqueda fijo $(A(\infty, r))$, con lo cual es fácil calcular la fracción f de objetos relevantes recuperados para cualquier cantidad de trabajo, $f = \frac{A(t,r)}{A(\infty,r)}$.

El proceso se puede repetir para cualquier r' y t', lo cual permite obtener varios puntos de la función de costo de un criterio específico, fijando el radio de búsqueda y variando la cantidad de trabajo permitida. La Figura 5.38 muestra los resultados obtenidos a través de los experimentos realizados en la base de datos de documentos (los mismos de la sección anterior), y la Figura 5.39 muestra los resultados obtenidos con el modelo de comparación, usando 100 consultas ($\frac{t}{n}$ vs. $\frac{A(t,r)}{A(\infty,r)}$). La diferencia entre ambos gráficos es muy pequeña.

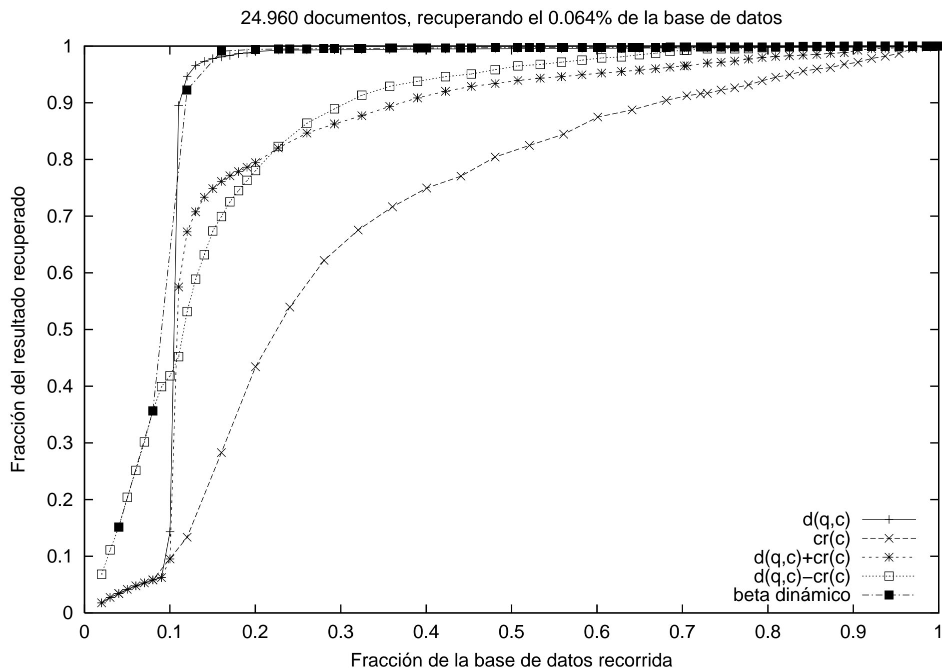

Figura 5.38: Fracción de objetos recuperados en función de la fracción de la base de datos recorrida para los distintos criterios de *ranking*, 0,064% recuperado.

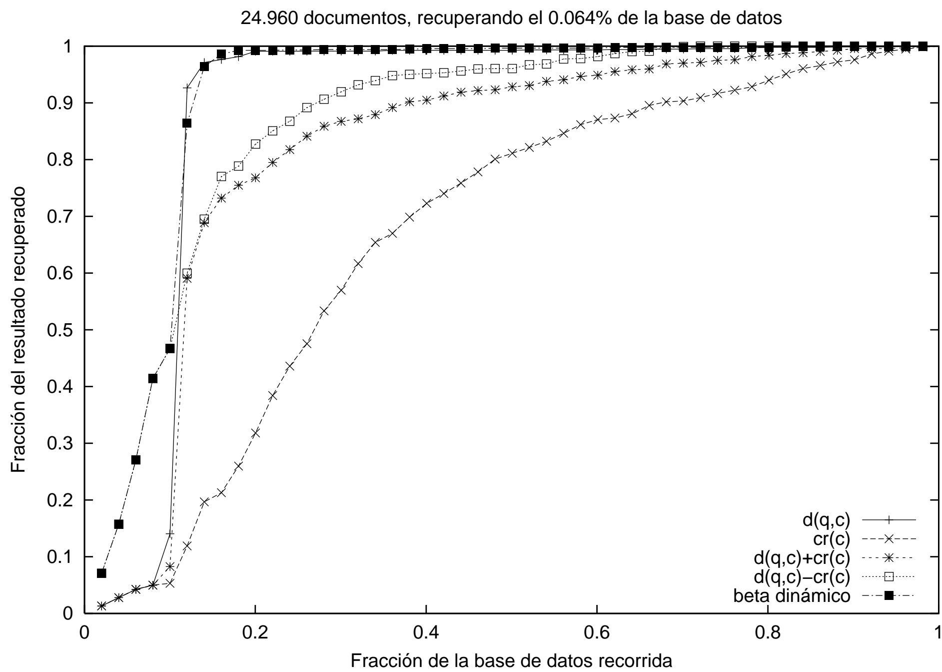

Figura 5.39: Resultado utilizando el modelo de comparación, 0,064% recuperado.

La desventaja que tiene el uso de este modelo de comparación es que requiere almacenar grandes cantidades de información, puesto que cada consulta contribuye con una cantidad de datos proporcional al tamaño de la base de datos. Esto se puede evitar discretizando los valores de $d(q, u)$ a s segmentos y definiendo una matriz de tamaño $s \times t$, en donde se almacenarán acumuladores para los valores discretizados de $(i, d(q, u))$. De esta forma, el costo total en espacio es de st, pero se puede perder precisión al momento de calcular $A(t, r)$.

Capítulo 6

Conclusiones

El objetivo de esta tesis fue desarrollar y proponer algoritmos probabilísticos para búsquedas en proximidad basados en particiones compactas, dado que estas estructuras de datos son las más eficientes para implementar algoritmos exactos de búsqueda cuando la dimensión intrínseca del espacio métrico es alta, y dado que los índices basados en particiones compactas tienen menores requerimientos en memoria que los basados en el uso de pivotes.

Los principales resultados obtenidos en esta investigación se pueden resumir en los siguientes puntos:

- Se definió un algoritmo probabilístico de búsqueda en proximidad basado en la búsqueda incremental de vecinos más cercanos [20], denominado *búsqueda incremental probabilística*, el cual tiene como parámetro el número de cálculos de distancia permitidos para realizar la búsqueda.
- Se definió un algoritmo probabilístico de búsqueda en proximidad denominado *ranking de zonas*, el cual ordena las particiones compactas según algún criterio, y luego realiza la búsqueda en ese orden, fijándose de antemano el número de cálculos de distancia permitidos.
- Ambos algoritmos probabilísticos son del tipo Monte Carlo, esto es, no aseguran encontrar todos los objetos relevantes, pero a medida que se permite realizar más trabajo (cálculos de distancia) la calidad de la respuesta mejora. Además, ambos algoritmos son 1-sesgados, puesto

que nunca reportan como relevantes objetos que no lo sean. Estos algoritmos se basan en un *modelo de tiempo de acotado de búsqueda*, pues se sabe a priori cuánto demorará la búsqueda, dado que el número de cálculos de distancia está acotado antes de empezarla.

- Para la técnica de *ranking* de zonas, se definieron distintos criterios de ordenamiento de zonas. Se propuso como heurística favorecer aquellas particiones compactas cercanas que posean una mayor "densidad" de elementos, como por ejemplo en el criterio denominado *beta dinámico*.

- Se realizaron experimentos en espacios vectoriales aleatorios de dimensión alta y en una base de datos de documentos, implementando los algoritmos probabilísticos en los índices SAT [23] y List of Clusters [13], comparando los resultados obtenidos con un algoritmo probabilístico basado en pivotes [14].

- Los resultados experimentales muestran que la técnica de *ranking* de zonas obtiene iguales o mejores resultados que el algoritmo basado en pivotes, necesitando este último mucha más memoria para poder ser competitivo.

- También se observa de los resultados experimentales que la eficiencia del método probabilístico depende del criterio utilizado para ordenar las zonas, y que para ciertos espacios métricos puede que un criterio funcione bien pero para otros no.

- Se propuso un modelo de comparación de criterios de *ranking* de zonas, en donde es posible simular búsquedas para cualquier radio y cuota de trabajo permitida para un conjunto de consultas dado sin tener necesidad de realizar todos los experimentos por separado.

En general son cuatro las ventajas que tienen los algoritmos probabilísticos basados en particiones compactas por sobre el algortimo probabilístico basado en pivotes. La primera ventaja se refiere al tamaño de los índices, dado que estructuras como el SAT o List of Clusters ocupan mucho menos memoria que un índice basado en pivotes, incluso cuando la cantidad de éstos sea pequeña (16, por ejemplo). La segunda ventaja es que, tanto el método de búsqueda incremental probabilística como el *ranking* de zonas, pueden mejorar la calidad de la respuesta si se les permite una mayor cantidad de trabajo, sin tener que realizar el cálculo desde cero, lo cual no ocurre con el algoritmo basado en pivotes, en donde antes de realizar la búsqueda se debe fijar el parámetro β; si se desea mejorar la calidad de la respuesta hay que modificar β y realizar la búsqueda de nuevo. La tercera ventaja es que con las técnicas basadas en particiones compactas se sabe exactamente cuánto demorará la búsqueda, puesto que se fija de antemano el número de cálculos de distancia permitidos, lo cual

no sucede con el algoritmo basado en pivotes (en el peor caso, éste puede recorrer toda la base de datos para responder una consulta). La última ventaja es que el algoritmo basado en pivotes empeora su rendimiento más rápido que el algoritmo basado en particiones compactas a medida que la dimensión del espacio aumenta.

La principal conclusión de este estudio es que es posible aplicar el enfoque de búsqueda en espacios métricos a aplicaciones reales en donde antes no era posible, respondiéndose búsquedas en proximidad en donde se pierden pocos objetos relevantes y se disminuye dramáticamente el tiempo de búsqueda con respecto al algoritmo exacto.

Por ejemplo, una de las aplicaciones más directas de los espacios métricos en Recuperación de la Información es la búsqueda de documentos relevantes a una cierta consulta [5], la cual puede ser vista como un conjunto de términos o como un documento completo. Como se describe en la Sección 2.2.2, los documentos y las consultas son representados por vectores, donde cada término es una coordenada cuyo valor es el peso del término en dicho documento. La distancia entre dos documentos se define como el ángulo formado entre sus vectores representantes, por lo que dos documentos que compartan términos importantes estarán cercanos bajo esa métrica.

A pesar que este enfoque es bastante directo, los espacios métricos no han sido utilizados para implementar búsquedas de documentos similares. Una razón para esto es que el espacio métrico resultante posee una dimensión intrínseca muy alta, por lo que todos los algoritmos exactos de búsqueda en proximidad prácticamente realizan una búsqueda exhaustiva para responder consultas por rango útiles, lo cual resulta ser impráctico en aplicaciones reales.

Éste es un caso en donde un algoritmo probabilístico puede ser de gran utilidad, dado que la relación de "relevancia" de un documento con respecto a otro conlleva un cierto grado de inexactitud y permite un cierto grado de "aproximación" en la respuesta, es decir, el recuperar la mayoría de los documentos relevantes puede ser suficiente y aceptable.

Los resultados experimentales obtenidos muestran que, utilizando la técnica de *ranking* de zonas implementada en un List of Clusters, es posible recuperar más del 99% de los documentos relevantes recorriendo sólo un 17% de la base de datos, y que es posible recuperar más del 94% de los documentos relevantes recorriendo sólo un 8% de la base de datos. Con la técnica probabilística basada en particiones compactas es posible, por primera vez, presentar una solución práctica a este

problema de Recuperación de la Información utilizando el enfoque de espacios métricos.

Otro aspecto interesante es el poder contar con una herramienta como el modelo de comparación descrito en la Sección 5.4. Hasta el momento, para comparar algoritmos de búsqueda en proximidad era necesario repetir el experimento para distintos radios de búsqueda. El modelo descrito es independiente del radio de búsqueda, lo que lo convierte en una poderosa herramienta para comparar criterios de ordenación de zonas, o incluso para comparar el método probabilístico con otras estructuras de datos utilizadas como índice.

Algunos puntos interesantes para abordar en futuras investigaciones son los siguientes:

- Utilizar otros esquemas de *clustering* para probar los criterios de *ranking* de zonas. Es razonable conjeturar que si los puntos pertenecientes al espacio métrico forman *clusters*, es posible que el método probabilístico obtenga mejores resultados utilizando técnicas de *clustering* más avanzadas.

- Investigar algoritmos *aproximados* de búsqueda de vecinos más cercanos en espacios métricos. Por ejemplo, en el caso de la búsqueda del vecino más cercano, desarrollar un algoritmo que asegure que el objeto recuperado en la búsqueda se encuentre a lo más a distancia $1 + \epsilon$ del vecino más cercano a la consulta, para algún parámetro $\epsilon > 0$.

- Poder calcular analíticamente la probabilidad de éxito de los algoritmos propuestos. Los resultados actuales se limitan a casos particulares de espacios métricos que satisfacen ciertas propiedades [17], al caso de algoritmos de búsqueda de vecinos más cercanos [2, 33, 30] y al caso particular de los espacios vectoriales [1].

Por último, cabe destacar que parte de los resultados obtenidos en esta investigación fueron aceptados para su presentación y publicación en el *9th International Symposium on String Processing and Information Retrieval (SPIRE'02)* [9] (ver Apéndice A).

Bibliografía

[1] S. Arya y D. Mount. Approximate range searching. En *Proc. 11th Annual ACM Symposium on Computational Geometry*, páginas 172–181, 1995.

[2] S. Arya, D. Mount, N. Netanyahu, R. Silverman, y A. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimension. En *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms (SODA'94)*, páginas 573–583, 1994.

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

[4] R. Baeza-Yates, W. Cunto, U. Manber, y S. Wu. Proximity matching using fixed-queries trees. En *Proc. 5th Combinatorial Pattern Matching (CPM'94)*, LNCS 807, páginas 198–212, 1994.

[5] R. Baeza-Yates y B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley, 1999.

[6] T. Bozkaya y M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. En *Proc. ACM SIGMOD International Conference on Management of Data*, páginas 357–368, 1997. Sigmod Record 26(2).

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

[8] W. Burkhard y R. Keller. Some approaches to best-match file searching. *Comm. of the ACM*, 16(4):230–236, 1973.

[9] B. Bustos y G. Navarro. Probabilistic proximity searching based on compact partitions. En *Proc. 9th International Symposium on String Processing and Information Retrieval (SPIRE'02)*. LNCS, Springer, 2002. Por aparecer.

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

[11] E. Chávez, J. Marroquín, y R. Baeza-Yates. Spaghettis: An array based algorithm for similarity queries in metric spaces. En *Proc. 6th International Symposium on String Processing and Information Retrieval (SPIRE'99)*, páginas 38–46. IEEE CS Press, 1999.

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

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

[14] E. Chávez y G. Navarro. Probabilistic proximity search: Fighting the curse of dimensionality in metric spaces. *Information Processing Letters*, 2002. Por aparecer. Versión preliminar en *ALENEX'01*, LNCS 2153.

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

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

[17] K. Clarkson. Nearest neighbor queries in metric spaces. *Discrete Computational Geometry*, 22(1):63–93, 1999.

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

[19] D. Harman. Overview of the Third Text REtrieval Conference. En *Proc. Third Text REtrieval Conference (TREC-3)*, páginas 1–19, 1995. NIST Special Publication 500-207.

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

[21] I. Kalantari y G. McDonald. A data structure and an algorithm for the nearest point problem. *IEEE Transactions on Software Engineering*, 9(5):631–634, 1983.

[22] L. Micó, J. Oncina, y E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (AESA) with linear preprocessing-time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.

[23] G. Navarro. Searching in metric spaces by spatial approximation. *The Very Large Databases Journal (VLDBJ)*, 2002. Por aparecer. Versión anterior en *SPIRE'99*, IEEE CS Press.

[24] H. Noltemeier. Voronoi trees and applications. En *Proc. International Workshop on Discrete Algorithms and Complexity*, páginas 69–74, Fukuoka, Japón, 1989.

[25] H. Noltemeier, K. Verbarg, y C. Zirkelbach. Monotonous Bisector* Trees – a tool for efficient partitioning of complex schenes of geometric objects. In *Data Structures and Efficient Algorithms*, LNCS 594, páginas 186–203. Springer-Verlag, 1992.

[26] F. P. Preparata y M. Ian Shamos. *Computational Geometry, an Introduction.* Springer-Verlag, 1985.

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

[28] J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40:175–179, 1991.

[29] E. Vidal. An algorithm for finding nearest neighbors in (approximately) constant average time. *Pattern Recognition Letters*, 4:145–157, 1986.

[30] D. White y R. Jain. Algorithms and strategies for similarity retrieval. Technical Report VCL-96-101, Visual Computing Laboratory, University of California, La Jolla, California, July 1996.

[31] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. En *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, páginas 311–321, 1993.

[32] P. Yianilos. Excluded middle vantage point forests for nearest neighbor search. In *DIMACS Implementation Challenge, ALENEX'99*, Baltimore, MD, 1999.

[33] P. Yianilos. Locally lifting the curse of dimensionality for nearest neighbor search. Technical report, NEC Research Institute, Princeton, NJ, 1999.

Apéndice A

Artículo Publicado

A continuación se presenta la publicación desarrollada durante el presente trabajo de tesis. El artículo titulado *Probabilistic Proximity Searching Algorithms Based on Compact Partitions* fue aceptado en el *9th International Symposium on String Processing and Information Retrieval (SPI-RE'02)*, y está próximo a publicarse en las actas de dicha conferencia en el mes de Septiembre del presente año. Estas actas serán publicadas por *Springer* en la serie *Lectures Notes in Computer Science.*

En el artículo se presentan los principales resultados obtenidos durante esta investigación.