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

MANIPULACIÓN DE ESTRUCTURAS MÉTRICAS EN MEMORIA SECUNDARIA

Roberto Uribe Paredes

2005

Agradecimientos

En esta etapa de mi vida me es grato recordar a todos aquellos que de una u otra manera han aportado a la culminación de este trabajo.

Primero, a mis padres Adolena y Manuel, por darme la oportunidad de desarrollarme durante los inicios de mi carrera.

A Marlies, mi esposa, por estar siempre conmigo, por su paciencia, su esfuerzo y su amor. A Atenea y Renato, mis hijos y la luz en mi camino, por su amor, ternura y júbilo y por permitirme quitarles un poco del tiempo que debería de haber estado con ellos.

A mis queridos amigos Carinne Santana y Humberto Carrasco, por estar cuando más se les necesita.

Quiero agradecer a mi profesor guía Gonzalo Navarro, por darme la oportunidad de trabajar con él, por todos sus aportes y conocimientos, fundamentales para el adecuado desarrollo de esta tesis, por su confianza y paciencia.

A mis compañeros del programa de Magíster y Doctorado del DCC, por hacer mi estadía más agradable y a Angélica Aguirre, jefa de estudios, por su permanente disposición a colaborar.

Al director del Departamento de Ingeniería en Computación de la Universidad de Magallanes, Dr. Carlos Árias M. por toda su ayuda y colaboración. A Mariela Aguilar, secretaria, por su constante preocupación, apoyo y ayuda durante mi ausencia de Punta Arenas. A mi colega Julio Águila, por su colaboración en la realización de algunas tareas personales.

Quiero agradecer muy especialmente a mi colega Nora Reyes, académica del Departamento de Informática de la Universidad Nacional de San Luis, Argentina, por su desinteresada ayuda cada vez que la necesitaba, por sus conocimientos aportados en la escritura de esta tesis.

Quiero agradecer formalmente a la Universidad de Magallanes representada en su rector Dr. Victor Fajardo Morales y al decano de la Facultad de Ingeniería Sr. Juan Oyarzo por las facilidades entregadas para la realización de mis estudios de postgrado.

Finalmente quiero agracedecer los aportes económicos tanto de mi universidad, del proyecto MECESUP MAG9901 del Ministerio de Educación de Chile y de las becas otorgadas por el Departamento de Ciencias de la Computación de la Universidad de Chile; todos ellos facilitaron el financiamiento de este trabajo.

Roberto Uribe Paredes. Punta Arenas, Chile, Abril 2004.

Resumen

La *búsqueda por similaridad* consiste en recuperar todos aquellos objetos dentro de una base de datos que sean parecidos o relevantes a una determinada consulta. Este concepto tiene una amplia gama de aplicaciones en áreas como bases de datos multimediales, reconocimiento de patrones, minería de datos, recuperación de información, etc. La búsqueda por similaridad o en proximidad se modela matemáticamente a través de un *espacio métrico*, en el cual un objeto es representado como una caja negra, donde la única información disponible es la *función de distancia* de este objeto a los otros. La función de distancia o *métrica* debe satisfacer ciertas propiedades, entre ellas la desigualdad triangular. La búsqueda por similaridad modela entonces cómo encontrar los objetos de la base de datos suficientemente cercanos a una consulta dada.

En general, el cálculo de la distancia es costoso, por ello el objetivo es reducir la cantidad de evaluaciones de distancia necesarias para resolver la consulta. Para esto existen numerosos métodos, lo cuales construyen estructuras llamadas *índices*, pagando un costo en preprocesamiento de los datos a fin de disminuir las evaluaciones de distancia al momento de la búsqueda.

Existen espacios donde buscar es intrínsecamente más difícil que en otros. Esto tiene relación con la distribución de la función de distancia y se conoce como *maldición de la dimensionalidad*. La mayoría de las técnicas tradicionales no son eficientes en estos espacios, llamados *de alta dimensionalidad*.

Existen dos caracteríscas poco comunes en las estructuras actuales. La primera es el *dinamismo*, es decir, la posibilidad de insertar y eliminar objetos una vez construida la estructura. La segunda característica, aún menos frecuente, es la manipulación de estas estructuras en memoria secundaria. No poseer dichas características hace poco factible la utilización de estas estructuras en aplicaciones reales.

La presente tesis aborda estos dos problemas presentando una nueva estructura denominada *Evolving GNAT* (*egnat*), la cual es completamente dinámica y optimizada para memoria secundaria. Ésta está basada en el *Geometric Near-neighbor Access Tree* (*gnat*) de Sergey Brin, es estática pero con buen desempeño en espacios de alta dimensionalidad.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Capítulo 1

Introducción

En Ciencias de la Computación, la búsqueda es uno de los problemas que ha concentrado gran parte de los esfuerzos de científicos y técnicos, y está presente prácticamente en toda aplicación. La búsqueda que se realiza más comúnmente es, "dada una consulta sobre un conjunto de datos, se recupera el valor que es igual a la consulta o query". Ésta es conocida como *búsqueda exacta* y se aplica en bases de datos tradicionales sobre datos estructurados. Una base de datos tradicional representa la información a través de entidades y relaciones (bases de datos relacionales [Cod70] y otros modelos basados en el anterior), donde cada entidad o relación se compone por registros o tuplas que representan al objeto, el cual está caracterizado por atributos o campos alfanuméricos. La búsqueda se realiza comparando la clave de búsqueda contra uno de estos atributos. Sobre estas bases de datos también se pueden realizar búsquedas más sofisticadas como consultas por rango sobre claves numéricas o prefijos sobre claves alfabéticas, sin embargo, éstas también se basan en el concepto de que dos claves son o no iguales o que existe un orden sobre las claves. A pesar de que en la actualidad las bases de datos permiten almacenar tipos de datos más complejos, como por ejemplo, imágenes, las búsquedas se realizan de igual manera sobre los atributos de tipo alfanumérico.

Con la rápida evolución de las tecnologías de la información han surgido nuevos depósitos no estructurados de datos tales como texto libre, imagen, sonido y video. Con la aparición de estos nuevos tipos de datos no sólo se hace necesario almacenarlos, sino también realizar búsquedas sobre ellos. Estructurar este tipo de datos es dificultoso ya sea manual o computacionalmente y restringe de antemano los tipos de búsqueda posibles. Más aún, realizar búsquedas exactas sobre estos datos sería poco útil. Por ejemplo, si se consultase por un elemento sobre una base de datos de imágenes, la consulta sólo podría encontrar su copia digital exacta en la base de datos. El verdadero interés reside, por ejemplo, en consultar sobre una base de datos de fotografías una imagen que contiene un rostro, donde no necesariamente existe una copia exacta de la misma fotografía; identificación de individuos a través de dispositivos biométricos, donde el dato consulta (voz, retina, etc) podría verse afectado por factores externos; encontrar una especie más parecida a otra en una base de datos de cadenas de ADN, etc.

Este tipo de búsqueda se denomina *búsqueda por similaridad* o *búsqueda en proximidad*. Se requiere recuperar un objeto más *parecido* o *relevante* a la consulta, en este sentido es posible que la consulta no sea parte de la base de datos.

Existe una amplia variedad de aplicaciones prácticas en diversas áreas donde es interesante utilizar búsqueda por similaridad, ejemplos de éstas son:

Consultas por contenido en bases de datos multimedia: En objetos tales como sonido, video e imagen, carece de interés realizar búsquedas exactas, esto debido a que la probabilidad de que dos objetos sean iguales es ínfima, para ello tendrían que ser copias digitales. Sin embargo, resulta interesante buscar objetos que sean similares entre sí, por ejemplo, la consulta por una huella digital donde ésta contenga información incompleta, es decir, falten algunos puntos que utilizan algunos mecanismos actuales de identificación.

Bioinformática: Los objetos clásicos de estudio en biología molecular son las secuencias de proteínas y cadenas de ADN, las cuales se pueden modelar a través de secuencias de caracteres. Los biólogos están interesados en encontrar, por ejemplo, mutaciones en distintas especies, o partes de una secuencia dada sobre un conjunto de secuencias más largas de proteínas.

Recuperación de información: Donde es necesario recuperar documentos relevantes a una consulta dada. Una forma tradicional de buscar documentos es determinar una medida de relevancia de acuerdo a la cantidad de veces que aparecen las palabras de la consulta en los documentos. Entonces, sería de interés poder recuperar documentos con contenido semántico similar, es decir, párrafos o frases distintas que signifiquen lo mismo.

Compresión y transmisión de audio y video: Durante la transmisión de una señal de audio o video se puede suponer que un cuadro (una imagen estática en un video) esta formado por subcuadros, entonces es posible enviar un primer cuadro completo para luego enviar de los siguientes cuadros sólo los subcuadros que tengan diferencias substanciales respecto de los previamente enviados.

Detección de plagios: Esta es una aplicación general basada en las anteriores, y cuyo objetivo es determinar si existe plagio de un objeto, por ejemplo, un artículo de otros o una melodía de otras.

La búsqueda por similaridad es un concepto unificador para la mayoría de las consultas de interés sobre datos no tradicionales. La similaridad se modeliza usando una *función de distancia* o *métrica*, la cual satisface las propiedades de *positividad*, *simetría* y *desigualdad triangular*, y el conjunto de objetos es llamado un *espacio métrico*. Los objetos de la base de datos son tratados como cajas negras donde sólo se conocen las distancias entre objetos.

Un caso particular de espacio métrico es el *espacio vectorial*, donde los objetos consisten de D coordenadas de valores reales, donde D representa la dimensión del espacio. Existen diversos

trabajos que aprovechan las propiedades geométricas de estos espacios, sin embargo, usualmente no se pueden generalizar a espacios métricos, donde la única información disponible o posible de calcular es la distancia entre un objeto y otro.

Existe dos tipos de consultas en búsqueda por similaridad:

- *Consulta por Rango*: el objetivo es recuperar todos los elementos de la base de datos que estén a cierta distancia (radio de tolerancia) de un objeto consulta.
- *k-vecinos más cercanos*: se desea recuperar los k elementos más cercanos a una consulta dada.

El segundo tipo de consulta se puede obtener a través del primero.

La forma trivial de responder a estas consultas es realizando una búsqueda exhaustiva sobre la base de datos, es decir, comparar cada uno de los elementos con la consulta y determinar cuál de ellos cumple la condición, por ejemplo que la distancia sea menor a un radio de tolerancia. En general, la función de distancia se considera computacionalmente costosa de calcular, por lo que durante la búsqueda, el objetivo general es reducir la cantidad de evaluaciones de distancia. Por ejemplo, la distancia entre dos tramas de video tiene un costo elevado comparado con otros tipos de operaciones realizadas durante la búsqueda.

Los algoritmos para búsqueda por similaridad en espacios métricos construyen a priori un estructura llamada *índice*, que permite reducir los cálculos de distancia durante la búsqueda. La construcción de un índice implica un costo adicional de preproceso de la base de datos, sin embargo, esto es en función de evitar posteriormente evaluaciones de distancia con algunos objetos, para lo que se usa la desigualdad triangular. Los algoritmos de búsqueda por similaridad se dividen en dos grandes áreas: algoritmos basados en pivotes y algoritmos basados en particiones compactas. En [CNBYM01] se presenta un marco unificador que describe y analiza las distintas soluciones para el problema de búsqueda por similaridad.

Para todas las aplicaciones anteriores lo relevante es definir la función de distancia a utilizar sobre cada tipo de objeto. La familia de distancias más utilizada para el caso de espacios vectoriales es la conocida como *distancias de Minkowski*.

1.1. Relevancia y Aportes de la Tesis

La existencia de aplicaciones reales sobre espacios métricos de alta dimensión genera un obstáculo para el diseño de técnicas de búsqueda eficiente. La mayoría de las técnicas tradicionales de indexación no tienen un buen desempeño sobre este tipo de espacios. La búsqueda por similaridad se torna más difícil a medida que crece la dimensión intrínseca de un espacio , este hecho es conocido como la *maldición de la dimensionalidad* [CNBYM01, Bri95].

En este contexto existen varios índices o estructuras métricas para búsquedas en espacios de alta dimensión. Sin embargo, hay dos características poco comunes en estas estructuras, la primera tiene

relación con características dinámicas, la mayoría de estas estructuras deben ser reconstruidas si existen nuevos objetos que indexar o si hubiese que eliminar objetos de la base de datos. La segunda característica, aún menos frecuente, dice relación con estructuras para manipulación de datos en memoria secundaria. En sí, dichas estructuras han sido diseñadas como prototipos, por lo que su utilización en aplicaciones reales no ha sido probada, más aún, considerando que en aplicaciones reales los objetos son de gran tamaño y/o la base de datos es considerablemente grande, donde es posible pensar que sólo un bajo porcentaje de datos podría entrar en RAM.

En este sentido existen estructuras para espacios multidimensionales que poseen estas características, pero que, sin embargo, no son adecuadas para los espacios mencionados.

Desde el punto de vista de dinamismo, algunos índices permiten inserciones eficientes una vez construidos y en los últimos años han aparecido algoritmos con capacidades dinámicas completas, como son el *M-tree* [CPZ97] y el *SAT* (*Spatial Approximation Tree*) [NR02]. El mayor problema en este caso es la eliminación de objetos, que resulta en exceso compleja sobre todo en estructuras de tipo árbol.

Desde el punto de vista de memoria secundaria las más conocidas son el *M-tree* y el *Slim-tree* [TTSF00] el cual está basado en el primero. El *M-tree* posee solapamiento de planos, es decir, los planos que particionan el espacio tienen intersecciones, esto implica que un objeto puede ser insertado en más de una posición dentro de la estructura, lo que repercute en la eficiencia durante la búsqueda. El *Slim-tree* reduce el grado de traslape o solapamiento de planos.

Objetivo General:

La presente tesis tiene como objetivo general abordar estos dos puntos, es decir, diseñar una estructura completamente dinámica y con buen desempeño en memoria secundaria.

Para esto se eligió como base la estructura denominada *gnat* (*Geometric Near-neighbor Access Tree*) [Bri95], ya que posee buen desempeño en espacios de alta dimensión y no tiene solapamiento de planos, por lo que resulta ser una buena alternativa para convertirla en una estructura dinámica sobre memoria secundaria.

Se considera que los aportes más relevantes de la presente tesis son:

- El diseño de un algoritmo eficiente de eliminación denominado *Planos Fantasmas*, aplicable a cualquier estructura de tipo árbol para búsquedas en espacios métricos.
- Lograr mantener la eficiencia de las búsquedas sobre una estructura que puede estar cambiando continuamente, además, el método propuesto para eliminaciones posee un excelente desempeño en memoria secundaria (sección 5.2 y 5.4.2.3).
- Conseguir una visión más profunda de los parámetros a considerar tanto para la aplicación de características dinámicas sobre estructuras métricas, como para su implementación sobre

memoria secundaria. Esto permite que futuros diseños de estructuras métricas para memoria secundaria tengan una base considerando los futuros problemas a enfrentar.

Finalmente, el aporte principal de la tesis es:

- Presentar una nueva estructura métrica denominada *egnat* (*Evolutionary gnat* o *gnat evolutivo*) optimizada para memoria secundaria, dinámica y con buen desempeño en búsquedas por similaridad sobre espacios de alta dimensión.

1.2. Organización de la Tesis

La organización del presente trabajo de tesis está dividido en tres partes:

- La primera corresponde a una descripción del marco teórico y conceptos básicos sobre el contexto en el que está basado la tesis (capítulo 2) y una descripción detallada del algoritmo original del *gnat* presentado en [Bri95], además de otras estructuras relevantes a la tesis (capítulo 3).
- La segunda parte es la presentación y análisis experimental del algoritmo de eliminación para el *gnat* (capítulo 4).
- La tercera y final es fundamentalmente la presentación de la estructura *egnat* (capítulo 5) como propuesta de solución al espacio requerido por la estructura original sobre memoria secundaria.

Capítulo 2

Marco Teórico

La similaridad se modeliza en muchos casos interesantes a través de un espacio métrico, y la búsqueda de objetos más similares a través de una búsqueda por rango o de vecinos más cercanos. En el presente capítulo se definen formalmente los conceptos referentes a búsquedas en espacios métricos, además de ejemplos de espacios y los temas relevantes a considerar respecto de dinamismo y memoria secundaria para estructuras métricas.

2.1. Espacios Métricos

Un *espacio métrico* es un conjunto X de objetos válidos, con una función de distancia $d : X^2 \to \mathbb{R}$, tal que $\forall x, y, z \in X$ cumple con las siguientes propiedades:

- Positividad : $d(x, y) \geq 0,\ x \neq y \Rightarrow d(x, y) > 0$.
- Simetría: $d(x, y) = d(y, x)$.

Estas propiedades sólo aseguran una definición consistente de la función, pero no pueden usarse para descartar objetos y ahorrar comparaciones de distancia durante una búsqueda por similaridad. Para que d sea realmente una métrica del espacio X debe satisfacer:

- Desigualdad triangular : $d(x, y) + d(y, z) \geq d(x, z)$.

Entonces, el par (X, d) es llamado *Espacio Métrico*.

2.2. Búsquedas por Similaridad

Sea $Y \subseteq X$ el conjunto de objetos que componen la base de datos. El concepto de búsqueda o consulta por similaridad o en proximidad consiste en recuperar todos los objetos pertenecientes a Y que sean parecidos a un elemento de consulta q que pertenece al espacio X.

Las consultas en proximidad sobre espacios métricos son básicamente dos [CNBYM01]:

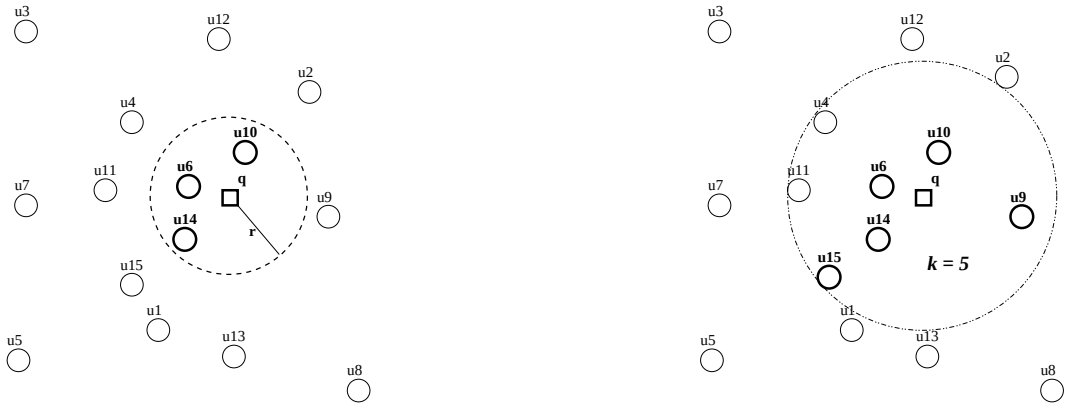


Figura 2.1: Ejemplos de consultas, por rango (izquierda) y por k-vecinos más cercanos (derecha) sobre un conjunto de puntos en $\mathbb{R}^2$.

Consulta por Rango $(q, r)_d$**:** Sea un espacio métrico *(X,d)*, un conjunto de datos finito $Y \subseteq X$, una consulta $q \in X$, y un rango $r \in \mathbb{R}$. La consulta por rango alrededor de q con rango r es el conjunto de puntos $y \in Y$, tal que $d(q, y) \leq r$.

Los k Vecinos más Cercanos $NN_k(q)$**:** Sea un espacio métrico *(X,d)*, un conjunto de datos finito $Y \subseteq X$, una consulta $q \in X$ *y un entero* k. Los k vecinos más cercanos a q son un subconjunto A de objetos de Y, donde la $|A| = k$ y no existe un objeto $y \in X - A$ tal que *d(y,q)* sea menor a la distancia de algún objeto de A a q.

El tipo básico de consulta es la consulta por rango. En la figura 2.1 se ilustran ambos tipos de consulta. Para mayor claridad las consultas están realizadas sobre un conjunto de puntos en $\mathbb{R}^2$ (espacio métrico). A la izquierda se muestra una consulta por rango con radio r y a la derecha una consulta por los *5*-vecinos más cercanos a q. En este último caso, también se gráfica el rango necesario para encerrar los 5 puntos. Entonces, se observa que dada una consulta q y una cantidad k (en este ejemplo *5*), es posible que existan distintas respuestas.

En el presente trabajo, el interés está centrado en la búsqueda por rango, debido a que es la base para resolver el otro tipo de consulta.

2.3. Indexación

Una manera trivial de responder a estos tipos de consulta es examinar exhaustivamente la base de datos Y, lo que toma tiempo $O(n)$. De hecho, esta es la única forma de realizar consultas por similaridad si no fuese posible construir una estructura que ayude a responder las consultas. Sin embargo, considerando que la función de distancia es *computacionalmente costosa* de calcular, no es práctico realizar una consulta de esta manera.

Por lo tanto, se hace necesario preprocesar la base de datos, para lo cual se paga un costo inicial de construcción de un *índice* a fin de ahorrar computaciones de distancia al momento de resolver la búsqueda. Esta estructura permitirá realizar búsquedas posteriores.

Un *algoritmo de indexamiento* es un proceso que construye una estructura denominada *índice*, el cual permite realizar búsquedas por similaridad evitando, en lo posible, tener que revisar toda la base de datos, ahorrándose evaluaciones de distancia al momento de realizar las consultas. Todos los algoritmos de búsquedas en espacios métricos utilizan la desigualdad triangular para descartar objetos, esto con el fin de evitar calcular la distancia real con el objeto de consulta. Esto último da una idea de la información que debería contener la estructura.

Hay dos casos principales que considerar: cuando el índice y los datos pueden ser mantenidos en memoria principal, y cuando es necesario utilizar memoria secundaria para almacenar los datos y/o el índice. Para los algoritmos de indexación en memoria principal, el objetivo central es reducir el número de cálculos de distancia. Para los algoritmos en memoria secundaria, además de reducir los cálculos de distancia, debe considerar también que los accesos a disco y movimiento de cabezales sean pocos. En este tópico en particular los avances han sido más lentos que los realizados para memoria principal.

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

$$T = \#\textit{evaluaciones de } d \times \textit{complejidad}(d) + \textit{tiempo extra de CPU} + \textit{tiempo de E/S}$$

En muchas aplicaciones la evaluación de la distancia es tan costosa que las demás componentes de la fórmula anterior pueden ser despreciadas. Sin embargo, para el este trabajo incluiremos el costo de la entrada y salida, es decir, la cantidad de accesos y movimientos en memoria secundaria (*read*, *write* y *seek*).

La tesis desarrollada orienta parte de los esfuerzos hacia este último problema, es decir, considerar que la base de datos es lo suficientemente grande para que el índice no entre completamente en memoria principal.

La otra característica importante que se considera en el presente trabajo, son las capacidades dinámicas que puedan poseer los índices métricos. Es decir, la posibilidad de inserción y eliminación en la estructura posterior a su construcción. Si bién, muchas estructuras soportan inserciones posteriores, existe sólo un pequeño conjunto de estructuras que implementan la eliminación. Éstas en su mayoría están diseñadas para memoria principal (ver sección 2.7).

Diseñar estructuras métricas que soporten eliminaciones resulta complejo, esto agrabado por la necesidad de su implementación en memoria secundaria.

2.4. Ejemplos de Espacios Métricos

A continuación se presentan algunos ejemplos de espacios métricos.

2.4.1. Espacios Vectoriales

Los espacios vectoriales son un caso particular de espacio métrico. Un espacio vectorial $\mathbb{R}^k$ es un conjunto de *k-tuplas* de números reales. Existen distintas funciones de distancia para espacios vectoriales. Entre las más comunes están las pertenecientes a la familia L_s o familia de distancias de *Minkowski*, que se define como:

$$L_s((x_1, ..., x_k)(y_1, ..., y_k)) = \left(\sum_{i=1}^{k} |x_i - y_i|^s \right)^{\frac{1}{s}}, \ 1 \leq s \leq \infty$$

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

- L_1, conocida como *distancia de Manhattan*:

$$L_1((x_1, ..., x_k)(y_1, ..., y_k)) = \sum_{i=1}^{k} |x_i - y_i|$$

- L_2, conocida como *distancia Euclidiana*:

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

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

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

En muchas aplicaciones los espacios métricos son en realidad espacios vectoriales, es decir, que los objetos son puntos *k-dimensionales* y que la similaridad se puede interpretar geométricamente. Un espacio vectorial permite más libertad al diseñar algoritmos de búsqueda, porque es posible usar la información de coordenadas y geometría que no está disponible en espacios métricos.

Para espacios vectoriales existen soluciones eficientes. Las más conocidas son los *Kd-trees* [Ben75, Ben79], *R-trees* [Gut84], *Quadtrees* [Sam84] y los *X-trees* [BKK96] y *TV-trees* [LJF94], que soportan consultas por similaridad, entre otros. Todas estas técnicas usan ampliamente la información de coordenadas para agrupar y clasificar puntos en el espacio. Sin embargo, éstas son muy sensibles a las dimensión del espacio vectorial. La complejidad de las búsquedas por rango depende exponencialmente de la dimensión del espacio [Cha94].

2.4.2. Diccionarios de Palabras

Para este caso, los objetos del espacio métrico son palabras en algún lenguaje específico. Para medir la distancia entre cadenas, una función posible es la conocida como *distancia de edición* o *distancia de Levenshtein*. Esta distancia se define como la cantidad de inserciones, eliminaciones o modificaciones de caracteres que hay que hacer sobre una palabra para convertirla en otra. Esta es una función de distancia discreta y tiene múltiples aplicaciones recuperación de texto, procesamiento de señales y bioinformática [Nav01].

2.4.3. Modelo Vectorial para Documentos

En recuperación de información [BYRN99] 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 representa el número total de términos de la colección. 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 j o 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

$$d_j = (w_{1j}, w_{2j}, ..., w_{ij})$$

.

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 vectores $\overrightarrow{d_j}$ y $\overrightarrow{q}$. Esta correlación puede ser cuantificada, por ejemplo, como el coseno del ángulo formado entre ambos vectores:

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

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

Los pesos de los términos pueden calcularse en 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:

$$f_{i,j} = \frac{freq_{i,j}}{\max\limits_{l=1..t}(freq_{l,j})}$$

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

Si se considera que los documentos son puntos en un espacio métrico, el problema de búsqueda de documentos similares a una consulta dada se reduce a una búsqueda por similaridad en el espacio métrico. Dado que $sim(d_j, q)$ es una medida de similaridad y no de distancia, se utiliza el ángulo formado entre los vectores $\overrightarrow{d_j}$ y $\overrightarrow{q}$, $d(d_j, q) = \arccos(sim(d_j, q))$, como función de distancia, la cual se denomina *distancia coseno* [BYRN99]. Así definido, se puede decir que el par $(\mathbb{D}, d)$ es un espacio métrico.

2.5. Dimensionalidad y Espacios Métricos

La búsqueda por similaridad en espacios métricos es más compleja cuando se realiza sobre los llamados *espacios de alta dimensión*. Muchas técnicas de indexación tradicionales para espacios vectoriales tienen una dependencia exponencial en la dimensión representacional k del espacio, es decir, a medida que la dimensión crece, dichas técnicas se vuelven menos eficientes.

Las técnicas más recientes han podido solucionar esto logrando que su complejidad dependa sólo de la *dimensión intrínseca* del espacio. Esto da a entender que existen espacios cuya dimensión representacional es alta, pero su dimensión intrínseca es baja. Un ejemplo de ésto es un plano embebido en $\mathbb{R}^{50}$, donde los tiempos de búsquedas se aproximarán más a los de un espacio de dimensión 2 que a un de dimensión 50.

La dimensión intrínseca de un espacio no es fácil de definir. Se dice que un espacio es de alta dimensión cuando su histograma de distancias es concentrado, es decir, la *media* es alta y la *varianza* pequeña, lo que indicaría que los objetos están todos más o menos a la misma distancia entre sí. En la figura 2.2 se muestra la diferencia entre histogramas de distancias para espacios de baja y alta dimensión intrínseca.

La dificultad de la búsqueda en espacios de alta dimensión radica en la baja cantidad de objetos que se pueden descartar. Un ejemplo extremo de esto es el siguiente.

Sea (X, d) un espacio métrico, donde:

- $\forall\, x,\ d(x, x) = 0$

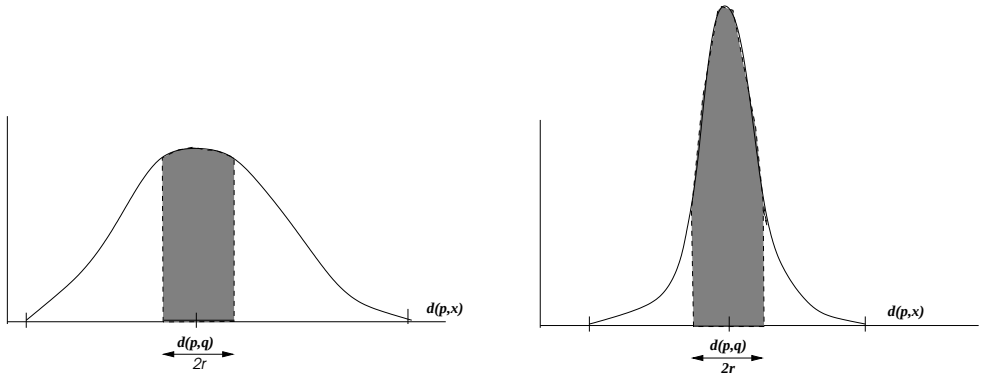


Figura 2.2: Histogramas de distancias para espacios de baja dimensión (izquierda) y alta dimensión (derecha).

- $\forall\, y \neq x,\ d(x,y) = 1$

Para una búsqueda $(q,r)_d$ en un espacio de estas características, la única información que se tiene al hacer una comparación $d(q,x)$ es si x es parte o no del resultado de la búsqueda. Sin embargo, esta información no sirve para evitar evaluaciones de distancia con los otros objetos del espacio. Es imposible, en este caso, evitar la búsqueda secuencial, independiente de la técnica de indexación utilizada.

Considerese lo siguiente, dado un espacio métrico (X,d), un elemento $p \in X$, una búsqueda $(q,r)_d$ y un índice basado en pivotes elegidos aleatoriamente. Los histogramas de la figura 2.2 representan dos posibles distribuciones de distancia de un pivote p para los valores de $d(p,q)$. La regla de eliminación permite descartar todos aquellos puntos y tales que $y \notin [d(p,q) - r, d(p,q) + r]$. La figura muestra intuitívamente el porque la búsqueda se torna más difícil cuando el histograma es más concentrado. Las áreas sombreadas representan los puntos que no podrán descartarse. Es decir, a medida que el histograma se concentra más alrededor de su media disminuye la cantidad de puntos que pueden descartarse usando como dato $d(p,q)$.

En [CNBYM01] se define la dimensionalidad intrínseca de un espacio métrico como $\rho = \frac{\mu^2}{2\sigma^2}$, donde μ y σ^2, son la media y la varianza del histograma de distancias. Esta definición es coherente con la noción de dimensión en un espacio vectorial con coordenadas uniformemente distribuidas.

2.6. Algoritmos de Indexación para Espacios Métricos

2.6.1. Modelo General de Indexación

Como se mencionó anteriormente, un algoritmo de indexación es un proceso *off-line* que construye una estructura de datos o índice, la cual permite ahorrar cálculos de distancia al momento de resolver

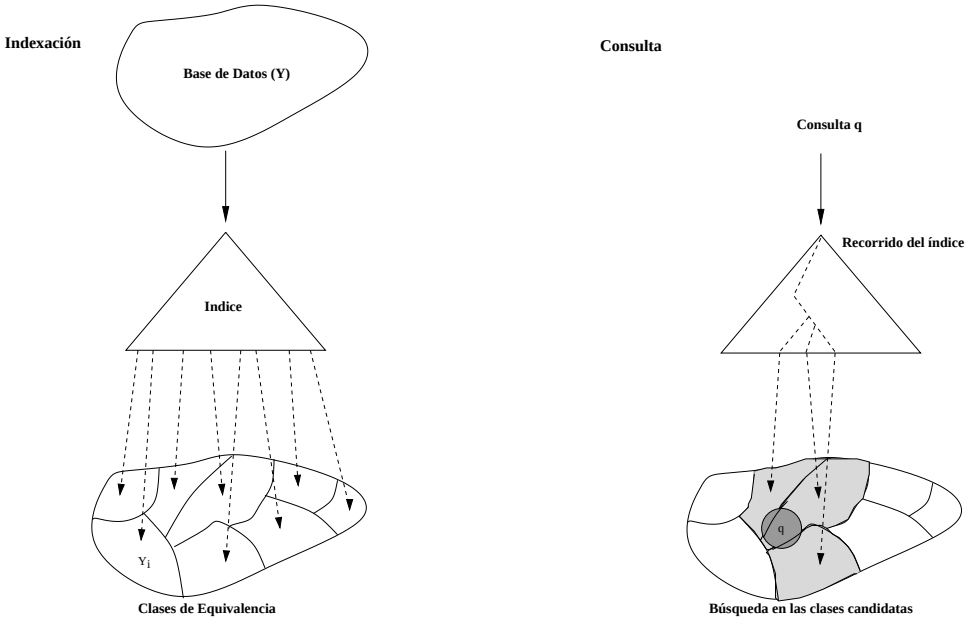


Figura 2.3: Modelo general para algoritmos de indexación. Izquierda: construcción del índice. Derecha: consulta sobre el índice.

una consulta. Entonces, lo importante es diseñar algoritmos de indexación que sean eficientes para de reducir las evaluaciones de distancia durante la búsqueda.

Los métodos para buscar en espacios métricos se basan principalmente en dividir el espacio empleando la distancia a uno o más objetos seleccionados. El no trabajar con las características particulares de cada aplicación tiene la ventaja de ser más general, pues los algoritmos funcionan con cualquier tipo de objeto.

Todos los algoritmos de indexación particionan la base de datos Y en subconjuntos. El índice permitirá determinar una lista de subconjuntos Y_i candidatos potenciales a contener objetos relevantes a la consulta. A cada subconjunto Y_i podría aplicársele el mismo método de particionamiento, en forma recursiva. Durante la búsqueda, se recorre el índice para obtener los conjuntos relevantes y luego se examina cada uno de ellos (en forma exhaustiva si no se ha continuado recursivamente la indexación). Todas estas estructuras trabajan sobre la base de descartar elementos usando la desigualdad triangular. En la figura 2.3 se muestra este modelo de indexación.

Para particionar la base de datos existen dos grandes enfoques que determinan los algoritmos de búsqueda en espacios métricos generales. Estos son: *algoritmos basados en pivotes* y *algoritmos basados en clustering o particiones compactas* [CNBYM01]. En la figura 2.4 se ilustra una clasificación [Rey02] de los distintos índices de acuerdo a sus características generales.

En la tabla 2.1 se muestra un detalle de las estructuras expuestas en la figura 2.4 con los nombres completos, la referencia a los artículos donde fueron propuestos y las abreviaturas usadas durante el

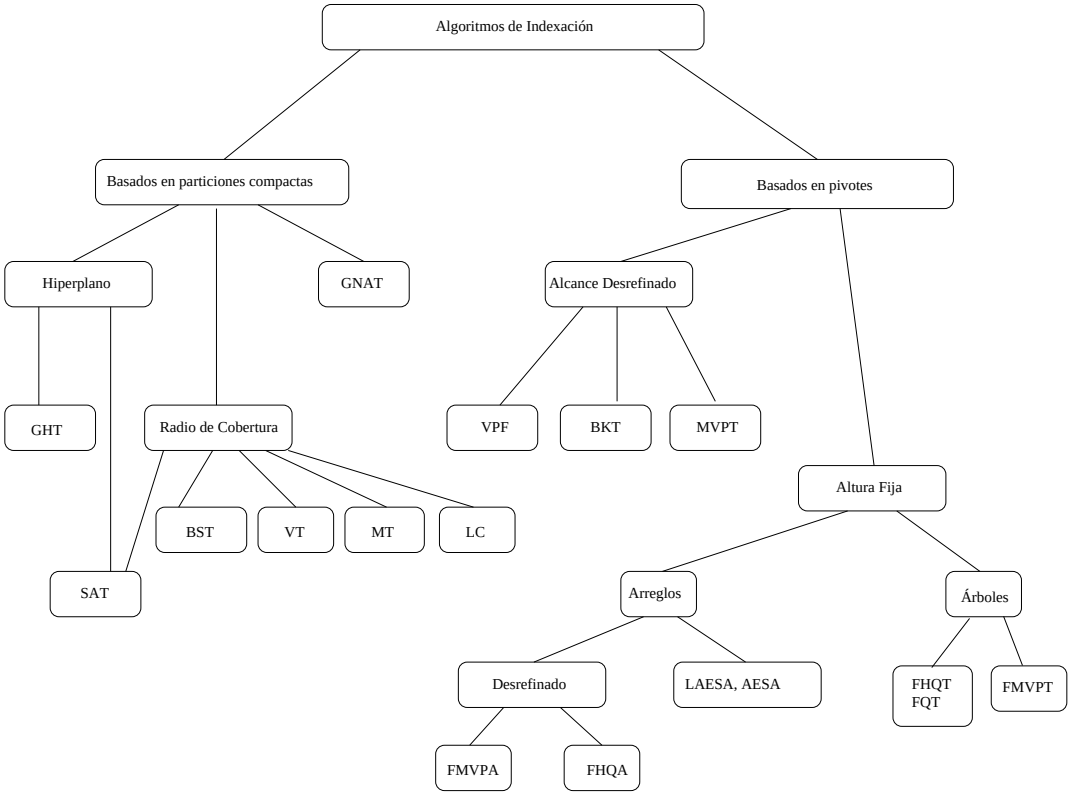


Figura 2.4: Taxonomía de los algoritmos existentes para búsquedas por similaridad en espacios métricos.

Abreviatura	Estructura	Referencia
BKT	Burkhard-Keller tree	[BK73]
FQT	Fixed Queries tree	[BYCMW94]
FQHT	Fixed Height FQ-tree	[BYCMW94, BY97, BYN98]
FQA	Fixed Queries arrays	[CMBY99, CMN01]
Metric Tree	Metric Tree	[Uhl91]
VP-tree	Vantage Point tree	[Yia93, Chi94]
MVP-tree	Multi-Vantage Point tree	[BO97]
VP-forest	Excluded Middle Vantage Point Forest	[Yia99]
BST	Bisector tree	[KM83]
GHT	Generalized-Hyperplane tree	[Uhl91]
GNAT	Geometric Near-neighbor Access tree	[Bri95]
VT	Voronoi tree	[DN87]
M-tree	M-tree	[CPZ97]
LC	List of Clusters	[CN00]
SAT	Spatial aproximation tree	[Nav02, Rey02]
AESA	Approximating Eliminating Search Algorithm	[Vid86]
LAESA	Linear AESA	[MOV94]

Tabla 2.1: Estructuras de datos para búsquedas en espacios métricos.

presente trabajo.

2.6.2. Algoritmos Basados en Pivotes

Un *pivote* es un objeto preseleccionado de la base de datos. Los pivotes sirven para filtrar objetos en una consulta utilizando la desigualdad triangular, sin medir realmente la distancia entre el objeto consulta y los objetos descartados.

- Sea $\{p_1, p_2, ..., p_k\} \in X$ un conjunto de pivotes. Se almacena para cada elemento x de la base de datos Y, su distancia a los k pivotes $(d(x, p_1), ..., d(x, p_k))$. Dada una consulta q, se calcula su distancia a los k pivotes $(d(q, p_1), ..., d(q, p_k))$.

- Si para algún pivote p_i se cumple que $|d(q, p_i) - d(x, p_i)| > r$, entonces por desigualdad triangular conocemos que $d(q, x) > r$, y por lo tanto no es necesario evaluar explícitamente $d(x, q)$. Todos los objetos que no se puedan descartar por esta regla deben ser comparados directamente con la consulta q.

Algunos algoritmos hacen una implementación directa de este concepto, y se diferencian básicamente en su estructura extra para reducir el costo de CPU de encontrar los puntos candidatos, pero no en la cantidad de evaluaciones de distancia. Ejemplos de éstos son: *AESA* [Vid86], *LAESA* [MOV94], *Spaghettis* y sus variantes [CMBY99, NN97], *FQT* y sus variantes [BYCMW94] y *FQA* [CMN01].

Las estructuras de tipo árbol utilizan esta técnica en forma indirecta. El árbol se va construyendo tomando el nodo raíz como pivote. Posteriormente el espacio se divide de acuerdo a la distancia de los objetos al pivote. Cada subárbol se construye recursivamente tomando un nuevo pivote de los elementos del subespacio. Las diferencias radican principalmente en la forma y tamaño de los espacios. La búsqueda realiza un backtrack sobre el árbol y utiliza la desigualdad triangular para minimizar los subárboles. Las estructuras que utilizan esta técnica son el *BKT* y sus variantes [BK73, Sha77], los *Metric trees* [Uhl91], *TLAESA [MOC96], VP-trees* y sus variantes [Yia93, BO97, Yia00].

2.6.3. Algoritmos Basados en Clustering o Particiones Compactas

Los algoritmos basados en clustering dividen el espacio en áreas, donde cada área tiene un *centro* o *split*. Se almacena alguna información sobre el área que permita descartarla completamente mediante sólo comparar la consulta con su centro. Los algoritmos de clustering son los mejores para espacios de alta dimensión, que es el problema más difícil en la práctica.

Existen dos criterios para delimitar las áreas en las estructuras basadas en clustering, *área de Voronoi* o *hiperplanos* y *radio cobertor* (*covering radius*). El primero divide el espacio usando hiperplanos y determina la partición a la cual pertenece la consulta según a qué zona corresponde. El criterio de radio cobertor divide el espacio en esferas que pueden intersectarse y una consulta puede pertenecer a más de una esfera.

Criterio de partición de Voronoi: Se selecciona un conjunto de puntos y se coloca cada punto restante dentro de la región con centro más cercano. Las áreas se delimitan con hiperplanos y las zonas son análogas a las regiones de Voronoi en espacios vectoriales.

- **Definición (*Diagrama de Voronoi*)**: considérese un conjunto de puntos $\{c_1, c_2, \ldots, c_m\}$ (centros). Se define el diagrama de Voronoi como la subdivisión del plano en m áreas, una por cada c_i. La q pertenece al área c_i si y sólo si la distancia euclidiana $d(q, c_i) \leq d(q, c_j)$ para cada c_j, con $j \neq i$.

Durante la búsqueda se evalúa $d(q, c_1), ..., d(q, c_m)$, se elige el centro c_i más cercano y se descartan todas las zonas c_j que cumplan con $d(q, c_j) > d(q, c_i) + 2r$, dado que su área de Voronoi no puede intersectar la *bola de consulta* con centro q y radio r. Se entiende por bola de consulta al conjunto de objetos que está a distancia máxima r de q.

Criterio de Radio Cobertor: El radio cobertor $rc(c_i)$ es la distancia entre el centro c_i y el objeto más alejado dentro de su zona. Entonces, se puede descartar la zona c_i si $d(q, c_i) - r > rc(c_i)$.

Algunas estructuras combinan estas técnicas, como el caso del *GNAT* [Bri95]. Otras sólo utilizan radio cobertor, como son los *M-trees* [CPZ97] y la *Lista de Clusters* [CN00]. Algunas que utilizan hiperplanos son GHT y sus variantes [Uhl91, NVZ92] y los Voronoi-trees [DN87, Nol89].

2.7. Estructuras Métricas y Dinamismo

El dinamismo es poco común en las estructuras de datos para espacios métricos [CNBYM01], sin embargo, algunas estructuras permiten inserciones eficientes una vez construidas. La eliminación resulta particularmente complicada, debido a que las estructuras podrían verse seriamente afectadas y habría que reconstruirlas parcial o totalmente, con el costo que conlleva.

La mayoría de las estructuras de tipo árbol no soportan eliminaciones, sobre todo en nodos internos. Esto debido a que de dichos nodos depende la organización de sus subárboles. En [CNBYM01] se muestra que las únicas estructuras que soportan completamente inserciones y eliminaciones son las de la familia *FQ* (*FQT*, *FQHT*, *FQA*) dado que no poseen nodos internos, *AESA* y *LAESA* ya que son sólo vectores de coordenadas, el *M-tree* que fue diseñado originalmente con capacidades dinámicas y una variante de los *GHTs* que soporta inserciones y eliminaciones [Ver95]. Finalmente, existe una nueva versión del SAT [NR02] que soporta capacidades dinámicas completas.

Para estructuras métricas, el problema de la eliminación no se puede solucionar utilizando la técnica de marcar el objeto como borrado para posteriormente hacer la eliminación definitiva de un conjunto de ellos. Esto, en primera instancia debido a que la calidad de la estructura se degradaría a medida que aumentan los elementos eliminados. En segundo lugar, porque en muchas aplicaciones los objetos son demasiado grandes y es inadecuado mantener utilizado ese espacio.

2.8. Estructuras Métricas y Memoria Secundaria

El problema de la implementación también en memoria secundaria no es trivial, de hecho la medida de eficiencia ya no es sólo la cantidad de evaluaciones de distancia realizadas, sino también los costos involucrados por accesos a disco. En disco, entre más secuenciales sean los accesos para realizar las búsquedas, más eficiente es el proceso, ya que tiene un costo muy elevado el realizar accesos en forma saltada dentro del archivo.

La mayor parte de estructuras para búsquedas por similaridad en espacios métricos están diseñadas para su implementación en memoria principal. En la actualidad sólo existen algunos algoritmos para espacios métricos generales diseñados para memoria secundaria. Estos son el M-Tree [CPZ97] y Slim-Tree [TTSF00], basado en M-Tree.

Una de las características principales del M-Tree es que es un árbol balanceado, sin embargo, uno de sus problemas es que está basado en radios cobertores y las áreas se intersectan, lo que provoca traslape o solapamiento (overlap) y por lo tanto menor eficiencia en la búsqueda, ya que las consultas deben recorrer más subárboles. El Slim-Tree es una estructura que reduce el grado de traslape del M-Tree.

Es fundamental encontrar nuevas estructuras y algoritmos que mejoren las búsquedas por similaridad en memoria secundaria, dado que hay muy pocas propuestas existentes. Además, este tipo de búsqueda generalmente se aplica a grandes bases de datos, ya sea multimediales, de texto o de otro tipo, lo que implica que todos los diseños pensados como prototipos para memoria RAM van a tener que ser modificados para disco.

Respecto de la eliminación, el problema se agrava dado que no sólo hay que considerar el costo de mover objetos o nodos dentro de una estructura provocado por alguna posible reorganización, sino que esto implica una serie de *escrituras* extras, y en general movimientos y lecturas adicionales en disco.

Otras consideraciones en memoria secundaria se refieren al tamaño de la estructura, la cual debe tener una correlación con la cantidad de objetos y el espacio ocupado por cada uno de ellos. Ligado a lo anterior, es importante también mantener una adecuada utilización del espacio dentro de las páginas de disco, de tal manera que estas estén en lo posible llenas.

En resumen, el diseño de una estructura métrica para memoria secundaria y con capacidades dinámicas completas requiere considerar los siguientes aspectos.

Costos de construcción: Junto con las evaluaciones de distancia, se debe considerar también los bloques leidos/escritos en disco y los movimientos de cabezales (seek).

Costos de búsqueda: Se deben considerar los cálculos de distancia, los reads y seeks en disco.

Costos de eliminación: Se deben considerar los mismos costos que para el caso de la construcción. Una especial atención requieren las escrituras en disco productos de posibles reoganizaciones de la estructura por eliminación de elementos claves para la mantención del índice.

Espacio utilizado: El tamaño completo de la estructura en disco, así como también una adecuada distribución del espacio dentro de los bloques de disco (porcentaje de nodos incompletos y promedio de uso del nodo). Se debe tomar en cuenta la reutilización de los bloques de disco disponibles producto de la eliminación de todos los datos contenidos en éstos.

Capítulo 3

Geometric Near-neighbor Access Tree y Otras Estructuras Relevantes

En este capítulo se mostrará en detalle la forma y características del ***gnat*** [Bri95], además de algunas otras estructuras que son de interés para el presente trabajo.

3.1. Características Generales

El ***gnat*** es una estructura basada principalmente en el diagrama de Voronoi, aunque igualmente usa radio cobertor. Es una generalización del *GHT* [Uhl91], el cual es construido seleccionando dos puntos clave y dividiendo el resto de los puntos de acuerdo a cuál de ellos está más cerca. Este proceso se realiza recursivamente en ambos hijos.

En el *gnat* se seleccionan k puntos claves denominados *centros* o *splits* para particionar el espacio $\{p_1, p_2, \ldots, p_k\}$, y cada punto restante es asignado al centro más cercano, definiéndose así el subárbol de influencia de cada centro. Cada subárbol es particionado recursivamente.

El objetivo del *gnat* es actuar como un modelo geométrico de los datos. Es decir, a partir del nodo raíz se obtiene una idea de los datos como espacio métrico y a medida que se avanza dentro de los subárboles, se va obteniendo una idea más acabada de la geometría de los mismos.

Para la búsqueda por rango en el *gnat*, se compara la consulta con cada centro, se determina que áreas están dentro del rango de influencia y se procede recursivamente en cada uno de esos subárboles, el resto de los centros se descarta.

3.2. Construcción del *Gnat*

La estructura *gnat* tiene la propiedad que el algoritmo de inserción original es en sí dinámico, es decir, como no es necesario conocer a priori la forma del árbol, al insertar un nuevo objeto, éste encuentra su lugar, independientemente de si la estructura está o no construida anteriormente.

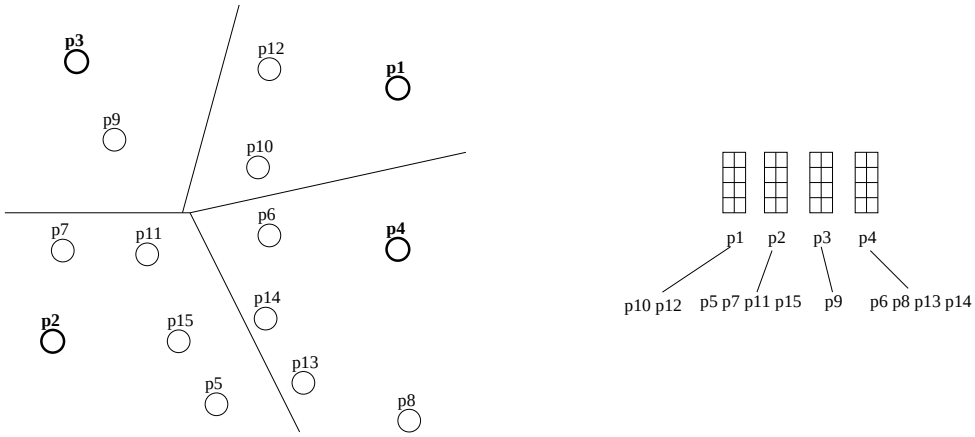


Figura 3.1: *gnat*: Construcción del árbol (aridad 4).

El algoritmo básico de la construcción del gnat es como sigue:

1. Se seleccionan k puntos (*centros*), $p_1, \ldots, p_k$ de la base de datos la cual se va a indexar.
2. Se asocia cada punto restante del conjunto de datos al centro más cercano a él. El conjunto de puntos asociados al split p_i se denota como D_{p_i}.
3. Para cada par de centros (p_i, p_j), se calcula el rango
   $range\left(p_i, D_{p_j}\right) = \left[min\_d\left\{\left(p_i, D_{p_j}\right)\right\}, max\_d\left\{\left(p_i, D_{p_j}\right)\right\}\right]$,
   la mínima y máxima distancia $d\left(p_i, x\right)$ donde $x \in D_{p_j} \cup \{p_j\}$.
4. El árbol se construye recursivamente para cada subconjunto D_{p_i}.

Cada conjunto D_{p_i} va a representar un subárbol cuya raíz es p_i, o lo que es lo mismo, cada D_{p_i} va a corresponder a la región de Voronoi cuyo centro es p_i. En la figura 3.1 se muestra un ejemplo de construcción del primer nivel de un gnat con k=4, también se muestra la tabla de rangos que debe ser almacenada para cada centro p_i. En este ejemplo se insertaron los puntos en orden al valor numérico que tienen.

Si se insertase un nuevo objeto p_{16} y quedara en el plano cuya raíz es p_4, entonces, esto crearía un nuevo nodo y haría aumentar el nivel del árbol como se muestra en la figura 3.2. Nótese que p_4 ya no es parte de estos subplanos. El algoritmo 1 muestra la inserción de un nuevo objeto sobre la estructura.

La construcción de las tablas de rangos para cada centro no implica un costo extra en evaluaciones de distancia, pero sí en almacenamiento, $\mathcal{O}\left(k^2\right)$ extra por nodo.

Usando la siguiente notación:

Algoritmo 1 *gnat*: inserción de objetos.

{Sea k el número máximo de centros por nodo}
inserta(Nodo P, TipoObjeto *dato*)

1: {Sea T el conjunto distancias al dato}
2: $T \leftarrow \emptyset$
3: **for all** $p_i \in P$ **do**
4: $T_i \leftarrow dist(p_i, dato)$
5: **end for**
6: **if** $|P| < k$ **then**
7: agrega dato a P
8: se actualizan los rangos
9: **else**
10: se cálcula la mínima distancia y se obtiene el p_{min}
11: se actualizan los rangos a p_{min}
12: inserta($D_{p_{min}}$, dato)
13: **end if**

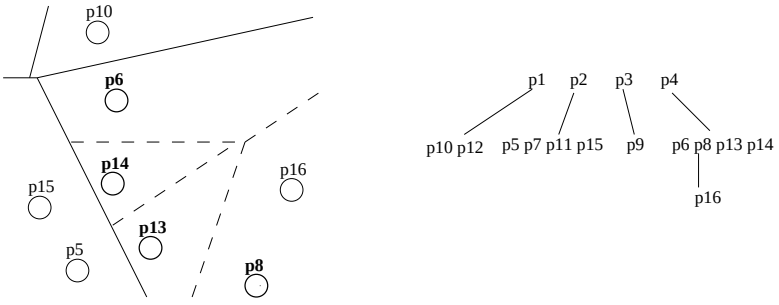


Figura 3.2: *gnat*: Inserción de un nuevo objeto.

- N : número de objetos.
- n : número de nodos.
- s : tamaño máximo requerido para almacenar un objeto (espacio de memoria).
- h : máximo grado de un nodo.
- k : grado promedio (N/n).
- k_2 : promedio de los k^2.
- l : profundidad promedio de un objeto dentro del gnat.

El espacio total requerido para almacenar la estructura es $\mathcal{O}(nk_2 + Ns)$, donde k_2 en la práctica no es mucho mayor que k^2.

Una de las principales desventajas del gnat comparado con otras estructuras es el tiempo de preprocesamiento. En un escenario ideal, con un árbol completamente balanceado toma $\mathcal{O}(Nk \log_k(N))$ evaluaciones de distancia. En la realidad, esto puede ser bastante más dependiendo del grado de los nodos, esto tomaría $\mathcal{O}(Nlh)$ cálculos de distancia. Obviamente, no todos los nodos van a tener grado h, y l tiende a ser un poco mayor a $\log_k(N)$.

La elección de los *centros* durante la construcción, resulta ser un tema interesante desde el punto de vista de las estructuras métricas, sin embargo, esto no es parte de los objetivos de esta tesis. Por lo tanto, para cada una de las estructuras mencionadas dicha elección será aleatoria.

3.3. Búsqueda en el *Gnat*

Una búsqueda en un *gnat* se realiza recursivamente como sigue:

1. Se supone que se desea buscar todos los puntos con distancia $d \leq r$ a la consulta q. Sea P la representación del conjunto de centros del nodo actual (inicialmente la raíz del *gnat*) el cual posiblemente contiene un vecino cercano a q. Inicialmente P contiene todos los puntos centros del nodo actual.
2. Se toma un punto p en P, se calcula la distancia *d(q,p)*. Si $d(q,p) \leq r$, se agrega p al resultado.
3. $\forall\, x \in P$, si $[d(q,p) - r, d(q,p) + r] \cap range(p, D_x)$ es vacío, entonces se elimina x de P.
4. Se repiten los pasos 2 y 3 hasta procesar todos los puntos en P.
5. Para todos los puntos $p_i \in P$, se procede recursivamente sobre D_{p_i}.

Algoritmo 2 *gnat*: búsqueda por rango r para la consulta q.

busquedarango(Nodo P, Consulta q, Rango r)

1: {Sea R el conjunto resultado}
2: $R \leftarrow \emptyset$
3: $d \leftarrow dist(p_0, q)$
4: **if** $d <= r$ **then**
5: se reporta p_0
6: **end if**
7: $range(p_0, q) \leftarrow [d - r, d + r]$
8: **for all** $x \in P$ **do**
9: **if** $range(p_0, q) \cap range(p_0, D_{p_x}) \neq \emptyset$ **then**
10: se agrega x a R
11: **if** $dist(x, q) <= r$ **then**
12: se reporta x
13: **end if**
14: **end if**
15: **end for**
16: **for all** $p_i \in R$ **do**
17: busquedarango(D_{p_i},q,r)
18: **end for**

El algoritmo 2 representa la búsqueda por rango sobre el *gnat*.

La razón que permite descartar subárboles durante la búsqueda en el paso 3 del algoritmo es lo siguiente:

Sea un punto $y \in D_x$. Si $d(y, p) < d(q, p) - r$, entonces, por desigualdad triangular, se tiene que $d(q, y) + d(y, p) \geq d(q, p)$, por lo que se deduce que $d(q, y) > r$. Análogamente, si $d(y, p) > d(q, p) + r$, se puede usar desigualdad triangular $d(y, q) + d(q, p) \geq d(y, p)$, para deducir $d(q, y) > r$ (figura 3.3).

Es interesante notar que para descartar los subárboles durante la búsqueda, el gnat usa un criterio distinto a los mencionados de hiperplano y radio cobertor. Este limita la partición asociada a un centro intersectando anillos alrededor de otros centros, lo que es similar a las técnicas usadas en los algoritmos basados en pivotes.

3.4. *Gnat* y Radio Cobertor

La construcción del gnat puede modificarse para evitar el espacio extra usado por la tabla de rangos. Para ello, puede construirse una versión simplificada del gnat, la cual almacenará sólo el radio cobertor de cada centro, en reemplazo de la tabla. Esto equivale a almacenar sólo $range\left(p_i, D_{pi_j}\right) = [0, rc(P_i)]$.

En la figura 3.4 se ilustra la forma del espacio para el gnat usando radio cobertor (usando los mismos puntos que los de la figura 3.1), las líneas rectas representan los planos de Voronoi para cada centro p_i, y los círculos representan los radio cobertores de cada centro. De la ilustración se

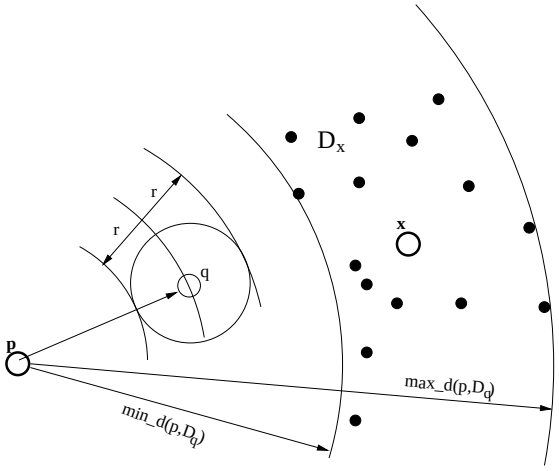


Figura 3.3: Descarte de subárboles usando rangos. Se descarta D_x ya que $d(q,p)+r < min\_d(p, D_x)$.

puede deducir que al agregar puntos nuevos cercanos a las fronteras de los planos, implicaría un aumento en los radios cobertores, por lo que habría mayor intersección y por ende mayor costo al momento de la búsqueda. En el capítulo 5 se muestran experimentalmente las diferencias entre estas dos implementaciones y cómo se ve afectado el rendimiento de las búsquedas para radio cobertor.

3.5. Otras Estructuras de Interés

Las siguientes estructuras son descritas por su relevancia a los temas centrales de la presente tesis, es decir, dinamismo y memoria secundaria.

3.5.1. *M-tree*

La primera estructura de datos métrica dinámica que apareció en la literatura fue el *M-tree* [CPZ97]. Además, esta estructura fue diseñada para un almacenamiento eficiente en memoria secundaria.

El M-tree es un árbol balanceado y paginado, es decir, usa páginas de tamaño fijo con cantidad de nodos u objetos variables. Además, es dinámico ya que permite inserciones y eliminaciones una vez construido sin degradar la búsqueda.

Esta estructura posee cierta similitud con el gnat. El árbol se construye eligiendo un conjunto de centros, los cuales almacenarán su radio cobertor. Cada nuevo objeto se inserta en el mejor subárbol, el cual se define como aquél en donde crece menos su radio cobertor. Si más de un centro cumple con esta condición, existen entonces distintas heurísticas que se aplican. Una es insertar el nuevo objeto a su centro más cercano, otra, la más usada, es elegir aquel centro cuyo subárbol posee menos datos, de esta manera se compensan los subárboles. Se procede recursivamente hasta insertar el objeto en

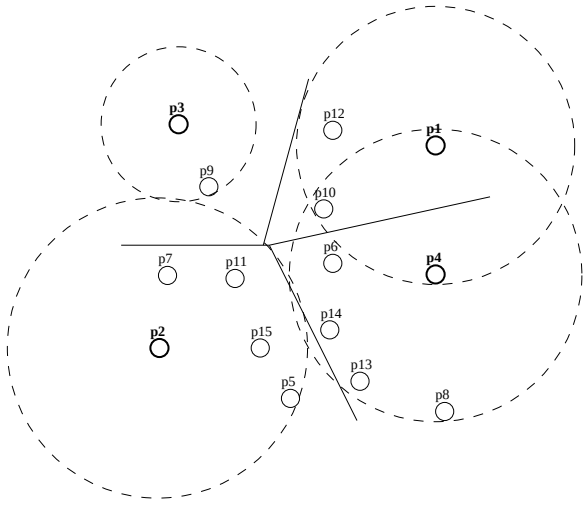


Figura 3.4: *gnat* con radio cobertor.

una hoja. Si el nodo rebalsa, la hoja se divide en dos y uno de sus objetos sube un nivel en el árbol, el cual recursivamente puede rebalsar. Como el árbol crece hacia arriba, es balanceado.

El M-tree posee dos tipos de nodos: los nodos internos y los nodos hojas. Los primeros mantienen la ruta a los objetos y son llamados *objetos ruteadores* (*routing objects*) y las hojas almacenan todos los objetos indexados.

La información general almacenada para un objeto ruteador es:

- O_r: objeto ruteador (o valor característico).
- $T(O_r)$: subárbol de O_r
- $ptr(T(O_r))$: puntero a la raíz de $T(O_r)$
- $rc(O_r)$: radio cobertor de O_r
- $d(O_r, P(O_r))$: distancia de O_r a su padre

Para los nodos hojas se almacena:

- O_j: objeto almacenado (o su valor característico).
- $Oid(O_j)$: identificador del objeto O_j
- $d(O_j, P(O_j))$: distancia de O_j a su padre

En la figura 3.5 se ilustra la construcción de un M-tree. En ella se puede ver cómo se intersectan las esferas (radios cobertores), siendo esto es lo que provoca el solapamiento o traslape de los planos.

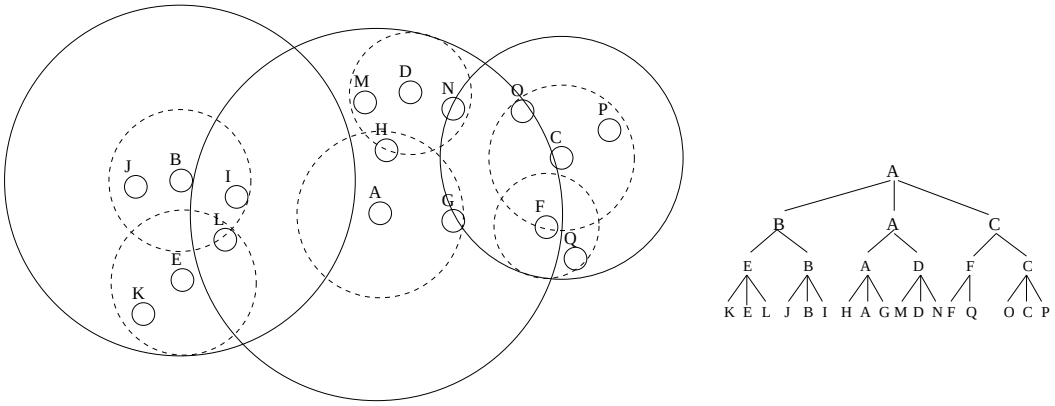


Figura 3.5: *M-tree*: Construcción de la estructura.

En otras palabras, al insertar un objeto, éste puede tener varias alternativas de subárboles donde ser insertado. Esta es una de las diferencias radicales con el gnat, en donde un nuevo objeto sólo tiene una única posición dentro del árbol, con la sola excepción del caso en que la distancia de un objeto es la misma a dos o más centros. Esta característica del M-tree hace que los datos se distribuyan mejor dentro del árbol y y pueda balancearse fácilmente. Sin embargo, una de sus desventajas es que incluso durante una búsqueda exacta necesitaría recorrer más de un camino para encontrar un objeto, lo que no ocurre con el gnat. En este sentido una estructura basada en el M-tree que reduce el grado de traslape es el Slim-tree [TTSF00].

Durante la búsqueda, se utiliza el radio cobertor para descartar zonas completas.

En general, los métodos de inserción y eliminación son parecidos a los de un B-tree, sin embargo, respecto de la eliminación existe poca o ninguna información disponible. La implementaciones originales fueron construidas usando un paquete llamado *GiST* (*Generalized Search Tree)* [HNP95], el cual tiene problemas para borrar.

3.5.2. *VoronoiTree*

El *Voronoi*-tree *o VT* [DN87] fue elegido por ser basado en clustering y por su parecido al M-tree y al gnat. El *VT* es estático, pero permite inserciones eficientes posteriores a su construcción.

Esta estructura se basa en el BST [KM83], el cual es un árbol binario que tiene dos centros, los objetos restantes se insertan al centro más cercano. El VT es un árbol ternario que va replicando el padre. Esto le otorga un mejor balance que el BST y le da la propiedad de que el radio cobertor se va reduciendo a medida que se baja en el árbol.

El algoritmo de construcción del VT se diferencia con el del gnat por lo siguiente:

- Si al insertar un nuevo objeto, se debe crear un nuevo nodo, entonces, este nodo tendrá al objeto y al elemento más cercano a él que pertenece al nodo padre (redundancia), de esta

manera cada raíz se va duplicando en su subárbol .

Cada centro almacena su radio cobertor, el cual utiliza para excluir subárboles durante la búsqueda, es decir, si $d(q, c_i) - rc(c_i) > r$, entonces, se puede descartar el subárbol de c_i.

La versión implementada para el presente trabajo es una generalización del VT, es decir, se eligen k centros para ir construyendo el árbol.

Como ejemplo de esto, supongamos el mismo espacio representado en la figura 3.1 sin los puntos p_{14} y p_{15}. El VT quedaría como se muestra en la figura 3.6.

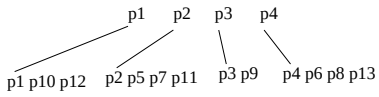


Figura 3.6: *Voronoi-tree*: Construcción de la estructura.

Si se agregara ahora p_{14} el plano se vería como en la figura 3.7. Aquí es importante notar que dentro del área de p_4 del nodo raíz del árbol, también hay una subárea con centro p_4.

3.5.3. Spatial Aproximation Tree

El Árbol de Aproximación Espacial o *SAT* es una estructura propuesta en [Nav99, Nav02] que se basa en un concepto diferente respecto de la mayoría de las estructuras métricas. La idea general, más que dividir el espacio de búsqueda, es aproximarse espacialmente a la consulta. Para ello se utilizan las búsquedas del vecino más cercano como una forma natural de aproximarse a la consulta.

En este modelo, dado un punto $q \in X$ (conjunto de objetos válidos del espacio métrico) y estando posicionado en algún elemento $a \in Y$ (base de datos) con $Y \subseteq X$, el objetivo es moverse a otro elemento de Y que esté más cerca espacialmente de q que a.

Las aproximaciones son realizadas sólo a través de vecinos. Cada elemento a de la base de datos tiene un conjunto de vecinos $N(a)$. La forma natural de presentar esta estructura es mediante un

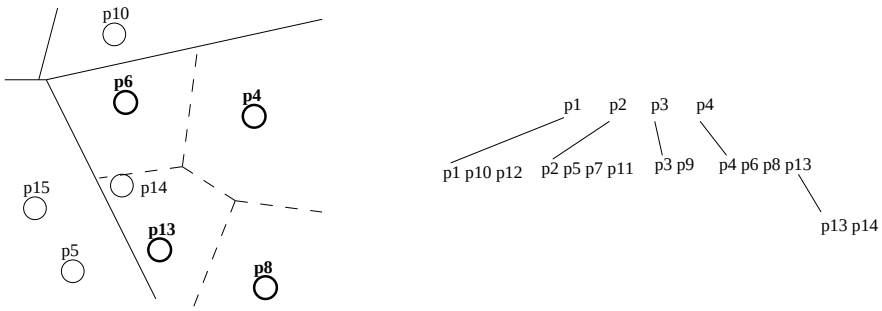


Figura 3.7: *Voronoi-tree*: Inserción de un nuevo objeto.

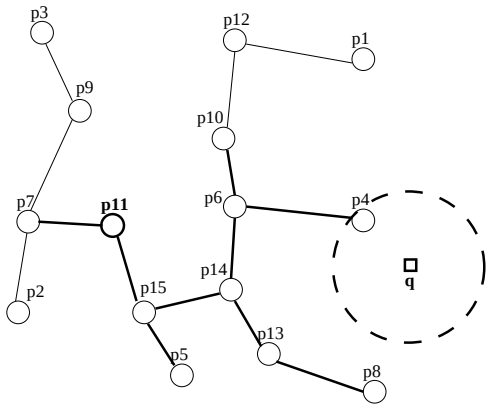


Figura 3.8: SAT: búsqueda por rango.

grafo dirigido, donde los nodos representan los objetos y se conectan a sus vecinos por medio de arcos. Para la búsqueda se elige arbitrariamente un nodo sobre el cual comenzar. Si ningún nodo está más cerca de q que a, entonces la respuesta es a. De otro modo se elige un nodo b vecino de a que esté más cerca de q que a, luego continúa la búsqueda desde b. El grafo debe tener suficientes aristas para garantizar que este método de búqueda funcione.

Construcción:

Para obtener el SAT se realiza un simplificación a esta idea, comenzando la búsqueda siempre por un nodo fijo, imponiendo la condición sólo para los $x \in Y$. De esta manera se reduce la estructura a un árbol. Para que la búsqueda funcione para todo $x \in X$ se debe hacer backtraking en el árbol. El árbol se define recursivamente, donde cada nodo a cumple que todos sus hijos están más cercanos a a que a cualquier otro hijo (originalmente a es la raíz).

Durante la búsqueda se toma el vecino más cercano c a q entre $a \cup N(a)$. Así, se puede descartar todos los vecino $b \in N(a) \,/\, d(q,b) > d(q,c) + 2r$. Para podar más la búsqueda se utiliza el radio cobertor $rc(a)$, no entrando en los subárboles tales que $d(q,a) > rc(a) + r$, siendo r el radio de búsqueda. En la figura 3.8 se muestra un ejemplo de búsqueda por rango, partiendo desde la raíz p_{11}, y recorriendo el árbol (arcos resaltados) hasta encontrar p_4 como respuesta a la consulta.

En su concepción original el SAT era esencialmente estático y no permitía inserciones posteriores en forma eficiente. En [NR02] se propone una versión completamente dinámica de esta estructura y en [Rey02] se muestran en detalle distintas alternativas de inserciones y eliminaciones sobre el árbol.

El SAT es una estructura cuya construcción necesita conocer previamente todos los elementos de Y. Al insertar un nuevo elemento, se debe bajar en el árbol siguiendo en cada paso el vecino más cercano hasta que el nuevo elemento deba volverse un vecino más del nodo corriente a, es decir, que el nuevo elemento este más cerca de a que de $N(a)$. Todo subárbol cuya raíz es a se debe reconstruir completamente, dado que algunos nodos que entraron por otro vecino podrían ahora preferir entrar

en el nuevo vecino.

En la versión dinámica se define una aridad máxima y la construcción se hace en forma incremental, almacenando para cada objeto un *timestamp*, que corresponde al orden de inserción de este objeto, con el propósito de usar este valor durante la construcción, búsqueda y eliminación. Al insertar un nuevo elemento x, se agrega como vecino al punto a más apropiado si la aridad de este nodo no es máxima, de ser así, se obliga a continuar por algún $N(a)$ que cumpla con las condiciones. En este punto se agrega x al final de la lista de vecinos de $N(a)$, se le coloca el *timestamp* actual y se incrementa el *timestamp* actual.

Durante la construcción, los nodos hijos de a y vecinos entre sí, tienen *timestamps* crecientes de izquierda a derecha, además se cumple que el padre siempre tiene *timestamp* más antiguo que sus hijos. Se debe notar que al insertar un nuevo objeto x no se sabe realmente si éste es vecino del primer nodo a que satisface que es más cercano a x que cualquier nodo de $N(a)$. Es posible que a haya tenido aridad máxima y por lo tanto el nuevo objeto fuera forzado a ser vecino de algún $N(a)$, lo que tendría implicancias durante la búsqueda.

En la figura 3.9 se muestra el proceso de construcción del SAT, antes y después de insertar un nuevo objeto. En esta figura la aridad máxima del árbol es 3. S se indican también los *timestamp* de cada elemento, obsérvese que al nuevo elemento x se le asigna el último valor del *timestamp*.

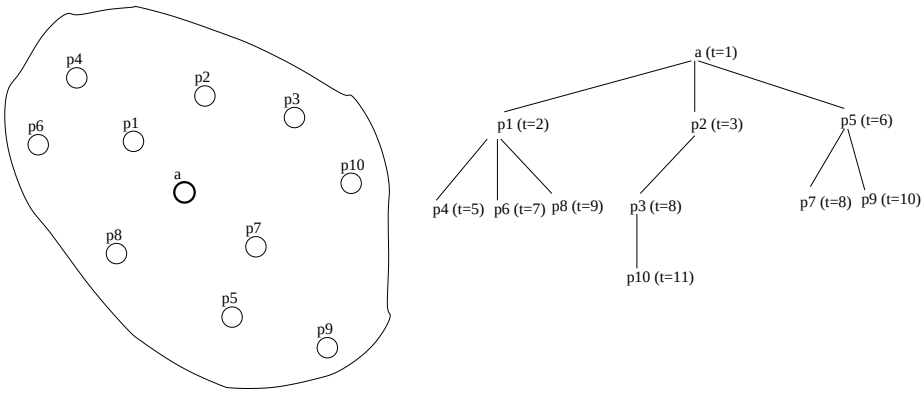
Búsqueda:

Durante la búsqueda se utilizan los *timestamps* para reducir el trabajo dentro de los vecinos más antiguos. Supóngase que $d(q, v_i) > d(q, v_{i+j}) + 2r$, entonces se tiene que entrar a v_i debido a que es más viejo. Sin embargo, sólo los elementos con *timestamp* más pequeño que el de v_{i+j} se deberían considerar cuando se busca dentro de v_i. Los elementos más nuevos ya se han comparado con v_{i+j} y por lo tanto no pueden estar dentro de v_i. Como los nodos padres son más viejos que sus descendientes, tan pronto como se encuentra un nodo dentro del subárbol de v_i con *timestamp* más grande que v_{i+j} se puede detener la búsqueda en esa rama, debido a que su subárbol es aún más joven.

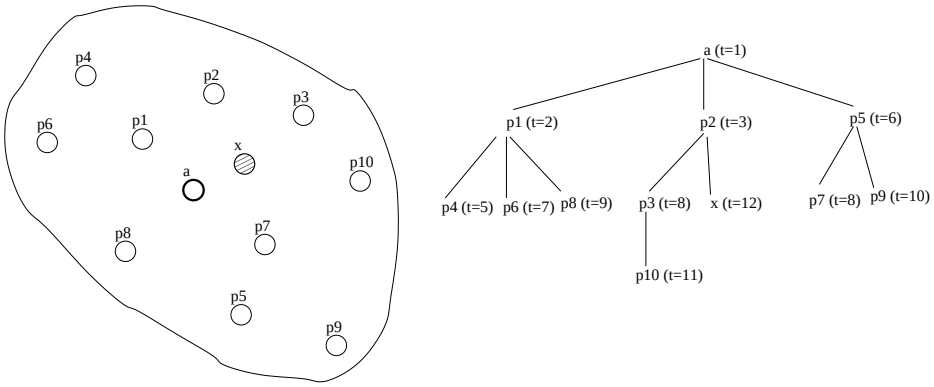
Eliminación:

La eliminación en el SAT se hace combinando dos métodos [NR02]. El primero de ellos es denominado *Nodos Ficticios*. En éste la idea principal es eliminar físicamente el objeto dejando su nodo dentro del árbol, de esta manera los costos de eliminación son bajos. El problema de este algoritmo es llevar una búsqueda consistente cuando un nodo no tiene un objeto. Para ello, durante las búsquedas, si se debe consultar por el objeto que se ubica en una posición indicada como ficticia, entonces se agrega su subárbol a la búsqueda, salvo que pueda ser descartado usando su *timestamp*. De todas maneras este método por sí solo aumenta notablemente los costos durante la búsqueda.

El otro método es reconstruir el subárbol del padre del nodo x eliminado, dejando el árbol tal como hubiese quedado si el nodo x eliminado nunca hubiese existido. Para este algoritmo se



(a) árbol



(b) Inserción de un nuevo objeto.

Figura 3.9: *SAT*: Instante en la construcción del árbol, antes (a) y después de insertar un nuevo objeto x (b).

recuperan del subárbol del padre de x todos los elementos cuyo *timestamp* sea mayor que el de x y se le mantienen sus *timestamps*, se los reinserta uno a uno en el árbol en orden decreciente de *timestamp*. También se puede usar la información de los *timestamps* para optimizar el proceso de reinserción.

Supóngase que x será eliminado, y sea $A(x)$ el conjunto de ancestros de x, es decir, todos lo nodos en el paso desde la raíz a x. Para cada nodo c que pertenece al subárbol con raíz x se tiene que $A(x) \subset A(c)$. Así, cuando se insertó el nodo c , se comparó con todos los vecinos en $A(x)$ cuyo *timestamp* era menor que el de c. Usando esta información se puede evitar evaluar las distancias a esos nodos cuando se visiten nuevamente al reinsertar c. Es decir, al buscar el vecino más cercano a c se sabe que el que está en $A(x)$ siempre es el más cercano, salvo que algún vecino sea más joven, en este caso también tendría que considerarse. Esto se hace funciona a medida que el camino para

reinsertar c este dentro de $A(x)$.

Se combinan las dos técnicas para compensar lo costoso de la eliminación con reconstrucción y la pérdida de eficiencia en la búsqueda por exceso de *nodos ficticios*. Para el espacio de palabras se obtiene un buen balance con una cantidad de nodos ficticios del orden de un 1 %, es decir, si un subárbol ha sobrepasado este porcentaje de nodos ficticios se lo reconstruye reinsertando sólo los elementos no ficticios.

Capítulo 4

Eliminación en el *Gnat*

Este capítulo tratará principalmente el problema de la eliminación sobre estructuras métricas, de la propuesta para la estructura *gnat*, de las características dinámicas aplicadas y de sus implicancias sobre la estructura.

4.1. Consideraciones en la Eliminación

El objetivo básico es poder tener una estructura que ofrezca total dinamismo y a su vez mantener la eficiencia en las consultas, inserciones y eliminaciones. En la definición del proceso de eliminación se deben tomar en cuenta ciertas premisas importantes, entre éstas:

1. Después de la eliminación, la estructura debe mantener las mismas características de antes, es decir, ser un gnat, o contener gnats, lo que permitirá inserciones, búsquedas por rango y exactas.

2. No degradar en demasía la eficiencia en las búsquedas, o permitir sólo un determinado aumento en el costo de búsqueda.

Se ha desechado desde el inicio la opción de marcar el elemento o nodo como borrado sin su eliminación física. Sin embargo, es posible mantener el nodo sin el objeto, un ejemplo de esto fue propuesto en [NR02] manteniendo *nodos ficticios*, para una versión dinámica de la estructura *SAT* (descrita en la sección 3.5.3). El problema de este método es que al haber nodos donde falten algunos objetos, no se puede realizar comparaciones, y por lo tanto habría que incluir dichos subárboles durante todas las búsquedas. Sin embargo, el problema mayor se produce durante la inserción, debido a que se presentan más de una alternativa donde insertar el objeto. Este método aplicado al gnat implicaría quebrar la forma de su construcción, es decir, provocaría solapamiento de planos.

Descartando inicialmente el caso en que el dato se encuentra en una hoja, lo cual no ofrece complicación en la eliminación y su dificultad radica principalmente en el manejo de memoria secundaria, se analizará el caso general, es decir, cuando el objeto se encuentra en un nodo interno del árbol.

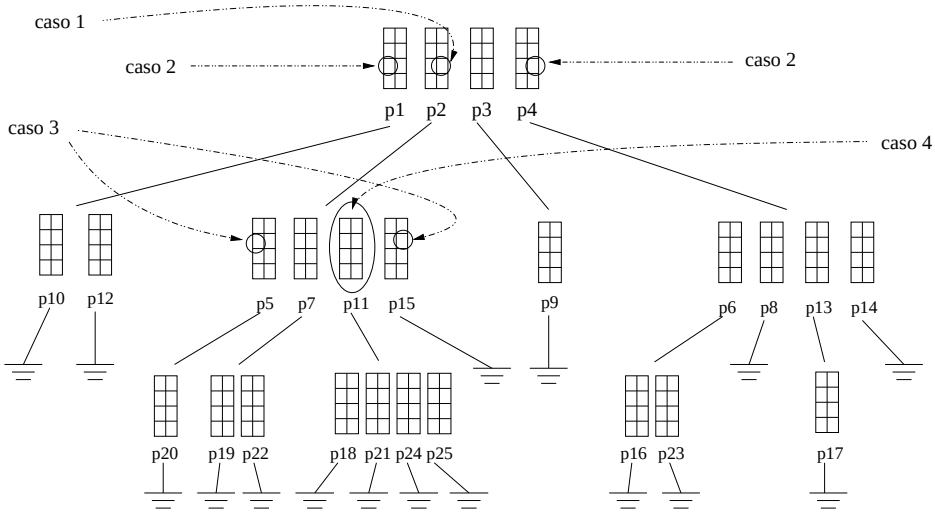


Figura 4.1: Casos problemas en la estructura gnat al eliminar el elemento p_{11}.

Existe una complejidad notoria en la eliminación de un dato en la estructura gnat. Hay que recordar que cada vez que se inserta un dato, éste es comparado con cada uno de sus ancestros y los hermanos de sus ancestros (ancestros son los centros de cada superzona a la que pertenece el objeto, es decir, todos los nodos que forman el camino desde la raíz hasta el objeto), y con sus propios hermanos, esto con el objetivo de obtener los rangos entre los centros y los subárboles adyacentes. Por lo tanto, durante la eliminación es muy posible que los rangos queden sobredimensionados, ya sea afectando al rango mínimo o máximo. Esto podría afectar a la estructura de varias formas:

1. Modificando los rangos de los planos de uno o más ancestros al que pertenece el objeto. Es decir, podría modificar el radio cobertor de uno o varios de sus padres.

2. Los rangos de los hermanos de dichos ancestros hacia el plano al cual pertenece el objeto.

3. Los rangos de los hermanos del objeto eliminado hacia el subárbol cuya raíz es el objeto.

4. Los rangos originales almacenados para el subárbol cuya raíz es el objeto eliminado ya no corresponderían. Dichas distancias fueron calculadas en base al objeto, por lo tanto las distancias almacenadas tanto para su radio cobertor como para los rangos a sus planos adyacentes (hermanos) corresponden a información erronea o que no se puede utilizar.

La figura 4.1 representa una estructura gnat incluyendo en el dibujo las tablas de rangos para cada uno de los centros. Esta figura muestra como es afectada la estructura producto de la eliminación del elemento p_{11}. Este es un ejemplo extremo, donde la eliminación del objeto genera todos los problemas mencionados anteriormente. Cada uno de los cuatro casos está representado por un círculo u óvalo.

En esta figura se puede ver, entre otros, como es afectado el radio cobertor del área cuyo centro es p_2 y el rango máximo del centro p_4 al conjunto cuyo centro contiene al elemento borrado (D_{p_2}).

4.2. Algoritmos de Eliminación

4.2.1. Reconstrucción de Subárboles

Cada vez que se elimina un elemento, el árbol debería mantener la estructura en su forma original, es decir, que tenga todas las propiedades de un *gnat*. Ello se consigue reconstruyendo el árbol.

Esto se podría conseguir usando un objeto cualquiera como reemplazo y recalculando todas las distancias a partir del nodo donde se encontraba el objeto eliminado (subárbol afectado). Sin embargo, esto tendría el mismo costo que la reconstrucción del subárbol por reinserción de todos sus descendientes.

La reconstrucción del subárbol puede ser a partir de la raíz o desde el nodo afectado, teniendo la primera la desventaja de aumentar las evaluaciones de distancia, pero con la garantía de que no existirían centros con rangos sobre dimensionados.

La necesidad de reinsertar todos los elementos del subárbol, es para mantener la coherencia de los rangos. Se deben incluir en la reconstrucción los subárboles adyacentes al centro eliminado. Esto debido a que es posible que un objeto de un área adyacente podría estar ahora más cerca del centro que ocupa el lugar del eliminado.

Finalmente este método resulta extremadamente costoso debido a la cantidad de evaluaciones de distancia por cada elemento a eliminar.

4.2.2. *Planos Fantasmas*

Esta idea consiste en reemplazar el objeto eliminado por otro que ocupe su lugar en el nodo, es decir, el nuevo objeto sería el centro del plano y conservaría los mismos rangos del elemento eliminado. Como no hay recálculo de rangos, la estructura cambiaría de forma a partir de este nodo produciéndose solapamiento (*overlap*) de las áreas, lo cual implica un nuevo problema a considerar en los métodos de inserción, eliminación y búsqueda.

La elección del objeto de reemplazo resulta interesante, porque dependiendo de la ubicación del elemento, el árbol puede resultar más o menos afectado. La alternativa general es elegir algún dato en el subárbol, de esta manera sólo el subárbol es afectado y el resto del árbol permanece intacto y conserva todas las propiedades de un *gnat*. Para entender mejor las distintas propuestas considere la figura 4.2 donde se muestra un área *P(S)*, cuyo centro es *S*. Las divisiones internas corresponderan a las subáreas interiores de *S*.

Este método se denomina *planos fantasmas* debido a que bajo un mismo subárbol coexitirán dos planos. Uno fantasma, cuya raíz es el objeto eliminado y sus hijos son aquellos que lo eran antes de eliminar dicho objeto. El otro plano, corresponde al plano real cuya raíz es el objeto que reemplazo

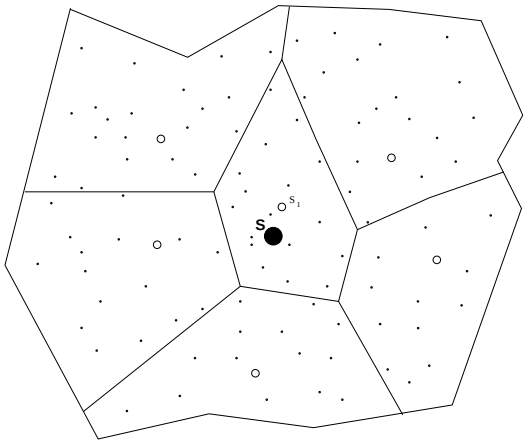


Figura 4.2: *gnat*: Partición del espacio, representación de subplanos.

al eliminado y cuyos hijos son verdaderamente los más cercanos a la nueva raíz. Algunos de los objetos del plano fantasma no pertenecerán al plano real. Durante la inserción, los nuevos objetos se compararán con el centro real.

4.2.2.1. Reemplazo por el Descendiente más Cercano

Una solución natural es elegir el elemento más cercano que sea *descendiente* del objeto eliminado, es decir, dentro de su subárbol. Esta alternativa se representa a través del siguiente algoritmo :

1. al encontrar el dato a eliminar, se marca el nodo y el centro como *afectado*, con el objeto de considerarlo en las búsquedas.

2. se busca el objeto más cercano al dato a eliminar (entre sus descendientes) (línea 5, algoritmo 3) .

3. se elimina el centro definitivamente al reemplazarlo por el objeto encontrado.

4. se conservan los rangos del conjunto del centro original.

5. se repite el proceso recursivamente sobre el subárbol del dato usado como reemplazo, hasta dejar el espacio eliminado en una hoja.

Supongamos que se desea eliminar el objeto S_1 de la figura 4.2, el cual es hijo de S y centro de su propia área. Con el algoritmo planteado, se elegiría el elemento señalado con la flecha en la figura 4.3, el más cercano a S_1. Con esto el plano sufriría un corrimiento como se muestra en la figura 4.3. Llamaremos plano fantasma a $P(S_1)$ antes de su eliminación, el cual es representado en la misma figura por líneas discontinuas.

Algoritmo 3 *gnat*: eliminación de un objeto q (alternativa 4.2.2.1).

eliminaobjeto(Nodo P, TipoObjeto q)

1: {Sea N el nodo que contiene al más cercano a q}
2: {Sea $N_{min} \in N$ el centro más cercano a q}
3: {Sea D_q subárbol con centro q}
4: búsqueda del elemento a eliminar
5: $N \leftarrow$ buscareemplazo(D_q,q)
6: se reemplaza q por N_{min} {se borra q definitivamente}
7: se marca como afectado $P_{N_{min}}$
8: **if** $N \neq hoja$ **then**
9: eliminaobjeto(D_N, N_{min})
10: **else**
11: borra $N_{min} \in N$ {definitivamente}
12: **end if**

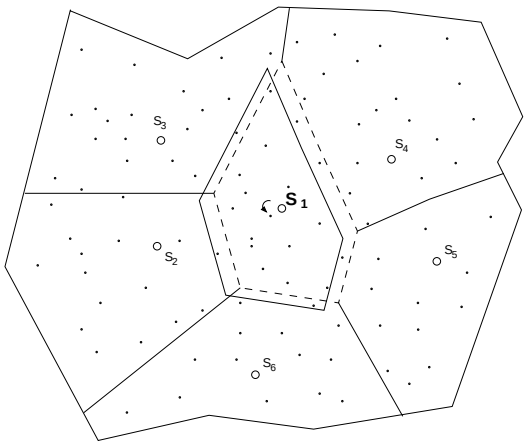


Figura 4.3: *gnat*: eliminación de un *centro*, vista de un *plano fantasma*.

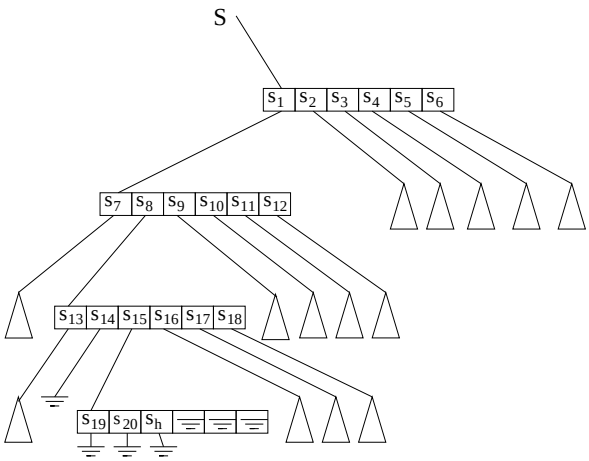


Figura 4.4: *gnat*: Estructura original, sin elementos eliminados.

La elección del más cercano a S_1 no garantiza que el plano no sufra un corrimiento, por lo tanto, es posible que varios puntos queden fuera del área real formado por el nuevo centro, y otros de áreas adyacentes queden dentro.

Como el proceso es recursivo, es muy probable que subplanos interiores sufran el mismo efecto, y por lo tanto, para una eliminación se crearían varios planos fantasmas dentro de un subárbol. Un caso extremo de esto es la eliminación de un objeto ubicado en la raíz.

Para visualizar cómo es afectada la estructura después de eliminar un split, considere la figura 4.4, la cual representa un árbol que no contiene datos eliminados. Esta figura podría representar el plano indicado en la figura 4.3 antes de eliminar objetos.

Si el centro eliminado es S_1, siendo el más cercano a él S_{15}, entonces la estructura quedaría según se representa en la figura 4.5, donde el asterisco representa la marca de *afectado*, tanto en el nodo como en el centro. El elemento S_h es el ubicado en una hoja que sería el último en moverse.

Es importante notar que, dada la eliminación de un sólo elemento, el árbol puede verse afectado en varios planos o subárboles interiores, dependiendo de la cantidad de niveles y la ubicación de los más cercanos.

4.2.2.2. Reemplazo por el más Cercano en el Nodo

Una segunda alternativa, sería el reemplazo del elemento a eliminar por el objeto más cercano, pero elegido dentro de todos los subárboles que salen del nodo. De esta manera, la superposición de planos sería mínima, sin embargo, esto podría afectar además a subárboles adyacentes, lo que deformaría aún más el *gnat*. Para lo anterior se modifica la línea 5 del algoritmo 3 pasando como parámetro todo el nodo al que pertenece q, y no sólo el hijo de q.

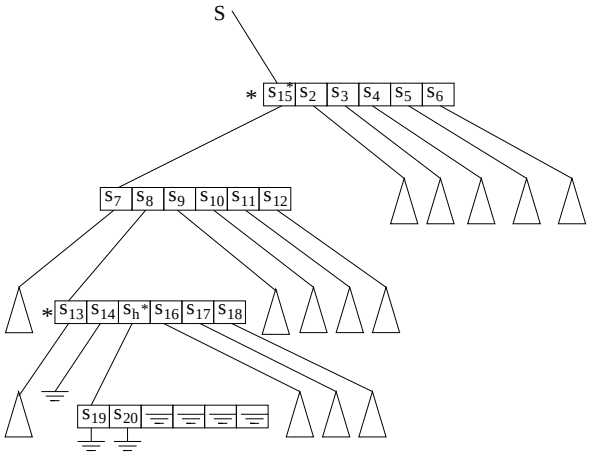


Figura 4.5: *gnat*: Árbol después de la eliminación de un centro, usando *planos fantasmas* y reemplazo del más cercano.

Para los dos métodos anteriores, hay que considerar los costos adicionales de búsqueda del más cercano. Además, tomando en cuenta que si el objeto de reemplazo no está ubicado en una hoja, se realiza una nueva búsqueda del más cercano para este nuevo objeto. Este proceso es recursivo hasta que uno de los objetos de reemplazo esté en una hoja. Este método implicaría que por cada eliminación existirían varios nodos y split marcados como afectados, lo que afectaría en los costos a los métodos de inserción y búsqueda.

Es importante señalar que, aunque un nodo se vea afectado, no necesariamente todos sus subárboles lo serán, lo que quiere decir que algunos de ellos siguen siendo *gnat*.

4.2.2.3. Reemplazo por Descendiente más Cercano Ubicado en una Hoja

Una tercera alternativa es el reemplazo por el objeto más cercano ubicado dentro de alguna hoja descendiente del elemento a borrar. Esto podría ocasionar un solapamiento mayor de los planos, pero con la garantía que se realiza sólo una búsqueda del más cercano y únicamente hay que recorrer el árbol hasta su primera hoja.

Otra característica importante de esto es que el movimiento del objeto no ocasiona overlap en los subárboles interiores, por lo tanto sólo un nodo y centro son afectados y no otros descendientes. Es decir, el subárbol completo a partir del elemento borrado sigue siendo un gnat.

4.2.2.4. Reemplazo por Último Descendiente Ubicado en una Hoja

La última alternativa es el reemplazo directo el elemento a borrar por el último descendiente que éste en alguna hoja del subárbol. Esta propuesta evita los cálculos de distancia ocasionados por la búsqueda del más cercano, es decir, costo 0 en términos de evaluaciones de distancia. Para esta alternativa basta eliminar la línea 5 del algoritmo 4 y elegir el último centro en N.

Algoritmo 4 *gnat*: eliminación de un objeto q (alternativa 4.2.2.3).

eliminaobjeto(Nodo P, TipoObjeto q)

1: {Sea $N_{min} \in N$ el centro más cercano a q}
2: {Sea D_q subárbol con centro q}
3: búsqueda del elemento a eliminar
4: se recorre el subárbol D_q hasta su primera hoja (N)
5: se obtiene N_{min}
6: se reemplaza q por N_{min} {se borra q definitivamente}
7: se marca como afectado $P_{N_{min}}$
8: borra $N_{min} \in N$ {definitivamente}

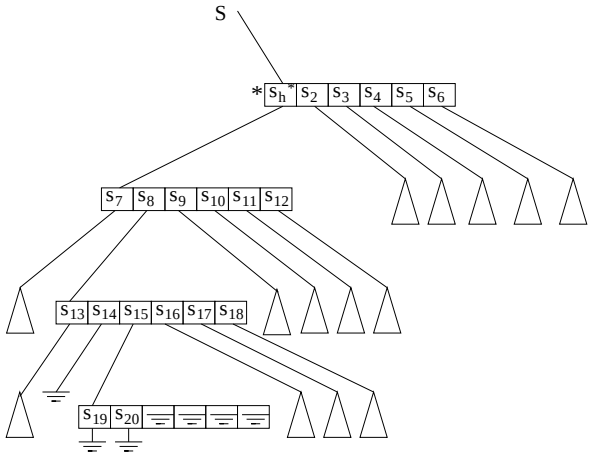


Figura 4.6: *gnat*: Árbol con una eliminación, usando *planos fantasmas* y reemplazo directo del último objeto ubicado en una hoja.

Al igual que con la alternativa anterior no existiría overlap en los subárboles interiores, es decir, sólo se crearía un plano fantasma como máximo por cada eliminación.

Para el árbol de la figura 4.4, aplicando este método la estructura quedaría como se muestra en la figura 4.6, considerando que la hoja donde se busca el reemplazante es la misma que para el primer algoritmo.

4.3. Búsquedas después de la Eliminación

Para el algoritmo de *planos fantasmas*, el definir una marca de centro *afectado* permite considerar a este centro en forma especial en los distintos métodos. La marca almacenará la distancia entre el objeto eliminado y su reemplazante, lo que indicaría cuánto se desplazó el plano.

Es interesante mostrar que para el método de *planos fantasmas*, el algoritmo de inserción después de eliminaciones no sufre modificación, aunque los nuevos datos serían insertados en el plano real y no el fantasma, es decir, como hijo de un centro que realmente existe. En otras palabras,

independientemente de si un centro contiene planos fantasmas (o sea, está marcado como afectado), siempre es posible agregarle datos.

En cantidad de datos, un plano real sólo puede crecer, y un plano fantasma sólo puede disminuir.

4.3.1. Búsquedas por Rango

Si se han realizado inserciones luego de eliminaciones, entonces, durante la búsqueda por rango al llegar a un nodo se aplica el algoritmo original sobre sus centros no marcados. Posteriormente se deben incluir en el conjunto respuesta los subárboles cuyos centros están marcados. De esta manera se incluyen las posibles respuestas que estén tanto en el plano real como en el fantasma. Sin embargo, dentro de los subplanos (o subárboles) de dicho centro, si existe un nodo no marcado, entonces en este subplano no hay solapamiento, por lo tanto se aplica el método original y se mantiene la eficiencia dentro de este subárbol.

4.3.2. Eliminación

Para buscar un elemento durante su eliminación no se utiliza el algoritmo de búsqueda con rango 0 sobre la estructura. Ésto, debido a que siempre es posible independiente del rango, que el conjunto respuesta tenga más de un centro. Considerando que un objeto sólo tiene una posición, entonces no es necesario buscar en más de un centro. Tampoco se utilizó el algoritmo original de inserción, ya que éste calcula la distancia a todos los centros de un nodo para obtener el más cercano.

Finalmente el método para buscar el objeto a eliminar utiliza una búsqueda por rango 0 reducida, es decir, sólo en el nodo y sin propagar la búsqueda sobre todos los centros resultantes. El objetivo de ésto sólo es reducir los cálculos de distancia. El algoritmo es:

1. Se busca objeto con rango 0 dentro de un nodo, sobre los centros no marcados.
2. Para el conjunto respuesta se calcula la distancia entre el objeto y cada uno de los centros.
3. Luego se elige el subárbol con la menor distancia.

Luego de eliminaciones e independiente de nuevas inserciones, es muy factible que al llegar a un nodo afectado, el objeto a buscar esté en el plano fantasma, pero no en el real, es decir, que su ubicación debería haber estado en algún plano adyacente. Entonces, debe incorporarse a la búsqueda los centros marcados como afectados.

En este método se busca el objeto sobre un subárbol, aquel que tiene la menor distancia con el objeto, más los subárboles cuyos centros tienen marcas de afectado. El proceso completo se muestra en el algoritmo 5.

Algoritmo 5 *gnat*: búsqueda (reducida) del objeto q para su eliminación.

Nodo busquedareducida(Nodo P, Consulta q, Rango r)

1: {Sea R el conjunto resultado}
2: {Sea T el conjunto distancias a q}
3: {Sea T_{min} distancia mínima a q}
4: {Sea p_{min} el centro que tiene distancia T_{min} q}
5: $R \leftarrow \emptyset$
6: $T_0 \leftarrow dist(p_0, q)$
7: **if** $T_0 = r$ **then**
8: return P
9: **end if**
10: $range(p_0, q) \leftarrow [d - r, d + r]$
11: **for all** $p_i \in P$ **do**
12: **if** $p_i.marca = false$ **then**
13: **if** $range(p_0, q) \cap range(p_0, D_{p_i}) \neq \emptyset$ **then**
14: se agrega p_i a R
15: $T_i \leftarrow dist(p_i, q)$
16: **end if**
17: **else**
18: se agrega p_i a R
19: $T_i \leftarrow dist(p_i, q)$
20: **end if**
21: **end for**
22: de R y T obtiene p_{min} y T_{min}
23: **if** $T_{min} = r$ **then**
24: return P
25: **else**
26: busquedareducida($D_{p_{min}}$, q, 0)
27: **end if**

4.3.3. Optimización

Si el grado de los nodos del árbol es elevado y existiesen muchos centros marcados dentro de un nodo, es posible estar incluyendo en las búsquedas más subárboles de los necesarios (todos los marcados).

Es posible evitar buscar en todos los marcados usando un factor de incertidumbre, es decir, si se reemplaza S_1 por S_h, entonces $I_{S_h} = d(S_h, S_1)$, el cual va a ser el valor almacenado en la *marca de afectado* (o borrado). Por lo tanto, cualquier decisión respecto de ese subárbol puede estar errada por $\pm I_{S_h}$. Por ejemplo, en una búsqueda de un dato a eliminar si $d(q, S_1) > d(q, S_2)$, se descarta S_1, ahora, si S_1 fue reemplazado por S_h, entonces igualmente podemos descartar S_h si $d(q, S_h) > d(q, S_2) + I_{S_h}$.

De igual manera en la búsqueda por rango se puede descartar un centro marcado usando el factor de incertidumbre modificando el algoritmo original de la siguiente manera:

$$\forall\, x \in P,\ si\ [Dist(q,p) - r - I_p, Dist(q,p) + r + I_p] \cap range(p, D_x)$$

Ahora bien, el conjunto D_x es posible que sea un plano fantasma, por lo tanto, $range(p, D_x)$ debería ser redefinido como: $[min\_d\,(p, D_x) - I_x, max\_d\,(p, D_x) + I_x]$. Entonces, si la intersección resulta falsa, se puede descartar el subárbol D_x.

4.3.4. Casos Especiales

Si bien los métodos propuestos evitan todo recálculo de distancia, existen casos especiales donde a bajo costo es posible mantener la estructura sin tantos planos fantasmas y rangos sobre dimensionados. Estos son:

1. Si el objeto a eliminar se encuentra en una hoja, no se crea un plano fantasma. Ésto ya que no tiene hijos y basta con eliminar los rangos de sus vecinos al centro borrado. En este caso no existen cálculos extra.

2. Si el centro a eliminar tiene un solo hijo. Se reemplaza por éste y se recalculan los rango de sus vecinos al nuevo centro, evitando así un nuevo plano fantasma y rangos sobre dimensionados.

3. Si el objeto reemplazo es el único hijo de un centro. Entonces, se recalculan las distancias de los vecinos al padre del objeto usado como reemplazo, evitando rangos sobre dimensionados.

Lo anterior implica un cálculo adicional de distancias, sin embargo, como se verá más adelante, no existirán lecturas y escrituras adicionales en disco.

4.4. Resultados Experimentales

Para los experimentos presentados en esta sección se seleccionaron las pruebas realizadas sobre dos espacios métricos. El primero, un diccionario de palabras en castellano de 86.061 objetos, donde la distancia utilizada es la *distancia de edición*, la cual entrega como resultado el número mínimo de inserciones, eliminaciones o reemplazos de caracteres para que una palabra sea igual a otra. El segundo es un espacio de vectores de coordenadas reales de dimensión 10 generados con distribución de *Gauss* con media 1 y varianza 0.1, cuya cantidad de objetos es de 100.000, para este espacio se utilizó la *distancia Euclidiana*. Se considera que ambos espacios son representativos del comportamiento del gnat. Los valores de $k = 20$ y $k = 19$ para el diccionario y el espacio de vectores respectivamente, fueron elegidos debido a que con estas cantidades de centros el nodo alcanza un tamaño lo más cercano a una página de disco de 4Kbytes, lo que servirá para los experimentos en memoria secundaria.

Los resultados mostrados utilizan los métodos planteados anteriormente sin la optimización propuesta en 4.3.3 y considerando los casos especial de la subsección 4.3.4.

4.4.1. Eliminaciones

Para los experimentos de eliminaciones, se construyó la estructura con el 90 % de los datos y sobre ella se eliminó el 10 o 40 % de los objetos, generados en forma aleatoria y en orden aleatorio, el restante 10 % se conserva para las búsquedas.

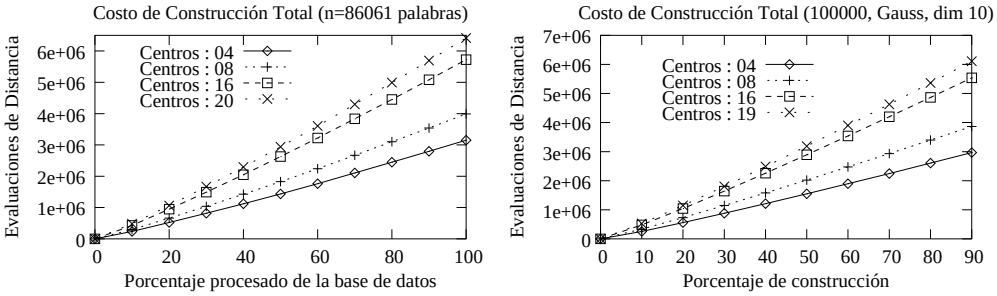
Para buscar el elemento que se eliminará se usará el método planteado en 4.3.2. Se incluye el gráfico de construcción para cada uno de los espacios a modo de referencia (figura 4.7). Los gráficos de las figuras 4.8 y 4.9 muestran los costos de eliminación en promedio por elemento para los diferentes espacios.

El método utilizado para la eliminación es el de *reemplazo por el último descendiente ubicado en una hoja* (4.2.2.4).

Se puede observar, en los gráficos de eliminación, que a mayor cantidad de datos eliminados las estructuras con mayor cantidad de centros tienden a comportarse mas eficiente que las de menor cantidad de centros.

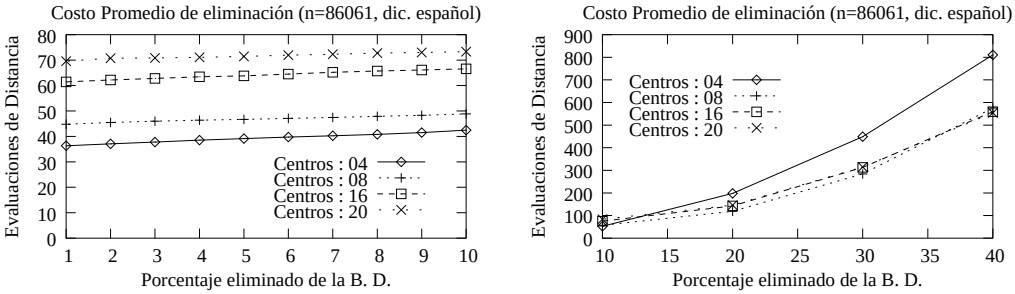
4.4.2. Búsquedas

Para los experimentos de búsquedas, se reservó un 10 % de objetos no insertados en la base de datos. Para el caso de los diccionarios, los rangos de búsqueda fueron 1, 2, 3 y 4. Para el caso de vectores, a priori se calculó lo rangos que recuperaban el 0.01 %, 0.1 % y 1 % de la base de datos. Para el caso de búsquedas con eliminaciones, se eliminó el 10 y 40 % de la base de datos y se insertó la misma cantidad; sobre esto se realizó la búsqueda del 10 % restante. Todos los costos promedios de búsquedas son presentados en la figura 4.10. Las búsquedas sobre la estructura sin eliminaciones



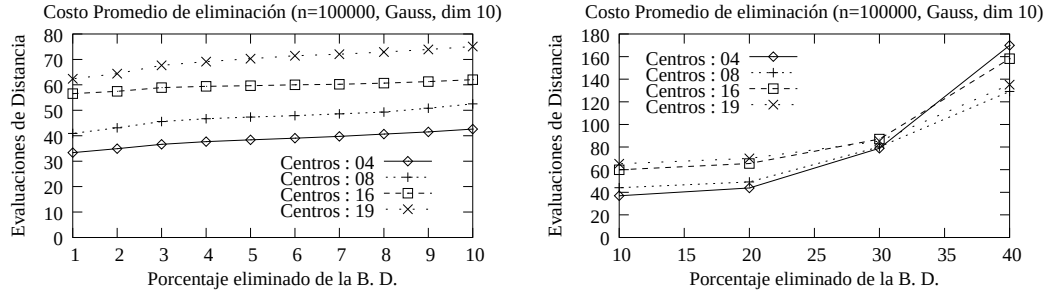
(a) Diccionario español (b) Espacio de Gauss

Figura 4.7: Costos totales de construcción para el diccionario español (86.061 objetos) y para el espacio de vectores de Gauss de dimensión 10 (90 % de objetos).



(a) 10 % de datos. (b) 40 % de datos.

Figura 4.8: Costos promedios de eliminación para el diccionario español.



(a) 10 % de los datos (b) 40 % de los datos

Figura 4.9: Costos promedios de eliminación para el espacio de Gauss.

corresponden a las subfiguras a y b. Con el 10 % eliminado y 10 % reinsertado en c y d. Con 40 % eliminado y 40 % reinsertado en e y f.

En los gráficos de búsquedas es observable el aumento de las evaluaciones de distancia producto de los objetos eliminados, las nuevas inserciones y de los planos fantasmas. La figura 4.11 se muestra el aumento en porcentaje para una búsqueda sin eliminaciones versus una búsqueda sobre la estructura con 40 % de los datos eliminados y la misma cantidad insertada posteriormente. Se puede ver que en el espacio de palabras es donde más se degrada la búsqueda con aumentos cercanos al 50 %. Para el espacio de Gauss el aumento es alrededor del 40 %. El porcentaje baja a medida que aumenta el rango de búsqueda producto de que, a mayor rango mayor cantidad de subárboles donde buscar y por lo tanto menor el efecto de los planos fantasmas.

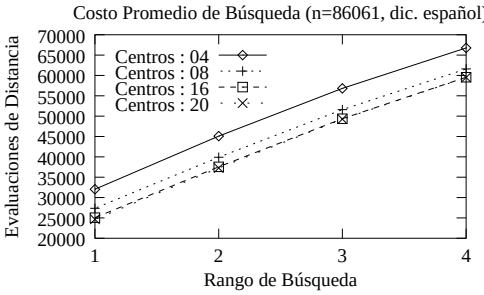
4.4.3. Optimización

Los distintos efectos de la optimización propuesta en 4.3.3 se muestran en las figuras 4.12 y 4.13 para la eliminación de objetos en el diccionario español. En las figuras 4.14 y 4.15 se muestra la información para la eliminación en el espacio de Gauss.

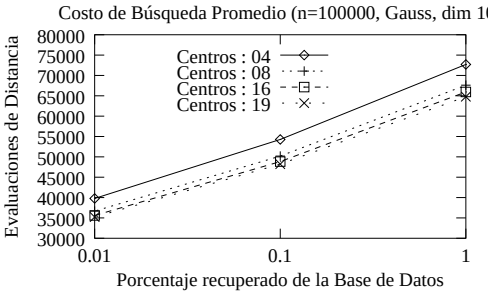
En estos experimentos de puede observar que a medida que aumenta la cantidad de datos eliminados, la optimización se vuelve más efectiva (a menor cantidad de datos eliminados, se hace menos necesaria). Se observa también un mejor comportamiento a medida que aumenta la cantidad de centros de la estructura. Para el espacio de Gauss con 40 % de datos eliminados e insertados los porcentajes son cercanos al 7 % para 19 centros, pues que se pueden descartar más subárboles a mayor cantidad de centros. En este espacio la excepción es con $k = 16$, donde el porcentaje es mayor, sin embargo, esto es debido que la estructura para 16 centros tuvo un peor comportamiento que con 19 durante la eliminación. Para el espacio de palabras ocurre algo similar.

Respecto de las búsquedas, la figura 4.16 muestra los cálculos de distancias para los distintos espacios con las distintas cantidades de datos eliminados e insertados. La figura 4.17 muestra la relación en porcentaje de la disminución de evaluaciones de distancia para los métodos optimizados versus los no optimizados.

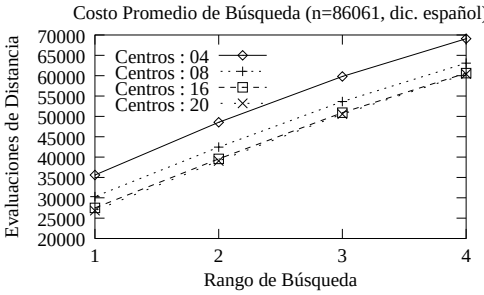
Durante las búsquedas, las estructuras con mayor cantidad de centros tuvieron un mejor desempeño. Por ello, al aplicar la optimización se observa que los porcentajes de reducción de evaluaciones de distancia son menores que para estructuras con baja cantidad de centros (ver figura 4.17). Al igual que con la eliminación a mayor cantidad de datos eliminados la búsqueda se hace más eficiente. Sin embargo, al aumentar los rangos de búsqueda es menor el efecto, dado que aumenta la probabilidad de intersecciones de la bola de consulta con los planos, por lo tanto la cantidad de subárboles descartados es menor.



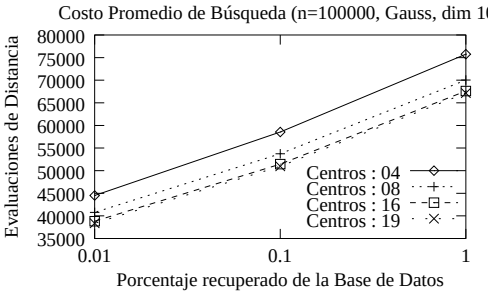
(a) Sin eliminaciones



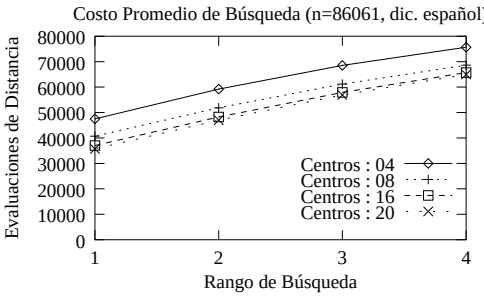
(b) Sin eliminaciones



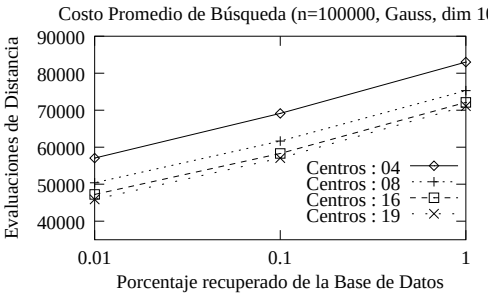
(c) 10 % eliminado y 10 % insertado.



(d) 10 % eliminado y 10 % insertado.

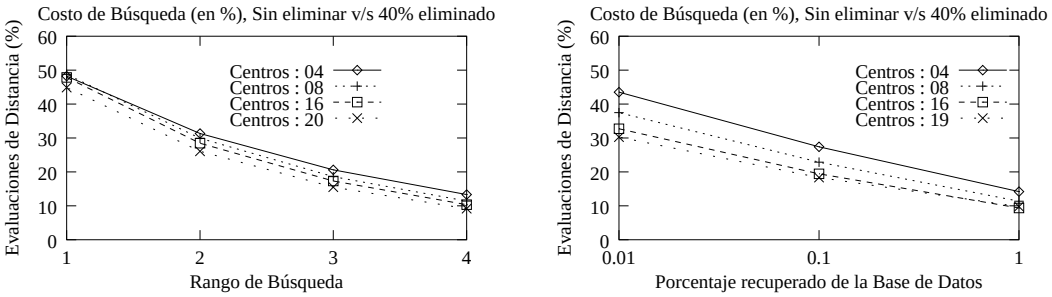


(e) 40 % eliminado y 40 % insertado.



(f) 40 % eliminado y 40 % insertado.

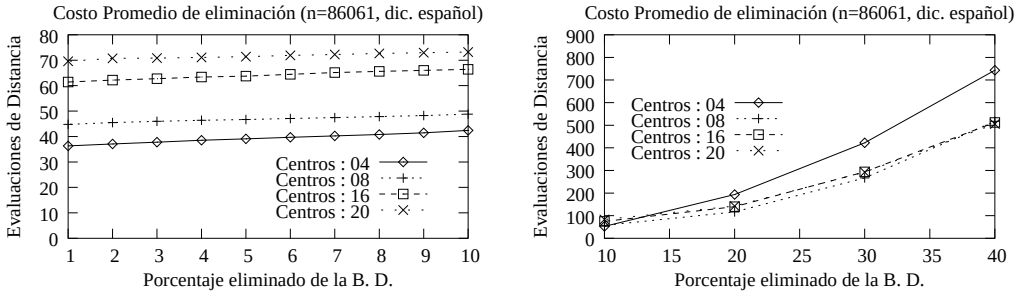
Figura 4.10: Costos de búsqueda promedio para el diccionario español (columna izquierda) y el espacio de Gauss de dimensión 10 (columna derecha).



(a) Diccionario español

(b) Gauss dim. 10

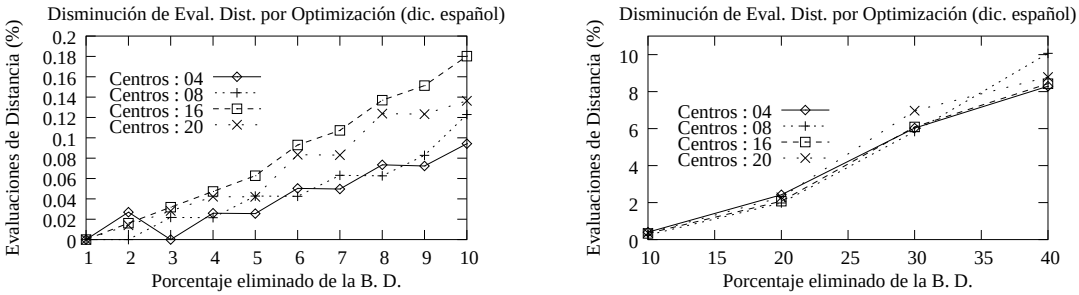
Figura 4.11: Aumento de evaluaciones de distancia (en %) para la búsqueda sin eliminaciones versus con 40 % eliminado e insertado. Para el diccionario español y para el espacio de Gauss.



(a) 10 % eliminado

(b) 40 % eliminado

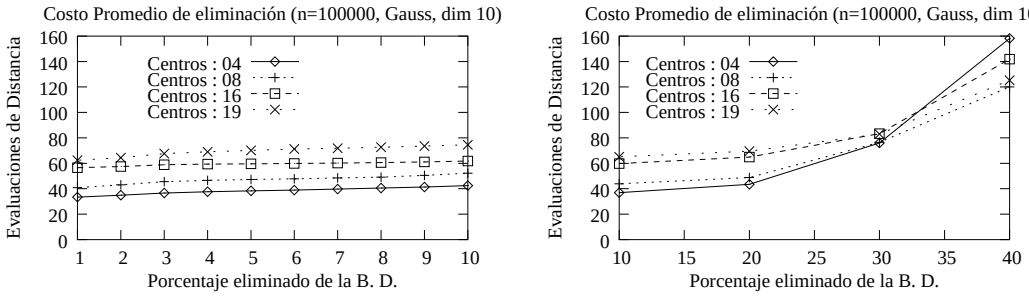
Figura 4.12: Costos promedios de eliminaciones para el diccionario español. Método optimizado.



(a) eliminación del 10 %

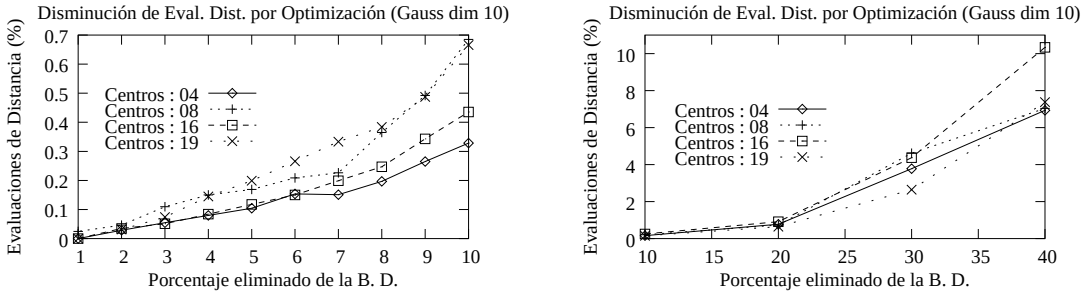
(b) eliminación del 40 %

Figura 4.13: Porcentajes de reducción de evaluaciones de distancia para métodos de eliminación sin y con optimización para el diccionario español.



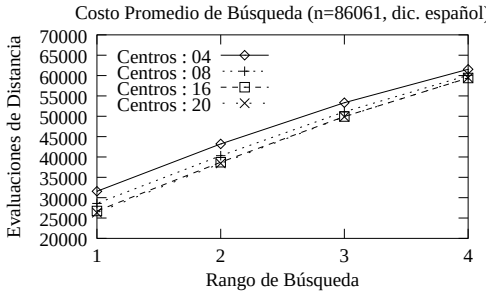
(a) 10 % eliminado (b) 40 % eliminado

Figura 4.14: Costos promedios de eliminaciones para el espacio de Gauss. Método optimizado.

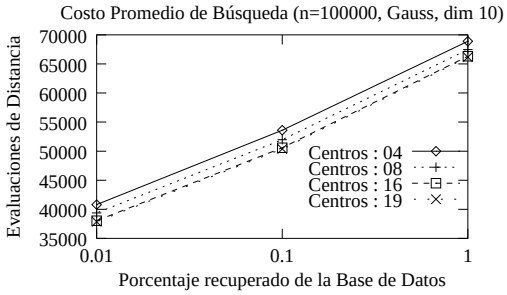


(a) 10 % eliminado (b) 40 % eliminado

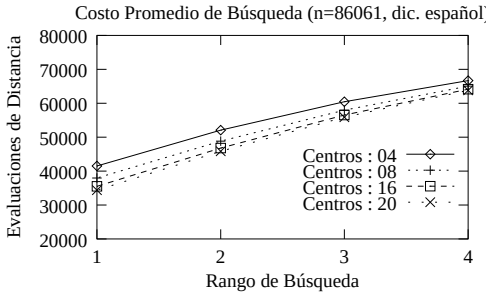
Figura 4.15: Porcentajes de reducción de evaluaciones de distancia para métodos de eliminación sin y con optimización para el espacio de Gauss.



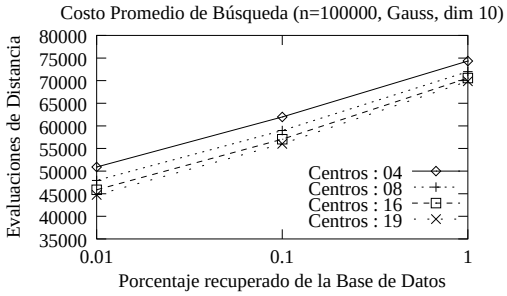
(a) Dic. español 10 % eliminado y reinsertado



(b) Gauss 10 % eliminado y reinsertado

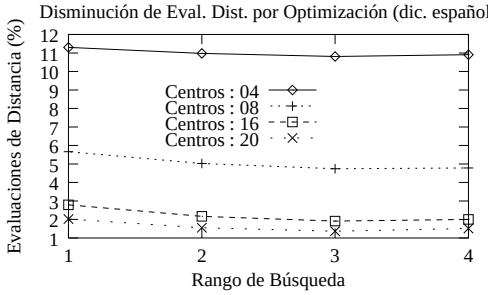


(c) Dic. Español 40 % eliminado y reinsertado

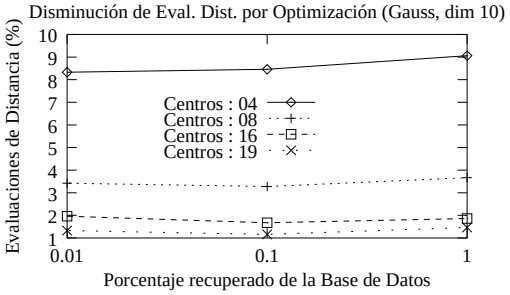


(d) Gauss 40 % eliminado y reinsertado

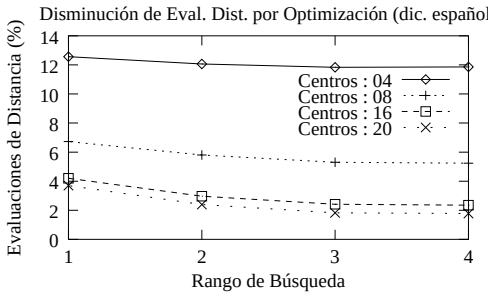
Figura 4.16: Búsquedas con métodos optimizados para el espacio de palabras y el espacio de Gauss.



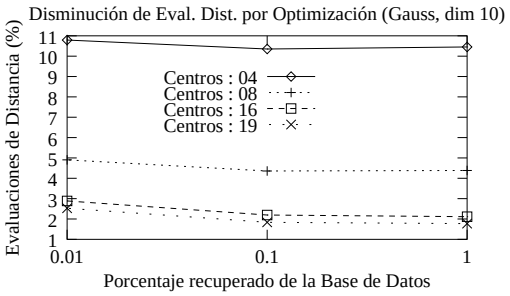
(a) Dic. español 10 % eliminado y reinsertado



(b) Gauss 10 % eliminado y reinsertado

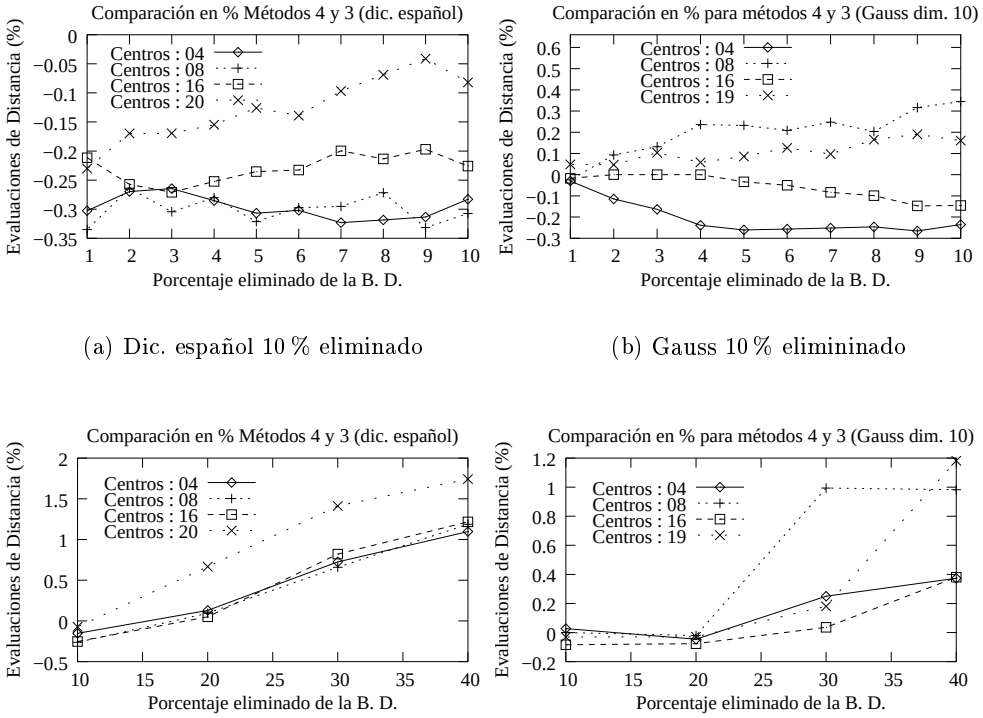


(c) Dic. español 40 % eliminado y reinsertado



(d) Gauss 40 % eliminados y reinsertados

Figura 4.17: Porcentajes de evaluaciones de distancia para la búsqueda entre versiones no optimizadas y optimizadas para el espacio de palabras y para el espacio de Gauss.



(a) Dic. español 10 % eliminado

(b) Gauss 10 % elimininado

(c) Dic. español 40 % eliminado

(d) Gauss 40 % eliminado

Figura 4.18: Variación porcentual en eliminaciones para el espacio de palabras y el espacio de Gauss. Método 4 versus método 3.

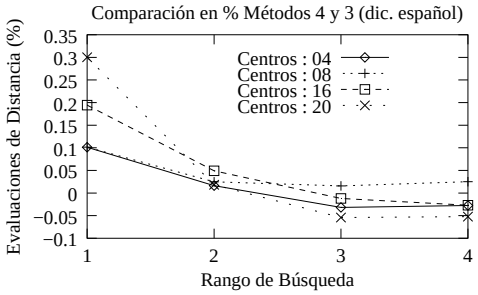
4.4.4. Alternativa 4.2.2.4 versus 4.2.2.3

Para una mejor visualización de los resultados obtenidos usando la tercera alternativa propuesta, de *reemplazo por el objeto más cercano en una hoja*, se muestran las figuras con las diferencias porcentuales entre ambos métodos.

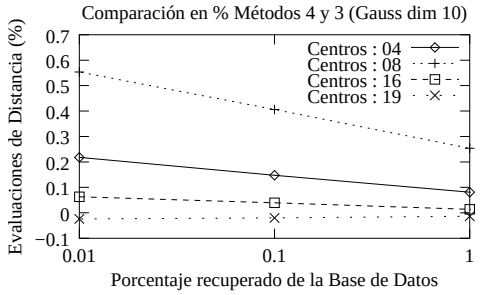
En las figuras 4.18 y 4.19 los porcentajes positivos indican un mejor comportamiento de la alternativa 3, por el contrario, los valores negativos implican que la alternativa 4 es superior.

En los gráficos de eliminaciones las curvas se presentan irregulares producto de la mínima diferencia que existe entre ambos métodos. Se puede observar en este caso que la alternativa 4 es mejor que la 3 para bajos porcentajes de datos eliminados. Sólo para el espacio de Gauss se obtienen mejores valores (porcentajes positivos) para las estructuras con 10 % eliminado, sin embargo, este aumento es para estructuras con mayor cantidad de centros y sólo bordea el 0.3 %.

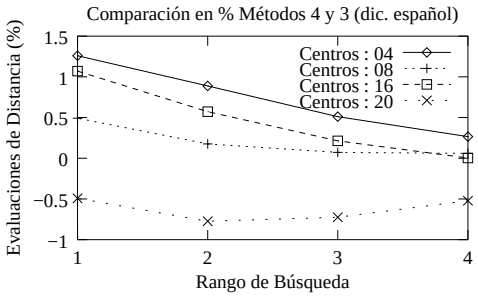
A medida que aumenta la cantidad de datos eliminados, los métodos implementados con la alternativa 3 comienzan a ser más eficientes que los generados con la alternativa 4. Esto es producto que el factor de incertidumbre (sección 4.3.3) crece un poco menos que con la otra alternativa, y por lo tanto es más eficiente la búsqueda del dato a eliminar.



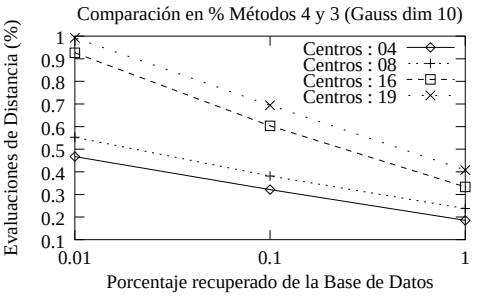
(a) Dic. español 10 % eliminado e insertado



(b) Dic. español 10 % eliminado e insertado



(c) Dic. español 40 % eliminado e insertado



(d) Dic. español 40 % eliminado e insertado

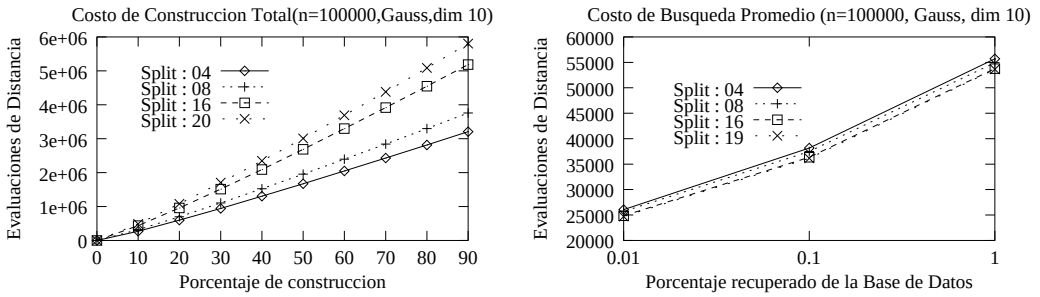
Figura 4.19: Variación porcentual en búsquedas por rango para el espacio de palabras y el espacio de Gauss. Método 4 versus método 3.

Para los métodos de búsqueda por rango (figura 4.19), la variante 4.2.2.3 tiene un rendimiento levemente mejor que la alternativa 4.2.2.4, sin embargo, a medida que aumentan los rangos de búsqueda la mejora en el rendimiento decae considerablemente, llegando a valores peores que los generados por la alternativa 4.2.2.4. En este caso, el espacio de palabras es el que tiene el peor comportamiento.

4.4.5. *Planos Fantasmas* en otras Estructuras

Resulta interesante analizar el método de *planos fantasmas* sobre otras estructuras de tipo árbol. Desde este punto de vista, el *Voronoi-tree* o VT es un ejemplo ideal, primero por su parecido en forma y construcción al gnat y segundo porque al replicar las raíces de sus subárboles (ver sección 3.5.2), presenta un medio hostil para la aplicación del método.

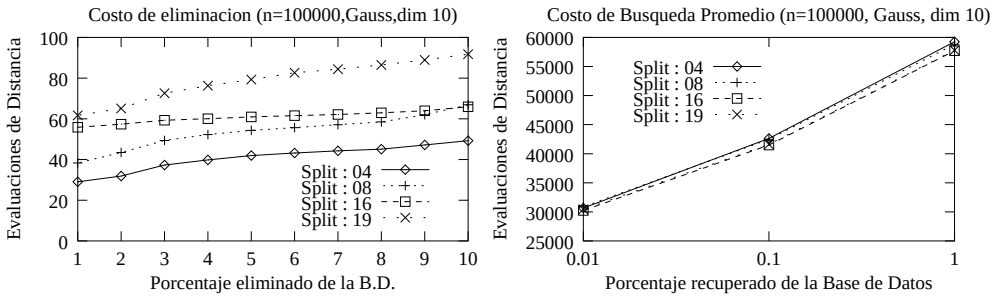
Lo primero a considerar es que, comparada con otras estructuras con igual cantidad de centros por nodo, el VT debería tener igual o mayor cantidad de nodos, producto de replicar la raíces hacia abajo en los subárboles. En segundo lugar, el hecho de tener valores repetidos en subárboles implica



(a) Voronoi-tree: construcción

(b) Voronoi-tree: búsqueda sin eliminaciones

Figura 4.20: *Voronoi-tree*: Costos totales de construcción y búsquedas promedio (sin eliminaciones) para el espacio de vectores de Gauss de dimensión 10, 90 % de objetos.



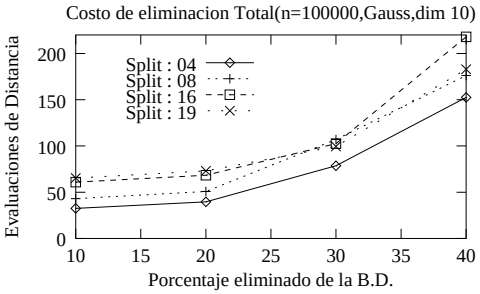
(a) 10 % eliminado

(b) búsqueda con 10 % eliminado y reinsertado

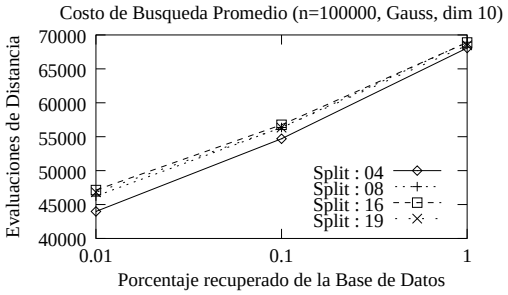
Figura 4.21: *Voronoi-tree*: Costos totales de eliminación para el 10 % y búsqueda promedio para el 10 % eliminado y reinsertado parreinsertadoa el espacio de Gauss.

que al eliminar uno de estos elementos ocurre un efecto de propagación de planos fantasmas. Es decir, al reemplazar el objeto eliminado, el objeto de reemplazo también debe replicarse, entonces, como este lleva consigo la marca de afectado, implica necesariamente que la marca se propaga. Por lo tanto, por una eliminación es posible tener varios planos fantasmas. Este efecto es más notorio a medida que el objeto eliminado está más cercano a la raíz del árbol.

En las figuras 4.20, 4.21 y 4.22 se muestran los costos de eliminación y búsquedas para el VT sobre el espacio de Gauss. Nótese que la forma de las gráficas de costos de eliminación son similares a las del *gnat*, sin embargo, producto de lo mencionado en el párrafo anterior, la cantidad de evaluaciones de distancia es superior. El método utilizado para la eliminación es el propuesto en 4.2.2.4. Para la búsqueda del elemento a eliminar y para las búsquedas por rango se utilizó la optimización (sección 4.3.3). De esta manera de pueden comparar las distintas gráficas de costo con las presentadas en la sección 4.4.3 para el *gnat*.



(a) 40 % eliminado



(b) búsqueda con 40 % eliminado y reinsertado

Figura 4.22: *Voronoi-tree*: Costos totales de eliminación para el 40 % y búsqueda promedio para el 40 % eliminado y reinsertado para el espacio de Gauss.

A pesar de las dos características mencionadas que provocan un aumento en la cantidad de cálculos de distancia y que hacen del VT una estructura no ideal para la aplicación de planos fantasmas, se puede ver a través de los mismos gráficos que aún así el método propuesto logra buenos resultados, tanto para la eliminación como para la búsqueda por rango.

4.4.6. Observaciones

Para entender la forma que va adquiriendo la estructura posterior a las eliminaciones, se presentan las siguientes tablas que reflejan información interesante en este respecto para ambos espacios utilizados. En las figuras 4.1 y 4.5 se muestra información general para la construcción de ambos espacios y la eliminación del 10 % de los datos. La columna *Total inicial* indica la cantidad de nodos después de haber creado la estructura con el 90 % de los datos, la columna *Total-n %* muestra la cantidad de nodos que quedan en la estructura después de eliminar n nodos, *borrados* representa la cantidad de nodos definitivamente eliminados, la columna *afectados* indica la cantidad de nodos que aún quedan en la estructura que tienen una *marca de afectado*, es decir, cada nodo afectado contiene a lo menos un *plano fantasma*.

Después de insertar la misma cantidad de datos sobre la estructura (reflejado en las tablas 4.2, 4.4, 4.6 y 4.8), se muestra la información de nodos *borrados* y nodos *nuevos*, que indica si aparecen nuevos nodos o por el contrario, la estructura queda con menos nodos, en relación a la cantidad generada inicialmente en la construcción. En este aspecto, el espacio de Gauss muestra un mejor comportamiento, es decir, los nuevos datos se ubican en mejores posiciones dentro del árbol, lo que hace que distribución sea mejor y por lo tanto los datos se insertan en nodos que tienen aún espacio para dichos objetos.

La forma de las estructuras es independiente de los métodos utilizados para buscar los elementos a eliminar, como también para las alternativas elección del reemplazo (alternativas 4.2.2.4 y 4.2.2.3).

Nro. Splits	Total (90 % inicial)	Total-10 %	borrados	afectados
4	28481	25148	3333	4062
8	20062	17571	2491	2720
16	14953	12982	1971	1561
20	13939	12145	1794	1256

Tabla 4.1: (Dic. Español): construcción con el 90 % de los datos y eliminación del 10 %.

Nro. Splits	Total	borrados	nuevos	afectados
4	28574	0	93	4062
8	20145	0	83	2720
16	15038	0	85	1561
20	14069	0	130	1256

Tabla 4.2: (Dic. Español): eliminación del 10 % de los objetos y reinserción del 10 %.

Nro. Splits	Total-40 %	borrados	afectados
4	15336	13145	8693
8	10589	9473	4314
16	7957	6996	1841
20	7381	6558	1385

Tabla 4.3: (Dic. Español): eliminación del 40 % de los objetos.

Nro. Splits	Total	borrados	nuevos	afectados
4	28447	34	0	8693
8	20030	32	0	4314
16	15020	0	67	1841
20	13898	41	0	1385

Tabla 4.4: (Dic. Español): eliminación del 40 % de los objetos y reinserción del 40 %.

Split	Total (90 % inicial)	Total - 10 %	borrados	afectados
4	33901	29835	4066	4516
8	24256	21293	2963	2923
16	19022	16560	2462	1683
19	17885	15318	2567	1506

Tabla 4.5: (Gauss, dim. 10): construcción con el 90 % de los objetos, eliminación del 10 % y sin reinserción.

Split	Total	borrados	nuevos	afectados
4	33695	206	0	4516
8	24055	201	0	2923
16	18726	296	0	1683
19	17575	310	0	1506

Tabla 4.6: (Gauss, dim. 10): después de eliminar el 10 % y reinsertar el 10 %.

Split	Total	borrados	afectados
4	18107	15794	9853
8	12735	11521	4691
16	9646	9376	2003
19	8611	9274	1671

Tabla 4.7: (Gauss, dim. 10): eliminación del 40 %, características de la estructura después de eliminar y antes de reinsertar.

Split	Total	borrados	nuevos	afectados
4	33289	612	0	9853
8	23841	415	0	4691
16	18070	952	0	2003
19	16869	1016	0	1671

Tabla 4.8: (Gauss, dim. 10): características de la estructura después de eliminar y después de reinsertar el 40 %.

4.5. Discusión

En este capítulo se ha presentado una versión dinámica de la estructura *gnat*, la cual permite realizar inserciones y eliminaciones eficientemente sin afectar en demasía la calidad de las búsquedas.

Es importante destacar que con la opción de reemplazo por el último objeto ubicado en una hoja, la cantidad de evaluaciones de distancia se reduce a uno, la cual es para mantener el factor de incertidumbre como *marca de afectado*. El costo extra sólo es el recorrido del árbol hasta su primera hoja. Sin embargo, existe un costo extra en buscar los próximos elementos a eliminar.

Respecto de los experimentos, se puede concluir, como era de esperar, que la búsqueda se degrada a aumentar el porcentaje de objetos eliminados. Sin embargo, el aumento en las evaluaciones de distancia para la búsqueda respecto de la estructura sin eliminaciones sigue siendo adecuado.

De los gráficos de eliminación se puede determinar que para el caso del espacio de palabras cuando la cantidad de eliminaciones supera el 20 % y en el caso de vectores el 35 %, la búsqueda del elemento de reemplazo se degrada bastante. En este caso, el espacio de palabras tiene el peor comportamiento superando las 500 evaluaciones para la búsqueda de un elemento usando el método de eliminación optimizado.

La optimización tiene un mayor desempeño en los métodos de eliminación, con comportamientos similares en ambas estructuras y con reducciones en las evaluaciones de distancia del orden del 8 % para 40 % de datos eliminados.

Respecto de la búsqueda, la optimización afecta positivamente a las estructuras con la menor cantidad de centros, sin embargo, esto es producto del bajo rendimiento de estas estructuras durante la búsqueda por rango.

Al comparar las alternativas propuestas en 4.2.2.4 y 4.2.2.3. Se puede concluir que la alternativa de *reemplazo por el objeto más cercano ubicado en una hoja* (4.2.2.3), tiene un mejor comportamiento durante la búsqueda de objetos para su eliminación. Sin embargo, esta mejora sólo alcanza al 1 % en el mejor caso, para estructuras con eliminaciones sobre el 35 a 40 % de los datos.

Tampoco se logra observar una mejora considerable en las búsquedas por rango para la alternativa 3. Logra leves mejoras para estructuras con altos porcentajes de datos eliminados y con bajos rangos de búsquedas.

Finalmente, es importante recordar que ambas alternativas no propagan los planos fantasmas, y en el pero caso genera un plano fantasma por dato eliminado.

La diferencia del método de *planos fantasmas* con el de *nodos ficticios* propuesta en [NR02], es que el primero efectivamente contiene un objeto en el lugar donde fue eliminado un dato, por lo tanto es posible hacer comparaciones durante la eliminación, búsqueda y reinserción de objetos y minimizar el subárbol usando la optimización propuesta en la sección 4.3.3 evitando entrar cada vez en éste.

Una pregunta relevante sobre este método sería: **¿cuanto es el máximo desplazamiento que puede moverse un plano?**.

El máximo desplazamiento I_S (*marca de afectado*) que puede tener un plano $P(S)$ es su radio cobertor, es decir, $I_S \leq max\_d(S, D_s)$, donde D_S es el conjunto de todos elementos que están en $P(S)$. Por lo tanto, el plano en el peor caso siempre queda *anclado* al elemento más lejano en $P(S)$, independientemente de cuántos elementos hayan sido borrados del plano.

Considerando la eficiencia del método de *planos fantasmas* propuesto para eliminaciones y de las alternativas de *"reemplazo por el último descendiente ubicado en una hoja*" y *de "reemplazo por el descendiente más cercano ubicado en una hoja*" , su forma y la simplicidad, se plantea como un método general para eliminaciones en nodos internos en estructuras de tipo árbol, pruebas de esto se ven en los experimentos realizados sobre el VT y sería sencillo de aplicar a otras se similares características, como por ejemplo el SAT.

Finalmente, en el próximo capítulo se analizará también, la viabilidad de este método sobre estructuras en memoria secundaria.

Capítulo 5

Gnat en Memoria Secundaria

En el presente capítulo se analizará la estructura *gnat* desde la perspectiva de memoria secundaria, en particular la viabilidad del método de *planos fantasmas* sobre memoria secundaria.

En memoria secundaria, la medida de costo ya no es solamente la cantidad de evaluaciones de distancia que se realizan, sino también los accesos a disco, ya sea lecturas, escrituras o movimientos de cabezal (*read*, *write* y *seek*). Además, como se verá más adelante, es importante considerar el porcentaje de uso de las páginas de disco y tamaño de la estructura generada, entre otros.

Una de las medidas importantes mencionadas, la reducción de los movimientos del cabezal en memoria secundaria, es resuelta en estructuras para búsquedas exactas agrupando los objetos de acuerdo a un orden en la base de datos (por ejemplo, alfabético en el caso de palabras). Con esto los accesos se realizan en forma secuencial al momento de la búsqueda y se evita costo extra por saltos adicionales dentro del archivo. Esto resulta inadecuado para estructuras de búsqueda por proximidad, porque no se puede mantener un orden del mismo tipo que para búsquedas exactas. Ahora bien, es posible mantener cierta secuencialidad de accesos organizando los objetos de acuerdo a alguna característica, por ejemplo, su cercanía a algún otro objeto (la raíz del árbol al que pertenecen). La mayoría de la estructuras métricas implementan esto en forma natural, sin embargo, esto no garantiza una disminución de los saltos dentro de un archivo y todo va a depender de la intersección de los planos de la estructura y del rango de búsqueda. En el caso de un espacio métrico, un objeto podría estar cerca de varios otros objetos, más aún considerando espacios de alta dimensión, por lo tanto, durante una búsqueda para un mismo rango éste aparecería en la respuesta para varias consultas, entonces daría lo mismo agruparlo a cualquiera de los puntos consultados, no pudiéndose impedir saltos para una u otra consulta.

5.1. Construcción

En una primera fase, se implementó el *gnat* como originalmente fue diseñado. Para ello se supuso que tanto la base de datos como el índice no entran completamente en RAM. Para los experimentos

presentados en este capítulo se determinó que el tamaño máximo de memoria RAM disponible para almacenamiento de nodos sería de 2 Mbytes. Esto permite definir un parámetro ($MAXNODOS$) que indicará la cantidad máxima de nodos permitidos en memoria principal.

Durante la construcción de la estructura y una vez alcanzada la máxima capacidad de RAM, se graba toda la estructura a disco y se comienzan a usar métodos de inserción para memoria secundaria. Esto permite un uso eficiente de la memoria manteniendo parte de la estructura en RAM y parte en disco, accesando el segundo sólo cuando sea necesario.

En RAM siempre se mantiene la raíz y los primeros niveles, esto a diferencia de otras estructuras tradicionales de administración de páginas de disco, las que mantienen cierto ranking de páginas más accesadas, las cuales son las que se mantienen en RAM. En estructuras métricas de tipo árbol como el gnat, las inserciones y búsquedas siempre comienzan por la raíz y los primeros niveles, por lo tanto esto resulta adecuado.

Para tener una visión general respecto del comportamiento de la estructura en memoria secundaria, a continuación se presenta información adicional para los experimentos realizados en el capítulo anterior. Se debe entender que la información siguiente es sólo referencial en el sentido que se supone que las medidas de costos consideradas están en base a los nodos de la estructura, es decir, lecturas y escrituras de nodos y movimientos entre nodos. En ese respecto se puede considerar que un nodo será equivalente a una página de disco, por lo que para cada experimento el tamaño de la página es distinto. Los valores de $k = 19$ y $k = 20$, para los espacios de vectores y de palabras respectivamente, representan la cantidad de splits que hacen que el nodo esté más cercano al tamaño real de una página de disco de 4.096 bytes (4Kb).

Los gráficos de costos muestran información complementaria respecto de los reads, seeks y writes para la construcción del espacio de palabras en castellano (86.061 palabras) y del espacio vectores de Gauss de dimensión 10 (100.000 vectores) presentados en la sección 4.4. Los gráficos de la figura 5.1 es información adicional a los de la figura 4.7 . Para visualizar mejor la información entregada en estos gráficos hay que relacionar tanto la cantidad de datos, la cantidad de nodos generados por la estructura, como la cantidad de evaluaciones de distancia. Por ejemplo, en los gráficos se revela el momento en que se disparan los métodos para inserción en memoria secundaria, los cuales dependerán de la capacidad de nodos en RAM para cada estructura. A menor cantidad de centros, menor tamaño del nodo y por ende mayor cantidad de éstos en memoria principal. En los gráficos de escrituras, se observa que al aproximarse al total de datos procesados, las curvas para las diferentes cantidades de splits tienden a ser similares. Para ello hay que considerar que después del disparo de los mecanismos para inserción en disco, un dato nuevo implica un write como mínimo. Sin embargo, al tener una estructura más nodos que otra, posee mayor cantidad de niveles o mayor cantidad de padres que podrían ser alterados en alguno de sus rangos producto de esa inserción, lo que implicaría escrituras adicionales.

En los cuadros 5.1 y 5.3 se muestra información general de las estructuras generadas con el 100 % de los datos en ambos casos. Las distintas columnas indican la cantidad de splits (o centros) del nodo,

el total de nodos generados, el tamaño en bytes del nodo, el espacio ocupado en memoria secundaria una vez construida totalmente la estructura, el porcentaje de nodos que se pueden mantener en RAM y la cantidad de nodos en RAM. Además, la última columna indica el número de niveles que pueden estar en memoria principal y el total de niveles generados por la estructura.

De la columna *Espacio* se desprende que la estructura es totalmente deficiente desde el punto de vista del tamaño de requerido para el almacenamiento. A mayor cantidad de centros, mayor es el espacio utilizado, sólo para el caso de $k = 4$ el tamaño es equivalente al espacio utilizado por un *M-tree*, sin considerar que el tamaño del nodo no es el de una página real de disco.

Los cuadros 5.2 y 5.4 ayudan a comprender este fenómeno. Lo que ocurre es que la estructura posee demasiados nodos que contienen muy pocos datos. Las columnas de estos cuadros representan la cantidad de centros, el total de nodos de la estructura, la cantidad de nodos incompletos (con cantidad de datos $<k$), el porcentaje de nodos incompletos, el promedio de uso del nodo (sólo considerando aquellos incompletos). La última columna representa el porcentaje de uso del nodo, de ésta, la primera subcolumna es considerando sólo los nodos incompletos y la segunda subcolumna es el porcentaje general de uso de un nodo (total de datos/total de nodos).

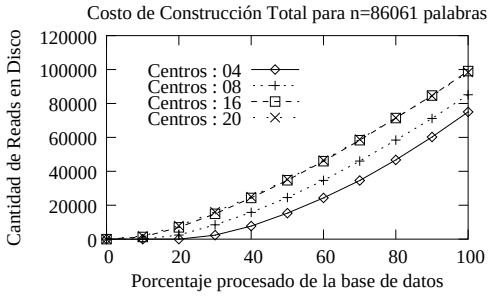
Para el caso de $k = 4$, se obtiene el mejor comportamiento, con porcentaje de incompletos inferior a 55 % (caso espacio de palabras) y porcentaje de uso del nodo sobre el 40 %. Esto resulta lógico debido que cada nodo tiene como mínimo un objeto, por lo tanto para $k = 4$ el porcentaje de uso es siempre mayor o igual a 25 %. Los casos más extremos son para $k = 20$ en el diccionario español y $k = 19$ para el espacio de vectores, esto es debido principalmente a que a mayor capacidad del nodo, más probable que éste no se llene. Además, considerando que un objeto tiene sólo una ubicación posible (lo que evita el solapamiento), no se podría compensar las cantidades de objetos por subárbol, como en el caso del *M-tree*, en el cual un objeto tiene más de una ubicación posible dentro del árbol.

Finalmente, la información proporcionada en los gráficos y cuadros mencionados da una visión más detallada de la forma y comportamiento de la estructura en memoria secundaria. En las siguientes subsecciones se mostrará el comportamiento en el caso de las eliminaciones y búsquedas.

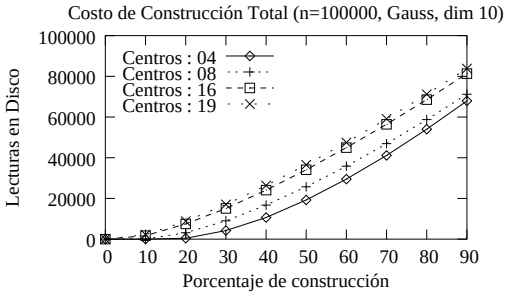
Los problemas de bajo promedio de uso de los nodos y del exceso de espacio utilizado por la estructura encontrados en está sección se tratarán en detalle en la sección 5.4.

5.2. *Planos Fantasmas* sobre Memoria Secundaria

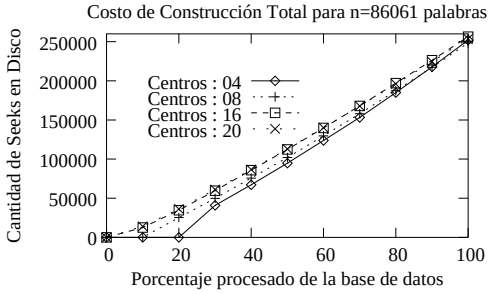
Para ver el efecto de las eliminaciones en memoria secundaria, hay que referirse a la información entregada en las tablas de la sección 4.4.6. De éstas se desprende que posteriormente a las eliminaciones y reinserciones de datos, la estructura mantiene su forma inicial, en tamaño y cantidades de nodos o páginas. En este respecto se puede observar que para el espacio de Gauss, luego de borrar e insertar datos, se mantienen páginas disponibles (columna *nodos borrados*). Esto indica una leve mejoría en el porcentaje de nodos incompletos y en el promedio y porcentaje de uso de los nodos.



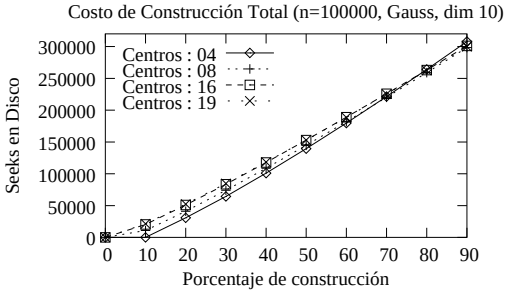
(a) Reads



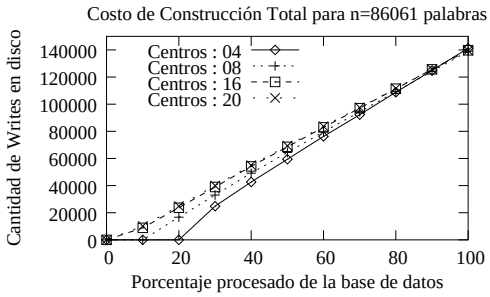
(b) Reads



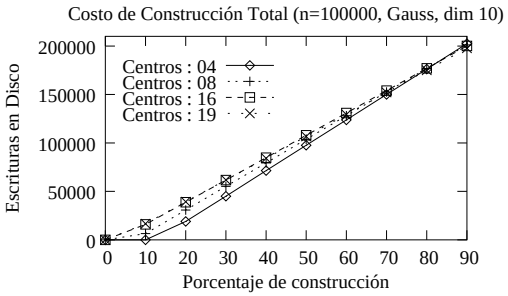
(c) Seeks



(d) Seeks



(e) Writes



(f) Writes

Figura 5.1: *gnat*: Diccionario Español y espacio de Gauss. Costo de construcción total de reads, seeks y writes en disco.

Splits	TotalNodos	SizeNodo	Espacio	% en RAM	NodosRAM	NivelRam
4	31544	304	9,15Mb	21,87	6899	7 de 21
8	22281	848	18,02Mb	11,10	2473	4 de 12
16	16857	2704	43,47Mb	4,60	776	3 de 8
20	15405	4016	59,00Mb	3,39	522	3 de 8

Tabla 5.1: Información general de la estructura para el diccionario Español.

Cuando la eliminación e inserción es del 40 %, esta mejoría aumenta y se ve un efecto similar en el espacio de palabras. Por ejemplo, para el caso del espacio de Gauss de los experimentos de la sección mencionada, el promedio de uso de los nodos incompletos para $k = 19$ era de 3,49. Después de eliminar y reinsertar 10 % de los datos éste quedó en 3,56 y en 3,72 para el 40 % eliminado y reinsertado. Los promedios de uso considerando todos los nodos fueron de 5,03, 5,12 y 5,34 respectivamente.

De todas maneras, este efecto es limitado y producto de que los nuevos objetos encontraron mejores ubicaciones debido a los movimientos que sufre el árbol con las eliminaciones, es decir, es posible que un centro actual sea más adecuado que uno ya borrado para un subárbol determinado.

Otra característica del método de planos fantasmas, es que para eliminaciones en una misma rama de un árbol, es posible obtener los objetos de reemplazo de una misma hoja, lo que provoca finalmente la eliminación de dicha hoja y por lo tanto tener nodos eliminados al finalizar la operación (por ejemplo, ver tabla 4.7), lo que disminuye el tamaño total de la estructura.

Los gráficos a continuación presentados muestran los costos totales de reads, seeks y writes sobre disco para eliminaciones del 10 % y 40 % de los objetos para ambos espacios. Es importante mencionar que el aspecto de las curvas para las lecturas y movimientos son proporcionales a las evaluaciones de distancia realizadas. Respecto de las escrituras, las curvas son lineales y el promedio para ambos porcentajes siempre se mantiene inferior a 2 escrituras por cada objeto eliminado, lo que demuestra un excelente comportamiento para esta medida de costo del método de *planos fantasmas* (una escritura cuando es hoja y dos cuando no).

En este respecto, el método de planos fantasmas supera enormemente a la opción de reconstrucción de subárboles, no solamente en el caso de las evaluaciones de distancia (del orden de los 63 millones para eliminación del 40 % en el espacio de palabras), sino también respecto de la cantidad de lecturas y escrituras necesarias para la reconstrucción luego de eliminaciones.

Los gráficos mostrados en esta sección corresponden a información adicional a los realizados en la sección 4.4.3, es decir, optimizados y tomando en cuenta los casos especiales indicados en la sección 4.3.4.

Los gráficos 5.2, 5.3 y 5.4 son información adicional al gráfico 4.12 de eliminaciones en el diccionario y los 5.5, 5.6 y 5.7 adicionales al gráfico 4.14 de eliminaciones en el espacio de Gauss. Todos los gráficos muestran los costos promedios por elemento para ambos porcentajes de datos eliminados.

Para los experimentos mostrados, la operación es como sigue: se realiza la carga de la estructura de memoria secundaria a memoria principal, es decir, la cantidad de nodos soportados en RAM, lo que depende de la cantidad de centros de cada estructura. En los gráficos los costos esta carga no es considerada. Posteriormente a esto se realizan las eliminaciones.

5.3. Búsquedas

Al igual que para los experimentos de eliminaciones, primero se carga parte del índice en memoria principal, de acuerdo a la estructura, y posteriormente se realizan las búsquedas. La carga del índice

Splits	Total	Incompletos	% Incompletos	PromUso	% del nodo usado	
4	31544	17103	54,22	1,65	41,36	68,21
8	22281	16670	74,82	2,47	30,87	48,28
16	16857	14669	87,02	3,48	21,75	31,91
20	15405	13787	89,50	3,90	19,48	27,93

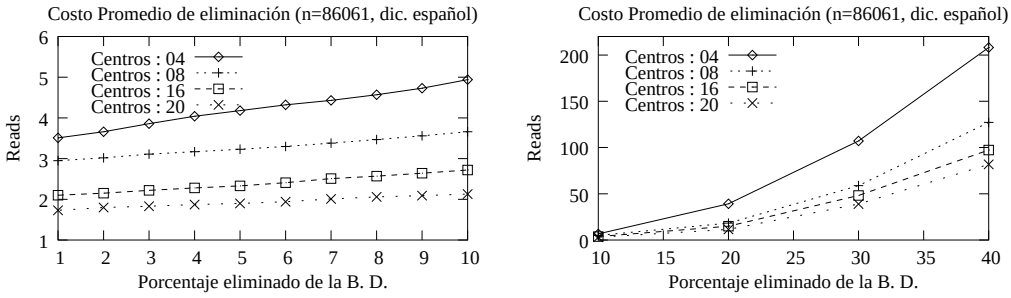
Tabla 5.2: Información de los nodos para el diccionario Español.

Splits	TotalNodos	SizeNodo	Espacio	%Ram	%Nodos	NivelRam
4	37606	352	12,62Mb	15,84	5958	7 de 14
8	27166	944	24,46Mb	8,18	2222	4 de 10
16	21155	2896	58,43Mb	3,42	724	3 de 7
19	19747	3892	73,29Mb	2,73	539	3 de 6

Tabla 5.3: Información general de la estructura para espacio de vectores de Gauss con dimensión 10 (100 % de los datos procesados).

Splits	Total	Incompletos	% Incompletos	Prom. uso	% del nodo usado	
4	37606	21440	57,01	1,65	41,20	66,48
8	27166	21153	77,87	2,45	30,67	46,01
16	21155	18809	88,91	3,32	20,76	29,54
19	19747	17803	90,16	3,54	18,64	26,65

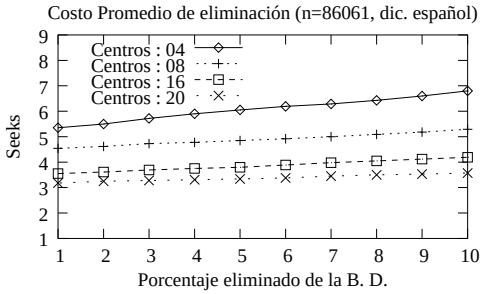
Tabla 5.4: Información de los nodos incompletos y promedio de uso del nodo para Gauss, dim. 10.



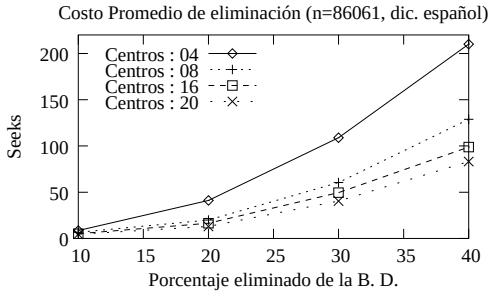
(a) 10 %

(b) 40 %

Figura 5.2: Costos promedios de lectura en disco por elemento para eliminaciones en el diccionario español.

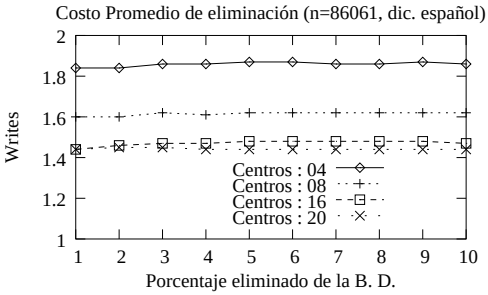


(a) 10 %

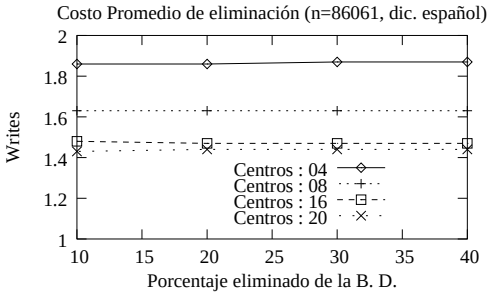


(b) 40 %

Figura 5.3: Costos promedios de seeks en disco para eliminaciones en el diccionario español.

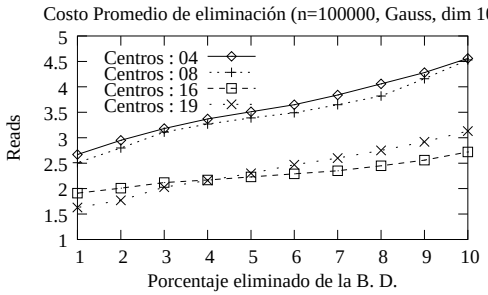


(a) 10 %

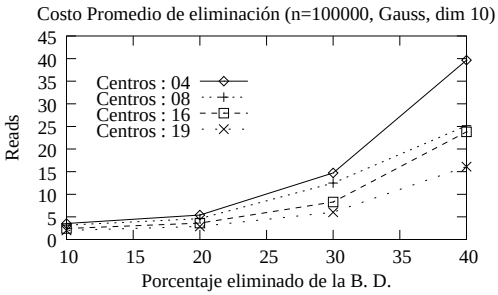


(b) 40 %

Figura 5.4: Costos promedios de escrituras sobre disco para eliminaciones en el diccionario español.

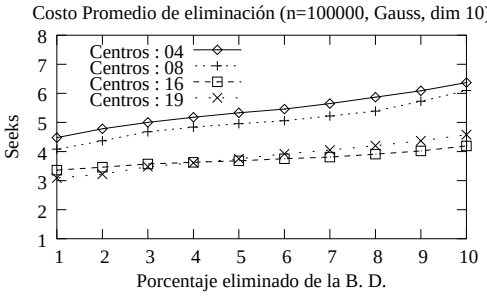


(a) 10 %

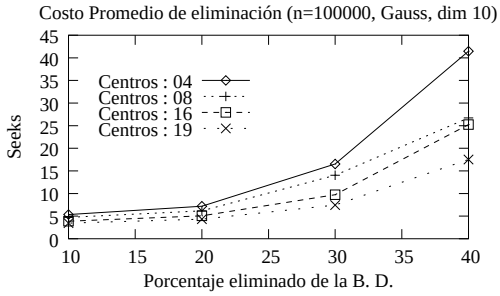


(b) 40 %

Figura 5.5: Costos promedios de lectura en disco para eliminaciones en el espacio de Gauss de dimensión 10.

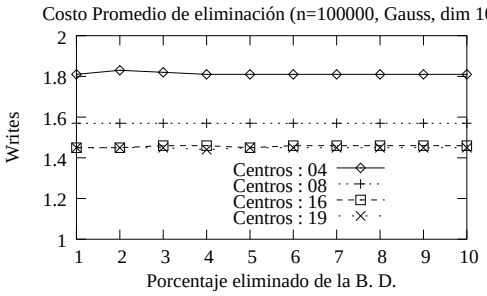


(a) 10 %

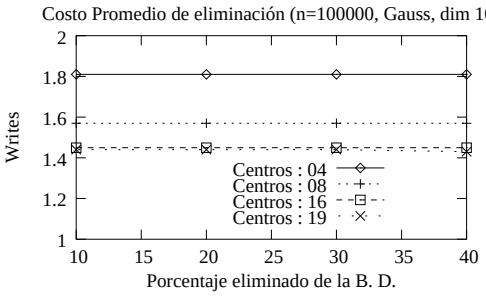


(b) 40 %

Figura 5.6: Costos promedios de seeks en disco para eliminaciones en el espacio de Gauss de dimensión 10.



(a) 10 %



(b) 40 %

Figura 5.7: Costos promedios de escrituras en disco para eliminaciones en el espacio de Gauss de dimensión 10.

no es considerada dentro de los costos de búsqueda porque se realiza una sola vez. Los gráficos de la figura 5.8 corresponden a la información complementaria para los costos en evaluaciones de distancia de la búsqueda por rango para el diccionario en castellano y el espacio de Gauss presentados en la figura 4.16. Los gráficos *a* y *b* en información complementaria para los gráficos *a* y *b* de la figura 4.10.

5.4. Uso de Páginas y Espacio en Memoria Secundaria

En esta sección se analizará cómo resolver el problema del bajo promedio de uso de las páginas de disco y en consecuencia el exceso de espacio requerido para almacenar la estructura.

5.4.1. Análisis y Alternativas

En la sección 5.1 se dedujo que el mayor problema de la estructura en estudio es referente al mal uso del espacio dentro de una página y por ende el elevado tamaño de la estructura al finalizar la construcción.

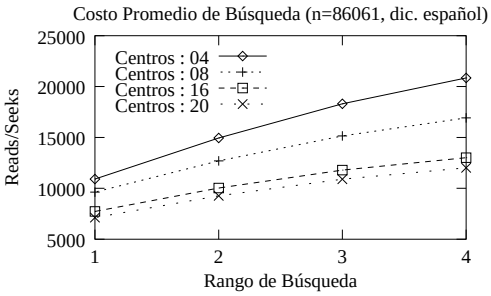
En experimentos sobre el espacio de palabras se demostró que una buena elección de centros puede reducir el número de nodos del árbol. En este experimento se eligió la raíz del árbol con los centros más adecuados de un subconjunto de la base de datos. La cantidad de nodos se redujo en un 0.98 %, y en un 1.07 % los nodos incompletos, por lo tanto aumentó del promedio de uso del nodo. Ahora bien, la elección de los centros más adecuados para un nodo es un tópico aparte en estructuras métricas, y en el caso de una estructura como el gnat, esto lograría balancear el árbol, pero no necesariamente aumentar notablemente el promedio de uso. Además, lo anterior conlleva un cálculo extra en evaluaciones de distancia en el preproceso de la base de datos, y en el caso de estructuras dinámicas no sería sostenible en el tiempo, debido a los continuos cambios que sufre el índice producto de eliminaciones y nuevas inserciones. Por lo tanto, una división homogénea del espacio o una buena distribución de los datos dentro del árbol no es una buena vía para la reducción del espacio requerido por la estructura.

Una alternativa que resulta interesante es utilizar la versión del gnat con radio cobertor. Con esto se reduce la cantidad de información que posee cada centro y el espacio para almacenarlo, y por ende aumenta la cantidad de datos que puede almacenar un nodo.

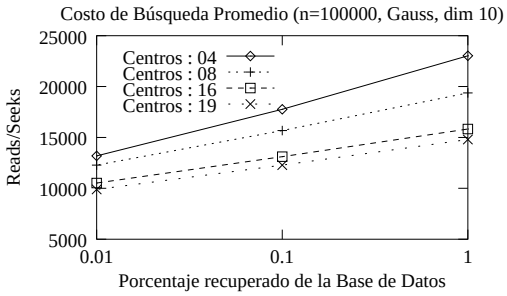
5.4.1.1. *Gnat* con Radio Cobertor

Cabe destacar que, en el proceso de construcción las estructuras que usan radio cobertor disminuyen las escrituras y tienen costos similares o mejores respecto de las cantidades de evaluaciones de distancia.

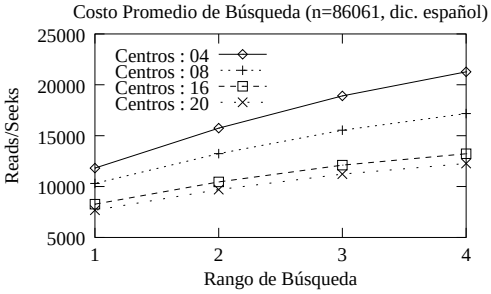
Para tener una idea de las diferencias entre el *gnat* con tabla de rangos (original) y con radio cobertor, se muestran una serie gráficos comparativos de distintos experimentos. La figura 5.9



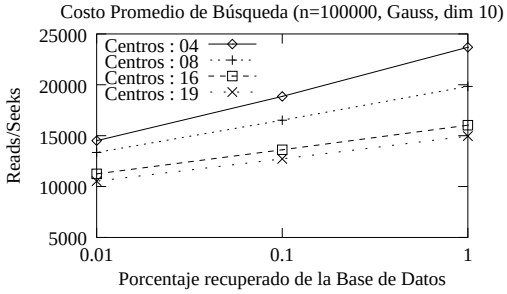
(a) Sin eliminaciones



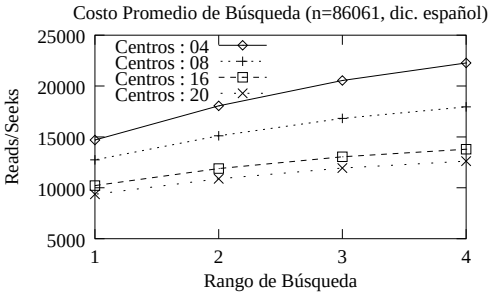
(b) Sin eliminaciones



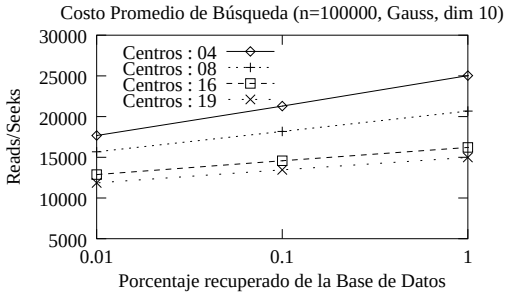
(c) 10 % eliminado y reinsertado



(d) 10 % eliminado y reinsertado

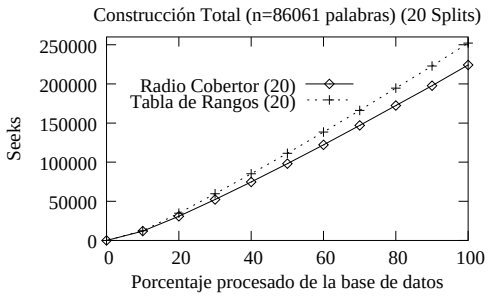


(e) 40 % eliminado y reinsertado

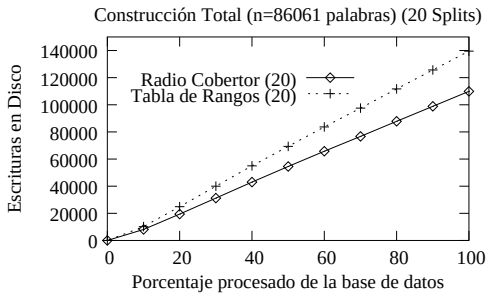


(f) 40 % eliminado y reinsertado

Figura 5.8: Lecturas y movimientos en disco para búsquedas por rango sobre el diccionario en castellano (columna izquierda) y el espacio de Gauss de dimensión 10 (columna derecha).



(a) Seeks 20 centros



(b) Writes 20 centros

Figura 5.9: Movimientos y escrituras para *gnat* con tabla de rangos v/s radio cobertor para el diccionario en castellano.

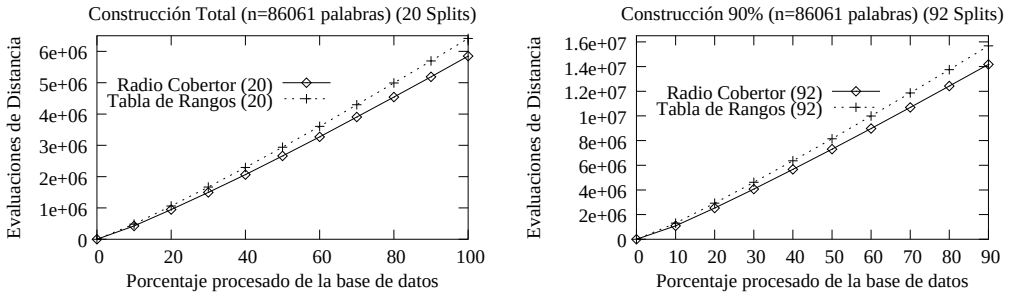
corresponde a los resultados para la construcción con 20 centros por nodo para el diccionario en castellano. Cabe recordar que para radio cobertor con 20 centros un nodo no completa una página de 4Kbytes. Sin embargo, para los accesos a disco se consideró que para ambos casos la cantidad de nodos en memoria principal era la misma. La figura 5.9 representa la cantidad de movimientos de cabezal y las escrituras; las lecturas son equivalentes dado que el objeto siempre se ubica en la misma posición si los datos son insertados en el mismo orden, independientemente del método para descartar subárboles (rangos o cobertor). Esto también indica que el árbol tiene la misma forma.

Los experimentos arrojaron resultados positivos desde el punto de vista de los accesos a disco y del espacio total requerido por la estructura, 13.864.500 bytes (nodo de 900 bytes) versus los 61.866.480 bytes (nodo de 4016 bytes) para el gnat con tabla de rangos.

En términos reales, las comparaciones deberían hacerse entre páginas del mismo tamaño, es decir, para el *gnat* con tabla de rangos de 20 centros por nodo, su equivalente con radio cobertor es de 92 centros por nodo. Las figuras 5.10 y 5.11 muestran una serie de gráficos comparativos entre estas distintas cantidades de centros. Estos gráficos demuestran de manera experimental lo dicho anteriormente, es decir, para igual cantidad de centros, si bien la opción de radio cobertor en la construcción las evaluaciones de distancia disminuyen, para la búsqueda por rango éstas tienen un aumento notorio.

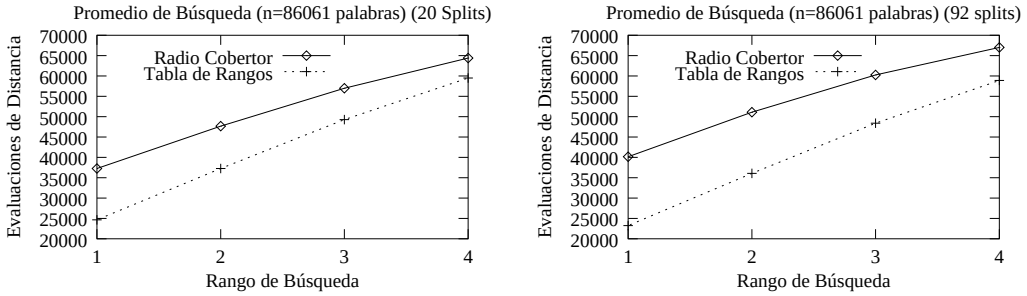
Finalmente en el gráfico de la figura 5.12 se observan las diferencias en las cantidades de lecturas entre el *gnat* cobertor (92 centros) versus *gnat* con tabla de rangos (20 centros). Esta figura es relevante, dado que aquí las páginas son iguales y como el gnat con radio cobertor genera menos nodos (9536) la cantidad de lecturas es bastante menor que el caso de uso de una tabla de rangos. La estructura final para la construcción con radio cobertor (92 centros) es de 38.792.448 bytes para un nodo de 4068 bytes, lo que logra disminuir bastante el espacio requerido.

Si se observan los gráficos de evaluaciones de distancia (5.10 y 5.11), se puede deducir que el comportamiento del gnat con radio cobertor aumenta considerablemente esta medida de costo (para



(a) construcción 20 centros (b) construcción 92 centros

Figura 5.10: Evaluaciones de distancia para la construcción del *gnat* con tabla de rangos v/s radio cobertor para el diccionario en castellano.



(a) busqueda por rango (20 centros) (b) busqueda por rango (92 centros)

Figura 5.11: Evaluaciones de distancia para la búsqueda por rango sobre *gnat* con tabla de rangos v/s radio cobertor sin eliminaciones para el diccionario en castellano.

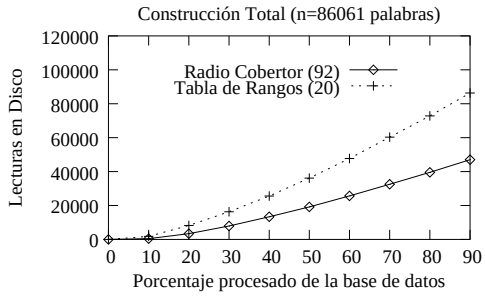
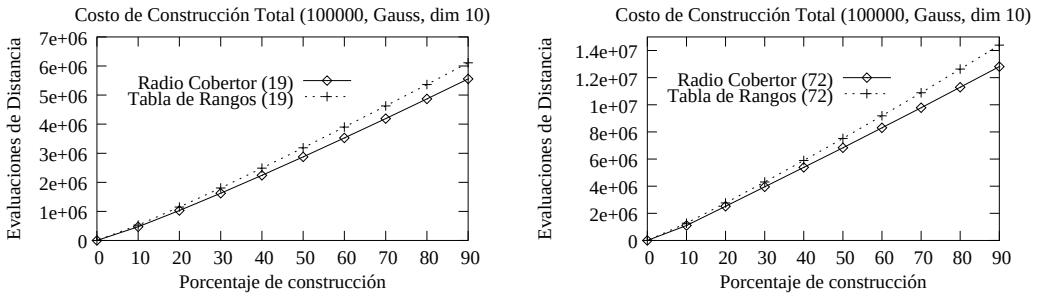


Figura 5.12: Lecturas *gnat* cobertor (92 centros) v/s *gnat* con tabla de rangos (20 centros).



(a) 19 centros

(b) 72 centros

Figura 5.13: Evaluaciones de distancia para la construcción del *gnat* con tabla de rangos v/s radio cobertor para el espacio de Gauss de dimensión 10.

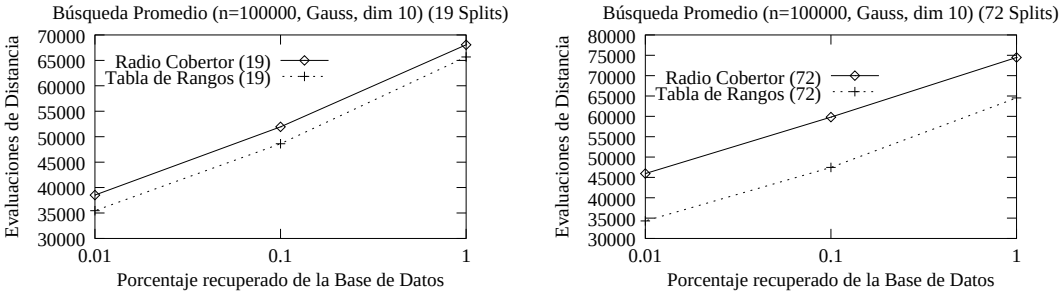
páginas de tamaño similar), tanto en la construcción como para las búsquedas. Para ello hay que comparar las versiones tabla de rangos de 20 v/s cobertor de 92. Es importante mencionar que en el caso de la versión original del gnat, si se aumenta la cantidad de centros, se disminuyen los costos de búsqueda. Sin embargo, usando radio cobertor, esto no resulta igual, lo que quiere decir que para estructuras con radio cobertor no necesariamente un aumento en la cantidad de centros implica una disminución en la cantidad de evaluaciones de distancia al momento de la búsqueda. Lo anterior es debido a que para las estructuras con radio cobertor las aridades adecuadas para buscar son mucho menores a 92 centros.

Sumado a lo anterior, el problema del promedio o porcentaje de uso del nodo no ha sido superado, sólo se ha disminuido la cantidad de nodos, producto de la mayor capacidad del nodo. Los valores obtenidos para el diccionario con radio cobertor y número de centros 92 son:

- Cantidad total de nodos: 9536
- porcentaje de nodos incompletos: 98.1 %
- promedio de uso del nodo (total 92): 7.4

Para tener una referencia del comportamiento sobre un espacio de vectores, se presentan en las figuras 5.13 y 5.14 resultados experimentales para el espacio de vectores de Gauss de dimensión 10 para páginas de tamaño equivalente. Los tamaños de la estructura para los experimentos de Gauss fueron 32.849.564 bytes (4.052 bytes por nodo) para radio cobertor de 72 centros versus 76.885.324 bytes (3.892 bytes por nodo) del gnat con tabla de rangos con 19 centros.

En la siguiente subsección se describirá una modificación a la estructura para superar el problema del bajo promedio de uso del nodo.



(a) 19 centros

(b) 72 centros

Figura 5.14: Evaluaciones de distancia para la búsqueda por rango sobre *gnat* con tabla de rangos v/s radio cobertor sin eliminaciones para el espacio de vectores con distribución de Gauss de dimensión 10.

5.4.2. *Gnat Evolutivo (Evolutionary Gnat)*

Uno de los objetivos de haber elegido el *gnat* como estructura es que ésta no posee solapamiento se planos, producto de que cada objeto al ser insertado tiene sólo una posición posible dentro de la estructura.

Al no tener solapamiento las búsquedas son menos costosas, sin embargo, esta importante característica es también la que provoca el bajo promedio de uso de los nodos y por lo tanto el tamaño excesivo de la estructura.

Para aumentar el promedio de uso y mantener la característica anteriormente mencionada, se propone una nueva estructura. Ésta poseerá distintos tipos de nodos o páginas sobre las cuales se realizarán distintas operaciones. Se definen tres tipos de nodos, los cuales son:

1. *Nodo Bucket*
2. *Nodo gnat con Bucket*
3. *Nodo gnat*

La información general de cada nodo es:

- ubicación en memoria secundaria.
- ubicación en memoria secundaria del padre.
- tipo del nodo.

Cada uno de estos nodos, bajo ciertas condiciones, mutará o evolucionará al siguiente tipo de nodo.

El nodo *bucket* es una bolsa que contiene sólo objetos sin ninguna información adicional, la mayor cantidad posible. El nodo *gnat con bucket* tiene las tablas de rangos pero los centros no tienen hijos,

este posee un sólo hijo que se desprende del nodo y es del tipo bucket. El nodo *gnat* es tal como la definición original de la estructura, además, los centros de éste pueden apuntar a nodos de tipo gnat, gnat con bucket o bucket. Las restricciones impuestas a las mutaciones de los nodos son: un nodo *gnat* es un nodo completamente evolucionado, por lo tanto siempre permanece del mismo tipo, un nodo de tipo *bucket* no puede tener hijos y por lo tanto sólo posee padres del tipo *gnat* o *gnat con bucket*.

Supongamos las siguientes definiciones:

- MAX_SPLITS: indica la cantidad máxima de splits que posee un gnat.
- MAX_SIZE_BUCKET: cantidad máxima de objetos que puede tener un bucket.

Un nodo siempre nace como nodo *bucket* y el evento que lo hace evolucionar es cuando éste se llena de objetos. En este momento se convierte en un nodo *gnat con bucket*, es decir, un nodo que tiene la forma de un gnat, pero que posee sólo un hijo. Al ocurrir este evento se reinsertan los primeros MAX_SPLITS elementos del nodo bucket original sobre el gnat; el resto de los objetos de copia al hijo *bucket* y no se calculan las distancias sobre la parte *bucket*.

Las transformaciones o mutaciones sólo ocurren cuando se llenan los nodos de tipo *bucket*, por ejemplo, si un *gnat con bucket* llena su hijo *bucket*, entonces, los objetos del hijo se reinsertan todos sobre la parte *gnat* transformándolo en un *gnat* completo con las tablas de rangos actualizadas. En términos reales en este caso los splits del gnat apuntarán inicialmente sólo a buckets.

Con esto, el promedio de uso del *nodo bucket* después de transformaciones será siempre mayor o igual a $MAX\_SIZE\_BUCKET - MAX\_SPLITS + 1$ si es hijo de un *gnat con bucket* y a $(MAX\_SIZE\_BUCKET + 1)/MAXSPLITS$ si es hijo de un nodo *gnat*. Los nodos *gnat y gnat con bucket* siempre estarán llenos (100 % de uso del nodo). Si existen eliminaciones posteriores, entonces los promedios podrían bajar y sería posible encontrar los otros tipos de nodos no completamente llenos. En la figura 5.15 se muestra el proceso de transformación de un nodo. Inicialmente se deja de lado la alternativa de tener sólo dos tipos de nodos, es decir, un nodo *bucket* que evolucione a un nodo *gnat*, esto dado que el promedio sería en ese caso peor, $(MAX\_SIZE\_BUCKET - MAX\_SPLITS + 1)/MAXSPLITS$.

Algoritmo:

1. Se insertan objetos en el nodo hasta MAX_SIZE_BUCKET (la primera vez sobre la raíz del árbol),
2. Al insertar el objeto siguiente se provoca la primera transformación (paso *a* de la figura 5.15). Esto genera dos nodos, el primero con las tablas de rangos calculadas en base a los primeros MAX_SPLITS objetos (parte *gnat*), el segundo con el resto, insertado sin realizar cálculos (parte *bucket*).

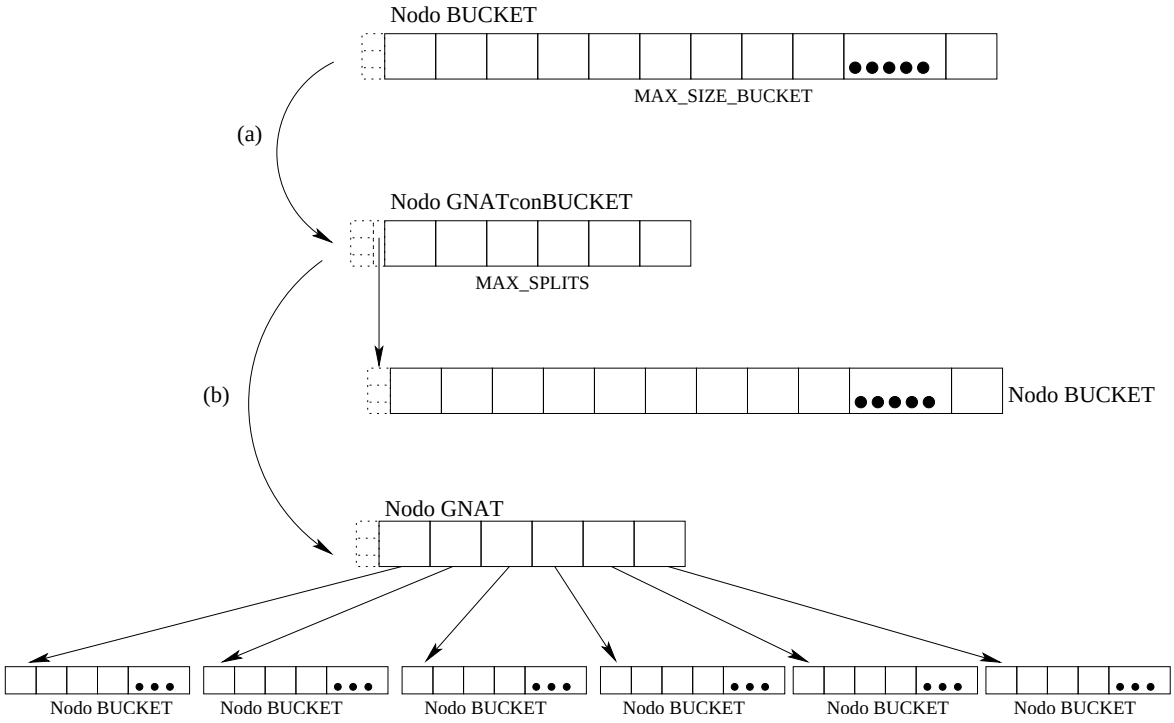


Figura 5.15: *egnat*: mutaciones de un nodo.

3. Los nuevos objetos se insertan objetos sobre la parte *bucket*, sin realizar evaluaciones de distancia sobre el padre. El siguiente objeto después de llenarse este nodo provoca la segunda transformación (paso *b* de la figura 5.15), la cual reinserta todos los objetos de la parte *bucket* (nodo *bucket* hijo de un *gnat con bucket*) sobre su padre, actualizando los rangos y creando un máximo de *MAX_SPLITS* hijos de tipo *bucket*. El nodo *bucket* original desaparece.

4. Después de la segunda transformación cada vez que se llena un nodo de tipo *bucket* se repite el proceso recursivamente.

5.4.2.1. Comparaciones *Gnat* vs *Gnat Cobertor* vs *Egnat*

A continuación se presenta información comparativa entre las distintas versiones de la estructura para los espacios de vectores de Gauss de dimensión 10 de 100.000 objetos y el espacio de palabras en español de 86.061 datos. Las estructuras son: el gnat en su versión original, gnat con radio cobertor y el gnat evolutivo. Para las tres estructuras el tamaño del nodo es el mismo y representa una página de disco de 4096 bytes. La tabla 5.6 muestra detalles de la construcción para las tres estructuras. Nótese que si bien el porcentaje de páginas o nodos incompletos sigue siendo elevada, los promedios de uso de la página van en aumento, siendo el gnat evolutivo el de mejor promedio. En este respecto, debe llamar la atención que para el espacio de palabras el promedio supera la capacidad máxima de un nodo. Este efecto es producido debido a que la capacidad de un nodo de tipo bucket es muy

	Vectores de Gauss de dim. 10			Diccionario Castellano		
	gnat	gnat Cobertor	egnat	gnat	gnat Cobertor	egnat
MAX_SPLITS	19	72	19	20	92	20
% páginas incompletas	90,16	96,99	93,78	89,50	98,08	93,86
Prom. uso del nodo	5,06	12,34	17,82	5,59	9,02	21,21
Nro Páginas	19.747	8.107	5.611	15.405	9.536	4.057
Nivel máximo	5	3	4	7	3	5
Cálculos de Distancia	6.888.432	14.348.952	5.700.150	6.414.657	16.049.768	5.111.072

Tabla 5.6: Construcción: valores comparativos entre *gnat*, *gnat cobertor* y *egnat* con el 100 % de los datos.

	vectores de Gauss	diccionario en castellano
MAX_SIZE_BUCKET	102	145
Cantidad de buckets	5.267	3.809
Cantidad de gnat con bucket	45	29
cantidad de gnat	299	219
promedio de uso bucket	17,75	21,29
prom. de uso de bucket con padre gnat con bucket	91,51	135,9

Tabla 5.7: *egnat*: detalles de construcción.

superior a uno de tipo gnat, lo cual influye en dicho cálculo.

En relación a la construcción, esta nueva estructura posee buen rendimiento, tanto desde el punto de vista del espacio (número de páginas de disco) como desde las cantidades de cálculos de distancia, lecturas, movimientos de cabezales y escrituras en disco. Los gráficos de la figura 5.16 muestran el desempeño de los distintos algoritmos para el proceso de construcción.

El problema de esta estructura radica en la búsqueda. La figura 5.17 muestra el desempeño de los distintos algoritmos para la búsqueda por rango. De los gráficos se observa un dramático aumento en la cantidad de evaluaciones de distancia para la nueva estructura. Observando la información adicional proporcionada en la tabla 5.7, se puede deducir que este incremento se debe a la gran cantidad de nodos de tipo bucket generados para ambas estructuras, como estos nodos no poseen información adicional, entonces, al llegar a uno de estos nodos se debe realizar la consulta directa sobre cada uno de los objetos.

Para disminuir el costo en evaluaciones de distancia, la alternativa natural sería eliminar una de las transformaciones de la estructura y hacer que ésta sólo tenga dos tipos de nodo, un nodo de tipo bolsa o bucket que pueda evolucionar inmediatamente a un nodo de tipo gnat. Con esto la cantidad de gnats dentro de la estructura aumenta y es posible discriminar más subárboles durante la búsqueda, esto sin embargo, implicaría un baja en el promedio y por ende un aumento del espacio utilizado por la estructura. En el cuadro 5.8 se muestra los detalles de construcción para el espacio

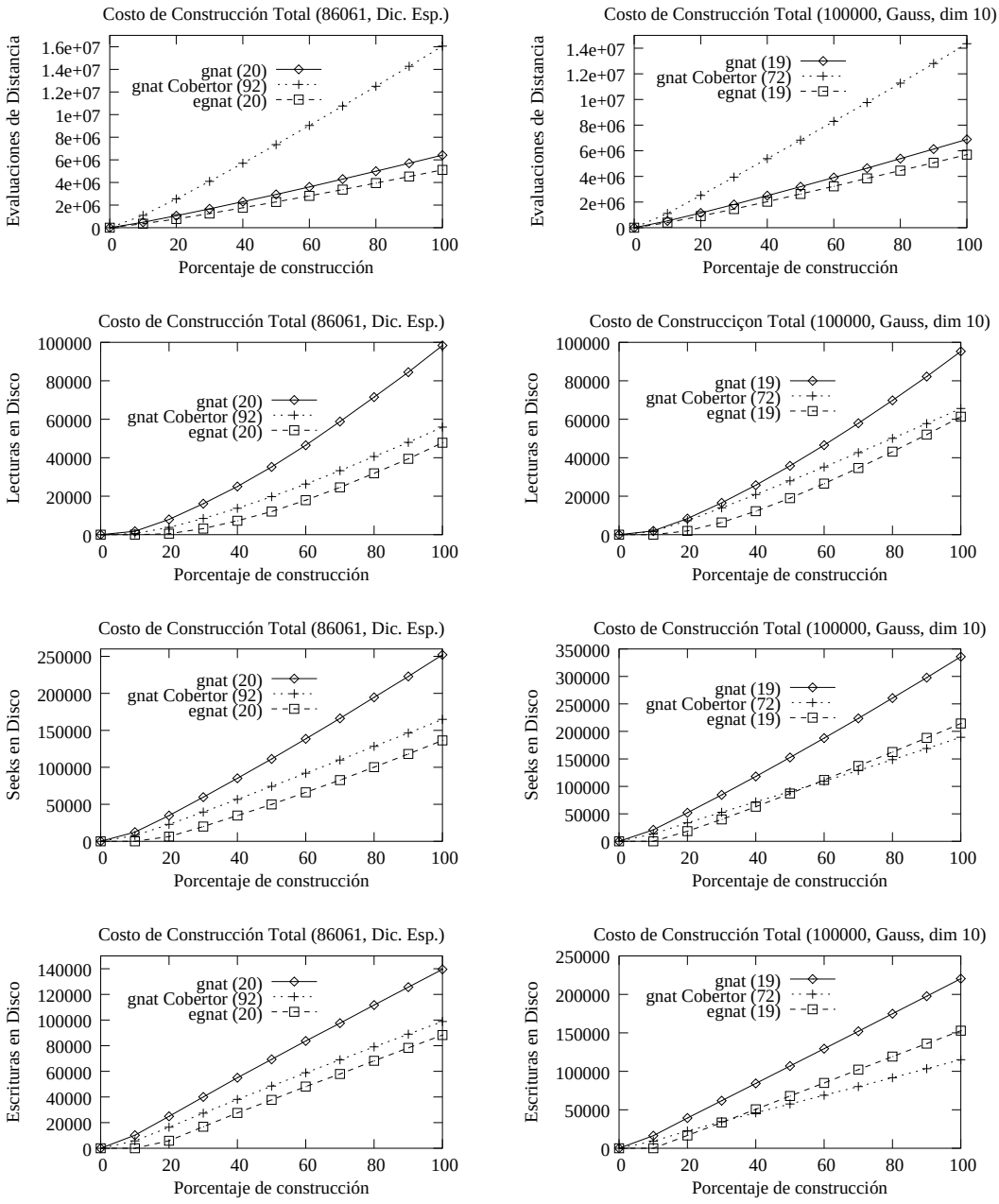


Figura 5.16: Comparativas de construcción para el diccionario en Español (izquierda) y el espacio de Gauss de dimensión 10 (derecha) con el 100 % de los datos: *gnat*, *gnat cobertor* y *egnat*.

	vectores de Gauss
Nro. de Páginas	6.354
% de páginas incompletas	94,56
Prom. de uso del nodo	15,74
Cantidad de buckets	6.008
cantidad de gnat	344
promedio de uso bucket	15,55

Tabla 5.8: *egnat* (versión 2): detalles de construcción.

de Gauss para la versión modificada del egnat, en éste la cantidad de centros por nodo y el tamaño del bucket es igual a la versión inicial.

De la figura 5.18 que muestra la construcción y búsqueda sobre el espacio de Gauss para las dos versiones del egnat, se observa que la ganancia es mínima al evitar una mutación en el nodo, esto producto de que la cantidad de nodos bucket sigue siendo alta.

En resumen, la única manera de reducir las evaluaciones de distancia es agregando información adicional a los objetos ubicados en los buckets. Por supuesto esta información no deben ser los rangos, dado que sería como el gnat original, mantener los radios cobertores tampoco sería de utilidad debido que igual implicaría la consulta directa sobre todos los objetos. En la siguiente sección se presentará una versión final para el gnat evolutivo que cumplirá con todos los requisitos esperados para una estructura métrica.

5.4.2.2. *Egnat* Versión Final

La idea de agregar información extra al bucket es debido a que si bien el promedio de uso en éstos es alto, no es superior a 20 en el caso de vectores de Gauss, siendo que el bucket tiene capacidad para 102 objetos en este espacio.

La idea final para la información que tendrán las bolsas (buckets) es propia de las estructuras basadas en pivotes, es decir, mantener la distancia del objeto a un pivote cualquiera. En este caso el pivote será el split o centro del cual es hijo el bucket. Lo interesante de esto es que no se paga un costo adicional en evaluar esta distancia ya que fue previamente calculada para determinar donde insertar el objeto. Lo anterior tampoco reduce en exceso el tamaño del bucket, por ejemplo en el caso de los vectores de Gauss de dimensión 10 el bucket quedo con capacidad para 92 y los nodos gnat conservan los 19 splits.

Manteniendo la distancia al centro se puede usar la desigualdad triangular para evitar el cálculo directo de la distancia de un objeto consulta sobre los objetos ubicados en los buckets de la siguiente manera:

- Sea q el objeto consulta, p el split que posee un hijo bucket, s_i los elementos ubicados dentro del bucket, r el radio de búsqueda e I_p la marca de borrado insertada en p si este fuese un

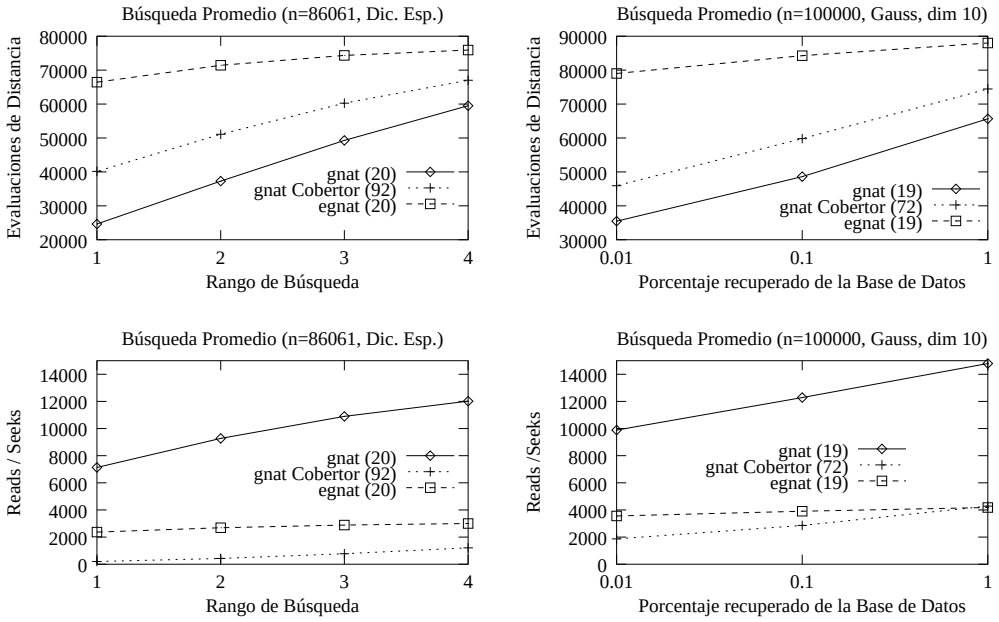


Figura 5.17: Comparativas de búsquedas promedio por rango para: *gnat, gnat cobertor* y *egnat* (versión inicial).

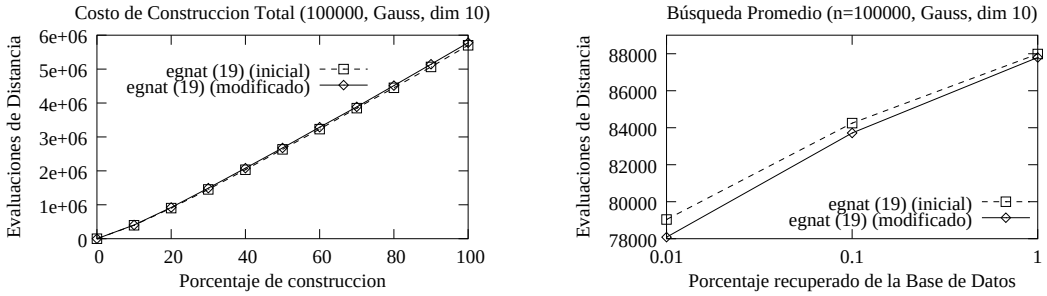


Figura 5.18: egnat: versión inicial (tres tipos de nodos) vs modificado (dos tipos de nodos).

plano fantasma (I: distancia del objeto eliminado al objeto usado como reemplazo), entonces, si

$$Dist(s_i, p) > Dist(q, p) + r + I_p \qquad o \qquad Dist(s_i, p) < Dist(q, p) - r - I_p$$

se puede determinar que el objeto s_i no está dentro del rango de búsqueda, de lo contrario hay que necesariamente hacer la comparación directa.

Esto se utiliza también para la búsqueda por rango 0 en las bolsas, lo que reduce considerablemente la cantidad de evaluaciones al momento de eliminar objetos.

Finalmente la estructura *egnat* tendrá la forma planteada inicialmente con las siguientes características:

- Tendrá dos tipos de nodo: nodo bucket y nodo gnat.
- El nodo bucket evolucionará directamente a un nodo de tipo gnat.
- Los objetos dentro de un bucket mantienen la distancia al centro del cual son hijos.
- El nodo gnat es de la forma original, es decir, contiene tablas de rangos.

Para el egnat se determinó que no tendrá involución, es decir, una vez transformado un bucket en gnat, este no podrá volver a ser bucket durante los procesos de eliminación. Esto principalmente debido a que los costos de transformación y de creación de las tablas de rango ya fueron absorbidos durante la construcción, por lo tanto durante futuras reinserciones este costo es evitado. Por lo demás, si un nodo involucionara, habría un costo adicional para obtener las distancias $Dist(s_i, p)$.

La tabla 5.10 muestra la nueva información para la versión final del egnat en la construcción usando el 100 % de los datos. La tabla 5.11 muestra los detalles de construcción de la nueva versión del egnat. Se puede observar en estas tablas un aumento en las páginas de disco con respecto a la versión inicial, pero sigue siendo aún considerablemente menor al gnat y menor que el gnat con radio cobertor. En las evaluaciones de distancia para la construcción sigue teniendo un mejor desempeño que las otras dos.

La figura 5.19 muestra información general para la construcción de la nueva estructura. En esta figura se puede ver que la nueva versión del egnat supera en casi todas las medidas de costo a las otra dos estructuras.

La figura 5.20 muestra los costos para la búsqueda por rango. Resulta interesante notar que la nueva estructura mantiene un excelente desempeño en ambos espacios, superando incluso a la versión original del gnat en el cálculo de evaluaciones de distancia. Desde el punto de vista de las lecturas en disco, es superado sólo por la versión que usa radio cobertor, sin embargo, la diferencia entre éstas es baja.

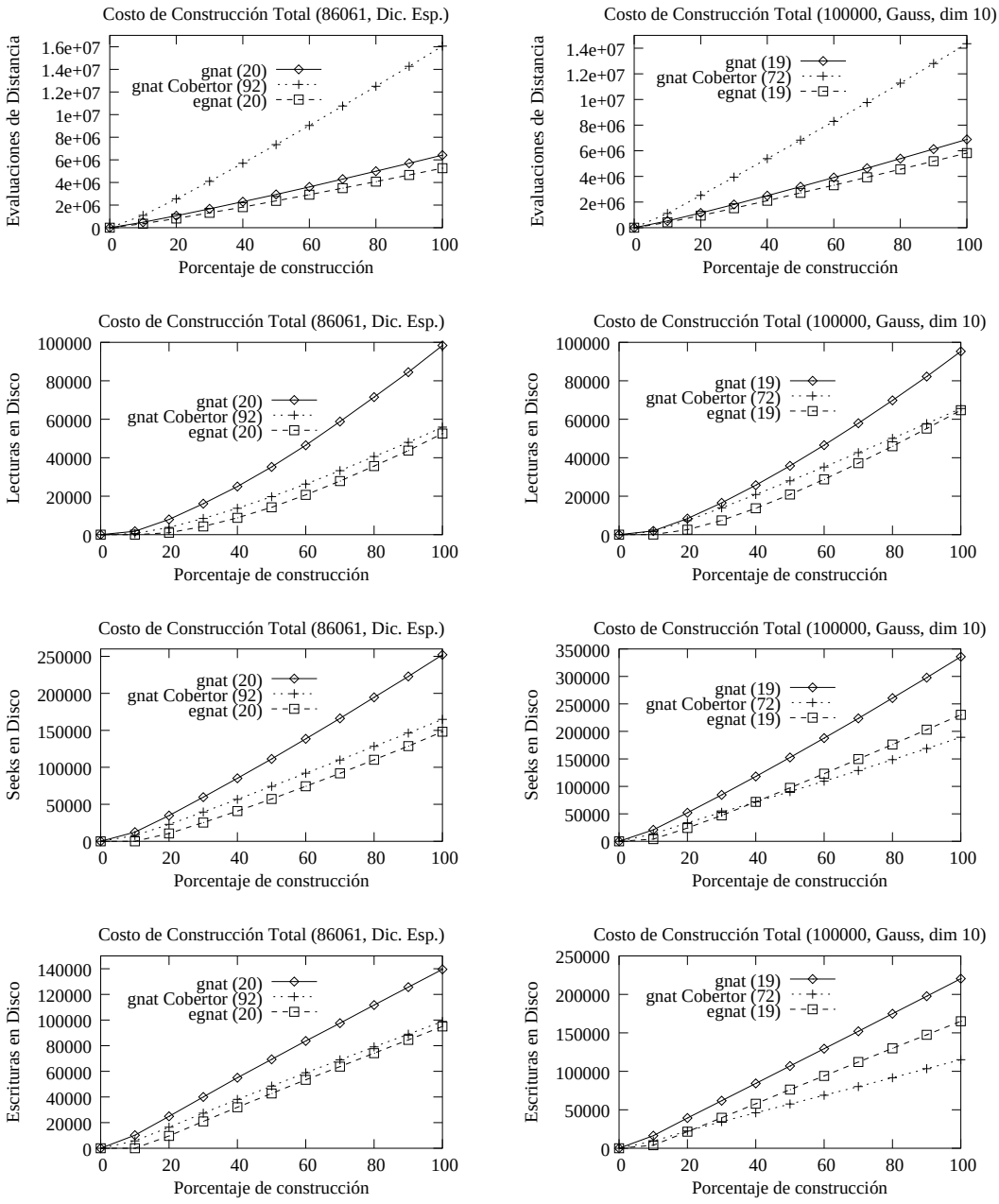


Figura 5.19: *egnat* (versión final): Costos totales de construcción para el diccionario de palabras (izquierda) y para el espacio de Gauss (derecha).

	Vectores de Gauss de dim. 10			Diccionario Castellano		
	gnat	gnat Cobertor	egnat	gnat	gnat Cobertor	egnat
MAX_SPLITS	19	72	19	20	92	20
% Nodos incompletos	90,16	96,99	94,57	89,50	98,08	94,48
Prom. uso del nodo	5,07	12.335	14,76	5.59	9,02	17,23
Nro Páginas	19.747	8,11	6.773	15.405	9.536	4.996
Nivel máximo	5	3	4	7	3	5
Cálculos de Distancia	6.888.432	14.348.952	5.818.458	6.414.657	16.049.768	5.260.224

Tabla 5.10: Construcción: valores comparativos entre *gnat*, *gnat cobertor* y *egnat* (versión final) con el 100 % de los datos.

	vectores de Gauss	diccionario en Español
MAX_SIZE_BUCKET	92	127
Cantidad de buckets	6.405	4.720
cantidad de gnat	368	276
promedio de uso bucket	14,52	17,06

Tabla 5.11: *egnat* (versión final): detalles de construcción (100 % de los datos).

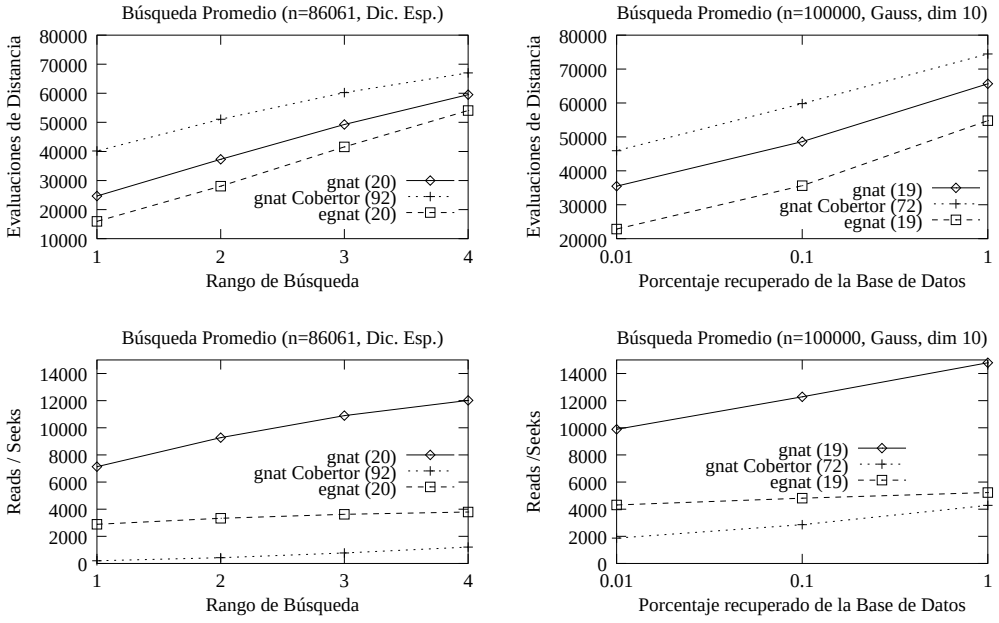


Figura 5.20: *egnat* (versión final): Búsquedas sin eliminación.

5.4.2.3. *Egnat* y *Planos Fantasmas*

Para la estructura egnat con eliminaciones, se definieron las siguientes características:

- Un nodo bolsa no tiene planos fantasmas, debido a que no tiene hijos.
- Una vez que un nodo bolsa evolucionó a uno del tipo gnat, queda permanentemente de este tipo. Esto independiente de si se eliminan todos sus hijos o queda incompleto.

Para los experimentos de eliminación sobre la versión final se consideró el siguiente caso especial:

- Un padre (gnat) de un nodo bolsa no tiene planos fantasmas. Es decir, al eliminar el centro, éste se reemplaza por uno de sus hijos en la bolsa y se recalculan todas las distancias reinsertando los objetos de la bolsa. Esto es para evitar la propagación de los planos fantasmas y para discriminar más subárboles durante la búsqueda.

Como se puede ver en las figuras 5.21 y 5.22, el caso anterior provoca un aumento en los cálculos de distancia durante la eliminación, siendo el espacio de palabras el de peor comportamiento durante la eliminación del 40 % de los datos. Sin embargo, este es el único costo adicional para la estructura. En la misma figura se puede observar que los costos de acceso a disco para el egnat son mucho menores que para el gnat.

Lo más relevante de la eliminación en la estructura egnat es que, si bien la búsqueda del dato a eliminar es más costosa que en la estructura original, producto del recálculo; las búsquedas por rango sobre la estructura con planos fantasmas son mejores.

Las figuras 5.23 y 5.24 muestran la información producida para las búsquedas con 10 % y 40 % eliminado y reinsertado sobre los dos espacios de ejemplo.

Es interesante notar el buen desempeño de la nueva estructura con el 10 % de datos eliminados y reinsertados, tanto en evaluaciones de distancia como en lecturas en disco. Con 40 % de datos eliminados la nueva estructura tiende a un comportamiento similar al gnat en el caso de las evaluaciones de distancia, manteniendo su ventaja en los accesos a disco, donde es notoriamente mejor.

5.4.2.4. Comparaciones *Egnat* vs *M-tree*

Como se menciona en la sección 3.5.1 el M-tree es una estructura dinámica y específicamente diseñada para memoria secundaria. Para la construcción del M-tree se modificó el código de la versión 0.911 (obtenida de http://www-db.deis.unibo.it/Mtree/). Se consideraron los siguientes parámetros permitidos por la estructura:

- MIN_UTIL: este parámetro es usado para garantizar un mínimo de llenado de una página. Para estos experimentos se utilizó 10 % y 40 % (el rango es entre 10 % y 50 %).

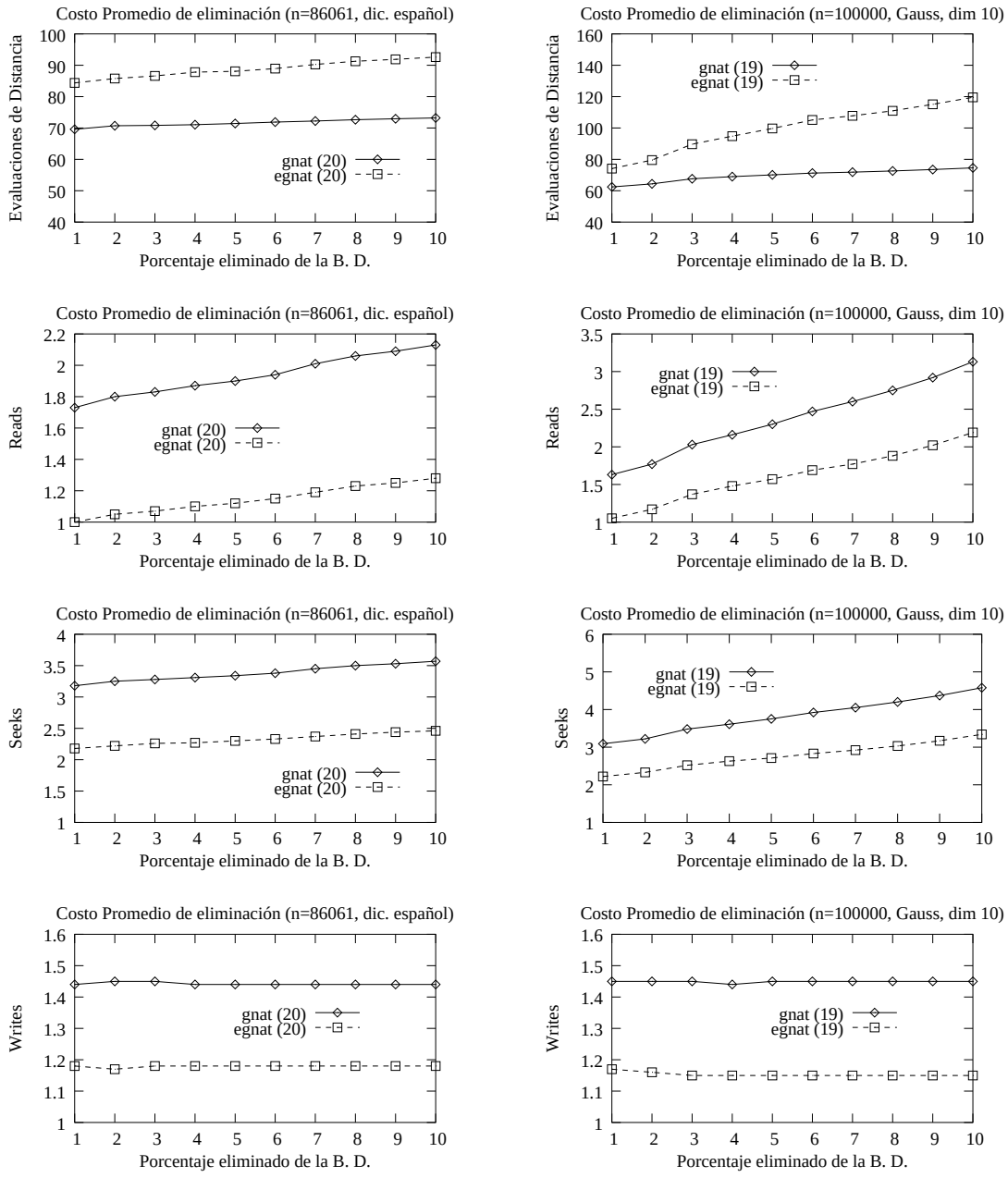


Figura 5.21: *egnat* (versión final): Costos promedios de eliminación del 10 % de los datos.

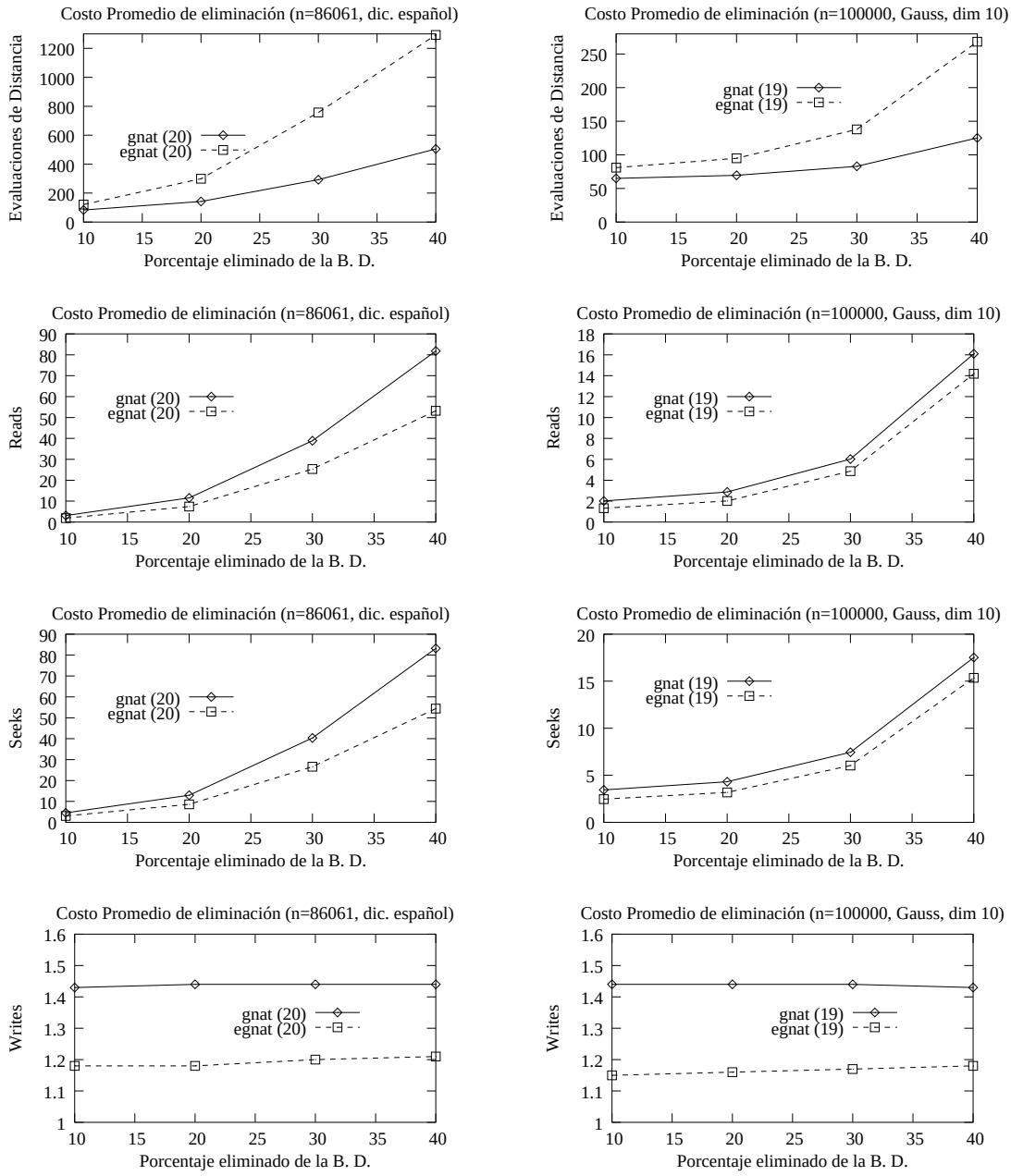


Figura 5.22: *egnat* (versión final): Costos promedios de eliminación del 40 % de los datos.

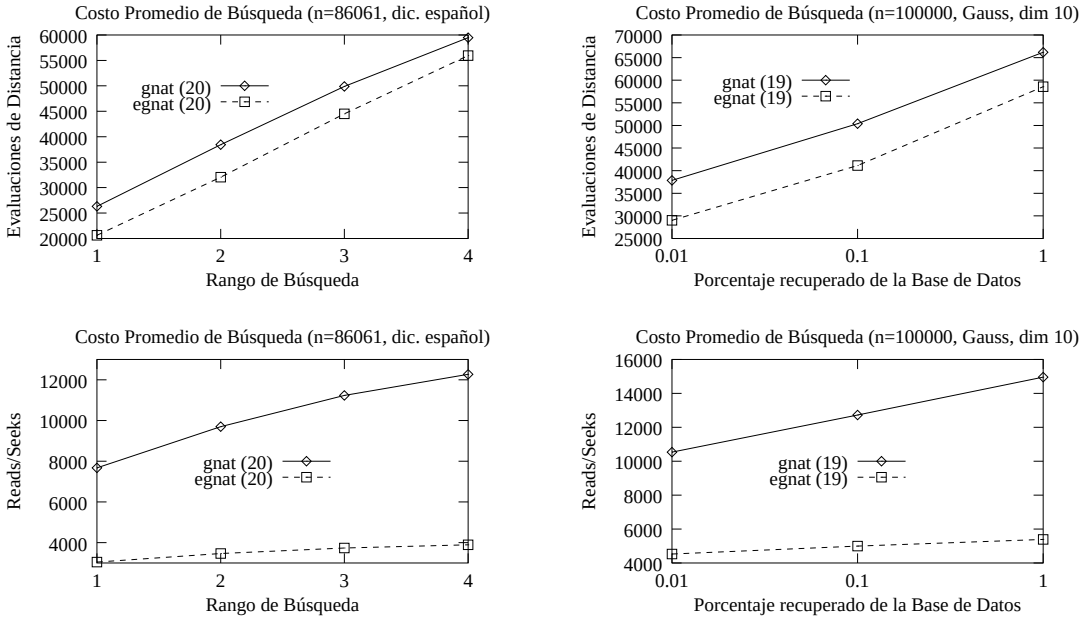


Figura 5.23: *egnat* (versión final): Búsquedas con 10 % de eliminaciones y reinserciones sobre los espacios de palabras (izquierda) y Gauss (derecha).

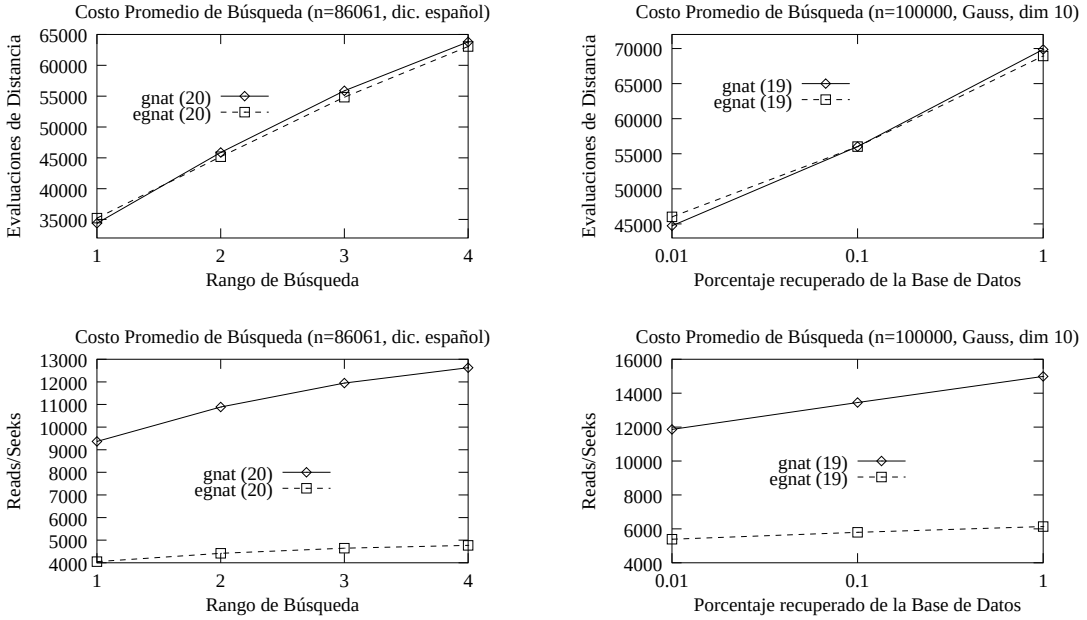


Figura 5.24: *egnat* (versión final): Búsquedas con 40 % de eliminaciones y reinserciones sobre los espacios de palabras (izquierda) y Gauss (derecha).

- SPLIT_FUNCTION: este parámetro permite elegir la forma de la partición de un nodo al llenarse. Se eligió para los experimentos la opción de generar un hiperplano general. Existen tres alternativas:

  - G_HIPERPL : estrategia de partición en un hiperplano generalizado.
  - BAL_G_HIPERPL: estrategia de partición en un hiperplano generalizado balanceado.
  - BALANCED: estrategia balanceada.

Para el parámetro SPLIT_FUNCTION, los últimos dos métodos permiten mantener los nodos perfectamente balanceados, sin embargo, producen un alto radio cobertor, lo que implica un efecto negativo en la búsqueda. El método elegido G_HIPERPL reduce la cantidad de evaluaciones de distancia en la búsqueda, sin embargo, aumenta cantidad de nodos de la estructura. Este parámetro puede ser sintonizado con MIN_UTIL para reducir este efecto. Las políticas de partición no están detalladas en [CPZ97], éstas pueden ser encontradas en [Pat99].

La estructura egnat tiene una ventaja considerable respecto al M-tree desde el punto de vista de los costos de búsqueda. La figura 5.25 muestra los costos de búsqueda para los dos espacios de prueba teniendo el egnat el mejor desempeño en el espacio de Gauss. Respecto de las lecturas en disco, el M-tree tiene una leve ventaja sobre el egnat debido principalmente al menor tamaño de la estructura.

La estructura egnat tiene un mayor desempeño sobre el M-tree en el espacio de Gauss. A modo de referencia, los tiempos de búsqueda para ambos programas ejecutados sobre el mismo equipo realizando las búsquedas de la figura 5.25 fueron de 15 minutos para el egnat y de 3 horas 3 minutos para el M-tree con MIN_UTIL de 0.4.

La figura 5.26 muestra la información de los costos de construcción para las estructuras. Respecto de las evaluaciones de distancia y a las escrituras en disco, la diferencia entre el egnat y el M-tree es levemente favorable al egnat en el espacio de palabras y levemente al M-tree en el espacio de Gauss. Para el caso de las lecturas, el M-tree es considerablemente más costoso dado que los algoritmos de inserción tienden a ahorrar espacio en disco al encontrar un mejor lugar para el objeto.

5.5. Discusión Final

En el presente capítulo se presentó una nueva estructura denominada *gnat evolutivo* o *egnat* y se demostró experimentalmente que muestra un excelente desempeño en las búsquedas, comparado tanto con la versión original del *gnat*, como con la versión modificada usando *radio cobertor*, como así también con otras estructuras como el *Voronoi-tree* y el *M-tree*. Desde el punto de vista de la construcción su comportamiento es similar, aunque en algunos casos con costos levemente más altos.

En relación al espacio utilizado en memoria secundaria el gnat utilizaba demasiado espacio para almacenar la estructura. Este problema fue reducido con la nueva estructura propuesta, sin embargo,

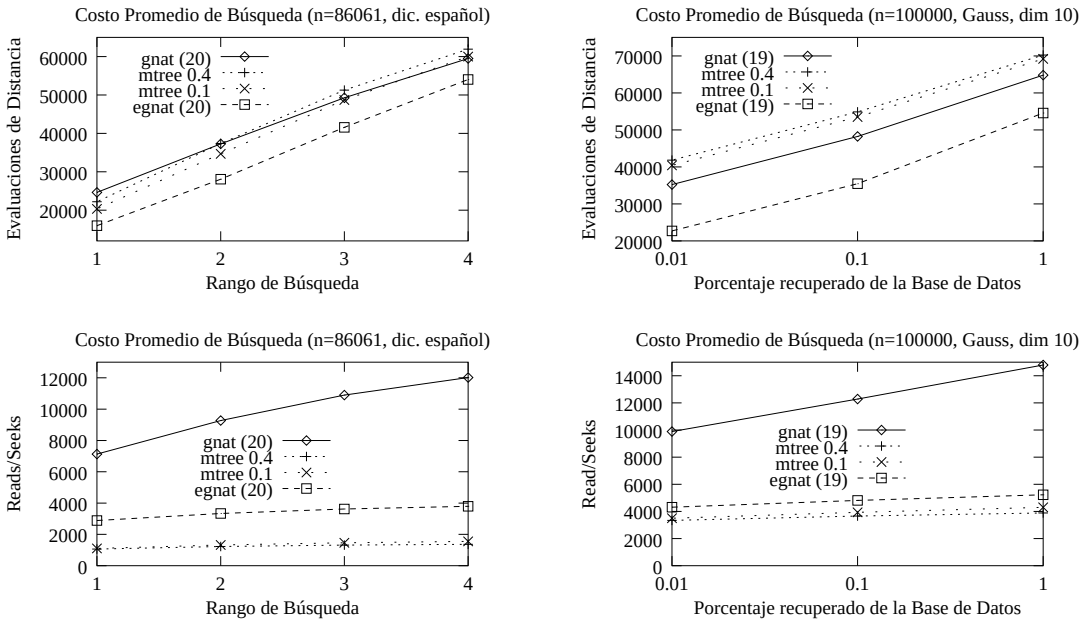


Figura 5.25: Comparativas de costos de búsqueda *gnat*, *M-tree* (MIN_UTIL=0.4), *M-tree* (MIN_UTIL=0.1) y *egnat*, para los dos espacios de prueba.

en relación al M-tree aún la cantidad de páginas de disco es mayor. Esta diferencia en favor del M-tree (tabla 5.13) representa la única desventaja importante del egnat.

Una solución natural a este problema es compartir la página de disco entre varios nodos. La tabla 5.13 muestra este método aplicado sobre el gnat original logrando valores adecuados en cantidad de páginas para el caso de 4 centros por nodo con 12 nodos por página para el espacio de palabras y 12 nodos por página para el espacio de Gauss. Para los experimentos con 8 centros por nodo los resultados son peores que en el egnat. Tanto para 4 centros como para 8 centros, los costos en evaluaciones de distancia para el gnat se conservan correspondiendo a los entregados en las figuras 4.7, 4.10 y 4.16, siendo los costos similares o peores al M-tree y mucho peores respecto al egnat.

Finalmente, una alternativa es aplicar la idea de compartir páginas en el egnat. Para evitar aumentar los costos de búsqueda al tener menos centros por nodo, los nodos que compartirán páginas serán aquellos del tipo *bucket*, es decir, si el nodo es de tipo *gnat* utilizará la mayor cantidad de centros posibles, pero si es bucket lo compartirá con otros nodos del mismo tipo. De esta manera los costos búsqueda permanecerán intactos y se mejorará la utilización del espacio.

Esta idea es basada principalmente en el hecho de que los promedios de uso de un nodo bolsa no son superiores a 14,52 para el caso del espacio de Gauss y de 17,06 para el espacio de palabras. Por lo tanto las páginas podrían contener por ejemplo 4 bolsas, de esta manera la estructura aplicada a vectores tendría bucket de 23 centros y la estructura aplicada a palabras 31 centros. En la tabla 5.13 se muestra la información para el egnat y el egnat con bolsa compartiendo páginas (versión B). Para este último experimento, la cantidad de páginas para el egnat es la suma de los nodos gnat más las

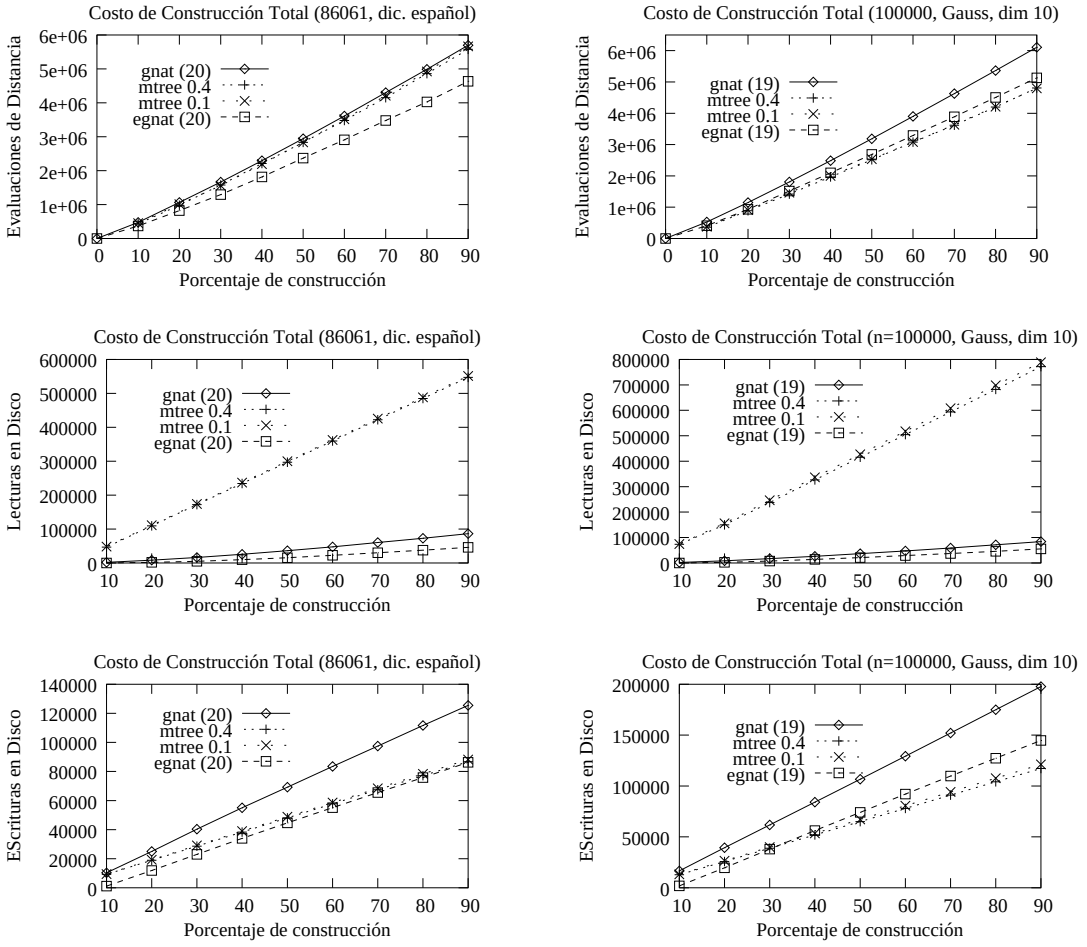


Figura 5.26: Costos de construcción para las estructuras *gnat*, *M-tree* (MIN_UTIL=0.4), *M-tree* (MIN_UTIL=0.1) y *egnat*, para los dos espacios de prueba.

	Vectores de Gauss de dim. 10				Diccionario Español			
	centros	nodos x pág.	nro. pág.	b x obj	centros	nodos x pág.	nro. pág.	b x obj
gnat (4)	4	12	3.339	136.77	4	12	2.633	125,32
gnat (6)	6	7	5.722	234,37	6	7	3.648	173,62
gnat (8)	8	4	10.010	410,01	8	4	6.384	303,84
M-tree 0.4	-	-	4.461	182,72	-	-	1.585	75,44
M-tree 0.1	-	-	5.095	208,69	-	-	1.856	88,33
egnat	19	1	6.773	277,42	20	1	4.996	237,78
egnat B	19	4 (bolsas)	5.936	243,14	20	4 (bolsas)	4.157	197,85
egnat B	19	6 (bolsas)	4.822	197,51	20	6 (bolsas)	3.789	180,33

Tabla 5.13: Construcción: valores comparativos para páginas compartidas entre *gnat*, *M-tree, egnat* y *egnat* (versión final) con el 100 % de los datos.

	vectores de Gauss		diccionario en Español	
Buckets por pagina	4	6	4	6
MAX_SIZE_BUCKET	23	15	31	21
Cantidad de buckets	17.518	17.676	12.475	13.852
cantidad de gnat	1.556	1876	1.038	1.480
cantidad total de páginas	5.936	4.822	4.157	3.789
promedio de uso bucket	4,02 (17,48 %)	3,64 (24,27 %)	5,23 (16,87 %)	4,08 (19,43 %)

Tabla 5.14: *egnat* (versión final B): detalles de construcción para páginas con capacidad para 4 y 6 bolsas (100 % de los datos).

páginas que contienen nodos de tipo bucket, en este caso 4 y 6 buckets por página. La misma tabla también muestra la columna promedio de byte por objeto (b x obj).

Para la alternativa mencionada para el egnat la información obtenida se muestra en la tabla 5.14, de ésta se desprende que al ser más pequeñas las bolsas, éstas se llenan más rápido, lo que implicó un aumento de los nodos de tipo gnat y de los bucket, pero ahora estos últimos utilizarían un cuarto o un sexto de la página.

De la tabla 5.13 se desprende que el M-Tree puede utilizar mejor el espacio dentro de una página en el espacio de palabras, siendo menos competitivo en el espacio de Gauss. Finalmente se puede indicar que la estructura egnat y el egnat con páginas compartidas por varios nodos logran reducir el espacio utilizado en disco y acercarse o mejorar los valores del M-tree.

Desde el punto de vista de los accesos a disco, cabe señalar que independiente de la cantidad de nodos por página, lo relevante es la cantidad total de nodos. Es decir, a mayor cantidad de nodos, mayor cantidad de accesos a disco. Esto se puede observar al tomar como ejemplo las estructuras gnat con 12 nodos por página versus egnat de un nodo por página. Si bien la primera tiene más

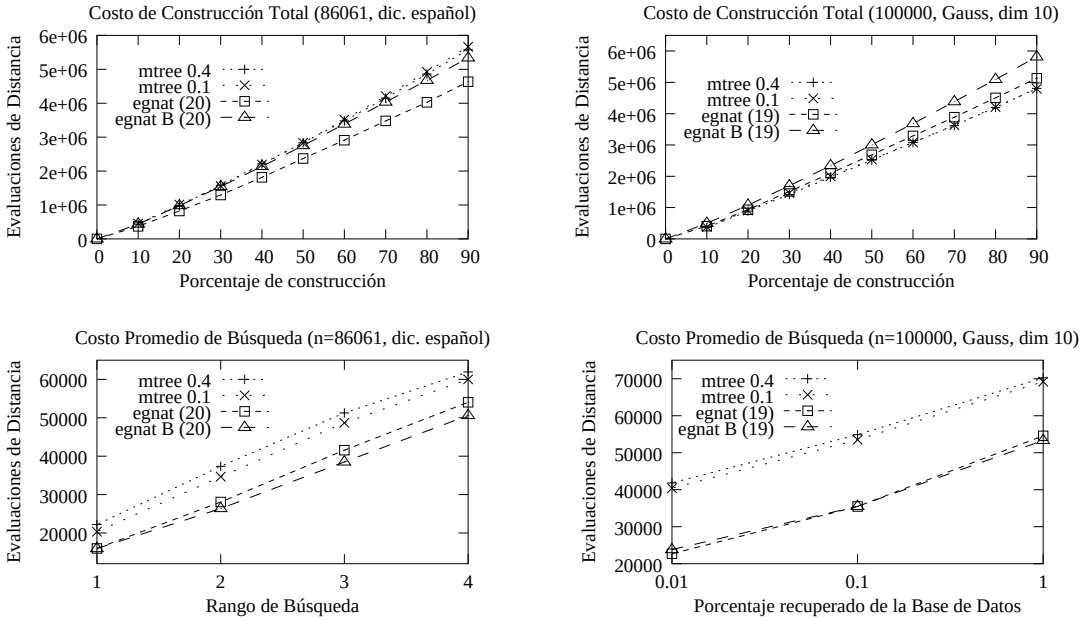


Figura 5.27: Costos de construcción y búsqueda entre el M-Tree y el egnat en su versión normal y con páginas compartidas (B).

nodos en RAM, también tiene más nodos en disco, en este caso mucho más (ver tablas 5.1 y 5.3). Lo anterior, sumado a que la probabilidad de tener un nodo en RAM en alguna página que se haya cargado recientemente, es en extremo baja, implica necesariamente un aumento en los accesos a disco. Los costos de lectura para el gnat de 4 centros (12 nodos por página) se muestran en la figura 5.8. Por lo tanto, al aumentar la cantidad de nodos en la versión B del egnat, aumenta la cantidad de accesos a disco de esta estructura.

Para tener una visión del comportamiento de la estructura egnat con páginas compartidas por bucket, se presenta la figura 5.27, donde de muestra el cálculo de distancias para la construcción y búsqueda. De esta figura se desprende que no hay una pérdida notoria en evaluaciones de distancia y aventaja al egnat en la búsqueda por rango sobre el diccionario español. La identificación en los gráficos para esta estructura es *egnat B*, que corresponde a 4 centros por nodo con 12 nodos por página.

La figura 5.28 muestra la información respecto de los accesos a disco para la construcción y búsqueda por rango correspondiente a los datos mostrado en la figura 5.27 de la estructura egnat versus egnat versión B de 4 centros y 12 nodos por página.

La figura 5.29 muestra la información comparativa para la eliminación en la estructura egnat modificada (versión B, de 4 centros y 12 bucket por página).

Las últimas figuras demuestran que el costo de aumentar la cantidad de nodos en la estructura se ve reflejado principalmente en los accesos a disco. Las evaluaciones de distancia se mantienen y en algunos casos mejoran.

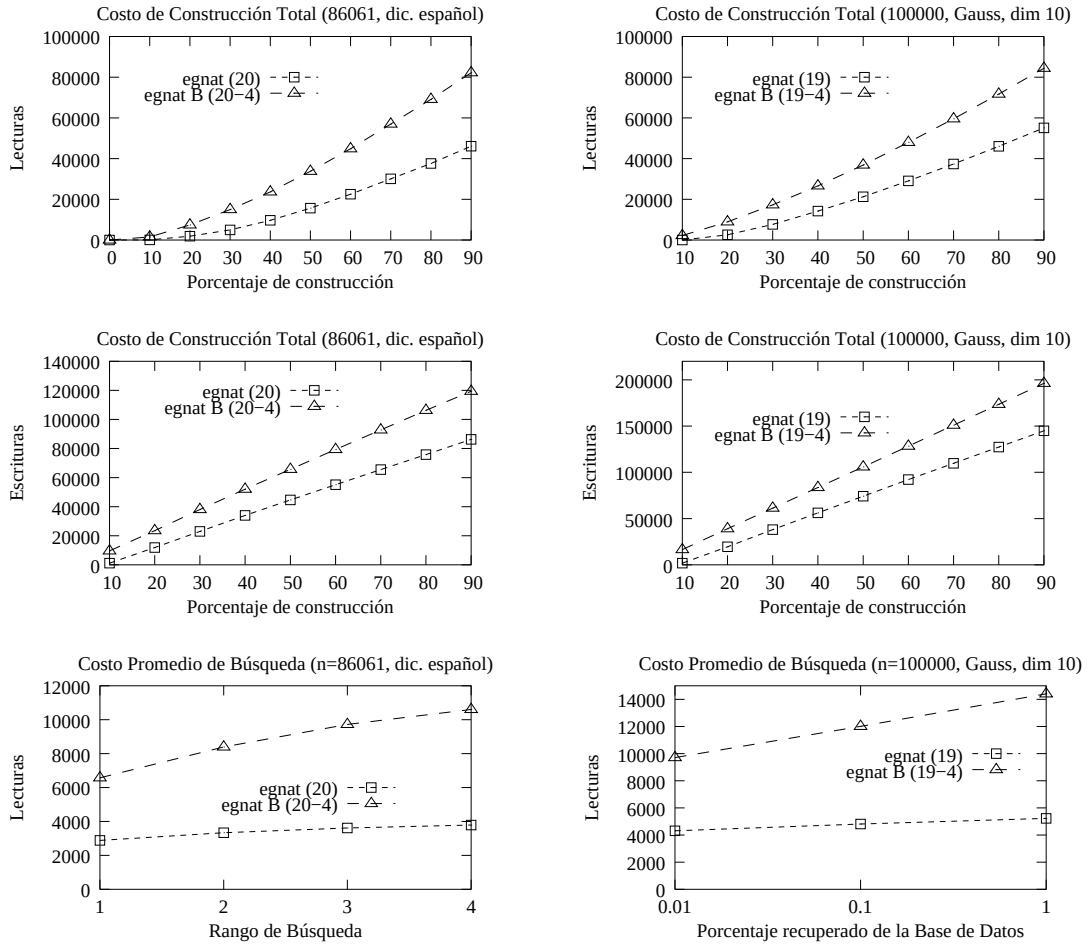


Figura 5.28: Accesos a disco para la construcción y búsqueda por rango para el egnat versus egnat versión B.

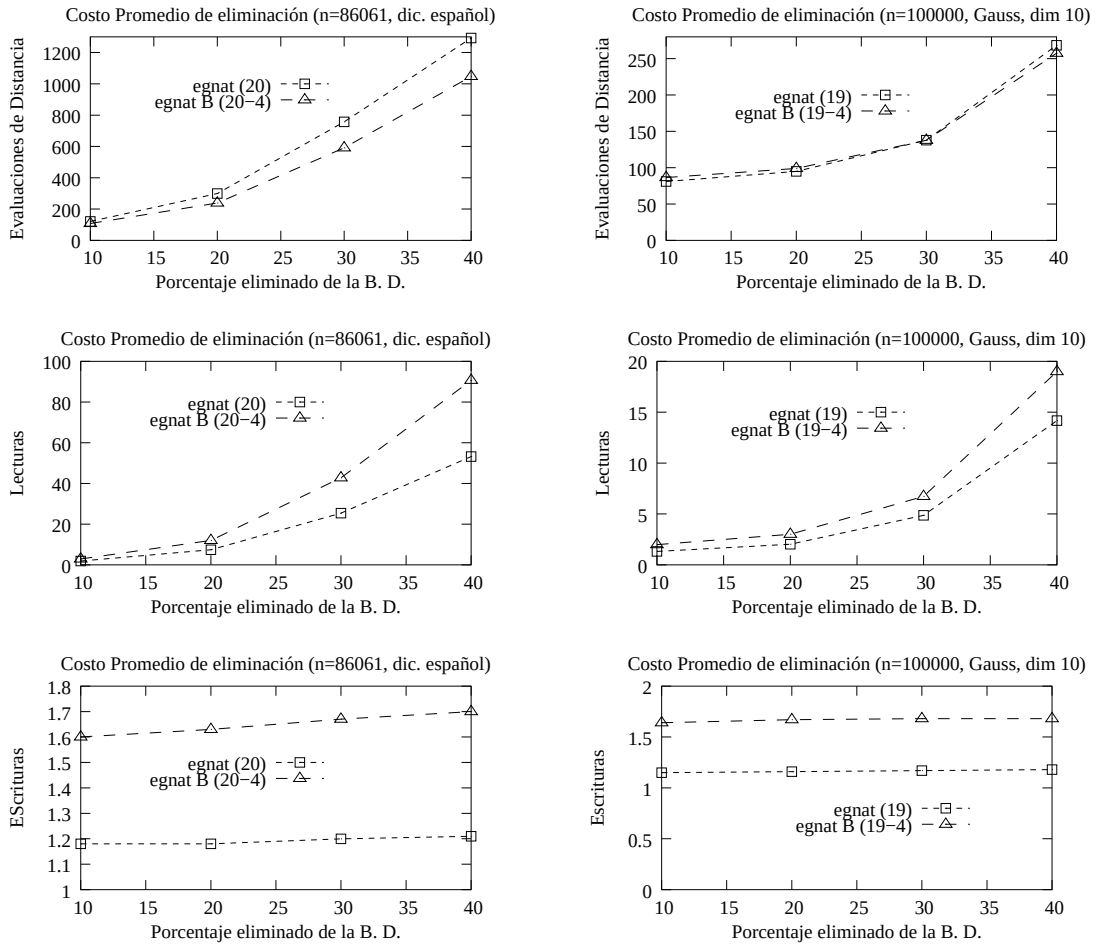


Figura 5.29: Comportamiento de la estructura egnat versión B en la eliminación.

Capítulo 6

Conclusiones

El objetivo principal de la tesis fue desarrollar una estructura de datos completamente dinámica, competitiva para búsquedas por similaridad en espacios métricos de alta dimensionalidad y con buen desempeño en memoria secundaria. Para ello se eligió como base la estructura denominada *Geometric Near-neighbor Access Tree* (*gnat*), perteneciente al grupo de algoritmos basados en particiones compactas. La elección de esta estructura fue debido al buen desempeño en espacios de alta dimensión. Existen innumerables aplicaciones reales donde son necesarias estructuras con las características mencionadas.

Este capítulo presenta, en términos generales, los aspectos más relevantes del trabajo realizado, así como los aportes y los trabajos para desarrollar a futuro.

6.1. Aspectos Relevantes y Aportes

La mayoría de las estructuras desarrolladas para búsquedas por similaridad son estáticas, más aún, están diseñadas como prototipos para memoria principal. Por lo tanto, la necesidad de estructuras con las características anteriores es de relevancia para aplicaciones de tipo real.

En este sentido, se considera como el aporte más relevante de la tesis:

- Presentar una nueva estructura métrica denominada *egnat* (*Evolutionary gnat* o *gnat evolutivo*) optimizada para memoria secundaria, dinámica y con buen desempeño en búsquedas por similaridad sobre espacios de alta dimensión.

El aporte fundamental de la tesis está basado en una serie de resultados interesantes que resuelven problemas desde el punto de vista del dinamismo como de la necesidad de utilización de memoria secundaria, estos son:

- El diseño de un algoritmo eficiente de eliminación denominado *Planos Fantasmas* con excelente desempeño en memoria secundaria (sección 5.2 y 5.4.2.3).

- El generar un algoritmo de eliminación aplicable a cualquier estructura de tipo árbol para búsquedas en espacios métricos. Éste fue comprobado sobre la estructura denominada Voronoi-tree (sección 4.4.5).
- Desarrollar una estructura dinámica que permite inserciones y eliminaciones y lograr mantener la eficiencia en las búsquedas sobre una estructura que puede estar continuamente cambiando. Proyectar el dinamismo a memoria secundaria con adecuados niveles de accesos a disco.
- Se proponen distintos métodos de reemplazo de objetos para el algoritmo de *Planos Fantasmas*. Obteniendo que los dos métodos más óptimos de implementar son: *reemplazo por el último objeto ubicado en una hoja* y *reemplazo por el objeto más cercano ubicado en una hoja*.
- Se analizaron y obtuvieron distintas alternativas para disminución de la cantidad de páginas de disco requerida por las estructuras. Entre éstas: *gnat* con radio cobertor, *gnat* con páginas compartidas por varios nodos, la nueva estructura *egnat* y una alternativa de *egnat* con páginas compartidas por varios buckets.
- De la estructura egnat se puede inferir que toda estructura puede mutar, es decir, toda estructura tiene la capacidad de modificar el tipo de nodo para ajustarlo a los requerimientos de espacio o en función de los costos de evaluaciones de distancia.
- Con la nueva estructura se logró superar ampliamente los costos en términos de evaluaciones de distancia respecto de la estructura *gnat* original y de la estructura *M-Tree*, la cual es una de las pocas que posee características de dinamismo en memoria secundaria. Así también, se logró una adecuada utilización del espacio en memoria secundaria con valores similares a los del *M-Tree*.
- Se logró combinar adecuadamente algunas ideas de estructuras métricas basadas en pivotes sobre estructuras basadas en clustering. Es el caso de mantener la distancia al padre. Para la estructura *egnat*, como parte del método de construcción se conserva la distancia al padre cuando los objetos están en las hojas. Esta misma idea se probó con excelentes resultados sobre la estructura *Voronoi-tree*.
- Conseguir una visión más profunda de los parámetros a considerar tanto para la aplicación de características dinámicas sobre estructuras métricas, como para su implementación sobre memoria secundaria. Esto permite que futuros diseños de estructuras métricas para memoria secundaria tengan una base considerando los futuros problemas a enfrentar.

6.2. Trabajos Futuros

A continuación se presentan algunas investigaciones y proyectos que se están realizando en la actualidad, como también algunas extensiones futuras en base al trabajo de tesis:

- Generar nuevos algoritmos para mejorar los costos de búsquedas sobre la nueva estructura propuesta.
- Generar nuevas alternativas para la disminución del tamaño de la estructura *egnat* sin degradar la búsqueda.
- Probar otros mecanismos de construcción de la nueva estructura, como los propuestos por Sergey Brin para la versión original del gnat [Bri95].
- Obtención de distintas alternativas de administradores de página para la estructura propuesta.
- Probar distintas alternativas para compartir páginas, para las diferentes estructuras.
- Analizar y comparar el método de planos fantasmas sobre distintas estructuras de tipo árbol. En este caso, sería interesante probar este método sobre la estructura denominada *sa-tree* [Nav02], la cual ha demostrado tener buen desempeño en búsquedas sobre espacios de alta dimensión.
- Analizar y experimentar el método de planos fantasmas sobre otros tipos de estructuras métricas, como ser las del tipo arreglos.
- Desarrollar una versión paralela del egnat y determinar distintos métodos de distribución de la estructura.
- Analizar y experimentar el concepto de mutación sobre distintas estructuras.

Finalmente se estima que la presente tesis junto con proponer una nueva estructura constituye un aporte relevante para obtener una visión más clara de los aspectos a considerar sobre dinamismo y memoria secundaria en estructuras métricas.

Bibliografía

[Ben75] J. Bentley. Multidimensional binary search trees used for associative searching. In *Communications of the ACM*, pages 18(9):509–517, 1975.

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

[BK73] W. Burkhard and R. Keller. Some approaches to best-match file searching. *Communication of ACM*, 16(4):230–236, 1973.

[BKK96] S. Berchtold, D. Keim, and H. Kriegel. The x-tree: an index structure for high-dimensional data. In *Proc. 22nd Conference on Very Large Database (VLDB'96)*, pages 28–39, 1996.

[BO97] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *ACM Conference on Management of Data (ACM SIGMOD'97)*, Sigmod Record 26(2), pages 357–368, 1997.

[Bri95] Sergei Brin. Near neighbor search in large metric spaces. In *the 21st VLDB Conference*, pages 574–584. Morgan Kaufmann Publishers, 1995.

[BY97] R. Baeza-Yates. Searching: An algorithmic tour. In A. Kent and J. Williams, editors, *Encyclopedia of Computer Science and Technology*, volume 37, pages 331–359. Marcel Dekker Inc., 1997.

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

[BYN98] R. Baeza-Yates and G. Navarro. Fast approximate string matching in a dictionary. In *5th South American Symposium on String Processing and Information Retrieval (SPIRE'98)*, pages 14–22. IEEE CS Press, 1998.

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[Cha94] B. Chazelle. Computational geometry: A retrospective. In *26th ACM Symposium on the Theory of Computing (STOC'94)*, pages 75–94, 1994.

[Chi94] T. Chiuch. Content-based image indexing. In *The 20th Conference on Very Large Databases (VLDB'94)*, pages 582–593, 1994.

[CMBY99] E. Chavéz, J. Marroquín, and R. Baeza-Yates. Spaghettis: An array based algorithm for similarity queries in metric spaces. In *6th International Symposium on String Processing and Information Retrieval (SPIRE'99)*, pages 38–46. IEEE CS Press, 1999.

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

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

[CNBYM01] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José L. Marroquín. Searching in metric spaces. In *ACM Computing Surveys*, pages 33(3):273–321, September 2001.

[Cod70] E. F. Codd. A relational model of data for large shared data bank. *Communications of ACM*, 13(6):377–387, June 1970.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree : An efficient access method for similarity search in metric spaces. In *the 23st International Conference on VLDB*, pages 426–435, 1997.

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

[Gut84] A. Guttman. The r-trees: a dynamic index structure for spatial search. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[HNP95] J. M. Hellertein, J. F. Naughton, and A. Pfeffer. Generalized search trees for databases systems. In *21st Conference on Very Large Databases (VLDB)*, Zurich, Switzerland, 1995.

[KM83] I. Kalantari and G. McDonald. A data structure and an algorithm for the nearest point problem. *IEEE Transactions on Software Engineering*, 9(5), 1983.

[LJF94] King-Ip Lin, H. Jagadish, and C. Faloutsos. The tv-trees - an index structure for high-dimensional data. *The Very Large Databases Journal (VLDBJ)*, 3:517–542, Oct. 1994.

[MOC96] L. Micó, J. Oncina, and R. Carrasco. A fast branch and bound nearest neighbor classifier in metric spaces. *Pattern Recognition Letters*, 17:731–739, 1996.

[MOV94] L. Micó, J. Oncina, and 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.

[Nav99] Gonzalo Navarro. Searching in metric spaces by spatial approximation. In *String Processing and Information Retrieval (SPIRE'99)*, Cancún, México, Setiembre 1999. IEEE CS Pres.

[Nav01] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[Nav02] Gonzalo Navarro. Searching in metric spaces by spatial approximation. *The Very Large Databases Journal (VLDBJ)*, 11(1):28–46, 2002.

[NN97] S. Nene and S. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):989–1003, 1997.

[Nol89] H. Noltemeier. Voronoi trees and applications. In *International WorkShop on Discrete Algorithms and Complexity*, pages 69–74, Fukuoka, Japan, 1989.

[NR02] Gonzalo Navarro and Nora Reyes. Fully dynamic spatial approximation trees. In *the 9th International Symposium on String Processing and Information Retrieval (SPIRE 2002)*, pages 254–270, Springer 2002.

[NVZ92] H. Noltemeier, K. Verbarg, and C. Zirkelbach. Monotonous bisector* trees - a tool for efficient partitioning of complex schemes of geometric object. In *Data Structures and Efficient Algorithms*, LNCS 594, pages 186–203, Springer-Verlag 1992.

[Pat99] Marco Patella. *Similarity Search in Multimedia Databases.* PhD thesis, Dipartamento di Electtronica Informatica e Sistemisca, Università degli Studi di Bologna, Bologna, Italy, 1999.

[Rey02] Nora Reyes. Índices dinámicos para espacios métricos de alta dimensionalidad. Master's thesis, Universidad Nacional de San Luis, Argentina, 2002.

[Sam84] H. Samet. The quadtree and related hierarchical data structures. In *ACM Computing Surveys*, pages 16(2):187–260, 1984.

[Sha77] M. Shapiro. The choice of reference points in best-match file searching. *Communications of ACM*, 20(5):339–343, 1977.

[TTSF00] Caetano Traina, Agma Traina, Bernhard Seeger, and Christos Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. In *VII International Conference on Extending Database Technology*, pages 51–61, 2000.

[Uhl91] J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. In *Information Processing Letters*, pages 40:175–179, 1991.

[Ver95] K. Verbarg. The c-tree: A dinamically balanced spatial index. *Computing Suppl.*, 10:323–340, 1995.

[Vid86] E. Vidal. An algorithm for finding nearest neighbor in (approximately) constant average time. *Pattern Recognition Letters*, 4:145–157, 1986.

[Yia93] P. Yianilos. Data structures and algoritms for nearest neighbor search in general metric spaces. In *4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 311–321, 1993.

[Yia99] P. Yianilos. Excluded middle vantage point forests for nearest neighbor search. In *DIMACS Implementation Challenge, ALENEX'99*, Baltimore, MD, 1999.

[Yia00] P. Yianilos. Locally lifting the curse of dimensionality for nearest neighbor search. In *11th ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, 2000.