Manipulación de Estructuras Métricas en Memoria Secundaria*

Roberto Uribe Paredes**
Depto. de Ingeniería en Computación
Centro de Investigación de la Web
Universidad de Magallanes, Bulnes 01855, Punta Arenas, Chile
(ruribe@ona.fi.umag.cl)

Prof. Guía: Dr. Gonzalo Navarro
(gnavarro@dcc.uchile.cl)

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.
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 a fin de disminuir las evaluaciones de distancia al momento de la búsqueda.

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.

El siguiente trabajo presenta una nueva estructura denominada *Evolutionary GNAT* (*egnat*) o *GNAT Evolutivo*, la cual es completamente dinámica y optimizada para memoria secundaria. Utiliza el método de *Planos Fantasmas* para la eliminación, el cual fue también diseñado en el presente trabajo. Está basada en el *Geometric Near-neighbor Access Tree* (*gnat*) de Sergey Brin, que es estático pero con buen desempeño en espacios de alta dimensionalidad.

Palabras claves: bases de datos, estructuras de datos, algoritmos, espacios métricos, consultas por similaridad.

1. Introducción

1.1. Antecedentes

Uno de los problemas de gran interés en ciencias de la computación es el de "búsqueda por similaridad", es decir, encontrar los elementos de un conjunto más similares a una muestra. Esta búsqueda es necesaria en múltiples aplicaciones, como ser en reconocimiento de voz e imagen, compresión de video, genética, minería de datos, recuperación de información, etc. En casi todas las aplicaciones la evaluación de la similaridad entre dos elementos es cara, por lo que usualmente se trata como medida del costo de la búsqueda la cantidad de similaridades que se evalúan.

Interesa el caso donde la similaridad describe un espacio métrico, es decir, está modelada por una función de distancia que respeta la desigualdad triangular. En este caso, el problema más común y

*Tesis presentada y aprobada en la Escuela de Postgrado de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile, Chile.
**Parcialmente financiado por el Proyecto MECESUP MAG9901, Mineduc, Chile.

difícil es en aquellos espacios de "alta dimensión" donde el histograma de distancias es concentrado, es decir, todos los objetos están más o menos a la misma distancia unos de otros.

El aumento de tamaño de las bases de datos y la aparición de nuevos tipos de datos sobre los cuales no interesa realizar búsquedas exactas, crean la necesidad de plantear nuevas estructuras para búsqueda por similaridad o búsqueda aproximada. Asimismo, se necesita que dichas las estructuras sean dinámicas, es decir, que permitan agregar o eliminar elementos sin necesidad de crearla nuevamente, así como también que sean óptimas en la administración de memoria secundaria.

1.2. 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.

Definición 1 (*Espacios Métricos*): Un espacio métrico es un conjunto X con una función de distancia $d : X^2 \to R$, tal que $\forall x, y, z \in X$,

1. $d(x,y) \geq 0 and d(x,y) = 0 ssi x = y$. *(positividad)*
2. $d(x,y) = d(y,x)$. *(Simetría)*
3. $d(x,y) + d(y,z) \geq (d(x,z)$. *(Desigualdad Triangular)*

Definición 2 (*Consulta por Rango*): Sea un espacio métrico *(X,d)*, un conjunto de datos finito $Y \subseteq X$, una consulta $x \in X$, y un rango $r \in R$. La consulta de rango alrededor de x con rango r es el conjunto de puntos $y \in Y$, tal que $d(x,y) \leq r$.

Definición 3 (*Los k Vecinos más Cercanos*): Sea un espacio métrico *(X,d)*, un conjunto de datos finito $Y \subseteq X$, una consulta $x \in X$ *y un entero k*. Los k vecinos más cercanos a x son un subconjunto A de objetos de Y, donde la $|A| = k$ y no existe un objeto $y \in A$ tal que *d(y,x)* sea menor a la distancia de algún objeto de A a x.

El objetivo de los algoritmos de búsqueda es minimizar la cantidad de evaluaciones de distancia realizadas para resolver la consulta. 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 [CNBYM01].

Existen distintas estructuras para buscar en espacios métricos, las cuales pueden ocupar funciones discretas o continuas de distancia. Algunos son BKTree [BK73], MetricTree [Uhl91], GNAT [Bri95], VpTree [Yia93], FQTree [BYCMW94], MTree [CPZ97], SAT [Nav02], Slim-Tree [TTSF00].

Algunas de las estructuras anteriores basan la búsqueda en pivotes y otras en clustering. En el primer caso se seleccionan pivotes del conjunto de datos y se precalculan las distancias entre los elementos y los pivotes. Cuando se realiza una consulta, se calcula la distancia de la consulta a los pivotes y se usa la desigualdad triangular para descartar candidatos.

Los algoritmos basados en clustering dividen el espacio en áreas, donde cada área tiene un *centro*. Se almacena alguna información sobre el área que permita descartar toda el área 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, *hiperplanos* y *radio cobertor* (*covering radius*). El primero divide el espacio en particiones de *Voronoi* y determina el hiperplano al cual pertenece la consulta según a qué centro corresponde. El criterio de radio cobertor divide el espacio en esferas que pueden intersectarse y una consulta puede pertenecer a más de una esfera.

Definición 4 (*Diagrama de Voronoi*):
Considérese un conjunto de puntos $\{c_1, c_2, \ldots, c_n\}$(centros). Se define el diagrama de Voronoi como la subdivisión del plano en n áreas, una por cada c_i, tal que $q \in$ al área c_i sí y sólo sí la distancia euclidiana $d(q, c_i) < d(q, c_j)$ para cada c_j, con $j \neq i$.

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

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, ya sean lecturas, escrituras o movimientos de cabezal. 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. Los 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.

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.

Para este artículo 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 estos espacios muestran claramente el comportamiento del egnat.

2. *Geometric Near-neighbor Access Tree*

La estructura base utilizada en el desarrollo de la tesis fue el *GNAT* [Bri95], sobre la cual se implementaron en primera instancia los métodos necesarios para convertirla en una estructura dinámica.

El **gnat** es una estructura basada principalmente en el diagrama de Voronoi, aunque igualmente usa radio cobertor. Es una generalización del *Generalized Hyperplane Tree* (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 clave para particionar el espacio $\{p_1, p_2, \ldots, p_k\}$, cada punto restante es asignado a la clave más cercana, definiéndose así el subárbol de influencia. Cada subárbol es particionado recursivamente.

2.1. Construcción del *gnat*

La estructura gnat tiene la propiedad que el algoritmo de inserción original [Bri95], 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, independiente de si la estructura está o no construida anteriormente.

La construcción básica del gnat es como sigue:

1. Se seleccionan k puntos (*splits*), $p_1, \ldots, p_k$ de la base de datos la cual se va a indexar.

2. Se asocia cada punto restante del conjunto de datos al split más cercano a él. El conjunto de puntos asociados al split p_i se denota como D_{p_i}.

3. Para cada par de split (p_i, p_j), se calcula el rango $range\left(p_i, D_{p_j}\right) = \left[min\_d\left(p_i, D_{p_j}\right), max\_d\left(p_i, D_{p_j}\right)\right]$, una mínima y máxima distancia $Dist\left(p_i, x\right)$ donde $x \in D_{p_j} \cup \{p_j\}$.

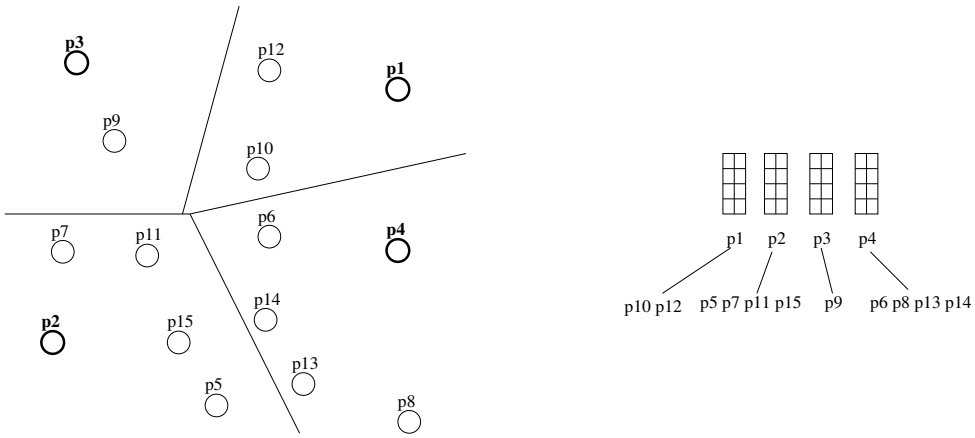


Figura 1: gnat: partición del espacio, representación de subplanos.

4. El árbol se construye recursivamente para cada elemento en 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 al plano de Voronoi cuyo centro es p_i. En la figura 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.

2.2. Búsqueda en el *gnat*

En una búsqueda en un *gnat* se asume que se desea encontrar todos los puntos con distancia $d \leq r$ a el punto q. Sea P la representación del conjunto de puntos splits del nodo actual (inicialmente la raíz del *gnat*) el cual posiblemente contiene un vecino cercano a q. Inicialmente P contiene todos los puntos split del nodo actual. La búsqueda se realiza como se muestra en el algoritmo 1.

Algoritmo 1 *gnat*: búsqueda por rango r para la consulta q.

busquedarango(Nodo P, Consulta q, Rango r)

```
1:  {Sea R el conjunto resultado}
2:  R ← ∅
3:  d ← dist(p_0, q)
4:  if d <= r then
5:     se reporta p_0
6:  end if
7:  range(p_0, q) ← [d − r, d + r]
8:  for all x ∈ P do
9:     if range(p_0, q) ∩ range(p_0, D_{p_x}) ≠ ∅ 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 ∈ R do
17:    busquedarango(D_{p_i},q,r)
18: end for
```

3. Eliminación en el *gnat*

El objetivo final de este trabajo 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. Esto no es aceptable, debido a que, especialmente en espacios métricos, en la mayoría de las aplicaciones los objetos son muy grandes (por ejemplo imágenes), y es indispensable eliminarlo físicamente. 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* . 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, entre otras cosas se presenta 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.

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 2 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}).

Por lo anterior, la reconstrucción del árbol o de los subárboles afectados implicarían un excesivo costo en el recálculo de los rangos, por esto, esta opción fue desechada.

3.1. *Planos Fantasmas*

Para esta estructura se diseñó un método de eliminación denominado *Planos Fantasmas* (o *Ghosts Hyperplanes*), el cual propone reemplazar el objeto eliminado por otro que ocupe su lugar en el nodo *sin modificar* los rangos originales del objeto eliminado. Una vez hecho el reemplazo, se marca el nodo y el centro como *afectado*. Como no hay recálculo de rangos, la estructura cambiaría de forma a partir de este nodo produciéndose solapamiento (*overlap*) de los planos, lo cual implica un nuevo elemento a considerar en los métodos de inserción, eliminación y búsqueda.

Este método se denomina *planos fantasmas* debido a que bajo un mismo subárbol coexistirán dos planos.

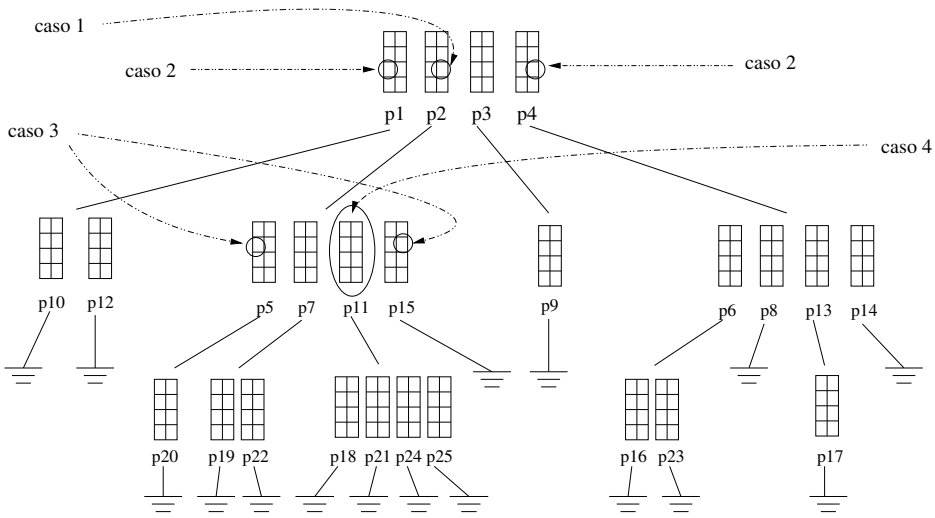


Figura 2: Casos problemas en la estructura *gnat* al eliminar el elemento p_{11}.

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 reemplazó 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.

La elección del objeto de reemplazo resulta interesante, esto porque, dependiendo de la ubicación del elemento, el árbol puede resultar más o menos afectado. La alternativa general es elegir algún dato a partir de este nodo, de esta manera, sólo un subárbol es afectado y el resto del árbol permanece intacto y conserva absolutamente todas las propiedades de un *gnat*. La figura 3 (b) representa el solapamiento de planos al eliminar el objeto S_1 de la figura 3 (a) al reemplazarlo por otro. Entonces llamaremos plano fantasma a $P(S_1)$ antes de su eliminación, el cual es representado en la misma figura por líneas no continuas.

Se proponen cuatro alternativas de elección del objeto de reemplazo. Cada alternativa reemplaza en forma definitiva el objeto eliminado por otro, marcando el nuevo centro (subárbol) como afectado y conservando los rangos originales. La marca permite considerar los subárboles en forma especial en los procesos de eliminación y búsqueda. Las diferentes alternativas planteadas son:

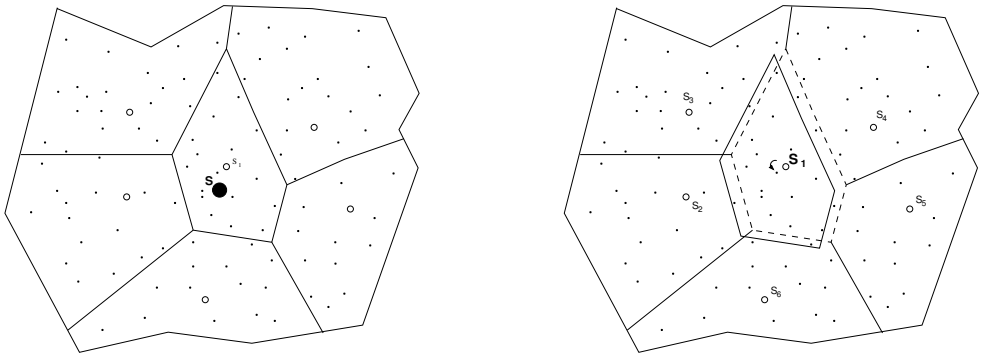
1.- Reemplazo por el descendiente más cercano: Una solución natural es elegir el elemento más cercano que sea *descendiente* del objeto eliminado, es decir, algún elemento dentro de su subárbol. Esta alternativa evita que el corrimiento del plano sea mayor. Sin embargo, la elección del más cercano a S_1 no garantiza que el plano siga igual, por lo tanto, es posible que varios puntos queden fuera del plano real formado por el nuevo dato y otros de planos adyacentes queden dentro (ver figura 3 (b)).

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, además de los producidos por la replicación del proceso de eliminación.

Para visualizar como es afectada la estructura después de eliminar un centro, considere la figura 4, la cual representa un árbol que no contiene datos eliminados. Esta figura podría representar el plano indicado en la figura 3 (a) antes de eliminar objetos.

Si el centro eliminado es S_1, siendo el más cercano a él S_{13}, entonces la estructura quedaría según se representa en la figura 5 (a), 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 se puede ver afectado en varios



(a) Vista de planos sin eliminaciones

(b) Eliminación de un centro, vista de un *Plano Fantasma*

Figura 3: partición del espacio, representación de subplanos para todas las estructuras

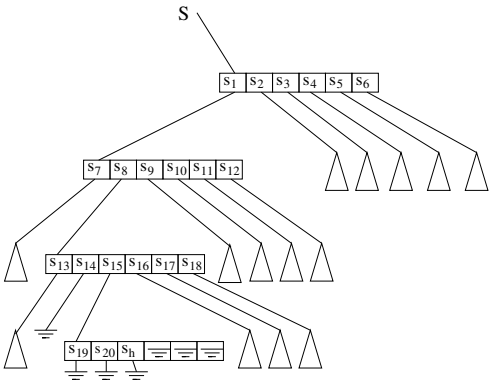


Figura 4: *gnat*: Estructura original, sin elementos eliminados.

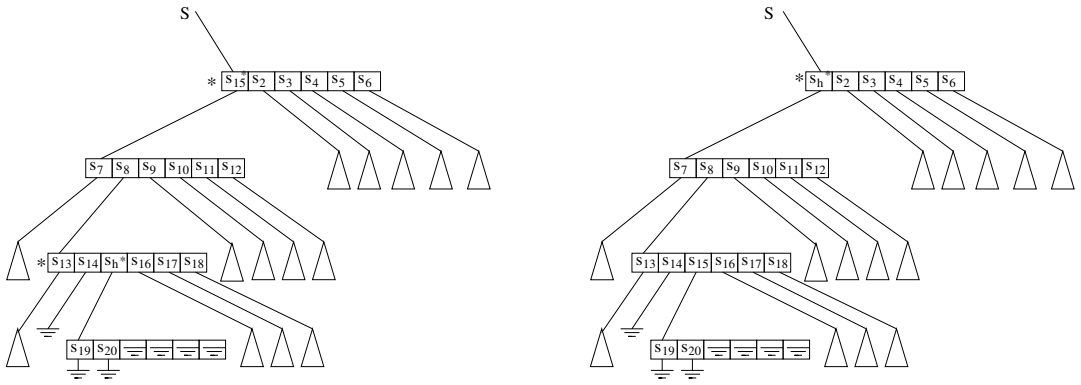
planos o subárboles interiores, esto dependiendo de la cantidad de niveles y la ubicación de los más cercanos.

2.- Reemplazo por el más cercano en el nodo: Una modificación a esto, sería el reemplazo del más cercano, pero 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 árbol.

Para los dos métodos anteriores, hay que considerar los costos adicionales por 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 último 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 centros marcados como afectados, además de las réplicas, lo que afectaría en los costos a los métodos de eliminación 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 *gnats*.

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



(a) reemplazo del más cercano

(b) reemplazo directo del último hijo ubicado en una hoja.

Figura 5: *gnat*: Árbol después de la eliminación de un centro usando planos fantasmas.

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.

4.- Reemplazo por descendiente 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.

Las dos últimas alternativas tienen dos características importantes. La primera, es que sólo basta recorrer el árbol hasta su primera hoja. La segunda, que el movimiento del objeto no ocasiona overlap en los subárboles interiores, por lo tanto, se puede decir que sólo un nodo y las replicas del centro borrado son afectados y no otros descendientes, es decir, el subárbol completo a partir del elemento borrado sigue siendo un *gnat*.

El árbol de la figura 4 aplicando el último método, dejaría la estructura como se muestra en la figura 5 (b), considerando que la hoja donde se busca el reemplazante es la misma que para el primer algoritmo.

3.2. 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 almacena la distancia entre el objeto eliminado y su reemplazante, lo que indicaría cuanto se movió 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.

3.2.1. Búsquedas por rango y eliminaciones

Si se han realizado inserciones luego de eliminaciones, entonces la manera trivial de resolver la búsqueda por rango es incluir en el conjunto respuesta el subárbol cuyo centro está marcado, de esta manera se incluyen las posibles respuestas que estén tanto en el plano real como en el fantasma.

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

Una optimización a lo anterior evita buscar

en todos los marcados usando un factor de incertidumbre. Por ejemplo, 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 modificando el algoritmo original de la siguiente manera: $\forall\, x \in P$, *si* $[Ddist(q,p) - r - I_p, dist(q,p) + r + I_p] \cap range(p, D_x)$. 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.

Cabe recordar que si dentro de los subplanos (o subárboles) de dicho centro, existiese un nodo no marcado, entonces en este subplano no hay solapamiento, por lo tanto se aplican los métodos originales, lo que logra mantener la eficiencia dentro de este subárbol.

3.3. Resultados Experimentales

Las figuras 6 y 7 muestran el comportamiento del método de Planos Fantasmas durante la búsqueda y eliminación usando la alternativa 4. La figura 6 muestra la búsqueda por rango sin eliminaciones y con 10 y 40 % de objetos eliminados. La figura 7 muestra los costos promedios de eliminación.

En [Uri05] (documento original de la tesis) se demuestra experimentalmente que la alternativa 4 se comporta mejor que la tercera en la mayoría de los casos, salvo algunas excepciones en la eliminación. Durante la búsqueda, a bajos rangos y con porcentajes reducidos de datos eliminados la tercera alternativa tiene un comportamiento un poco mejor que la cuarta opción. Sin embargo, esta mejora sólo alcanza a un 1 % y decae a medida que aumenta el rango de búsqueda y la cantidad de datos eliminados, siendo finalmente superado por la última alternativa. Desde el punto de vista de la memoria secundaria, con las alternativas 3 y 4 se garantiza que las escrituras en disco no serán más de dos.

El método de planos fantasmas también demostró buen desempeño en la estructura *Voronoi-Tree* ([Uri05]). Esta estructura resultó ser ideal para realizar pruebas del método, primero, por su parecido al *gnat* y segundo, porque durante la construcción de este árbol, se replican las raices de sus subárboles, lo que presenta un medio hostil para la implementación del algoritmo. En el *Voronoi-Tree,* si se elimina uno de los elementos repetidos, se propaga el plano fantasma.

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.

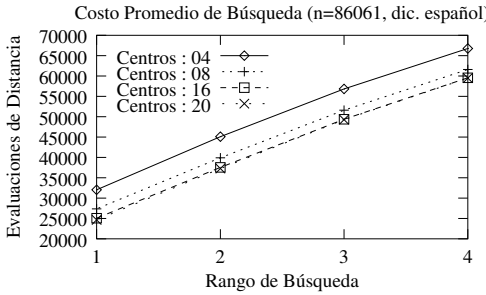
De los gráficos de eliminación se puede determinar que para el caso del espacio de palabras cuando la cantidad de eliminaciones supera el 20 % y en el caso de vectores el 35 %, la búsqueda del elemento de reemplazo se degrada bastante.

4. *gnat* en Memoria Secundaria

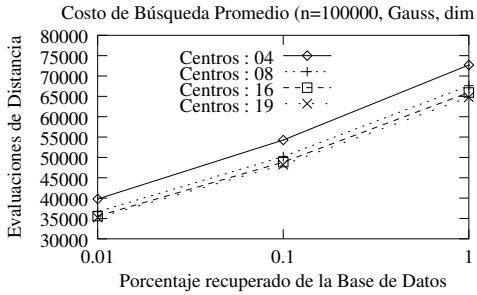
En una primera fase, se implementó el *gnat* como originalmente fue diseñado. Para ello se supuso que tanto la base de datos como el índice no entran completamente en RAM. Para los experimentos en memoria secundaria se determinó que el tamaño máximo de memoria RAM disponible para almacenamiento de nodos sería de 2 Mbytes. Esto permite definir un parámetro (*MAXNODOS*) que indicará la cantidad máxima de nodos permitidos en memoria principal. Para este caso, se mantiene en RAM la raíz del árbol y algunos niveles, esto ya que siempre todos los procesos comienzan por la raíz.

Desde el punto de vista de los procesos, es decir, inserción, eliminación y búsqueda, el comportamiento en disco fue adecuado. Por ejemplo, el método de planos fantasmas tubo un promedio de entre 1,4 y 1,9 escrituras por dato para los dos espacios y para las distintas cantidades de centros e independiente de la cantidad de objetos eliminados. Las lecturas y movimientos de cabezal iban en aumento a medida que la cantidad de datos eliminados aumentaba. Para ambos espacios eran similares los valores para los *reads* y *seeks*, unos 200 por dato para palabras y del orden de 40 para el espacio de Gauss, cuando la cantidad de eliminados era del 40 % .

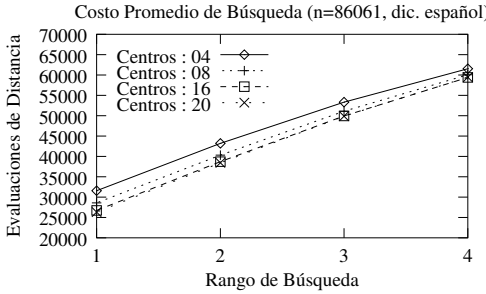
Para tener una visión general respecto del comportamiento de la estructura en memoria



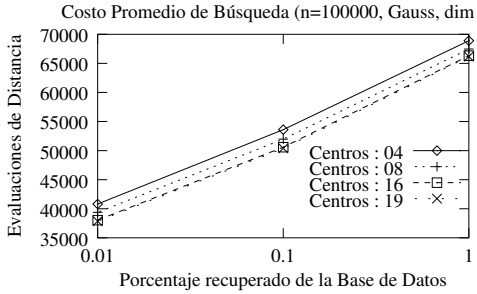
(a) Sin eliminaciones



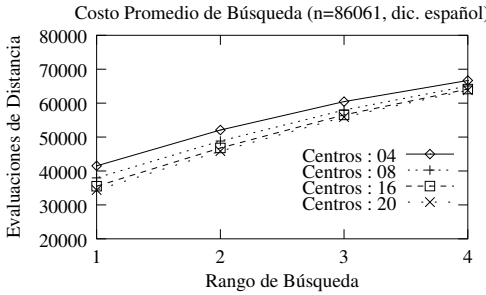
(b) Sin eliminaciones



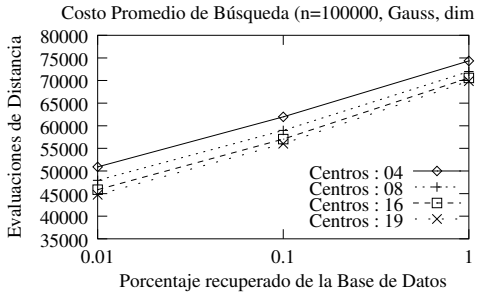
(c) Dic. español 10 % eliminado y reinsertado



(d) Gauss 10 % eliminado y reinsertado

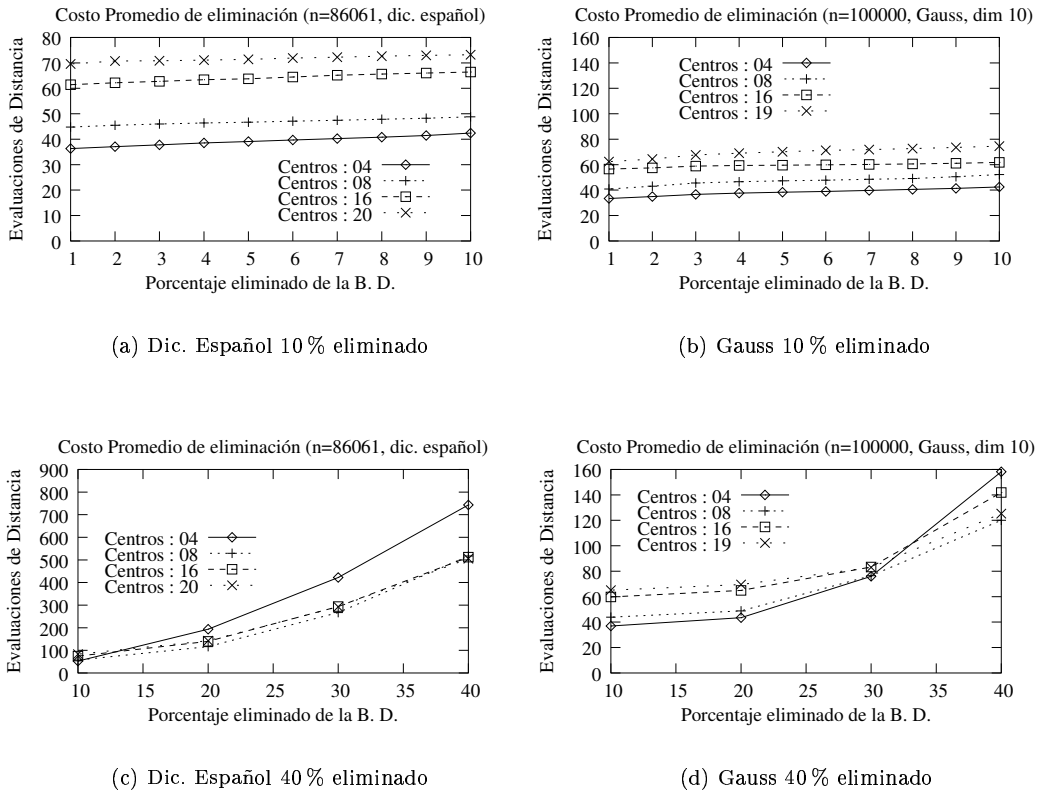


(e) Dic. Español 40 % eliminado y reinsertado



(f) Gauss 40 % eliminado y reinsertado

Figura 6: Búsquedas con métodos optimizados para el espacio de palabras y el espacio de Gauss.



(a) Dic. Español 10 % eliminado

(b) Gauss 10 % eliminado

(c) Dic. Español 40 % eliminado

(d) Gauss 40 % eliminado

Figura 7: Costos promedios de eliminaciones para el diccionario español y para el espacio de Gauss.

secundaria, se presentan las tablas 1 y 2 con la construcción usando el 100 % de los datos. Se debe entender que la información presentada ahí 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).

De la columna *Espacio* de las tablas 1 y 2 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.

Las tablas 3 y 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

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 1: Información general de la estructura 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 2: Información general de la estructura para espacio de vectores de Gauss con dimensión 10.

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.

4.1. *gnat Evolutivo (Evolutionary gnat)*

Para superar el problema del excesivo uso de espacio en disco, se implementó una versión modificada del *gnat*, el *gnat con radio cobertor*. Esta alternativa elimina las tablas de rangos, lo que permite aumentar la candidad de centros por nodo. Para nodos de 4kbytes, la capacidad aumento a 92 centros para el espacio de palabras y para el espacio de vectores a 72. Si bien, logra bajar el espacio en disco considerablemente, por ejemplo, de 15.405 páginas a 9.536 páginas para el caso de las palabras, no lo reduce lo suficiente. Esta estructura mantiene el problema del bajo porcentaje de uso del nodo, además, hay una baja considerable en los costos de búsqueda.

Para poder utilizar el máximo espacio disponible en un nodo y además mantener el desempeño mostrado por el *gnat*, se propone una nueva estructura. Ésta poseerá distintos tipos de nodos o páginas sobre las cuales se realizarán distintas operaciones.

Una primera versión de esta estructura define tres tipos de nodos: *nodo bucket, nodo gnat con bucket y nodo gnat.*

La información general de cada nodo es: ubicación en memoria secundaria, ubicación en memoria secundaria del padre y 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.

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 3: Información de los nodos para el diccionario Español.

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

- 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 8 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, $(MAX\_SIZE\_BUCKET - MAX\_SPLITS + 1)/MAXSPLITS$.

Para esta primera versión del *egnat* se logró reducir la cantidad de páginas a un 26,34 % en el caso del espacio de palabras y a un 28,41 % para el caso de los vectores de Gauss, además de disminuir los costos de construcción, en términos de evaluaciones de distancia, a valores que bordean el 80 % en ambas estructuras.

Si bien esta idea permite una mayor capacidad de los nodos y por ende una mayor concentración de datos, el comportamiento de la primera versión de la estructura durante la búsqueda es en extremo deficiente. En la tabla 5 se muestran detalles de la construcción del *egnat*. Se puede observar que la cantidad de buckets es en extremo alta, por lo que el incremento de las evaluaciones de distancia durante la búsqueda es debido a que al llegar a un nodo bucket se debe realizar el cálculo directo entre la consulta y los elementos en el bucket.

Una alternativa natural es eliminar una mutación para aumentar el número de *gnats* y poder discriminar más elementos durante la búsqueda. Sin embargo, el efecto de esto es menor debido a la aún gran cantidad de buckets.

Finalmente, la forma de disminuir los cálculos de distancia durante la búsqueda es agregar información adicional al bucket. La información que tendrán las

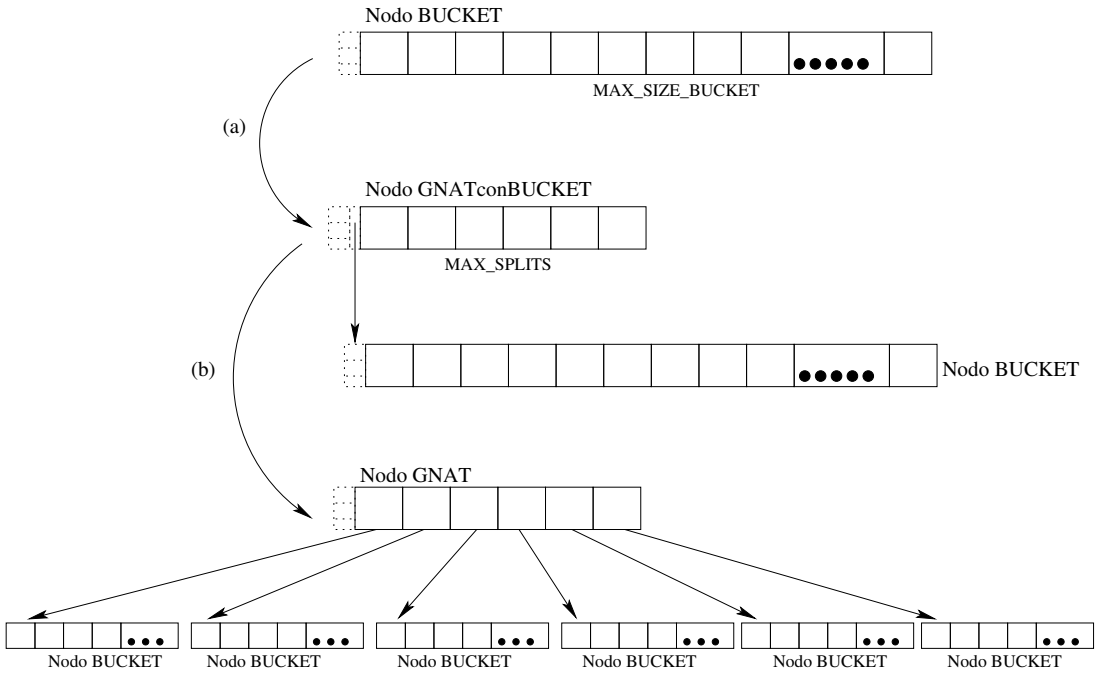


Figura 8: *egnat*: mutaciones de un nodo.

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.

Los detalles de construcción se verán en las próximas subsecciones.

4.1.1. Construcción del *egnat*

La estructura *egnat* se construye usando el método de inserción del *gnat*, con la diferencia de que el *egnat* es una estructura donde los nodos nacen como bolsas o buckets y cuya única información es sólo la distancia al padre, al estilo de las estruturas basadas en pivotes (un solo pivote). Este mecanismo es el que permite disminuir el espacio requerido en disco para almacenar la estructura y logra mantener un buen desempeño en términos de evaluaciones de distancia. Si un nodo se completa, éste evoluciona de un *nodo bolsa* a un *nodo gnat*, reinsertando los objetos de la bolsa al nuevo *nodo gnat*.

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*, éste no podrá volver a ser bucket durante los procesos

	Espacio de Gauss	Espacio de palabras
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: *egnat*: detalles de construcción.

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 6 muestra que si bien aumento levemente la cantidad de páginas del *egnat*, este aún es competitivo versus las otras estructuras.

Para el espacio de vectores de Gauss, el máximo tamaño del bucket fue de 92 centros, la cantidad buckets generados 6.405, el promedio de uso del bucket fue 14,52 y la cantidad de gnat fue 368. Para el espacio de palabras en español, el tamaño del bucket fue de 127, el promedio de uso de éste 17,06, la cantidad de *gnats* 276 y la cantidad de buckets 4.720.

4.1.2. Búsqueda en el *egnat*

Una búsqueda en un *egnat*, se realiza recursivamente usando el algoritmo de búsqueda del *gnat*. con la siguiente modificación:

- En el caso de que la búsqueda se realizara sobre un nodo de tipo bolsa, al mantener 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, entonces, si

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

$$Dist(s_i, p) < Dist(q, p) - r$$

  - 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.

5. Resultados experimentales

Los experimentos realizados con la nueva versión del *egnat* muestran que supera ampliamente al *gnat* y al *gnat con radio cobertor*. Durante la construcción superó en todas las medidas de costo a las otras estructuras. En la eliminación con planos fantasmas, el *gnat* superó al *egnat* en costos de distancia, pero en accesos a disco fue muy superior el *egnat*. La información respecto de la búsqueda se muestra en las figuras 9 y en la figura 10. Durante la búsqueda con 40 % de datos eliminados y reinserción del 40 %, el *egnat* es levemente superior en cálculos de distancia y muy superior en costos de lecturas en disco.

La figura 11 muestra gráficos comparativos entre el *egnat* y el *M-Tree*. El M-Tree es una de las pocas estruturas dinámicas y diseñadas para memoria secundaria. En los gráficos los valores de MIN_UTIL de 0,4 y 0,1 son parámetros del *M-Tree* que garantizan un 40 % y un 10 % de utilización del nodo o página de disco. En esta figura se demuestra que el egnat es muy superior en términos de evaluaciones de distancia y levemente peor en términos de lecturas en disco.

Durante la construcción el *M-Tree* superó levemente al *egnat* sólo en cálculos de distancia y escrituras en el espacio de Gauss, en el espacio de palabras, siempre fue mejor el *egnat*.

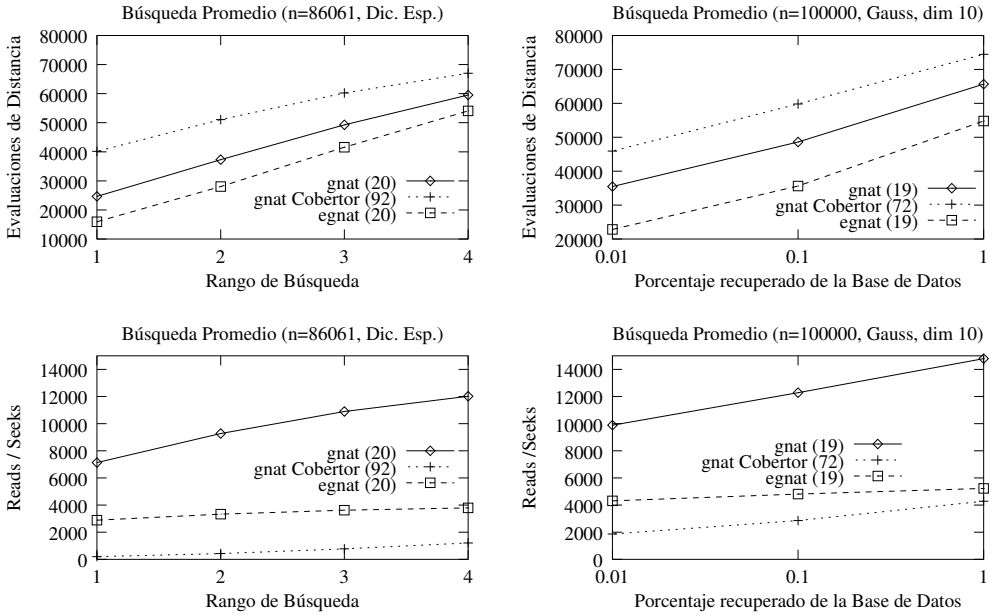


Figura 9: *egnat* (versión final): Búsquedas sin eliminación.

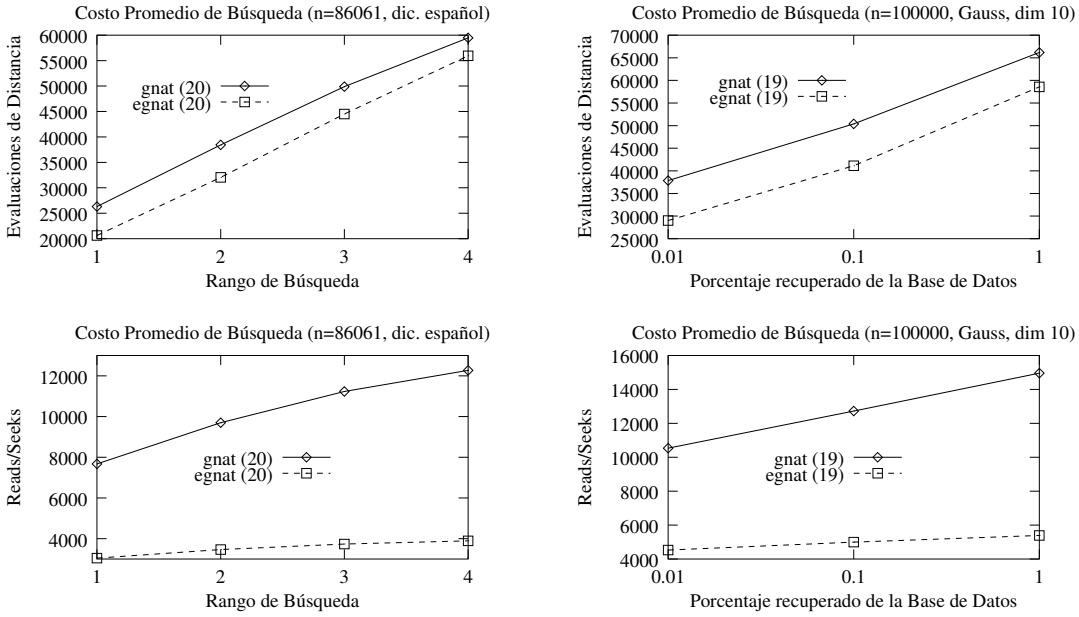


Figura 10: *egnat* (versión final): Búsquedas con 10 % de eliminaciones y reinserciones sobre los espacios de palabras (izquierda) y Gauss (derecha).

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

6. Discusión Final

Para el presente trabajo también se analizaron ciertos casos adicionales en función de disminuir aún más la cantidad de espacio requerido por la estructura, esto es debido a que el *M-Tree* mantenía cierta ventaja con respecto al *egnat*, sin embargo, esta sería la única desventaja importante del *egnat*.

Una solución natural a este problema es compartir la página de disco entre varios nodos. La tabla 7 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 anteriormente, siendo los costos similares o peores al *M-tree* y mucho peores respecto al *egnat*.

La misma idea de compartir páginas se puede aplicar al *egnat*. Si el nodo es de tipo *gnat* utilizará la mayor cantidad de centros posibles, pero si es bucket, compartirá la página 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 7 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 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 8, 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 7 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*.

Finalmente, el *egnat* se comporta similar o mejor a su versión B.

7. 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.

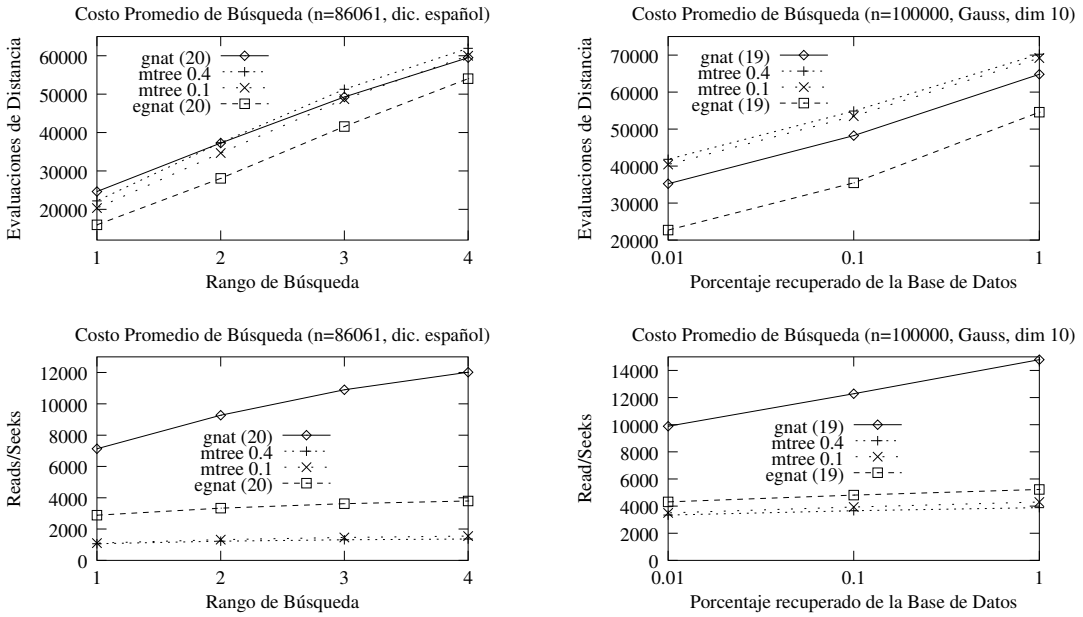


Figura 11: Comparativas de costos de búsqueda *gnat*, *M-tree* (MIN_UTIL=0.4), *M-tree* (MIN_UTIL=0.1) y *egnat*, para los dos espacios de prueba.

	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 7: 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 8: *egnat* (versión final B): detalles de construcción para páginas con capacidad para 4 y 6 bolsas (100 % de los datos).

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

7.1. 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.

- 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.

- 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.

7.2. 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 y generar nuevas alternativas para la disminución del tamaño de la estructura 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.
- Analizar y comparar el método de planos fantasmas sobre distintas estructuras de tipo árbol, como también con 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.

Referencias

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

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

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

[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.

[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.

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

[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.

[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.

[Uri05] Roberto Uribe. Manipulación de estructuras métricas en memoria secundaria. Master's thesis, Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile, Santiago, Chile, Abril 2005.

[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.