Una Estructura Dinámica para Búsqueda en Espacios Métricos

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)

Gonzalo Navarro [**]
Depto. de Ciencias de la Computación
Centro de Investigación de la Web
Universidad de Chile, Blanco Encalada 2120, Santiago, Chile
(gnavarro@dcc.uchile.cl)

Resumen

El *gnat* o Geometric Near-neighbor Access Tree es una estructura de datos para búsquedas por similaridad en espacios métricos [Bri95]. Esta estructura es prometedora dado que se ha demostrado que tiene buen desempeño en espacios de alta dimensión, sin embargo, es estática, es decir, no esta diseñada para la inserción y eliminación de objetos una vez construida, lo que implica que no puede ser usada en una serie de aplicaciones interesantes.

El presente trabajo describe la propuesta de una versión dínamica del gnat con el diseño e implementación de métodos de inserción y eliminación en el árbol. Finalmente se demuestra que es posible dar pleno dinamismo a la estructura y ofrecer un método adecuado y de bajo costo para la eliminación, además, manteniendo un buen desempeño en la búsqueda.

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

[*]Parcialmente financiado por el Proyecto MECESUP MAG9901, Mineduc, Chile.
[**]Parcialmente financiado por el Nucleo Milenio Centro de investigación de la Web, Proyecto P01-029-F, Mideplan, Chile.

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 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.

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 en 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].

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 si y sólo si la distancia euclideana $d(q, c_i) < d(q, c_j)$ para cada c_j, con $j \neq i$.

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.

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 subarboles, el resto de los centros se descarta.

El dinamismo es poco común en las estructuras para espacios métricos [CNBYM01], sin embargo, algunas estructuras permiten inserciones eficientes una vez construidas. Respecto de 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. Resultados experimentales de estructuras dinámicas pueden encontrarse en [NR02].

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 ambos espacios muestran claramente el comportamiento del gnat.

2. Geometric Near-neighbor Access Tree

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, este 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}.

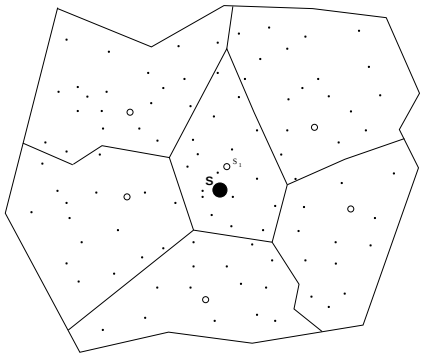

Figura 1: gnat: partición del espacio, representación de subplanos.

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 $x \in D_{p_j} \cup \{p_j\}$.

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 algún momento en la construcción de un plano *P(S)*, cuyo split o centro es S, las divisiones internas corresponderían a los subplanos interiores de S.

2.2. Búsqueda en el gnat

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

1. Se asume que se desea buscar todos los puntos con distancia $d \leq r$ a el punto x. Sea P la representación del conjunto de puntos split del nodo actual (inicialmente la raíz del *gnat*) el cual posiblemente contiene un vecino cercano a x. Inicialmente P contiene todos los puntos split del nodo actual.

2. Se toma un punto p en P, se calcula la distancia *d(x,p)*. Si $d(x,p) \leq r$, se agrega p al resultado.

3. $\forall\, q \in P$, si $[d(x,p) - r, d(x,p) + r] \cap range(p, D_q)$ es vacío, entonces se elimina q 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}.

3. Eliminación en el gnat

3.1. 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, y ahora reinserciones y eliminaciones. Entonces, 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 reinserciones, 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 la estructura *sa-tree*.

Descartando inicialmente el caso en que el dato se encuentra en una hoja, lo cual no ofrece complicación en la eliminación, 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, este es evaluado con cada uno de sus ancestros y los hermanos de éste (ancestro es el centro de cada superplano al que pertenece el objeto), y con sus propios hermanos, esto con el objetivo de obtener los rangos (*frontera*) 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 y sus propios planos.

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 hacia el subárbol cuya raíz es el objeto.

4. Los rangos originales almacenados del objeto para su plano y para sus planos adyacentes (hermanos), ya no corresponderían.

3.2. Algoritmos propuestos

3.2.1. Reconstrucción de subárboles

Una primera alternativa busca mantener la estructura en su forma original, es decir, que tenga todas las propiedades de un *gnat*.

Esto se podría conseguir usando un objeto cualquiera como reemplazo y recalcular 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 splits con rangos sobredimensionados.

La necesidad de reinsertar todos los elementos del subárbol, es para mantener la coherencia de los rangos, sin embargo, es posible encontrar algunos subárboles que podrían insertarse completos, pero

no implicaría que se eximieran de algún cálculo de distancia.

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

3.2.2. Planos Fantasmas

En esta idea lo que se hace es 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 los planos, lo cual implica un nuevo elemento a considerar en los métodos de inserción y búsqueda.

La elección del objeto de rremplazo 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*.

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 se representa a través del siguiente algoritmo :

1. al encontrar el dato, se marca el nodo y el split como *afectado*, con el objeto de considerarlo en las búsquedas.
2. se busca el objeto más cercano al dato (entre sus descendientes)
3. se elimina el split definitivamente al reemplazarlo por objeto encontrado.
4. se conservan los rangos del conjunto del split original.
5. se repite el proceso recursivamente sobre el subarbol 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 1, el cual es hijo de S y centro de su propio plano. Con el algoritmo planteado, se elegiría el elemento señalado con la flecha en la figura 2, el más cercano a S_1, con esto el plano sufriría un corrimiento como se muestra en la figura 2. Entonces llamaremos plano fantasma a $P(S_1)$ antes de su eliminación, el cual es representado en la misma figura por líneas no contínuas.

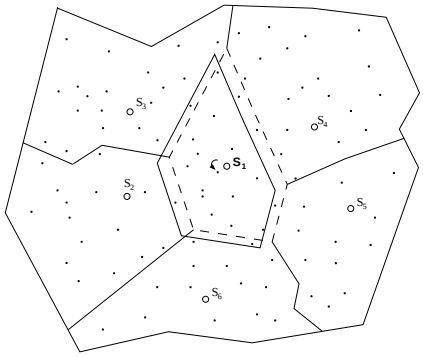

Figura 2: gnat: eliminación de un *split*, vista de un *plano fantasma*.

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.
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.
Para visualizar como es afectada la estructura despues de eliminar un split, considere la figura 3, la cual representa un árbol que no contiene datos eliminados. Esta figura podría representar el plano indicado en la figura 2 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, donde el asterisco representa la marca de *afectado*, tanto en el nodo como en el split. El elemento S_h es el ubicado en una hoja que sería el último en

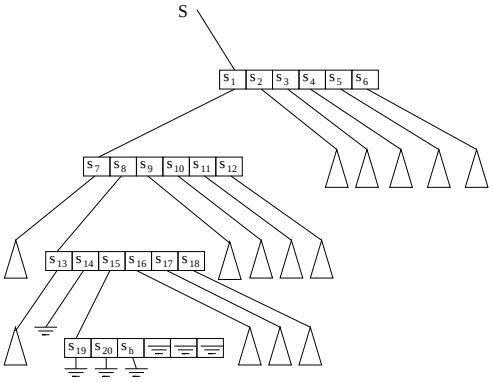

Figura 3: gnat: Estructura original, sin elementos eliminados.

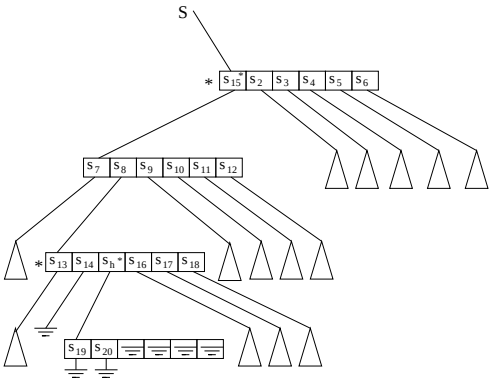

Figura 4: gnat: Árbol después de la eliminación de un Split, usando planos fantasmas y reemplazo del más cercano.

moverse.

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

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 superposicíon de planos sería mínima, sin embargo, esto podría afectar además a subárboles adyacentes, lo que deformaría aún más el *gnat*.

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 split marcados como afectados, lo que afectaría en los costos a los metodos 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*.

Reemplazo por descendiente hoja : Una tercera alternativa es el reemplazo directo por un objeto ubicado en alguna hoja del arbol cuya raíz es el elemento a borrar, esto podría ocasionar un solapamiento mayor de los planos, pero con la garantía que no hay que hacer evaluaciones de distancia y sólo basta 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, se puede decir que sólo un nodo y split son afectados y no otros descendientes, es decir, el subárbol completo a partir del elemento borrado sigue siendo un gnat.

Para el árbol de la figura 3, aplicando este método la estructura quedaría como se muestra en la figura 5, considerando que la hoja donde se busca el reemplazante es la misma que para el primer algoritmo.

3.3. Búsquedas después de la eliminación

Para el algoritmo de *planos fantasmas*, el definir una marca de *split afectado*, permite considerar a este split en forma especial en los distintos métodos. La marca almacena la distancia entre el objeto eliminado

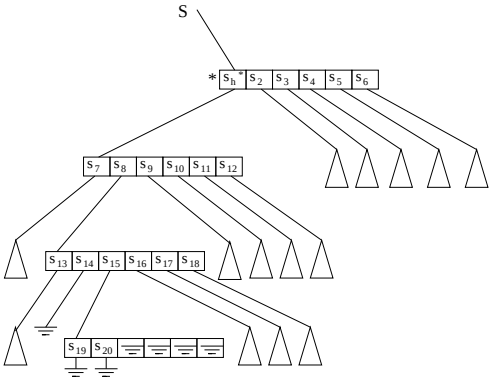

Figura 5: gnat: Árbol con una eliminación, usando planos fantasmas y reemplazo directo de una hoja.

y su reemplazante, lo que indicaría cuanto se movió el plano.

El método 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 split que realmente existe.

3.3.1. Búsquedas por rango

Si se han realizado inserciones luego de eliminaciones, entonces durante la búsqueda por rango al llegar a un nodo marcado, se debe incluir en el conjunto respuesta el subárbol cuyo split está marcado, 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 split, 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.

3.3.2. Eliminación y búsquedas exactas

Para una búsqueda exacta, luego de eliminaciones e independiente de nuevas inserciones, el algoritmo varía debido a que 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, al llegar a un nodo afectado, se analizaron dos alternativas:

1. La primera, es realizar la búsqueda del más cercano dentro del nodo, tomando como primer split, uno no marcado.

2. La segunda, es proceder con el algoritmo original, pero agregar a la búsqueda, los subárboles cuyos centros están marcados.

La primera opción fue descartada dado que en la búsqueda por rango, independiente si existen planos fantasmas o gnat reales se comporta de la misma manera y no descarta subárboles sin marca, además de las consiguientes evaluaciones de distancia.

La segunda variación resulta a la vista más eficiente, dado que se agregan a la búsqueda sólo aquellos subárboles marcados, por lo demás, si aparecen nodos no marcados, lo que quiere decir que no existe overlap, el proceso dentro de éstos es directo, o sea, el árbol a revisar es aquel con la menor distancia entre el dato y los splits del nodo.

3.3.3. Optimización

Si el grado 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).

Es posible evitar buscar en todos los marcados usando un factor de incertidumbre, es decir, si se reemplaza S_1 por S, entonces $I = d(S, S_1)$, el cual es el valor almacenado en la marca de borrado. Por lo tanto, cualquier decisión respecto de ese subárbol puede estar erróneo por $\pm I$. Por ejemplo, en una búsqueda exacta si $d(q, S_1) > d(q, S_2)$, se descarta S_1, ahora, si S_1 fue eliminado, entonces igualmente podemos descartar S si $d(q, S) > d(q, S_2) + I$.

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

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

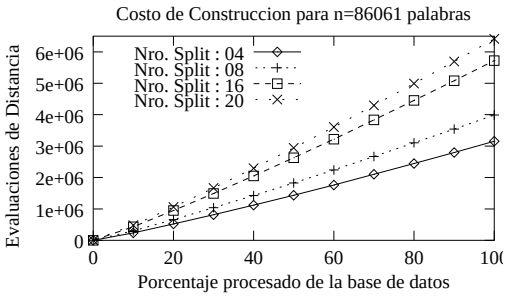

Figura 6: Costos totales de construcción para el diccionario Español.

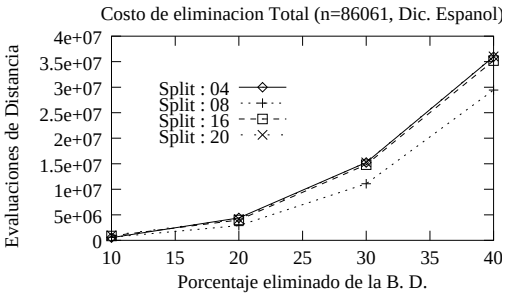

Figura 8: Costos totales de eliminación del 40 % para el diccionario Español.

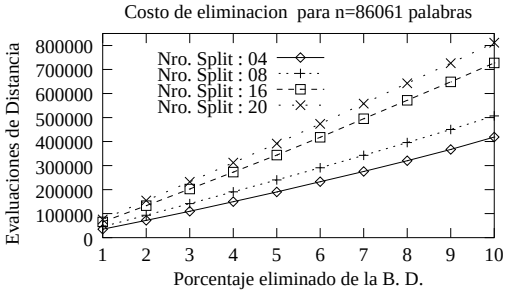

Figura 7: Costos totales de eliminación del 10 % para el diccionario Español.

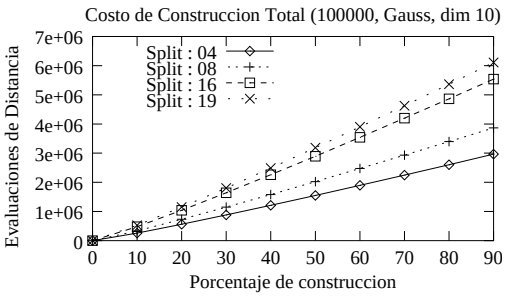

Figura 9: Costos totales de construcción para el espacio de vectores de Gauss de dimensión 10, 90 % de objetos.

4. Resultados experimentales

4.1. Eliminaciones

Para los experimentos de eliminaciones, se construyo 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.

Para la eliminación se usa la búsqueda exacta, que es igual a los métodos usados para la construcción, se incluye el gráfico de construcción para cada uno de los espacios a modo de referencia (figuras 6 y 9).

El método utilizado para la eliminación es el de *reemplazo por un decenciente ubicado en una hoja*.

4.2. Búsquedas

Para los experimentos de búsquedas, se reservó un conjunto del 10 % de objetos, el cual es el que se buscará. 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 el rango que recuperaban el 0.01, 0.1 y 1 por ciento 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 reinsertó la misma cantidad, sobre esto se realizó la búsqueda del 10 % restante.

El método de búsqueda utilizado en los experimentos es el propuesto inicialmente en 3.3.1.

5. Conclusiones

Se ha presentado una versión dinámica de la estructura *gna-tree*, la cual permite realizar inserciones y eliminaciones eficientemente sin afectar la calidad de las búsquedas.

Tanto los métodos de inserción, búsqueda y eliminación para las tres variaciones de *planos fantasmas*, son los mismos. Es importante destacar que con la opción de reemplazo por un objeto ubicado

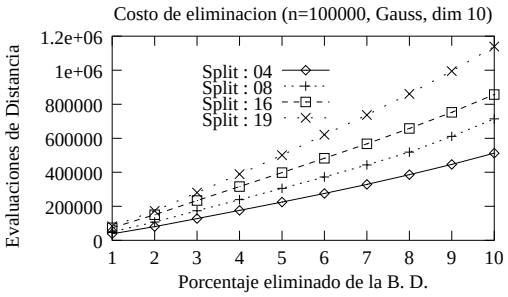

Figura 10: Costos totales de eliminación del 10 % para el espacio de Gauss.

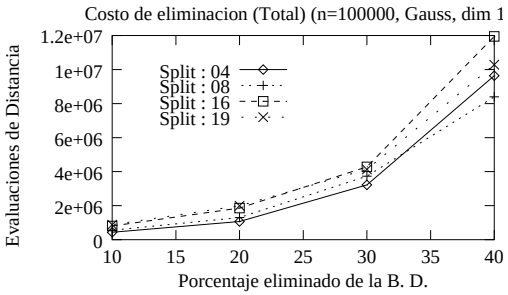

Figura 11: Costos totales de eliminación del 40 % para el espacio de Gauss.

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.

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 respecto de la estructura sin eliminaciones sigue siendo adecuado.

Las alternativas propuestas también han sido inicialmente las más eficientes en el trabajo que se realiza en la actualidad, que es proveer de una estructura dinámica eficiente que trabaje en memoria secundaria, aquí también son relevantes junto con las evaluaciones de distancia, los accesos a disco y el tamaño de la estructura en disco.

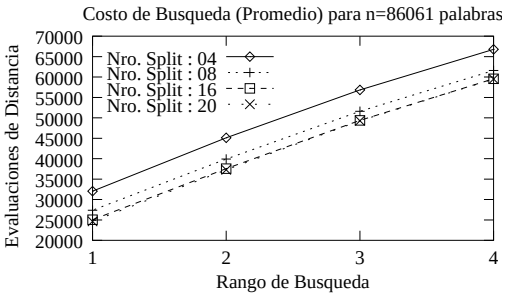

Figura 12: Costos de búsqueda para el diccionario español, sin eliminaciones.

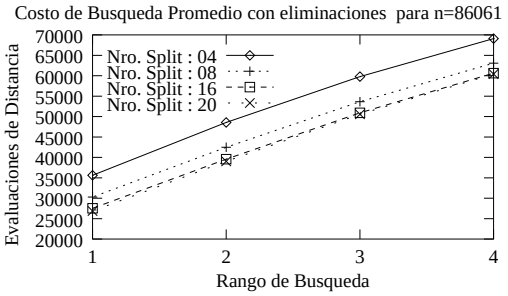

Figura 13: Costos de búsqueda para el diccionario español, con el 10 % eliminado y reinsertado.

Referencias

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

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

[BYCMW94] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixedqueries trees. In *5th Combinatorial Pattern Matching (CPM'94)*, 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, 2001.

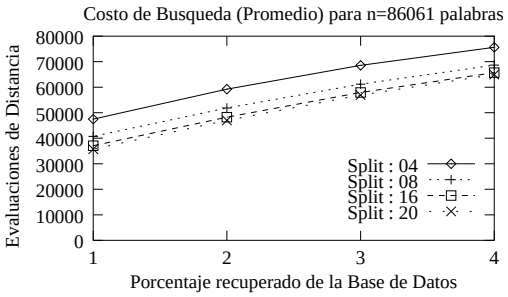

Figura 14: Costos de búsqueda para el diccionario español, con el 40 % eliminado y reinsertado.

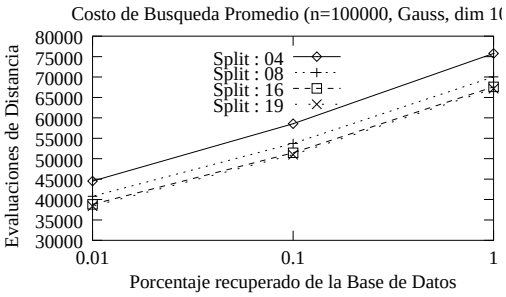

Figura 16: Costos de búsqueda para el espacio de Gauss, con el 10 % eliminado y reinsertado.

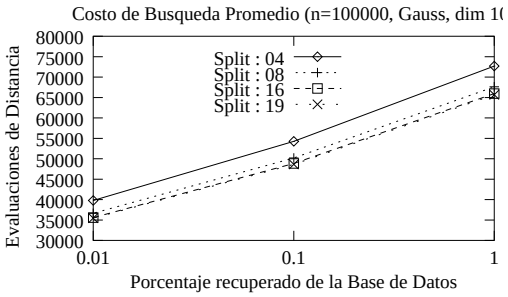

Figura 15: Costos de búsqueda para el espacio de Gauss, sin eliminaciones.

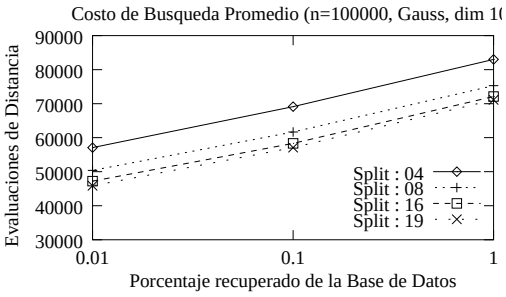

Figura 17: Costos de búsqueda para el espacio de Gauss, con el 40 % eliminado y reinsertado.

[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)*, 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, 2002.

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

[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.