Una Gráfica de Proximidad para Búsqueda de Similaridad

Edgar Chávez and Héctor Tejeda

Escuela de Ciencias Físicas y Matemáticas, Universidad Michoacana, Morelia, Michoacán, México,
{elchavez,htejeda}@fismat.umich.mx, Con apoyo de CONACyT, proyecto R-28923A

Resumen En este trabajo se presenta un grafo para búsqueda de similaridad. Esta estructura es construida en forma dinámica, con la finalidad de permitir inserciones y borrados, además de que no es necesario conocer previamente los elementos para construir su índice, lo cual es muy deseable en ciertas aplicaciones de búsqueda de similaridad. Se presentan tres heurísticas en donde dos de ellas son dinámicas. Estas estructuras que se generan son elementos del superconjunto del *árbol de aproximación espacial,* (SAT) que ha probado ser de utilidad en espacios métricos de dimensión alta.

1 Introducción

La búsqueda de similaridad es una generalización de la búsqueda exacta, en donde se permite buscar con tolerancia en una base de datos. Esta generalización del modelo permite realizar peticiones por contenido en objetos multimedia, tales como en huellas digitales, reconocimiento de voz, buscar datos multimedia, hacer reconocimiento de patrones (vía el clasificador k-nn), comprimir audio y vídeo en el esquema MPEG, realizar búsqueda por contenido en bases de datos textuales y es de amplio uso en un gran número de aplicaciones computacionales y matemáticas. Un trabajo reciente del análisis de éste problema y las distintas soluciones que han aparecido, se encuentra en [3].

Entre las soluciones mas eficientes que se han encontrado, destaca la idea de *aproximación espacial* [6] SAT por sus siglas en inglés, en donde se construye un árbol que adopta la geometría del espacio que se pretende indexar. SAT es un grafo de proximidad, inspirado en el dual del diagrama de Voronoi [1]. La búsqueda se realiza siguiendo las aristas del grafo. Una de las partes débiles del algoritmo descrito en [6] es que el índice no es dinámico; esto es, se deben tener todos los elementos de la base para poder construir el árbol, y como consecuencia de lo anterior, la inserción o el borrado de cualquier elemento de la base de datos implica la reconstrucción del índice.

Por otra parte, en [5] se propone una solución alternativa a SAT (gráfica de covecindad), que es una estructura dinámica para búsqueda de similaridad, ésta permite hacer inserciones y borrados y vuelve dinámica la estructura de datos.

En este trabajo se presenta un grafo, que es sencillo y útil, ya que resuelve los problemas anotados previamente, al permitir inserciones y borrados, con lo cual la estructura de datos se vuelve dinámica. Así mismo, se plantean algunas heurísticas que permiten mejorar la eficiencia de la búsqueda. Otra característica importante es que se puede construir mas rápidamente que el SAT.

2 Definición del Problema

El conjunto $\mathbb{X}$ denotará el universo de objetos *válidos*. Un subconjunto finito de ellos, $\mathbb{U}$ de tamaño $n = |\mathbb{U}|$, será llamado el *diccionario*, que es el conjunto de objetos donde se hará la búsqueda. Este conjunto es un espacio métrico, con una distancia d que respeta las propiedades usuales de simetría, positividad y desigualdad triangular.

Existen básicamente tres tipos de peticiones de interés en los espacios métricos:

(a) ***Petición por rango*** $(q, r)_d$. Recupera todos los elementos que estén dentro de una distancia r respecto a q. Esto es, $\{u \in \mathbb{U} / d(q, u) \leq r\}$.
(b) ***Petición del vecino más cercano*** $NN(q)$. Recuperar los elementos mas cercanos a q en $\mathbb{U}$. Esto es, $u \in \mathbb{U} / \forall v \in \mathbb{U}, d(q, u) \leq d(q, v)$. En algunos casos es suficiente con un elemento (en espacios continuos hay normalmente sólo uno). Se puede también proporcionar una distancia máxima r^* tal que si el elemento más cercano esta a una distancia más allá de r^*, entonces no sea devuelto.

(c) ***Petición de los k-vecinos cercanos*** $NN_k(q)$. Recuperar los k elementos más cercanos a q en $\mathbb{U}$. Esto es, recuperar un conjunto $A \subseteq \mathbb{U}$ tal que $|A| = k, \forall u \in A, v \in \mathbb{U} - A, d(q,u) \leq d(q,v)$. Para el caso de que haya elementos a la misma distancia, se puede devolver cualquier conjunto de k elementos que satisfagan la condición.

Se puede observar que cualquier tipo de petición puede ser resuelta examinando todo el diccionario $\mathbb{U}$. De hecho, si no se permite el preprocesamiento de los datos, entonces la revisión exhaustiva es la única forma de proceder. Una forma de evitar lo anterior, es construir una estructura de indexamiento con los datos. En este trabajo se considera que la función de distancia d es costosa de calcular, y por lo tanto el número de cálculos hechos será la medida de la complejidad del algoritmo propuesto.

Un *algoritmo de indexamiento* es por lo general un procedimiento fuera de linea (off-line), el cual genera una estructura de datos (o "índice") que permite ahorrar cálculos de distancia en las peticiones de proximidad que se harán. La construcción de la estructura de datos tiene un coste elevado, pero dicho coste queda amortizado al ahorrar evaluaciones de distancia en las busquedas de proximidad que se realizan posteriormente y que, en principio, serán muy numerosas. La meta es, por lo tanto, diseñar algoritmos eficientes de indexamiento con el objetivo de reducir el número de evaluaciones de distancia. La estructura que se obtiene usa la desigualdad triangular para descartar elementos. Para una completa revisión del estado del arte en algoritmos de indexamiento basados en distancia el lector puede consultar [3].

3 Motivación

El diagrama de Voronoi es un grafo de proximidad que asigna a cada elemento de la base de datos la región del espacio que define su zona de influencia, es decir, la región asignada es aquella en donde los puntos del espacio tienen como vecino más cercano al elemento de la base de datos. En la frontera de cada región de influencia se encuentran aquellos elementos que "comparten" el vecino más cercano.

3.1 El Grafo de Delaunay

En este grafo dos puntos tendrán un arco cuando compartan una frontera. Se puede emplear el grafo de Delaunay para realizar consultas de proximidad usando la siguiente idea:

Sea a un elemento arbitrario de la base de datos, q un elemento de consulta y $N(a)$ el conjunto de vecinos de a en el grafo de Delaunay.

- Si $d(q, v_i) \geq d(q, a)\ \forall\ v_i \in N(a)$ entonces a es el vecino más cercano de q. Terminar.
- Hacer $a \leftarrow v_i$ si v_i es el más cercano a q en $N(a)$. Repetir el paso anterior.

Usando la definición del grafo de Delaunay es posible mostrar que el algoritmo anterior es correcto. Se podría pensar en generalizar el grafo de Delaunay para espacios métricos arbitrarios, y utilizar el algoritmo anterior par realizar la búsqueda del vecino más cercano. Debido a que sólo se cuenta con la matriz de distancias (ya que es la única información que se puede obtener de un espacio métrico general) no es posible construir el grafo; en el trabajo de [6] se prueba formalmente que el único grafo que tiene esa propiedad, construido en base a la matriz de distancias es el grafo completo.

3.2 El Árbol de Aproximación Espacial (SAT)

El árbol de aproximación espacial está definido en forma recursiva; la propiedad que cumple la raíz (y en turno cada uno de los siguientes nodos) es que los hijos están más cerca de la raíz que de cualquier otro punto de la base de datos. Lo anterior puede indicarse como: $x \in N(a) \leftrightarrow \forall y \in N(a) - \{x\}, d(x,y) > d(x,a)$ A este tipo de construcciones recursivas se les denomina "clique". La figura 1 muestra el árbol de aproximación espacial para un conjunto pequeño de puntos en el plano.

De acuerdo a la definición del árbol de aproximación espacial se observa que se necesita tener al inicio a todos los elementos de la base de datos para la construcción, es decir, no es posible construir el grafo dinámicamente.

El costo de una búsqueda por rango empleando SAT es $O(n^{1-\Theta(1/\log \log n)})$

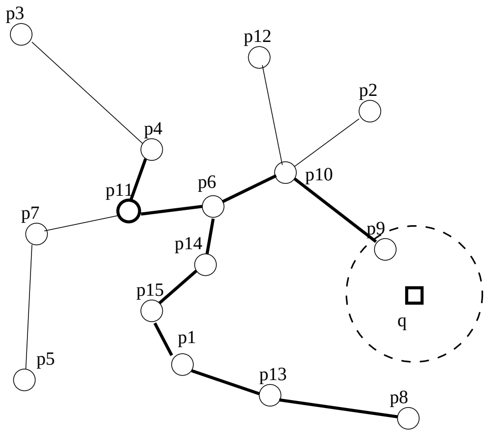

Figura 1: Ejemplo de búsqueda en el árbol SAT. Se encuentra remarcado la trayectoria hecha para una consulta q, la cual comenzó en p_{11}.

4 Propuesta

En este trabajo se presentan varias ideas con el afán de lograr mejoras en el SAT. Una de ellas es que se tiene una estructura que es dinámica, es decir se pueden realizar inserciones y borrados. Además la estructura presenta un buen desempeño tanto en dimensiones bajas como altas.

Una de las ideas principales de este trabajo descansa en el hecho de que un algoritmo que realiza búsqueda exacta puede ser extendido, para realizar búsqueda aproximada. Por otra parte, se emplea únicamente la función de distancia para construir una estructura que es empleada para discriminar puntos al momento de realizar la búsqueda.

4.1 Heurística RFN (*Random First Neighbor*).

En la heurística que se presenta se considera para la construcción del árbol que la inserción de datos se hace en forma aleatoria. Es decir, no se lleva a cabo ningún proceso previo a la construcción del mismo. Se debe hacer notar que ésta estructura es dinámica totalmente, ya que una vez que se ha terminado de construir el árbol con un conjunto $\mathbb{S}$ de elementos, si así se requiere se pueden agregar o eliminar elementos.

Algoritmo de Construcción.

(a) Se selecciona un punto del diccionario $\mathbb{S}$, que será el nodo raíz. A partir de este punto o nodo se iniciará la revisión en el árbol para cualquier punto que vaya a ser insertado.
(b) Se calcula la distancia entre el punto que se va a insertar y el nodo en el cual se está dentro del árbol ($d(pad, q)$). Se empieza con el primer hijo que haya sido insertado, calculándose su distancia con el punto a insertar $d(h_i, q)$. Se comparan las distancias $d(pad, q)$ y $d(h_i, q)$, si la distancia con el hijo es menor entonces moverse a ese hijo, caso contrario se revisa el siguiente hijo h_{i+1}. Cada vez que se mueva hacia un hijo, éste paso se repetirá considerando al hijo ahora como al padre.
(c) Una vez que ya no sea posible moverse en el árbol, se deberá a una de dos posibles razones, el nodo que se está revisando no tiene hijos, o bien, no existe ningún hijo que mejore la distancia con el punto que se está insertando, por lo tanto, el nuevo punto deberá conectarse al nodo que se llegó.

Se muestra a continuación la construcción del grafo de acuerdo al procedimiento de construcción. El ejemplo que se muestra es considerando 20 puntos en el plano cartesiano y usando la función de distancia euclidiana $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. Se insertará un punto a la vez, en orden aleatorio.

En la figura 2 se muestran los puntos que se insertarán, y el orden en el cual serán ingresados.

De acuerdo con el algoritmo descrito previamente, el punto con el número 1 es la raíz, cuando se inserta el segundo punto, como solamente existe un punto (la raíz) en el árbol, éste es conectado con el primero. El árbol resultante es el que se muestra en la figura 3.

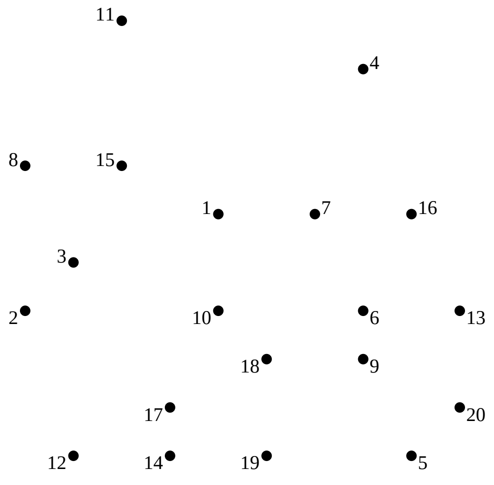

Figura 2: Puntos que se insertarán en el árbol.

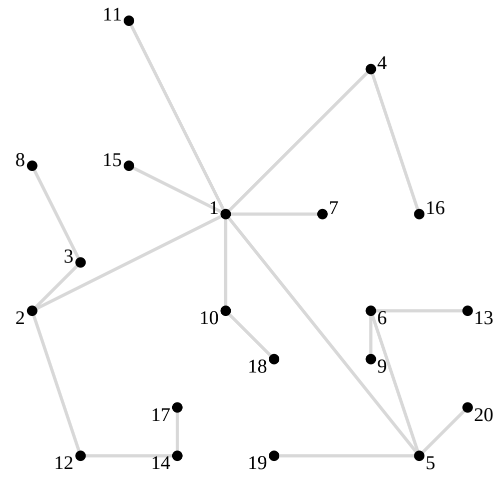

Figura 3: Árbol obtenido con la heurística RFN

Búsqueda Exacta. Con las construcción anterior siempre es posible hacer búsqueda exacta utilizando solo calculos de distancia, siguiendo la rama que mejora la distancia al objeto que buscamos. Esto funciona porque en la estructura se va guardando implícitamente el orden en el que fueron agregados los hijos para algún nodo, de esta forma si se vuelve a repetir el proceso de inserción para un elemento que ya esta en el árbol, siempre que se recorra el árbol por las mismas aristas.

El recorrido del árbol puede terminar en algún nodo por tres posibles causas:

- La distancia entre la consulta y el nodo es cero;
- Se llega a una hoja del árbol, por lo cual ya no se puede seguir bajando más niveles; o
- Para algún nodo padre en el que nos encontremos, no existe ningún hijo que mejore la distancia respecto a la petición.

Búsqueda por rango. La búsqueda por rango se puede obtener como una extensión de la búsqueda exacta, es decir, se pueden seguir los mismos criterios, teniendo en cuenta que para moverse a una rama se deberá satisfacer la condición de que $d(h_i, q) \leq d(pad, q) + r$, en donde r indica el radio o la tolerancia de la petición. Como no existe más información dentro del árbol para descartar ramas a visitar, entonces se deben revisar aquellas ramas que cumplan la condición anotada previamente.

La búsqueda puede concluir dentro de una rama una vez que se llegue a una hoja, o bien, por que no existe algún hijo que mejore la distancia respecto al padre considerando el radio r.

Análisis de complejidad Se analiza la estructura que se genera como resultado de haber aplicado la heurística RFN. El análisis que se presenta se ha simplificado al suponer que la distribución de nodos que van en cada subárbol es el mismo que en el espacio global.

Suponiendo que se tenga un número n de nodos ya introducidos se indicará cual es el costo promedio de insertar un nuevo elemento. El cálculo que se obtendrá estará en términos de los cálculos de distancia hechos. Sea a la aridad de cada nodo en el árbol y sea h la altura total del árbol. Un nuevo nodo siempre se compara con la raíz y en promedio con la mitad de sus hijos, por lo que el coste medio viene descrito por la recurrencia

$$C_n = \frac{a}{2} + C_{\frac{n}{a}}$$

cuya solución es

$$C_n = \frac{a}{2} \log_a(n) = O(\log n)$$

tomando $n = a^h$.

El coste de construcción del árbol es:

$$\sum O(\log i) = \log(n!) = O(\frac{n}{\log n})$$

Una vez que se tiene construido el árbol, el costo de hacer una búsqueda exacta es igual a la de insertar un elemento, para conocer el costo de una búsqueda por rango se deben revisar varias ramas, designando por k al número de ramas visitadas, que es directamente proporcional al radio de búsqueda r se tiene que:

$$C_n = \frac{a}{2} + kC_{\frac{n}{a}}$$

expandiendo la suma

$$C_n = \frac{a}{2} + k\left(\frac{a}{2} + kC_{\frac{n}{a^2}}\right)$$

y hasta el último nivel

$$C_n = \frac{a}{2}k^0 + \frac{a}{2}k^1 + \ldots + \frac{a}{2}k^h$$

entonces

$$C_n = \frac{a}{2}\left(1 + k^1 + \ldots + k^h\right)$$

y si $k > 1$

$$C_n = \frac{a}{2}\left(\frac{k^h - 1}{k - 1}\right) = O(k^h) = O(n^{\log_a k})$$

En general, $C(n) = O(max \log n, n^{\log_a k})$. Puesto que $1 \leq k < a$, se sigue que el coste de la busqueda por rango es en cualquier caso sublineal.

4.2 Heurística SFN (*Sort First Neighbor*).

En esta variante se propone un grafo con inserción ordenada respecto a la raíz. Para la construcción se requiere que se conozcan previamente todos los elementos, ya que antes de iniciar el proceso de inserción los elementos que forman el diccionario son ordenados respecto a un punto que se escoge como la raíz, y los primeros puntos que son insertados en el grafo son los que están más cerca de la raíz. Esto está motivado por el hecho de que se observó experimentalmente que la complejidad de la búsqueda depende de la distancia a la raiz.

El algoritmo de búsqueda exacta y aproximada funciona igual que el algoritmo original. Como el conjunto de datos tiene que estar ordenado respecto a la raiz el costo de construcción sube por el término $n \log n$ necesario para ordenar el conjunto de datos.

4.3 Heurística RBNFN (*Random Best Neighbor First Neigbor*)

De la heurística SFN se deduce que se obtiene un mejor árbol al ordenar los elementos del diccionario antes de ser introducidos. La desventaja que presenta es que no es un árbol dinámico, ya que se requiere tener todos los elementos del diccionario para construir el árbol.

Con el afán de seguir teniendo una estructura dinámica y que mejore a la heurística RFN, se presenta un grafo en donde la inserción es aleatoria y los criterios que se emplean para determinar en que rama se insertan los nodos, dependen del nivel del árbol, esto es, en el primer nivel se inserta en el mejor hijo y en los otros niveles en la primera rama que mejore la distancia. Esto es un compromiso entre la construcción del SAT y el de la heurística RFN.

Se muestra en la figura 4 el árbol obtenido después de haber insertado 20 puntos.

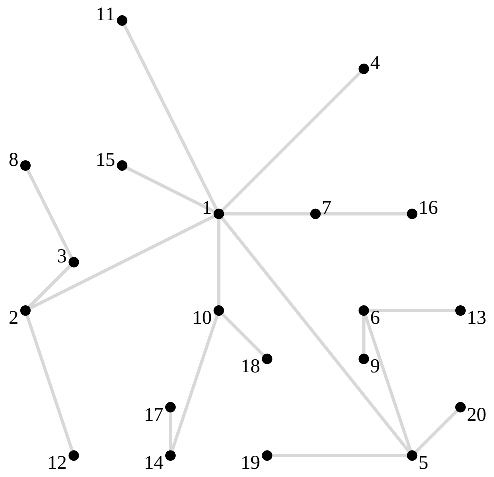

Figura 4: Arbol obtenido con la heurística RBNFN.

Algoritmo para búsqueda exacta En esta heurística no funciona que siempre que se desplace hacia el mejor hijo que mejore la distancia en el primer nivel, se llegue a un elemento del diccionario que haya sido previamente introducido, ya que como el proceso de inserción es aleatorio, se puede dar el caso de que más tarde se inserte algún punto como hijo de la raíz que sea un mejor hijo para algunos nodos ya insertados, en relación a los que fueron determinados al ser insertados dichos puntos.

La forma de resolver lo anterior es considerando todas aquellas ramas de los nodos hijos de la raíz que mejoren la distancia respecto a la raíz.

La condición que se cumple para que una petición sea exitosa, es que la distancia entre la petición y algún punto del árbol sea cero. Por lo que en el algoritmo de búsqueda exacta, ésta será una condición que se revisará siempre que se llegue a un nuevo nodo.

Búsqueda por rango La búsqueda por rango opera prácticamente de la misma forma que la búsqueda exacta, donde ahora se revisarán todas las ramas hijas que mejoren la función de distancia con respecto al padre tomando en cuenta el radio r, esto es, si para una hijo i se tiene que $d(h_i, q) \leq d(n, q) + r$, donde $d(n, q)$ es la distancia del nodo padre a la petición y $d(h_i, q)$ es la distancia del hijo a q.

4.4 Validación Experimental

En esta sección se presentan los experimentos realizados usando los árboles que se construyeron usando los algoritmos propuestos. En la primera sección se muestra el comportamiento de los mismos cuando son usados en espacios discretos, y en la segunda sección se muestran para un espacio continuo.

Espacio Discreto Se analizarán tres ejemplos en donde se usa la distancia de edición entre cadenas de caracteres, es decir, el número mínimo de inserciones, borrados y sustituciones para hacer que dos cadenas sean iguales. La forma como fueron construidas las peticiones, fue usando nuevamente las palabras del diccionario para pedir y ordenando las palabras respecto a la raíz.

En el primer análisis se emplea un diccionario del español, el cual contiene 86,061 palabras. Se grafica para cada elemento, el número de cálculos de distancia realizados. En la figura 5 se muestra la búsqueda aproximada con un radio $r = 1$. Como se observa en la figura el mejor desempeño lo tiene el árbol RBNFN.

El comportamiento del diccionario del español conforme las palabras se encuentran más alejadas de la raíz, se muestra en la figura 6 la gráfica para cada elemento, su distancia a la raíz contra los cálculos de distancia hechos.

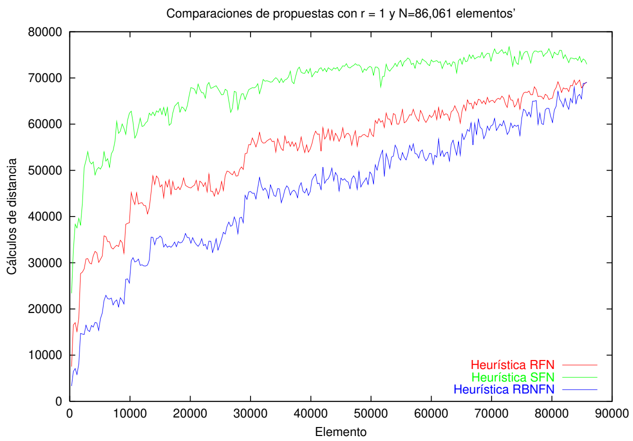

Figura 5: Búsqueda aproximada con $r = 1$.

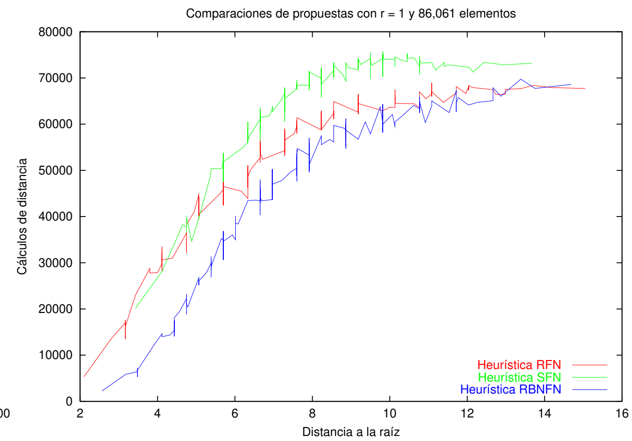

Figura 6: Comparaciones de heurísticas con radio $r = 1$. Distancia a la raíz contra cálculos de distancia.

En el segundo ejemplo que se analiza se muestra el comportamiento de las diferentes heurísticas empleando un diccionario que contiene palabras del inglés. Este diccionario contiene 494,048 palabras obtenidas del vocabulario de la colección Wall Street Journal de TREC, también bajo la distancia de edición.

En la figura 7 se muestra la búsqueda aproximada con un radio $r = 1$.

Para observar el comportamiento del diccionario de palabras del inglés conforme las palabras se encuentran más alejadas de la raíz, se grafica para cada elemento, su distancia a la raíz contra los cálculos de distancia.

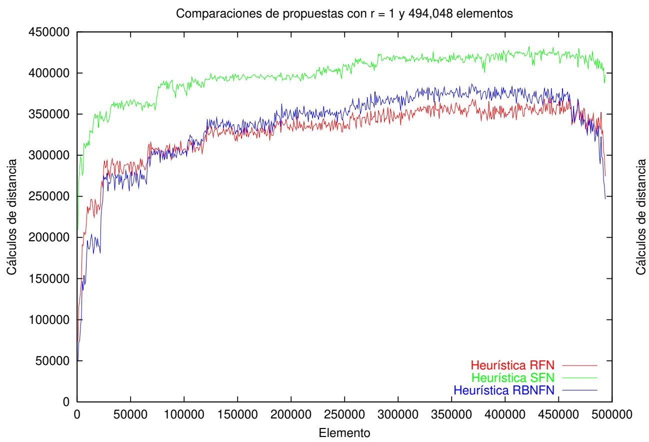

Figura 7: Búsqueda aproximada $r = 1$.

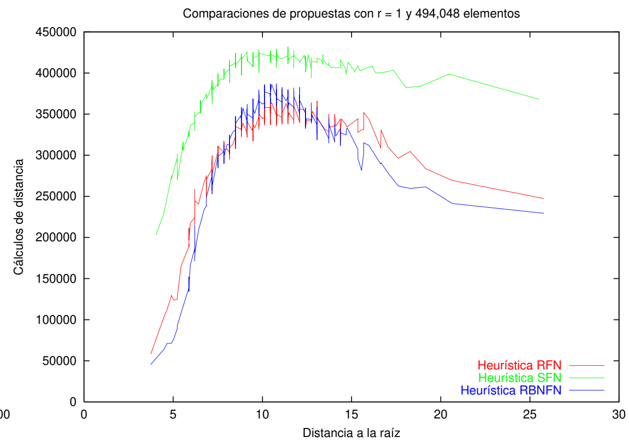

Figura 8: Comparaciones de heurísticas con radio $r = 1$. Distancia a la raíz contra cálculos de distancia.

Como parte del análisis que se hace en el espacio discreto se revisa, el comportamiento de los árboles construidos cuando se hacen peticiones con palabras que no están en el diccionario.

Este experimento se realizó empleando el mismo diccionario de palabras del español que en los experimentos anteriores. Del conjunto de palabras disponibles se tomaron aleatoriamente 70,000 palabras para formar los diccionarios con los que se construyeron los árboles, y el subconjunto restante (16,061 palabras) se emplearon para realizar las peticiones por rango.

Se muestra en la figura 9 el comportamiento que tienen las heurísticas con un radio igual a 1.

Se analiza también en la figura 10 como influye la distancia en las peticiones que se hacen respecto a la raíz del árbol construido. De la figura podemos observar como la heurística SFN permanece casi constante independientemente de la distancia a la raíz de la petición.

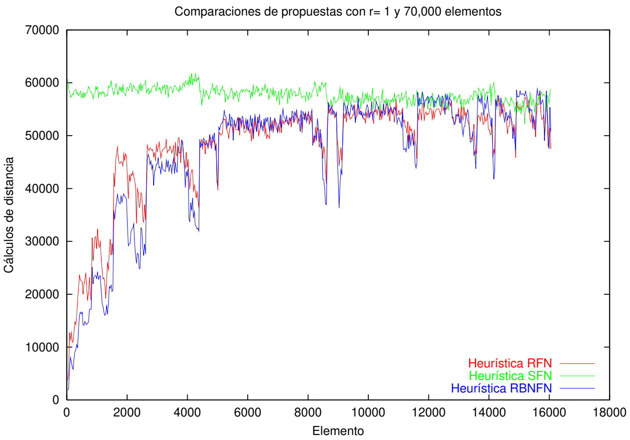

Figura 9: Comportamiento de las heurísticas con $r = 1$ y búsqueda de palabras que no están en el diccionario.

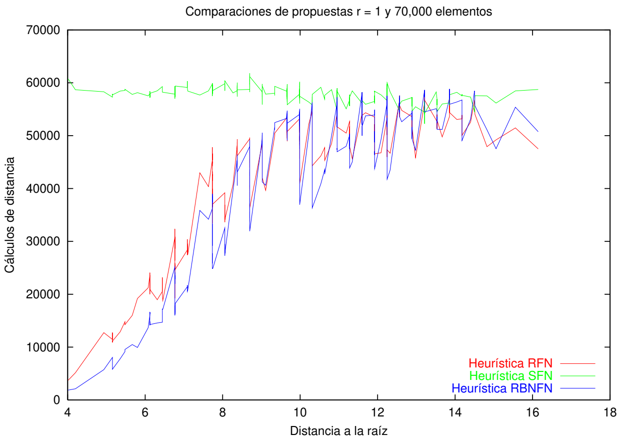

Figura 10: Distintas heurísticas con un radio pequeño.

4.5 Comparación de Heurísticas Vs. Otros Algoritmos

Para observar el comportamiento que tienen las heurísticas que son propuestas en el trabajo, se compara el comportamiento de nuestros algoritmos con el del [6] SAT, [2] GNAT y [4] FQA. En [3] se hace una revisión del GNAT y del FQA, los cuales aquí no son comentados por cuestiones de espacio.

En la gráfica 11 se observa el comportamiento para el diccionario del español que contiene 86,061 palabras.

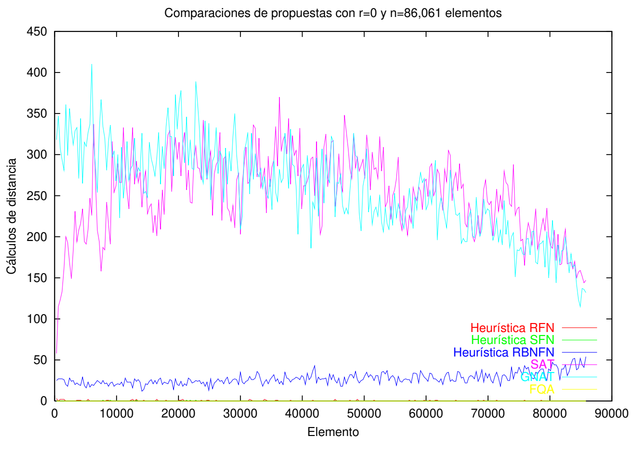

Figura 11: Comparación de las heurísticas con un diccionario de palabras del español.

Se muestra también el comportamiento de las heurísticas, que se tiene con el diccionario de palabras del inglés, se compara nuevamente con los mismos algoritmos que el ejemplo anterior (figura 12).

Finalmente se compara cuando se toma una parte del diccionario del español (70,000 palabras) y el resto del diccionario es empleado para realizar búsqueda exacta (figura 13)

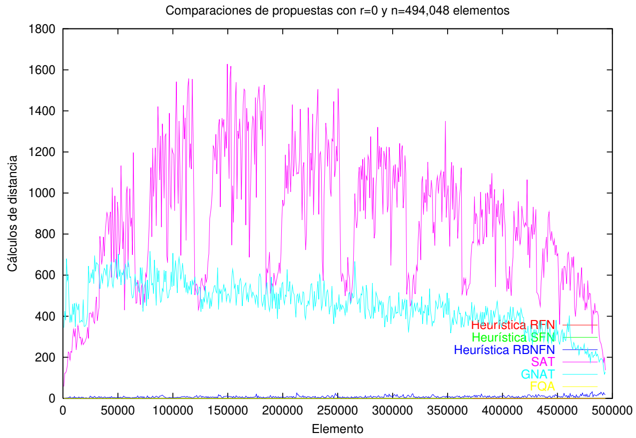

Figura 12: Comparación de las heurísticas con un diccionario de palabras del inglés.

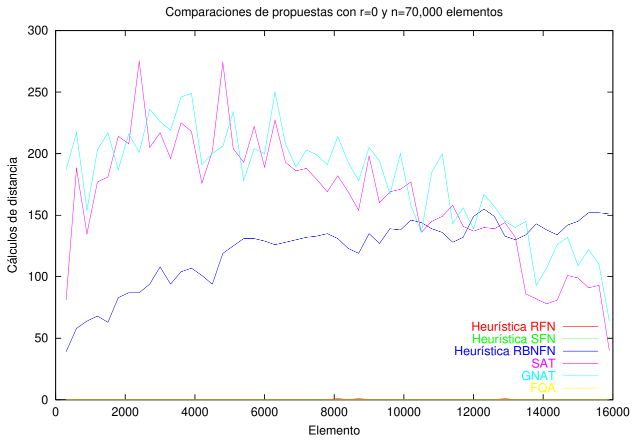

Figura 13: Comparación de las heurísticas tomando una parte del diccionario de palabras del español.

Del resultado de la comparación hecha de las heurísticas propuestas contra los algoritmos que usan aproximación espacial (GNAT y SAT), en búsqueda exacta podemos mencionar que las 3 heurísticas tienen un buen desempeño, salvo cuando se hace búsqueda de palabras que no están en el diccionario, la heurística RBNFN tiene un comportamiento inferior cuando las palabras estan muy alejadas de la raíz (ver figura 13).

5 Conclusiones y Trabajo Futuro

En espacios discretos la heurística RBNFN tiene el mejor desempeño. También tiene un buen desempeño la heurística RFN

Se observa que en espacios discretos conforme se esta más cerca de la raíz, se realiza un menor número de cálculos de distancia Por lo que es importante tener en cuenta la distancia de los elementos a la raíz. Una alternativa interesante se adivina de los experimentos, consiste en utilizar un grafo con más de una raiz, o con una raiz por cada cluster en los datos.

Dentro de los espacios discretos se observa el buen comportamiento que tienen los algoritmos de inserción aleatoria (RFN y RBNFN). Esta buena característica obedece a que los nodos no están muy ramificados, y para el caso de aquella estructura en donde se construye el árbol RBNFN, que es el que tiene mejor respuesta, obedece a que tiene una mejor distribución de los diferentes ramas del árbol, por lo que se evitan concentraciones innecesarias de ciertos puntos y se obtienen árboles de menor profundidad.

Queremos agradecer los acertados comentarios de los árbitros, que ayudaron a mejorar la calidad de la presentación.

Referencias

1. F. Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3), 1991.
2. S. Brin. Near neighbor search in large metric spaces. In *Proc. 21st Conference on Very Large Databases (VLDB'95)*, pages 574–584, 1995.
3. E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 2001. to appear.
4. E. Chávez, G. Navarro, and J. Marroquín. Overcoming the curse of dimensionality. In *European Workshop on Content-Based Multimedia Indexing (CBMI'99)*, pages 57–64, 1999. `ftp://garota.fismat.umich.mx/pub/-users/elchavez/fqa.ps.gz`.
5. M. García. E. Chávez, K. Figueroa. Una estructura dinámica para búsqueda de similaridad. Manuscript, 1999.
6. G. Navarro. Searching in metric spaces by spatial approximation. In *Proc. String Processing and Information Retrieval (SPIRE'99)*, pages 141–148. IEEE CS Press, 1999.