Algoritmos y Estructuras de Datos para Búsqueda de Objetos Similares

Virna Cuquejo

Centro Nacional de Computación
Universidad Nacional de Asunción, Paraguay
vcuquejo@cnc.una.py

Ricardo Baeza-Yates
Gonzalo Navarro
Depto. de Ciencias de la Computación
Universidad de Chile, Chile
{rbaeza,gnavarro}@dcc.uchile.cl

13 de septiembre de 2000

Resumen

Varias aplicaciones en computación buscan objetos en una base de datos que son similares a una consulta dada, desde el reconocimiento de patrones hasta la recuperación de objetos multimediales.

En este trabajo estudiamos algoritmos existentes que trabajan con criterios de similaridad en el espacio métrico para resolver este problema conocido como la "búsqueda del vecino más cercano". A la vez, introducimos un algoritmo de búsqueda "Híbrido" para este tipo de problema, basado en la idea de recorrer primero los caminos más prometedores, cálculo dependiente de la estructura de datos a la que es aplicada. Además, desarrollamos una base teórica necesaria y presentamos los resultados experimentales obtenidos.

Palabras clave: Búsqueda Similar, Espacio Métrico, Vecino más cercano, Consulta aproximada, Data mining.

1 Introducción

Hoy en día surgen aplicaciones en las que se desea buscar objetos similares en grandes bases de datos, como por ejemplo, encontrar una imagen similar a una dada.

El problema de buscar el vecino más cercano en una base de datos se puede ver aplicado en áreas como:

- Genética: encontrar secuencias de ADN o proteínas similares en alguna base de datos genética.
- Reconocimiento de voz: encontrar patrones vocales similares desde una base de datos de patrones vocales (Ej: bajo la transformada de Fourier).
- Reconocimiento de imágenes: encontrar imágenes similares (usando una métrica sobre una imagen dada desde una gran biblioteca de imágenes).
- Compresión de video: encontrar bloques en una imagen previa que son similares a bloques en una nueva imagen (usando métricas L1 o L2, posiblemente después de una transformada DCT).
- Minería de datos: encontrar series de tiempo aproximados o patrones no conocidos de una base de datos (ej. histórico de inventario).
- Recuperación de información: encontrar documentos relacionados a uno dado en una biblioteca digital.
- Detección de copia: encontrar sentencias similares a una consulta de un usuario en una gran base de datos de documentos (espacio métrico de strings).

- Base de Datos Médica: donde se almacenan imágenes $2d$ (Ej: rayos-X) e imágenes $3d$ (Ej: la tomografía computarizada). La identificación de casos anteriores con similares síntomas es muy útil en diagnósticos, enseñanza médica e investigación.

Para resolver esta búsqueda se indexan los datos con modelos de semejanza que definen un espacio vectorial y un espacio métrico.

En el espacio vectorial se supone que cada objeto puede ser representado como un punto d-dimensional. Se construye una estructura de datos que particiona el espacio en celdas que guardan los puntos de la base de datos. La búsqueda se inicia buscando la celda que contiene el nuevo punto en la estructura de datos. Luego, se realiza una búsqueda exhaustiva en esta celda para encontrar el punto más cercano (si es necesario se busca también en las celdas adyacentes). Sobresalen la estructura Kd-tree [Ben75] que propuso ideas que han sido utilizadas en varios métodos posteriores, y el R-tree [Gut84] que ha sido implementado en sistemas académicos y comerciales como POSTGRESS, ILLUSTRA e INGRES II. Desafortunadamente, cuando la dimensión del objeto crece, ambas estructuras requieren más espacio y las consultas se hacen más lentas.

En el espacio métrico, las estructuras son construidas con sólo la función de distancia que define el criterio de similaridad, ya que en algunos casos resulta difícil mapear cada objeto en un punto d-dimensional mientras se mantiene la precisión de la representación de semejanza o distancia entre objetos usando alguna métrica en este espacio. Destacan el BKT [BK73] y árboles métricos [Uhl91], donde el espacio es dividido jerárquicamente. En el nodo inicial se elige uno o varios puntos de datos. Luego se calculan las distancias entre el(los) punto(s) seleccionado(s) y el resto. Basándose en estas distancias, los puntos son separados en dos o más subárboles diferentes, siendo la estructura construída recursivamente. Pueden citarse además métodos como GNAT [Bri95], VPT [Yia93], FQT [BCM+94] y SAT [Nav99].

Algoritmos como AESA [Vid86] no suponen que los datos están estructurados y sólo hacen uso de las propiedades de una distancia dada.

Este trabajo presenta en la próxima sección un repaso de los conceptos teóricos. El algoritmo híbrido propuesto es introducido en la sección 3, mientras los resultados experimentales son presentados en la sección 4. Finalmente se presentan las conclusiones en la sección 5.

2 Conceptos Teóricos

2.1 Espacio Métrico

Es un espacio donde se tiene un conjunto $\mathbb{X}$ con una función de distancia d, $d : \mathbb{X} \times \mathbb{X} \to I\!R$ tal que $\forall\, x, y, z \in \mathbb{X}$:

$$\begin{aligned} d(x,y) &\geq 0 && \text{positividad} \\ d(x,y) = 0 \Leftrightarrow x &= y && \text{positividad estricta} \\ d(x,y) &= d(y,x) && \text{simetría} \\ d(x,y) + d(y,z) &\geq d(x,z) && \text{desigualdad triangular} \end{aligned}$$

Las tres primeras propiedades de similaridad permiten una definición consistente de la función de distancia. La desigualdad triangular permite descartar puntos imposibles en consultas aproximadas.

2.2 Optimización multiobjetivo

El presente trabajo pretende ser un material de referencia para decidir qué algoritmos son competitivos para una aplicación dada, los mismos deben minimizar el uso de recursos computacionales lo que nos conduce a los problemas de optimización.

Queremos minimizar básicamente dos variables estudiadas: el número de cálculos de distancia realizados y el tiempo de CPU consumido en la ejecución de los algoritmos, éstas obtenidas del proceso de construcción de la estructura de datos y de la consulta; es decir, tenemos cuatro objetivos a minimizar por lo que estamos ante un Problema de Optimización Multiobjetivo (POM).

Los problemas multiobjetivos se caracterizan por no tener una solución óptima única. La interacción entre los diferentes objetivos a cumplir da lugar a todo un conjunto de soluciones de compromiso que se conocen como las soluciones Pareto-óptimas. Esto nos lleva a definir algunos conceptos de Pareto [Aha80].

Definición 1 (Dominancia de Pareto) *Una solución x se dice que domina a y si y sólo si:*

1. *$f_j(x)$ no $\prec^1$ $f_j(y)$, y*
2. *$f_j(x) \succ f_j(y)$ para al menos un valor de j.*

Definición 2 (Conjunto Pareto-óptimo) *Un conjunto Pareto-óptimo es un conjunto de soluciones no dominadas que supone que la muestra incluye a todo el espacio de búsqueda sino es especificada otra cosa.*

Las soluciones paretos-óptimas son llamadas "no inferior", "admisible", o soluciones eficientes y sus vectores correspondientes "no dominados".

2.3 Selección de pivotes utilizados

Cuando se tiene un conjunto de datos se debe elegir un elemento a partir del cual ordenar al resto, este elemento seleccionado es llamado pivote o clave. Definimos el problema de la siguiente forma:

Sea S el espacio representado por el conjunto de datos estudiado y $X \subset S$. Se desea hallar k pivotes x_i para dividir el espacio X, para todo $i = 1, \ldots, k$ donde k es la cantidad de pivotes a seleccionar.

A continuación se presenta cada uno de los métodos aplicados en este trabajo:

1. Datos aleatorios escogidos de la base de datos (*Random*). Se elige x_i en forma aleatoria.
2. El último dato (*Last*). El último elemento de X es seleccionado como x_i, que lo hace dependiente del orden de los datos.
3. Punto de fuga (*Vp*). Se determina x_i tal que, x_i sea el de mayor dispersión entre los elementos de $X', X' \subset X$. Se utiliza X' debido a que el costo de este método de selección del pivote es orden $O(n^2)$. Para determinar el tamaño de X' se debe tener en cuenta que $|X'| \leq c \leq |X|$. En los experimentos realizados se considera un valor $c = 100$. El conjunto X' se forma seleccionando algún $x \in X$ aleatoriamente. La fundamentación teórica del método está detallada en [Yia93].
4. Puntos más alejados (*Farthest*). Este método es el sugerido en el trabajo que expone el GNAT [Bri95] que trabaja con k pivotes. La idea es que cada pivote seleccionado sea el elemento más alejado de todos los pivotes seleccionados anteriormente a él. Para ello se elige x_1 en forma aleatoria y el resto de los pivotes es seleccionado calculando la distancia para todo $x \in X$ con respecto a todo pivote ya seleccionado y el que tenga la mayor distancia será el siguiente pivote.

3 Algoritmo híbrido

Durante el estudio del estado del arte se encontró que el Metric Tree [Uhl91] y el Vp-tree[Yia93] particionan el espacio de igual forma, construyendo de esta manera estructuras idénticas, por lo que se decidió omitir la implementación del *Metric Trees*. Cabe señalar que ambos trabajos sugieren distintos algoritmos de búsqueda del vecino más cercano. Además, se pudo observar la misma idea de búsqueda presentada en [Uhl91] en los trabajos [HS97] para R-tree y en SAT [Nav99]. En [HS97] se probó que es más eficiente que el algoritmo original del R-Tree.

Como parte del presente trabajo se propone un algoritmo híbrido de búsqueda entre la idea básica presentada en [Uhl91] y las reglas de descarte particulares de cada estructura. A esta propuesta se la llamará "Algoritmo Híbrido". En la figura 1 se presenta el seudocódigo de dicho algoritmo.

La idea de este algoritmo consiste en visitar un nodo del árbol y almacenar todos sus subárboles en una cola de prioridad de modo que los más prometedores a contener el dato buscado queden hacia el inicio del mismo y, por lo tanto, sean los primeros en ser visitados en la próxima iteración del algoritmo.

La principal regla de descarte deriva de la desigualdad triangular y nos permite definir un punto de parada antes de recorrer todo el espacio de búsqueda. Además, cuando se determina el costo del subárbol se puede aplicar otra regla de descarte antes de insertar nuevos elementos a la cola de prioridad, evitando

[1] $\prec$ denota "es peor" y $\succ$ denota "es mejor"

```
NearNeigh( node Tree, object query ) {

     bestDistance = Tree.distance;
     bestObject   = Tree.object;

     priorityQueue.insert( bestObject );

     While( priorityQueue is not empty) {

         /* condicion del final del programa*/
         if ( priorityQueue.maximo() >= bestDistance )
            return bestObject;
         else {
           element = priorityQueue.nextElement();

           distance= distance(query,element);
           if ( distance < bestDistance)
              bestDistance = distance
              bestObject   = element;
           }

          /* Calcular el subarbol mas prometedor de cada nodo, debe ser
             adaptado segun la estructura del nodo y la aridad del arbol  */

          for( i=0 until arity-1) {
            calcular el costo de cada subrama
            /* esto es util para grandes aridades */
            if ( no se cumple regla de descarte )
                priorityQueue.insert(subrama, costo);
          }
     }
}
```

Figura 1: Seudocódigo del Algoritmo Híbrido de búsqueda del vecino más próximo

guardar ramas muy alejadas de la deseada. Esto último constituye un ahorro en tiempo de CPU sobre todo cuando se busca en estructuras de aridad mayor a dos.

Una diferencia esencial de este algoritmo con los presentados con cada estructura de datos estudiada es que seleccionan en cada nivel del árbol una subrama candidata y la recorren hasta llegar a una hoja, si no se halla el elemento deseado se selecciona otra subrama y se procede de la misma forma hasta encontrarlo, técnica similar a la "Búsqueda en profundidad"; sin embargo, este algoritmo utiliza una técnica equivalente a la "Búsqueda por niveles" sólo que en cada nivel recorre únicamente los hijos de las subramas más prometedoras. Además, los algoritmos existentes son dependientes de cada estructura de datos para la que fueron diseñados; sin embargo, el "Híbrido" tiene un algoritmo básico con adaptaciones necesarias para cada estructura de datos.

La implementación de este algoritmo en la estructura MVPT (*multiway VPT*) no resultó satisfactoria, alivianando las reglas de descarte se tiene un tiempo muy superior al algoritmo implementado originalmente; en caso contrario, no siempre se obtiene el dato más cercano sino el segundo. Este resultado erróneo se obtuvo en un 1% del tamaño de la muestra del experimento.

El anterior resultado determinó que el algoritmo Híbrido no sea implementado en las estructuras que trabajan con funciones discretas de distancias como el BKT, FQT y el FHQT. Esas estructuras, como el MVPT, poseen información adicional sobre la partición del espacio realizada por sus subárboles, lo que les permite incorporar en sus algoritmos de búsqueda una heurística para recorrer los subárboles más prometedores, por lo tanto, este nuevo método no logra optimizar la búsqueda en estos casos.

4 Resultados Experimentales y Análisis

En esta sección realizamos un estudio comparativo de los algoritmos existentes, las pruebas son direccionadas para medir primeramente el costo de preprocesamiento de estos algoritmos en tres aspectos: tiempo de CPU requerido para la creación de las estructuras de datos medido en milisegundos, los cálculos de distancias realizados durante el proceso y la cantidad de espacio requerido medido en bytes. Además, el rendimiento de los algoritmos de búsqueda del vecino más cercano en cuanto al tiempo de CPU requerido y la cantidad de cálculos de distancias realizados.

La cantidad de experimentos realizados varía para cada algoritmo de modo a proporcionar un 95% de confiabilidad de que el valor muestral obtenido difiere como máximo en un 5% del valor real.

Los algoritmos fueron implementados en el lenguaje Java y fueron ejecutados en computadores con procedores k6-2 de 350 MHz y 128 MB de RAM, con el sistema operativo Linux Red Hat 6.0 y se utilizó el *Java Runtime Environment* de IBM.

Se trabajó con un diccionario de 86.063 palabras de tamaño variable en español, utilizando la función de distancia de Edición. Se realizó el experimento para pivotes de palabras del diccionario con alteraciones de 1, 2 y 3 caracteres que representan el número de errores permitidos en nuestras pruebas y fueron promediados para algunos análisis. Esto es aplicable por ejemplo para realizar búsquedas en la *web*, donde es deseable encontrar la palabra correcta aún cuando el usuario introduzca la consulta con error o el documento buscado contenga errores, debido tal vez a ser el resultado de un proceso de captura digital a través de un *scanner*.

Sea p_0 una palabra del diccionario y p_k, la misma palabra p_0 con k errores (inserción, eliminación o cambio de k caracteres), para $k = 1, 2, 3$. Se desea encontrar p_0 a partir de p_k. Se pretende buscar la palabra más cercana a p_k que no necesariamente será p_0 ya que en la alteración puede asemejarse más a otra palabra del diccionario.

En la siguiente tabla se puede observar los algoritmos estudiados junto con el número de referencia asociado que aparecen en los gráficos con sus variantes de acuerdo al parámetro estudiado.

Referencia	Algoritmo	Aridad(a)/Altura(h)	Sel Pivote	Algoritmo
1	BKT [BK73]			
2	FQT [BCM+94]	h15, h20		
3	FHQT [BCM+94]			
4	SAT [Nav99]			
5	GNAT [Bri95]	a2, a50, a100, a200, a400	(f)arthest, (r)andom	1-original, 2-híbrido
6	VPT [Yia93]		(m) random, (v)p	1-original, 2-híbrido
7	MVPT	a2, a4, a6	(l)ast, (r)andom, (v)p	
8	AESA [Vid86]			

Se desarrolló una simulación del algoritmo AESA a sólo efecto de tener una cota mínima de cálculos de distancia, debido al alto costo de CPU del mismo. La estructura MVPT es una variación del VPT que soporta mayor aridad y sigue utilizando un solo pivote para particionar el espacio.

4.1 Resultados experimentales del GNAT

ARIDAD. En la figura 2 se puede notar que cuando trabajamos con el criterio de selección del pivote del punto más alejado obtenemos buenos resultados con las aridades bajas de 2 y 50; sin embargo, cuando trabajamos con pivotes aleatorios el proceso de búsqueda se beneficia con una mayor aridad, experimentalmente hallamos el límite de 200 para ambos algoritmos estudiados.

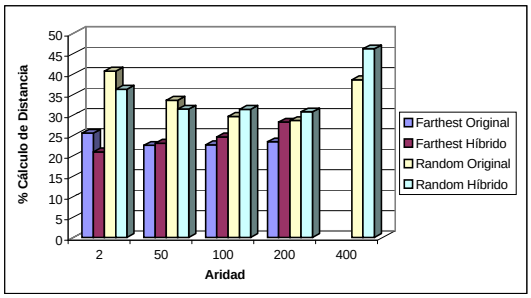
PIVOTE. Esta estructura tiene un mejor comportamiento con el método de selección del pivote del punto más alejado. Se debe tener en cuenta que el número de cálculos de distancias y el tiempo de CPU están muy por encima del método aleatorio pero se mantiene en un porcentaje aceptable de costo de preprocesamiento (figura 8).

ALGORITMO. El algoritmo de búsqueda "Híbrido" tiene mejor comportamiento que el algoritmo original con aridades bajas: 2 y 50. En aridades superiores el algoritmo original tiende a un mejor rendimiento, esto es debido que a mayor aridad desecha mayor cantidad de elementos por cada nivel visitado, el híbrido siempre debe calcular los costos de todos los hijos del subárbol más prometedor.

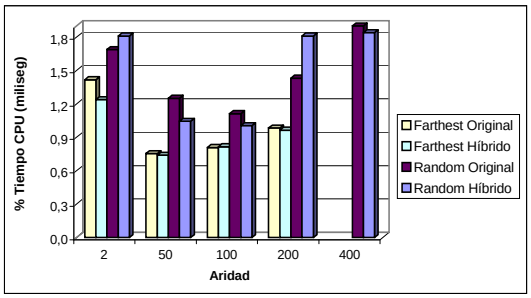
En la figura 2 se puede notar que el algoritmo "Híbrido" implementado en la estructura Gnat con aridad 2 logra un ahorro de aproximadamente 5% en el número de cálculos de distancias con respecto al algoritmo original. Este porcentaje disminuye con el incremento de la aridad en el GNAT, decayendo su rendimiento a partir de la aridad 100 pero preservando una menor desviación.

4.2 Resultados experimentales del VPT

PIVOTE. El criterio de selección de pivote a través del punto de fuga garantiza el mejor comportamiento de la estructura, aunque el costo de hallarlo es de $O(n^2)$ se puede observar que el trabajar con un subconjunto



(a) Número de cálculos de distancias



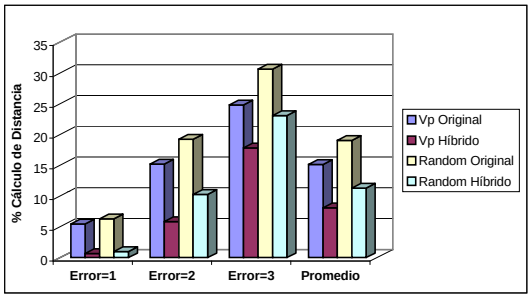
(b) Tiempo de CPU (milisegundo)

Figura 2: Resultados del GNAT según el método de selección del pivote y algoritmo de búsqueda estudiado

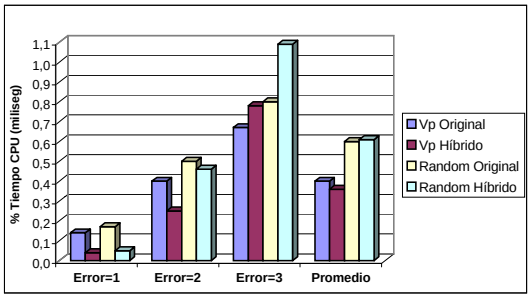
de tamaño constante se comporta muy bien y el orden pasa a $O(c^2)$ donde c es una constante, por lo tanto es de orden O(1).

Algoritmo. De la figura 3 se puede notar que el mejor comportamiento lo tiene el algoritmo "Híbrido" en cuanto a cálculos de distancias, en cuanto al tiempo de CPU se ve influenciado por el error de la consulta. Con un error de tres, si bien evalúa menos nodos, crece el sobrecosto inherente a la cola de prioridad.

Se puede decir que el mejor rendimiento ofrece el VPT construído seleccionando el pivote con el método del punto de fuga y aplicándole a dicha estructura el algoritmo de búsqueda "Híbrido". Además, utiliza una cantidad mínima de memoria para almacenar la estructura a un alto costo de preprocesamiento, requiriendo aproximadamente un 1008% más de cálculos de distancia que el método aleatorio de selección de pivotes.



(a) Número de cálculos de distancias



(b) Tiempo de CPU (milisegundo)

Figura 3: Resultados del VPT según el método de selección del pivote y algoritmo de búsqueda estudiado

4.3 Resultados experimentales del MVPT

Aridad. En la figura 4 se puede notar que la estructura con aridad 2 tiene el mejor comportamiento con excepción de la que se encuentra construída con el método aleatorio de selección de pivote que tiene su óptimo con aridad 4.

Pivote. El mejor comportamiento tiene la estructura construída con la selección del pivote del punto de fuga (vp).

Algoritmo. El mejor comportamiento se tuvo con una aridad de 2 y la selección del pivote del punto de fuga. Además, se tiene un rendimiento aceptable con una selección aleatoria y una aridad de 4, con la ventaja de un menor costo de preprocesamiento y a expensas de un pequeño incremento en el tamaño de la memoria ocupada por la estructura.

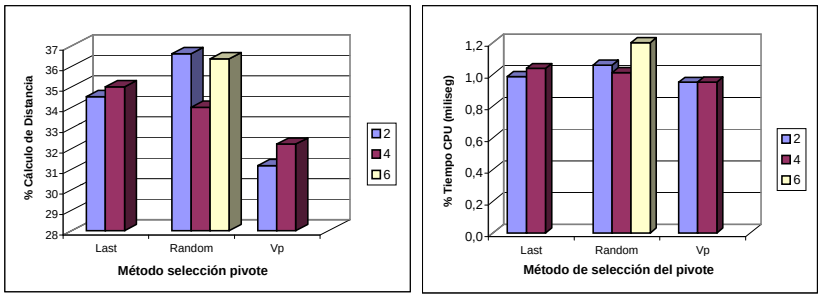


Figura 4: Resultados del MVPT por aridad estudiada y método de selección del pivote

4.4 Análisis estadísticos de conjunto

Los resultados empíricos obtenidos se pueden visualizar en la figura 5 de donde podemos decir que el algoritmo de mejor comportamiento es el algoritmo Híbrido implementado en el VPT con la selección del punto de fuga como pivote, además es el algoritmo Pareto-óptimo si consideramos sólo el tiempo de respuesta de las consultas. Esta selección requerirá una cantidad de cálculos de distancia que está entre el $6,27\%$ y el $9,73\%$ de la cantidad de elementos de la base de datos utilizada.

Se puede notar además, que la estructura VPT con todas las variantes implementadas está en los primeros puestos de este *ranking* de cálculos de distancias y además en la figura 6 se puede notar que tiene un rendimiento equivalente en cuanto al tiempo de ejecución. Este mismo comportamiento se mantiene en cada error permitido en la consulta, esto lo podemos observar en la figura 7.

Sin embargo, la estructura FHQT de altura 20 que se encuentra entre los algoritmos con menos del 20% de cálculos de distancias pasa a ocupar el último puesto en relación del tiempo de CPU, esto es debido al sobrecosto computacional de recorrer la altura del árbol.

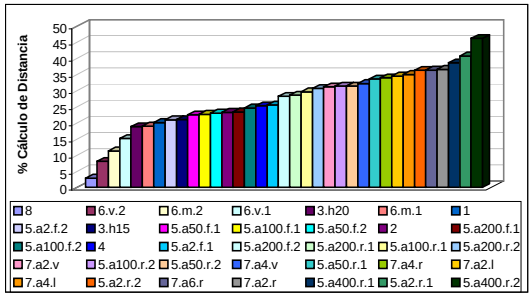


Figura 5: Porcentaje del promedio de cálculos de distancia ordenado ascendentemente

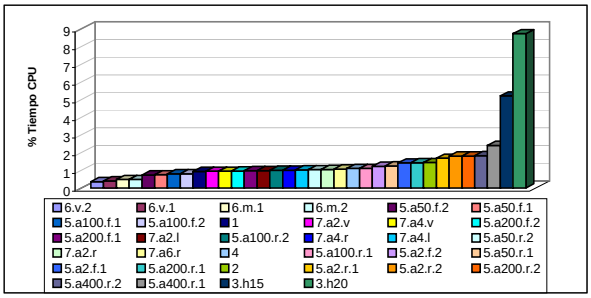


Figura 6: Porcentaje del promedio de tiempo de CPU ordenado ascendentemente

La figura 7 muestra que en forma mayoritaria cuando la palabra que se consulta difiere en 2 caracteres del más próximo de uno existente en la base de datos, es decir, el error de la consulta aumenta de 1 a 2, se puede observar que los cálculos de distancia aumentan en la mayoría de los casos al doble o más. Esto está de acuerdo a la intuición de que al aumentar el error, la cantidad de palabras que satisface el criterio de búsqueda crece en forma exponencial; es decir, si con error 1 encontraba n palabras, al aumentar el error a 2 por cada una de estas palabras se debería encontrar otras n, o sea n^2 palabras. Las excepciones son el 5.a100.f.1 y el 5.a200.f.1 ya que a mayor aridad descartan más cantidad de elementos en cada nivel, esto no se cumple para el algoritmo híbrido ya que al trabajar por niveles la aridad le afecta en forma negativa ya que constituyen más nodos a evaluar. Además, gracias a la selección de pivotes a tráves del método de los puntos más alejados, logra una mejor división del espacio por lo que cada celda del GNAT contiene los puntos que se encuentran espacialmente más cerca unos de otros.

Al aumentar el error permitido a 3, sin embargo, se puede notar que el incremento es marginal en la cantidad de cálculos de distancia, esto se debe esencialmente a que estos algoritmos agrupan los datos más cercanos espacialmente y gran porcentaje de las celdas recorridas con error 3 pertenecen al recorrido con error 2 también. En los algoritmos 3.h15, 3.h20, 6.m.2 y 6.v.2 no se duplica el número de palabras recorridas con respecto al error 2, pero sí se mantiene el nivel de crecimiento con respecto al error 1.

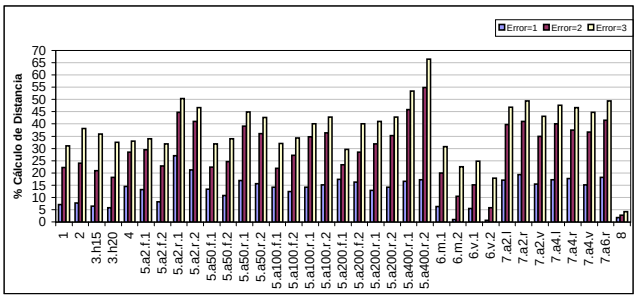


Figura 7: Porcentaje de cálculos de distancias según el error permitido

Costo de preproceso. Analizando los requerimientos de memoria RAM de las diferentes estructuras, en la figura 9 se puede notar que las estructuras que consumen menos memoria RAM son el GNAT y el VPT, entre 281% y 472% del tamaño del diccionario. El FQT y el FHQT constituyen las estructuras que requieren de mayor cantidad de memoria para su construcción.

En la figura 8 la estructura VPT con los métodos de selección de pivote del *vp* y *random*, 6.v y 6.m respectivamente, tienen un costo de CPU casi idéntico a pesar de que el primero realiza mayor cantidad de cálculos de distancias. Esto se debe al mayor costo computacional dado por el cálculo de la media.

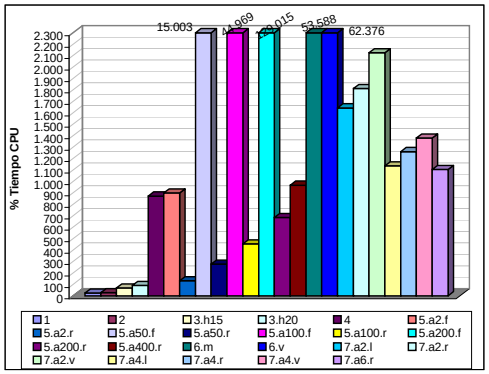
Con respecto al MVPT y el método de selección de pivote *vp* observamos que con aridad 2 realiza mayor número de cálculos de distancias que con aridad 4 porque en cada nivel del árbol se procesa menor cantidad de datos, la reducción drástica del tiempo de CPU deriva del hecho que aqui no se trabaja con la media.

4.5 Algoritmos que pertenecen al conjunto Pareto-óptimo

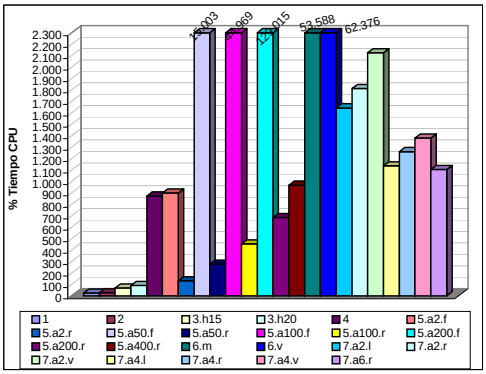
En la tabla 9 podemos visualizar los algoritmos estudiados con mejor relación de compromiso entre los cuatro objetivos propuestos de: minimizar tiempo de CPU y número de cálculos de distancias tanto para la consulta del vecino más cercano como para el tiempo de preprocesamiento o creación de las estructuras.

Si consideramos como factor de decisión el comportamiento del algoritmo en las consultas tenemos un único algoritmo componente del conjunto Pareto-óptimo que es el VPT con la selección del pivote a través del "Punto de Fuga" y el algoritmo de búsqueda "Híbrido" propuesto.

Cada uno de estos algoritmos son buenos dependiendo del compromiso entre recursos o variables que necesitamos optimizar entre el preprocesamiento y la consulta. Si el tiempo de respuesta en las consultas es lo primordial, el algoritmo 6.v.2 claramente sería lo indicado, el que ofrece un rendimiento más próximo al 6.v.2 es el 6.m.2 con un 39% más de cálculos de distancias pero con el ahorro de un orden de magnitud en el preproceso en la misma variable, similarmente para el algoritmo 6.m.1 que si bien tiene un 134% más



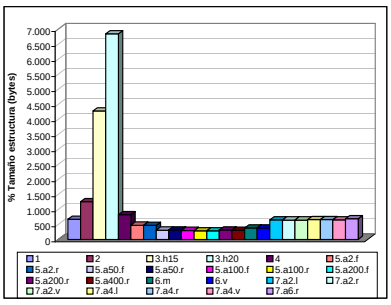
(a) Número de cálculos de distancias



(b) Tiempo de CPU (milisegundo)

Figura 8: Resultados del preproceso expresados en porcentajes según el número de elementos del diccionario

de cálculos de distancias que el anterior tiene un costo de CPU menor debido al bajo costo computacional de la función de distancia utilizada. Si además se necesita una relación de compromiso con el tiempo de preproceso se tiene los algoritmos 1 y 3.h20 pero si el tamaño de la memoria principal es otra limitación desecharíamos este último. En aplicaciones donde el costo del cálculo de la función de distancia es barata se puede querer minimizar esencialmente los tiempos de CPU en ambos procesos (preproceso y consulta), para este caso puede considerarse el uso de los algoritmos 5.a50.f.1 y 5.a50.f.2.



(a) Porcentaje del tamaño de las estructuras estudiadas según el tamaño del diccionario

Algoritmos	Consulta		Preproceso	
	% Número Distancia	% Tiempo de CPU	% Número Distancia	% Tiempo de CPU
1	20,14	0,94	712,16	24,57
3.h20	18,84	8,74	2000,00	89,56
5.a50.f.1	22,52	0,75	483434,22	15002,89
5.a50.f.2	23,07	0,74	483434,22	15002,89
6.m.1	19,01	0,48	3271,07	53587,88
6.m.2	11,32	0,49	3271,07	53587,88
6.v.2 (*)	8,12	0,36	32982,70	62375,64

(*) Algoritmo Pareto-óptimo teniendo en cuenta sólo variables de consulta

(b) Mejores algoritmos de compromiso

Figura 9: Tamaño de las estructuras y el conjunto Pareto-óptimo obtenido

5 Conclusión

El algoritmo de búsqueda propuesto "Híbrido" resultó ser mejor en las estructuras donde no se dispone de criterios para la selección del pivote. En general tiene mejor comportamiento con las aridades mínimas, esto se debe a que por cada subárbol más prometedor debo evaluar el costo de sus hijos insertándolos en la cola de prioridad, a mayor aridad mayor número de ramas a evaluar en cada nivel.

El valor óptimo del método de selección aleatoria del pivote tiende a mayor aridad, con otro método tiende hacia las mínimas aridades, así tenemos mejor comportamiento en los siguientes casos:

- Gnat con aridad 2 para el método del punto más alejado de selección del pivote y el algoritmo Híbrido.

- Gnat con aridad 200 con el método aleatorio de selección de pivotes con el algoritmo original.
- VPT con el método del punto de fuga y el algoritmo Híbrido.
- MVPT con aridad 2 y el método del punto de fuga.
- MVPT con aridad 4 y el método de selección aleatoria.

Los métodos de selección de pivote que eligen un objeto con mayor dispersión o el más alejado, en caso de trabajar con k pivotes, son los que mejor comportamiento presentaron pero a mayor costo de preprocesamiento.

Como trabajo futuro, consideramos realizar pruebas para otros tipos de datos como palabras de longitud fija y puntos d-dimensionales a fin de estudiar además la influencia de la dimensionalidad de los datos en los algoritmos estudiados.

Referencias

[Aha80] Ben-Tal, Aharon. "Characterization of Pareto and Lexicographic Optimal Solutions." *Multiple Criteria Decision Making Theory and Application 177.* Lecture Notes in Economics and Mathematical Systems, edited by G. Fandel and T. Gal, 1-11, Berlin:Springer-Verlag, 1980.

[Ben75] J.L. Bentley. *Multidimensional Binary Search Trees Used for Associative Searching.* Communications of the ACM, 18(9) : 509-517, setiembre 1975.

[BCM+94] R. Baeza-Yates, W. Cunto, U.Mamber y S. Wu. *Proximity Matching Using Fixed-Queries Trees.* 5th Symp. Combinatorial Pattern Matching, Springer Verlag LNCS 807, pág. 198-212, junio 1994.

[BK73] W. A. Burkhard y R. M. Keller. *Some approaches to best-match file searching.* Communications of the ACM, vol. 16, Abril 1973.

[Bri95] S.Brin. *Near Neighbor Search in Large Metric Spaces*, Proceedings of the 21st VLDB Conference, pág 574-584, 1995.

[Gut84] A. Guttman. *R-trees: A Dynamic Index Structure for Spatial Searching.* In ACM SIGMOD, pág :47-57, Junio 1984.

[HS97] Gísli Hjaltason y Hanan Samet. *Distance Browsing in Spatial Databases Using R-trees*.*, Febrero 1997.

[Nav99] G. Navarro. *Searching in Metric Spaces by Spatial Approximation.*Proc. String Processing and Information Retrieval (SPIRE'99), Cancún, Méjico, Setiembre 1999.

[Uhl91] J. Uhlmann. *Satisfying general proximity/similary queries with metric trees. Information Processing Letters.* 40(4) :175-9. November 1991.

[Yia93] P.N. Yianilos. *Data Structures and algorithms for nearest neighbor search in general metric spaces.* In ACM-SIAM Symposium on Discrete Algorithms, pág 311-321.1993.

[Vid86] E. Vidal. *An algorithm for finding nearest neighbours in (approximately) constant average time.* Patt. Recogn. Lett. 4(3), julio 1986.