Optimización de las búsquedas *kNN* en espacios métricos*

Luis G. Ares, Nieves R. Brisaboa
Alberto Ordóñez, Óscar Pedreira
Laboratorio de Bases de Datos
Universidade da Coruña
15010 A Coruña, España
{lgares,brisaboa,alberto.ordonez,opedreira}@udc.es

Benjamín Bustos
Departamento de Ciencias de la Computación
Universidad Nacional de Chile
Santiago 6511224, Chile
bebustos@dcc.uchile.cl

Resumen

La búsqueda por similitud es una de las operaciones más frecuentes en problemas que involucran el tratamiento de datos que no poseen una estructura completamente determinada. En esencia consiste en obtener los objetos más similares a uno dado. Los espacios métricos resultan adecuados para formalizar el problema, al trabajar con una función de distancia que permite determinar de manera rigurosa el grado de similitud entre los objetos. Una variante consiste en realizar la búsqueda de los k vecinos más cercanos o búsqueda *kNN (k-Nearest Neighbor)*. En este trabajo proponemos un nuevo algoritmo para realizar búsquedas *kNN* en espacios métricos de manera eficiente y aplicable a problemas reales, basándonos en minimizar el impacto de los factores que afectan negativamente al rendimiento de la operación de búsqueda. Las prestaciones de nuestro algoritmo quedan corroboradas por los resultados de la evaluación experimental, realizada sobre varias colecciones de datos reales.

1. Introducción

Los sistemas de información han manipulado casi exclusivamente a lo largo del tiempo datos de tipo numérico o alfanumérico, organizados en general en forma de tablas. El panorama en los últimos años ha cambiado significativamente gracias, en parte, a la proliferación de los servicios web que conllevan la necesidad de afrontar el tratamiento de contenidos multimedia y el de enormes volúmenes de texto. Encontramos ejemplos integrados en la vida cotidiana, como el desarrollo de los medios de información en formato digital o el crecimiento de las redes sociales, y otros más específicos como los avances en biocomputación, entre los que destaca el estudio de las secuencias de ADN, en los que surgen nuevos tipos de datos, habitualmente de naturaleza semiestructurada o incluso no estructurada [9].

En este contexto, el modelo clásico de búsqueda exacta no resulta adecuado, siendo necesario el desarrollo de nuevos planteamientos. Uno de los más importantes es la búsqueda por similitud, que consiste en recuperar de entre una colección de objetos, aquellos que son similares a uno determinado inicialmente. Por ejemplo, en el tratamiento de imágenes puede interesarnos obtener un número concreto de ellas que sean las más similares a otra dada; en una base de datos de huellas digitales, una aplicación común consiste en obtener las que resulten más parecidas a una determinada; en las redes sociales podemos utilizar la búsqueda por similitud para encontrar los perfiles de los usuarios más parecidos a otro dado para sugerir relaciones basadas en intereses comunes.

La manera trivial de resolver el problema de las búsquedas por similitud es realizar la comparación de cada elemento de la colección con el objeto en cuestión. Los elevados costes

*Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia e Innovación (PGE y FEDER) ref. TIN2009-14560-C03-02.

asociados a este planteamiento, han originado la aparición de diversas soluciones eficientes, muchas de ellas adaptadas al dominio de aplicación, careciendo por tanto de la generalidad necesaria para poder aplicarse a entornos que presentan características diferentes a las del inicialmente previsto.

Los espacios métricos proporcionan un marco de trabajo independiente del dominio de aplicación para realizar búsquedas por similitud de forma eficiente [6].

Un espacio métrico es un par (X, d) donde X es un universo de objetos y $d : X \times X \rightarrow \mathbb{R}^+$ es una función de distancia que cuantifica lo diferentes que son dos objetos en ese universo. Además, d debe cumplir las siguientes propiedades:

- (Estrictamente positiva) $d(x, y) \geq 0$ y $d(x, y) = 0 \Longleftrightarrow x = y$
- (Simetría) $d(x, y) = d(y, x)$
- (Desigualdad triangular) $d(x, z) \leq d(x, y) + d(y, z)$

En general el cálculo de la función de distancia resulta muy costoso, por lo que la eficiencia tiene como objetivo reducir al máximo el número de comparaciones entre el objeto de consulta y los objetos de la colección, centrándose en evitar que el proceso involucre a todos los elementos de la colección. Por ello los métodos de búsqueda por similitud construyen índices para poder descartar el mayor número posible de objetos durante las búsquedas, sin necesidad de compararlos directamente con la consulta.

Un espacio vectorial k-dimensional $\mathbb{R}^k$ es un caso particular de espacio métrico en el que los objetos son vectores en $\mathbb{R}^k$. En ambos casos, el subconjunto finito $U \subset X$, $n = |U|$ se conoce con el nombre de base de datos y será sobre el que se realicen las operaciones de indexación y búsqueda.

Existen dos operaciones de búsqueda consideradas básicas:

- Búsqueda por rango: recupera los objetos de la base de datos que están a una distancia del objeto de consulta, inferior o igual a un determinado radio de búsqueda; $R(q, r) = \{x \in U / d(x, q) \leq r\}$.
- Búsqueda de los k vecinos más cercanos: recupera los k objetos más próximos a la consulta; $kNN(q) = \{A \subseteq U, |A| = k, \forall x \in A, y \in U - A, d(x, q) \leq d(y, q)\}$.

Se han propuesto numerosos métodos de indexación y búsqueda que posteriormente han sido clasificados en dos grandes grupos: métodos basados en *pivotes* y métodos basados en *clustering* [6].

Los *métodos basados en pivotes* se basan en elegir un conjunto de objetos de referencia de la colección a indexar, conocidos con el nombre de pivotes, y almacenar las distancias entre ellos y el resto de objetos de la colección. Esta información se utiliza durante el proceso de búsqueda para descartar objetos sin compararlos directamente con las consultas. Supongamos que $P = \{p_1, p_2, \ldots, p_k\}$, $p_i \in U$ es el conjunto de pivotes seleccionados. Dada una consulta por rango $R(q, r)$ y un objeto $x \in U$, aplicando la desigualdad triangular tenemos que $|d(p_i, x) - d(p_i, q)| \leq d(x, q)$ para cualquier $p_i \in P$. Con esto se tiene una cota inferior de la distancia a la consulta, lo que además conlleva que una condición para descartar x sin compararlo directamente con la consulta es que:

$$l_b = |d(p_i, x) - d(p_i, q)| > r$$

para algún pivote p_i.

Los métodos basados en pivotes pueden clasificarse en:

- Métodos que almacenan información jerárquica sobre la colección que indexan: *Burkhard-Keller Tree* (BKT) [4], *Multi Vantage Point Tree* (MVPT) [2], etc.
- Métodos sin información jerárquica que solo almacenan distancias entre objetos y pivotes: *Approximating and Eliminating Search Algorithm* (AESA) [10], *Linear-AESA* (LAESA) [15], *Sparse Spatial Selection* (SSS) [3], *Unique Pivot Index* (UPI) [1], etc.

Los *métodos basados en clustering* dividen el espacio a indexar en un conjunto de regiones disjuntas denominadas *clusters*. *Bisector Tree* (BST) [12], *Generalized Hyperplane Tree* (GHT) [14], *Spatial Approximation Tree* (SAT) [11], *List of Clusters* (LC) [5], etc. son ejemplos de métodos basados en *clustering*

Los métodos basados en pivotes consiguen generalmente un rendimiento mayor en las operaciones de búsqueda que los basados en *clustering* pero a costa de requerir mucho más espacio para almacenar la estructura del índice. Los métodos basados en pivotes sin información jerárquica acentúan aún más las diferencias: son los más eficientes pero los que más espacio ocupan.

Para poder estimar y comparar el rendimiento de diferentes métodos de búsqueda por similitud, en [6] se propone la siguiente expresión de coste:

$$T = \#d \times \textit{complejidad de } d() + \textit{tiempo I/O} + \textit{tiempo extra CPU} \quad (1)$$

donde $\#d$ representa el número de evaluaciones de la función de distancia necesarias para resolver una consulta, *tiempo I/O* es el tiempo necesario para pasar el índice de disco a memoria y *tiempo extra CPU* es el tiempo que se necesita para procesar la estructura del índice una vez que está en memoria.

Podría simplificarse la expresión anterior considerando despreciables tanto el *tiempo I/O* como el *tiempo extra CPU*. Esta aproximación sería válida siempre y cuando el tamaño de las colecciones a indexar fuera relativamente pequeño, con lo que el tamaño de los índices sería también reducido, por lo que el *tiempo I/O* y el *tiempo extra CPU* podrían considerarse irrelevantes.

La situación es muy diferente si se trabaja sobre colecciones de gran tamaño, esto es, que poseen millones o decenas de millones de objetos. En este caso el tamaño del índice puede llegar a ser realmente voluminoso y los factores que antes se desestimaban (*tiempo extra CPU* y *tiempo I/O*), ahora se convierten en componentes determinantes de la expresión de coste. Por este motivo en los dominios de aplicación en los que es necesario manejar bases de datos de gran tamaño, la elección del método de búsqueda por similitud debe de estar determinada por la totalidad de los factores de la expresión de coste y no solo por $\#d$.

En este trabajo se propone un nuevo algoritmo para la resolución de consultas kNN que trata de minimizar el tiempo total T. El objetivo es, a la vez que reducir el número de evaluaciones de la función de distancia, minimizar el *tiempo extra CPU* mediante el desarrollo de un algoritmo lineal cuyo rendimiento sea adecuado para colecciones de datos de gran tamaño.

El resto del artículo está organizado de la siguiente forma: en la sección 2 se describen trabajos previos relacionados con búsquedas kNN en métodos basados en pivotes; en la sección 3 se detalla la propuesta del nuevo algoritmo; en la sección 4 se presentan los resultados obtenidos en la evaluación experimental y por último en la sección 5 se exponen las conclusiones y las líneas de trabajo futuras.

2. Antecedentes y trabajo previo

Como adelantábamos en la sección anterior, los métodos de búsqueda en espacios métricos se clasifican en basados en pivotes y en basados en clusters. Dentro del grupo de los basados en pivotes hay dos posibles aproximaciones. En la primera los métodos usan los pivotes para establecer una jerarquía sobre la colección de objetos que indexan, con el fin de determinar el orden en el que se comparan las consultas con los objetos. En este caso el índice tiene una estructura arbórea. En la segunda aproximación los métodos no crean esa jerarquía sobre el espacio, almacenando en una matriz las distancias entre los objetos indexados y los pivotes.

2.1. Estructura del índice y búsquedas *kNN*

En los métodos que almacenan información *jerárquica* los índices suelen estar implementados sobre estructuras de datos en forma de árbol. Para resolver búsquedas kNN utilizando este tipo de índices se puede recorrer el ár-

bol siguiendo estrategias *depth-first* o *breadth-first* [7], y se compara el objeto de consulta con aquellos objetos que están más cerca de la propia consulta. De esta forma se descartan regiones completas del árbol sin necesidad de comparar los objetos que contienen con la consulta. Este caso se puede dar, por ejemplo, cuando sabemos que todos los objetos de una determinada zona del árbol van a estar más alejados de la consulta que el objeto más alejado ya seleccionado, conocido como el actual k vecino más cercano.

En los métodos basados en pivotes con índices de estructura *matricial* existen diferentes propuestas para realizar búsquedas kNN. En [6] se describe una forma de implementar dicha operación utilizando búsquedas por rango con radio de búsqueda decreciente. El conjunto kNN se forma inicialmente con los primeros k objetos de la base de datos y se almacena la distancia entre la consulta q y el objeto de kNN más alejado, esto es, el actual k vecino más cercano. Para el resto de objetos $\{u_i \in U - kNN\}$ se determina, basándose en la cota inferior, si u_i está más próximo a la consulta que el actual k vecino más cercano, y en caso afirmativo, este se sustituye por el objeto u_i y se actualiza la distancia a dicho elemento.

La eficiencia de esta alternativa depende de q y de que el radio de búsqueda converja rápidamente al valor real entre la consulta y el k vecino más cercano.

Algoritmos de búsqueda como $AESA$ [15] y $LAESA$ [10] siguen una aproximación muy similar a esta. Ambos dividen el espacio de búsqueda en tres subconjuntos: los objetos seleccionados o conjunto kNN, los objetos descartados por estar más alejados de la consulta que el actual k vecino más cercano y los objetos que aún no han podido ser descartados. En cada iteración calculan las cotas inferiores l_b para cada objeto, pudiendo darse tres casos:

1. El objeto pasa a formar parte del conjunto kNN.
2. El objeto puede descartarse, ya que su l_b es mayor que la distancia entre la consulta q y el actual k vecino más cercano.
3. El objeto no se puede descartar.

El algoritmo prosigue mientras existan elementos que no han podido ser descartados. Un problema en ambos casos es que, en cada iteración, solo se utiliza un pivote para hallar las cotas inferiores de las distancias entre los objetos y la consulta, lo que da como resultado, en general, cotas poco ajustadas al valor real, originando una pérdida de rendimiento debida a que la distancia entre la consulta y el actual k vecino más cercano convergerá más lentamente a su valor real y con ello se descarten menos objetos en cada iteración.

En [13] se propone un cambio estructural en el índice con el objetivo de acelerar el proceso de convergencia. El índice almacena los objetos de la base de datos ordenados por su distancia al primer pivote. La búsqueda comienza por el objeto cuya distancia al primer pivote $d(o_i, p_1)$ es la más parecida a la distancia entre la consulta y ese pivote $d(q, p_1)$. El objetivo es procesar primero los objetos que se considera que están más próximos a la consulta, basándose en las distancias aportadas por las cotas, y lograr así que el radio de búsqueda converja lo más rápidamente al valor real. Como existe una relación de orden entre los objetos y el pivote, también la habrá entre los objetos y la consulta q, por lo que si el algoritmo encuentra un objeto que está a una distancia mayor que el actual k vecino más cercano, la búsqueda puede finalizar porque el resto de elementos que quedan por procesar estarán a una distancia aún mayor.

2.2. Fundamentos de un nuevo algoritmo

El cambio en la estructura del índice propuesto en [13] hace que no se pueda aplicar a métodos como $AESA$, $LAESA$ y SSS, pero aporta una idea que puede aprovecharse para diseñar un nuevo algoritmo aplicable a cualquier método de búsqueda basado en pivotes con estructura *matricial*.

En lugar de ordenar los objetos con respecto a uno de los pivotes, lo que conlleva un cambio estructural en el índice, sería más adecuado ordenar los objetos en función de su distancia a la consulta. La distancia real no se conoce, pero se ordena en función de las cotas inferiores que los pivotes ofrecen para dichas distancias.

De esta forma se consigue que la distancia al actual k vecino más cercano converja rápidamente al valor real de dicha distancia, ya que se minimiza el número de evaluaciones de la función de distancia para resolver la consulta, a la vez que se modifica el criterio de finalización de la búsqueda, porque puede terminar en cuanto se encuentra un objeto cuya distancia estimada a la consulta sea mayor que la distancia al actual k vecino más cercano.

El problema de esta aproximación es la necesidad de ordenar el vector de cotas para cada consulta. En el caso de manipular colecciones de tamaño moderado podría asumirse dicha operación, pero trabajando con colecciones extensas, habituales en muchas aplicaciones actuales, el tiempo necesario para realizar la ordenación hace que el proceso sea inaceptable.

Una solución podría consistir en diseñar un algoritmo que, por una parte, evitase la ordenación global del vector de cotas, lo que afectaría inicialmente a la expresión de coste reduciendo el *tiempo extra CPU*, y que, por otra parte, lograse minimizar el número de evaluaciones de la función de distancia para resolver una consulta, lo que se lograría procesando primero los objetos teóricamente más próximos a ella y pudiendo detener el proceso de búsqueda lo antes posible.

3. Estructura variable de intervalos

En este trabajo se propone un nuevo algoritmo que tiene como objetivo optimizar el rendimiento de las búsquedas kNN en espacios métricos, por lo que intenta minimizar la expresión de coste indicada en la ecuación 1 de la sección 1.

Nuestra propuesta, no solo trata de reducir al máximo el número de evaluaciones de la función de distancia necesarias para resolver una consulta, como hacen muchas de las descritas en la sección anterior, sino que intenta minimizar el *tiempo extra CPU* mediante el diseño de un método con complejidad lineal. Esta última característica hace que sea aplicable tanto a colecciones de pequeño tamaño, como sobre todo a grandes colecciones en las que la habitual complejidad $O(n \log n)$ convierte en inviables muchos métodos existentes. Además nuestro algoritmo se puede aplicar a cualquier índice con estructura *matricial* sin que resulte necesario realizar ninguna modificación en la estructura del índice.

La ordenación de los objetos de la base de datos se realiza en función de su distancia estimada a la consulta, pero se evita la ordenación completa de todos los objetos mediante la utilización de una estructura variable de intervalos en los que se distribuyen los objetos, haciendo que la ordenación real afecte a un número reducido de ellos. Esto reducirá el *tiempo extra CPU* en la expresión de coste.

La estructura variable de intervalos permite también resolver satisfactoriamente dos problemas adicionales; uno es la necesidad de que la distancia del actual k vecino más cercano a la consulta, converja con rapidez al valor real de dicha distancia, y el otro es garantizar que el proceso de búsqueda pueda finalizar con éxito cuanto antes, evitando que la comprobación involucre a objetos que finalmente no van a formar parte de los k vecinos más cercanos. Ambos problemas afectan al número de evaluaciones de la función de distancia, por lo que solucionarlos reducirá ese factor en la expresión de coste.

3.1. Descripción de la estructura

La solución se basa en distribuir las posibles distancias entre los objetos del espacio métrico en una serie de intervalos. Para ello se considera que M es la distancia máxima entre dos objetos cualesquiera del espacio métrico y que int es el tamaño de cada intervalo, con lo que el número de intervalos dependerá de ambos factores. Se tiene entonces que $S = \{s_i \,/\, i = 0, \ldots, \frac{M}{int} \wedge s_i = [i{*}int, (i{+}1){*}int)\}$ es el conjunto de intervalos definidos para ese espacio métrico en concreto.

El número de intervalos definido sobre el espacio será también el número de filas de una estructura matricial, representada en la figura 1, en la que a cada fila se le asocia una serie de columnas formando un cubo o *bucket* para cada fila. Cada cubo está compuesto por pares de la forma (oid, l_b), donde:

- *oid* se corresponde con el identificador unívoco de un objeto en la base de datos.

- l_b es la cota inferior para la distancia real entre el objeto *oid* y la consulta q.

La distribución de los objetos en los cubos viene dada en función de la distancia estimada a la consulta, de forma que un objeto está en el cubo b_i asociado al intervalo s_i, si el valor de su cota l_b está en el intervalo $[i*int, (i+1)*int)$.

3.2. Búsquedas *kNN*

Para resolver una consulta $kNN(q)$, el algoritmo que proponemos realiza los siguientes pasos:

1. Se inicializa la estructura descrita en la figura 1 en función de los parámetros M e int. Este proceso solo se realiza una vez, ya que no es necesario repetirlo al cambiar la consulta.

2. Se calcula la cota inferior l_b para la distancia entre cada objeto de la base de datos y la consulta q. Esta cota, junto al *oid* del objeto en cuestión, se añaden al cubo asociado al intervalo correspondiente, que será el determinado por el valor de la cota l_b.

   Una vez terminado este proceso, se cumple que todas las cotas inferiores l_b de los objetos almacenados en cualquier cubo b_i, son estrictamente inferiores a cualquier otra cota incluida en otro cubo b_j siendo $j > i$. Es decir, mediante la distribución en intervalos estamos obteniendo una ordenación parcial del vector de cotas, lo que nos permitirá procesar primero aquellos objetos más próximos a la consulta. De esta forma se logra que la distancia entre la consulta q y el k-ésimo vecino más cercano converja rápidamente a la distancia real, minimizando así el número de evaluaciones de la función de distancia necesarias para resolver la consulta, esto es, el parámetro $\#d$ de la expresión de coste de la ecuación 1.

3. El último paso consiste en procesar secuencialmente la lista de cubos. Los k primeros objetos se comparan directamente con la consulta y pasan a formar parte del conjunto kNN. Para el resto de objetos se procede de la siguiente forma:

   - Si la distancia entre la consulta q y el actual k vecino más cercano es menor que $i * int$ para algún i con $i = 1, \ldots, \frac{M}{int}$, se comprueba si la cota l_b asociada a cada objeto de ese cubo es menor que la distancia entre q y el actual k vecino más cercano. En caso afirmativo, se compara directamente el objeto con la consulta y si la distancia es inferior a la distancia entre q y el actual k vecino más cercano, este objeto pasa a formar parte de los k vecinos más cercanos.
   - Si por el contrario la distancia entre la consulta q y el actual k vecino más cercano es mayor o igual que $i * int$ para algún i con $i = 1, \ldots, \frac{M}{int}$, el algoritmo de búsqueda se detiene, ya que está garantizado que los objetos que pertenecen a cualquier cubo b_j con $j = i, \ldots, \frac{M}{int}$ están más alejados de la consulta que el actual k vecino más cercano.

Con esta propuesta hemos logrado los dos grandes objetivos que nos habíamos planteado al diseñar un algoritmo que optimizase las búsquedas kNN en espacios métricos:

- Minimizar el *tiempo extra CPU*, ya que el algoritmo presenta complejidad lineal $O(n)$, por lo que su aplicación resulta adecuada en colecciones de datos de gran tamaño.

- Reducir el número de evaluaciones de la función de distancia $\#d$, ya que el radio de búsqueda converge rápidamente al valor real de la distancia entre la consulta q y el k-ésimo vecino más cercano.

Aunque el algoritmo de búsqueda presentado en este trabajo alcanza unos resultados

altamente satisfactorios, proponemos aún una posible mejora para tratar de igualar su rendimiento, en cuanto al número de evaluaciones de la función de distancia, con el de los algoritmos de búsquedas kNN que ordenan completamente el vector de cotas. Dichos algoritmos consiguen una mayor eficiencia en cuanto a $\#d$, ya que en el momento en que encuentran un objeto con una cota que supera la distancia entre q y el actual k vecino más cercano, se detienen automáticamente. Esto es posible porque sus cotas están ordenadas. En nuestra propuesta inicial, las cotas están ordenadas si corresponden a objetos que pertenecen a cubos asociados a intervalos diferentes, pero no lo están si se trata de objetos en el mismo cubo. Por ese motivo al acceder a un cubo deben de procesarse todos sus elementos. Sin embargo, esta pérdida de rendimiento se puede evitar si ordenamos los elementos de un cubo antes de procesarlo. Aunque esto último requiere $O(n \log n)$, el impacto sobre el tiempo total T resulta inapreciable y a cambio se logra minimizar el número de evaluaciones de la función de distancia, alcanzando en este aspecto un rendimiento similar al que obtendríamos ordenando el vector de cotas en su totalidad.

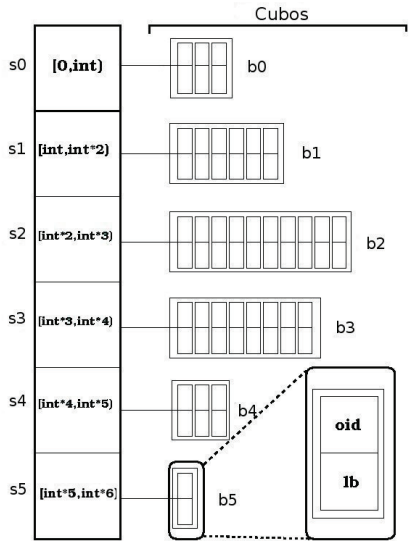

Figura 1: Vector de cubos correspondiente a los seis primeros intervalos.

4. Resultados experimentales

4.1. Entorno de pruebas

Para llevar a cabo la evaluación experimental de nuestra propuesta, realizamos experimentos con varios espacios métricos que representan colecciones de datos reales, y cuyas métricas son representativas de un amplio espectro de complejidad algorítmica. En concreto, las colecciones de prueba que utilizamos son:

- NASA: contiene 40.151 imágenes extraídas de repositorios de imágenes y vídeo de la NASA, cada una de ellas representada por un vector de características de 20 componentes. Como métrica para comparar dos imágenes se utilizó la distancia euclídea ($O(n)$, siendo n el número de componentes de cada vector).
- ENGLISH: contiene 69.069 palabras extraídas de un diccionario inglés. Se utilizó la distancia de edición [8], calculada como el número de símbolos a insertar, eliminar o substituir en una cadena para convertirla en otra ($O(m \times n)$, sindo m y n la longitud de las dos cadenas).

Para comparar nuestro algoritmo con otros descritos en el estado del arte, implementamos las tres estrategias que describimos para la búsqueda kNN, sobre los dos siguientes métodos de búsqueda basados en pivotes con estructura matricial: *SSS* [3] y *UPI* [1]. Los tres algoritmos que implementamos fueron: *cubos*, *cubos_sorted* (la versión que ordena los cubos antes de procesarlos) y la versión ya descrita en el estado del arte *sorted*, que ordena el vector de cotas completo antes de procesarlo.

También comparamos la eficiencia de los tres algoritmos con la de otros métodos, tanto basados en *clustering*, como es el método *LC: List of Clusters* [5], como con algún método basado en pivotes con estructura jerárquica: *Burkhard-Keller Tree (BKT)* [4].

Todas las pruebas se realizaron sobre una máquina con procesador AMD Athlon(tm) 64

Processor 3800+ (2,4 GHz), 8GB de memoria RAM y 512Kb de memoria *cache*, sistema operativo Ubuntu 8.04.4 LTS (kernel Linux 2.6.24-26-generic (x86_64)). La compilación se realizó con GNU C (gcc 4.2.4) con el flag -09 para optimización activado.

4.2. Análisis de los resultados

En esta sección presentamos los resultados que obtuvimos en las pruebas realizadas para evaluar el rendimiento de nuestro algoritmo. Para cada colección de datos y para cada valor del parámetro k (número de vecinos más cercanos recuperados para cada consulta), se indexó el 90 % de la colección y se utilizaron el resto de los objetos de cada colección (10 %) como objetos de consulta. Para el parámetro k de las búsquedas kNN, se determinaron los siguientes valores: $k = 10, 30, 50, 70, 90$. Los resultados se muestran tanto en función del número medio de evaluaciones de la función de distancia para resolver cada tanda de consultas, como en función del tiempo total T necesario para ejecutar cada bloque de consultas.

En las tablas 1 y 2 se muestran los resultados obtenidos en términos del número de evaluaciones de la función de distancia necesarias para una consulta. Para los métodos SSS y UPI, se muestran los resultados obtenidos ordenando el vector de cotas completo (S), utilizando la estructura de cubos sin ordenar (C), y utilizando la estructura de cubos y, además, ordenándolos (CS).

Como podemos ver en ambas tablas, nuestro algoritmo obtiene unos resultados prácticamente idénticos al de las versiones que necesitan realizar algún tipo de ordenación, ya sea del vector completo de cotas o de los cubos. Además, conseguimos mejores resultados que los que obtienen BKT y LC, dos métodos que utilizan información jerárquica para guiar al algoritmo de búsqueda.

Las figuras 2 y 3 muestran los tiempos de ejecución obtenidos en las dos colecciones para resolver las consultas. En la figura 2 podemos ver la importante diferencia de rendimiento que existe entre ordenar el vector completo de cotas (*SSS+S, UPI+S*) y utilizar la estrategia de ordenación parcial que proponemos

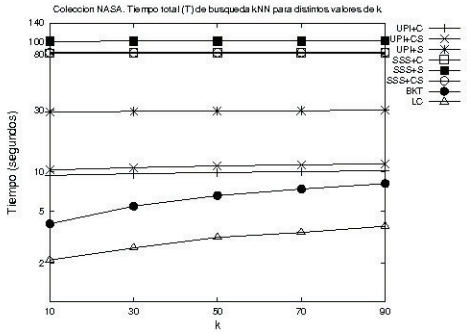

Figura 2: Tiempo total T en NASA.

en este trabajo. Aún así, los índices BKT y LC obtienen mejores tiempos de ejecución debido a que son índices que no necesitan procesar grandes matrices de distancias durante las consultas, además de por el bajo coste computacional de la función de distancia utilizada en la colección NASA. Sin embargo, observando la figura 3, que muestra los resultados para la colección ENGLISH (en la que el coste de la función de distancia as algo más alto) los métodos BKT y LC ya obtienen peores resultados que el índice *UPI*. La tendencia está clara, al ir aumentando la complejidad de la función de distancia, estos dos métodos empeoran rápidamente su rendimiento.

En esta colección, al igual que en la anterior, se obtienen mejores tiempos de ejecución para el algoritmo que no requiere ordenación *SSS+C, UPI+C*. La diferencia entre *UPI+C* o *UPI+CS* y *UPI+S* es de prácticamente el doble mientras que entre las mismas versiones para el índice *SSS* son algo inferiores. Esto último es debido a que el *tiempo extra CPU* necesario para procesar la matriz del índice (en el caso particular de SSS, de tamaño $n \times m$, siendo m el número de pivotes) es muy alto y atenúa el efecto del *tiempo extra CPU* necesario para ordenar el vector completo de cotas.

5. Conclusiones

En este trabajo hemos presentado un nuevo algoritmo para la búsqueda kNN que minimi-

k	UPI			SSS			BKT	LC
	C	CS	S	C	CS	S		
10	2717,07	2716,69	2716,69	252,17	252,12	252,12	4563,56	3168,57
30	3128,37	3128,23	3128,23	391,42	391,36	391,36	6137,43	3907,82
50	3383,43	3383,17	3383,17	498,53	498,50	498,50	7330,95	4457,59
70	3548,80	3548,56	3548,56	590,45	590,42	590,42	8157,54	4928,58
90	3680,16	3679,89	3679,89	671,53	671,48	671,48	8854,16	5326,33

Tabla 1: Número de evaluaciones de la función de distancia para la colección NASA.

k	UPI			SSS			BKT	LC
	C	CS	S	C	CS	S		
10	7606,29	7606,29	7606,29	1773,52	1773,52	1773,52	45272,48	22425,35
30	11337,93	11337,93	11337,93	4111,96	4111,96	4111,96	48986,34	31328,89
50	13535,82	13535,82	13535,82	6043,90	6043,90	6043,90	51105,59	35351,17
70	15172,63	15172,63	15172,87	7325,06	7325,06	7357,75	52142,25	37994,89
90	16503,63	16503,63	16503,72	8517,17	8517,17	8545,79	53116,28	39971,27

Tabla 2: Número de evaluaciones de la función de distancia para la colección ENGLISH.

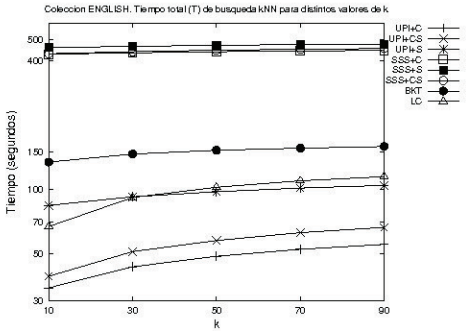

Figura 3: Tiempo total T en ENGLISH.

za el impacto de los factores que más negativamente afectan al rendimiento de la operación de búsqueda: el *tiempo extra CPU* y el número de evaluaciones de la función de distancia $\#d$. La reducción de ambos factores, en conjunto, permite que sea una opción válida aplicable a cualquier base de datos de cualquier tamaño, ya que la complejidad lineal del algoritmo junto con la minimización del factor $\#d$ son cuestiones deseables y fundamentales para determinar el rendimiento de cualquier método de búsqueda por similitud. Además, nuestro algoritmo puede ser visto como un *plug-in* para cualquier método de búsqueda basado en pivotes sin información jerárquica, ya que se puede aplicar a cualquiera de ellos sin requerir ningún tipo de cambio estructural sobre los índices que manejan.

Todas estas ventajas hacen de nuestra propuesta una opción válida y de gran interés para aplicaciones que necesiten realizar búsquedas por similitud en espacios métricos.

Referencias

[1] L. G. Ares, N. R. Brisaboa, M. F. Esteller, Ó. Pedreira, and Á. S. Places. Optimal pivots to minimize the index size for metric access methods. In P. Z. Tomas Skopal, editor, *2nd International Workshop on Similarity Search and Applications (SISAP'09)*, pages 74–80, Prague (Czech Republic), 2009.

[2] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proc. of the ACM International Conference on Management of Data (SIGMOD'97)*, pages 357–368, Tucson, Arizona, USA, May 1997. ACM Press.

[3] N. R. Brisaboa, A. Fariña, O. Pedreira, and N. Reyes. Similarity search using sparse pivots for efficient multimedia information retrieval. In *Proceedings of the 8th IEEE International Symposium on Multimedia (ISM2006)*, pages 881–888, San Diego, California, USA, 2006. IEEE CS Press.

[4] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, May 1973. ACM Press.

[5] E. Chávez and G. Navarro. An effective clustering algorithm to index high dimensional metric spaces. In *SPIRE '00: Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, page 75, Washington, DC, USA, 2000. IEEE Computer Society.

[6] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001. ACM Press.

[7] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems (TODS 2003), ACM Press*, 28(4):517–580, 2003.

[8] V. Levenshtein. Binary codes capable of correcting spurious insertions or deletions of ones. *Problems of Information Transmission*, 1:8–17, 1965. Kluwer Academic Publishing.

[9] C. D. Manning, P. Raghavan, and H. Schütze. *An introduction to information retrieval.* Cambridge University Press, 2008.

[10] L. Micó, J. Oncina, and R. E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (aesa) with linear pre-processing time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994. Elsevier.

[11] G. Navarro. Searching in metric spaces by spatial approximation. In *Proc. of String Processing and Information Retrieval (SPIRE'99)*, pages 141–148. IEEE CS Press, 1999.

[12] H. Noltemeier, K. Verbarg, and C. Zirkelbach. Monotonous bisector* trees - a tool for efficient partitioning of complex scenes of geometric objects. In *Data Structures and Efficient Algorithms, Final Report on the DFG Special Joint Initiative*, pages 186–203, London, UK, 1992. Springer-Verlag.

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

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

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