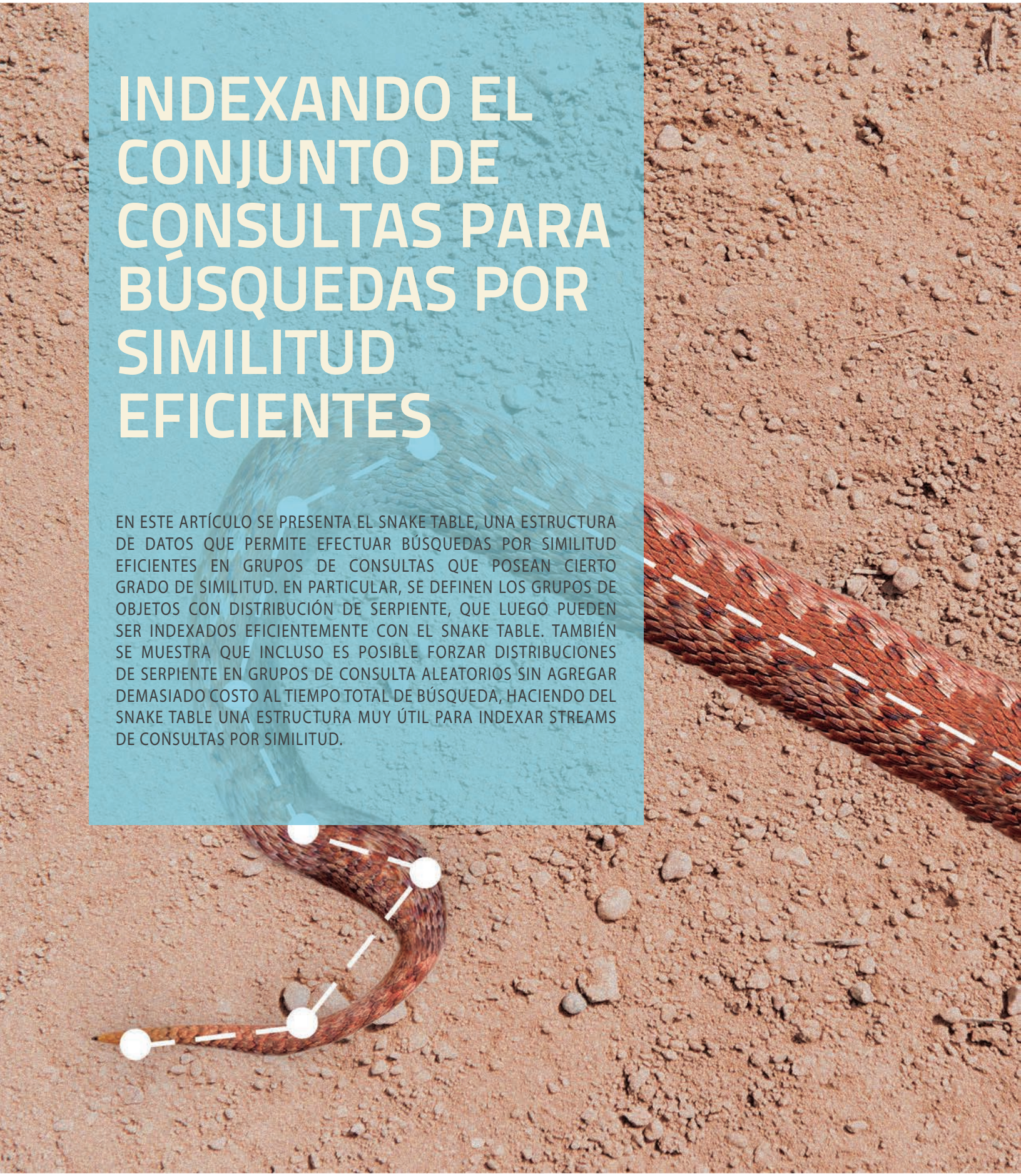
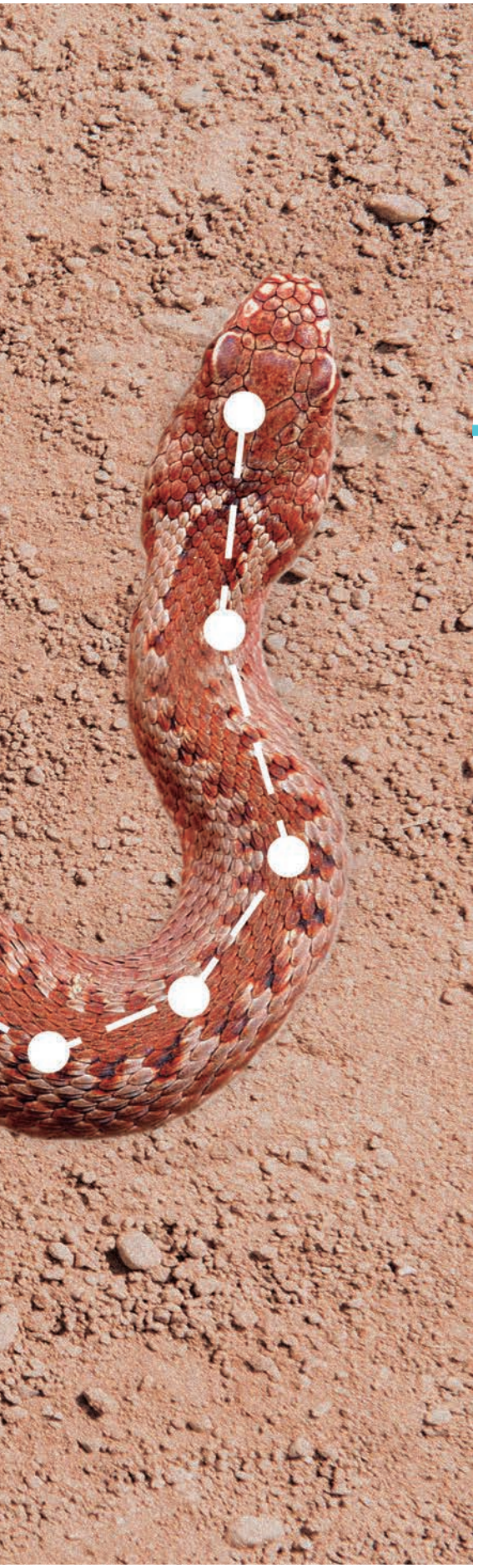
INDEXANDO EL CONJUNTO DE CONSULTAS PARA BÚSQUEDAS POR SIMILITUD EFICIENTES

EN ESTE ARTÍCULO SE PRESENTA EL SNAKE TABLE, UNA ESTRUCTURA DE DATOS QUE PERMITE EFECTUAR BÚSQUEDAS POR SIMILITUD EFICIENTES EN GRUPOS DE CONSULTAS QUE POSEAN CIERTO GRADO DE SIMILITUD. EN PARTICULAR, SE DEFINEN LOS GRUPOS DE OBJETOS CON DISTRIBUCIÓN DE SERPIENTE, QUE LUEGO PUEDEN SER INDEXADOS EFICIENTEMENTE CON EL SNAKE TABLE. TAMBIÉN SE MUESTRA QUE INCLUSO ES POSIBLE FORZAR DISTRIBUCIONES DE SERPIENTE EN GRUPOS DE CONSULTA ALEATORIOS SIN AGREGAR DEMASIADO COSTO AL TIEMPO TOTAL DE BÚSQUEDA, HACIENDO DEL SNAKE TABLE UNA ESTRUCTURA MUY ÚTIL PARA INDEXAR STREAMS DE CONSULTAS POR SIMILITUD.





BENJAMÍNBUSTOS

Profesor Asociado Departamento de Ciencias de la Computación, Universidad de Chile. Doctor en Ciencias Naturales de la Universität Konstanz, Alemania (2006); Magíster en Computación (2002) e Ingeniero Civil en Computación, Universidad de Chile (2001). Áreas de investigación: Bases de Datos Multimedia, Búsqueda por Similitud e Indexamiento de Bases de Datos Espaciales y Métricas.

bebustos@dcc.uchile.cl

ESTE TRABAJO [2] FUE REALIZADO EN CONJUNTO CON JUAN MANUEL BARRIOS (MIENTRAS REALIZABA SU DOCTORADO EN EL DCC) Y CON TOMAS SKOPAL, PROFESOR DE LA CHARLES UNIVERSITY IN PRAGUE.

BÚSQUEDA POR SIMILITUD BASADA EN CONTENIDO

El principal objetivo de la búsqueda por similitud es encontrar objetos que se parezcan. Ésta es una tarea que para un humano puede ser muy intuitiva de realizar. Por ejemplo, si tenemos tres fotos distintas, dos tomadas a una manzana y una tercera tomada a un plátano, un ser humano inmediatamente puede asociar las fotos de las manzanas como similares y separar la del plátano, que es completamente distinta. Sin embargo, enseñarle al computador a realizar este tipo de tareas es complejo y ha sido un tema de investigación que involucra distintas áreas como Visión Computacional, Aprendizaje de Máquina, Computación Gráfica, Recuperación de Información y Bases de Datos.

En el área de Búsqueda por Similitud de Objetos Multimedia (imágenes, vídeo, audio, objetos 3D, etc.), una estrategia genérica para resolver el problema de la búsqueda por similitud es utilizando el contenido mismo del objeto multimedia para realizar la búsqueda. Por ejemplo, si el objetivo es encontrar imágenes similares, éstas se procesan utilizando técnicas de computación gráfica para analizar sus colores, texturas, orientaciones de los bordes, y cualquier otra característica que se les pueda medir. Usualmente, dichas características corresponden a atributos numéricos que se agrupan para formar vectores característicos, también denominados descriptores. Los descriptores pueden ser globales (un solo vector caracteriza al objeto multimedia completo) o locales (un conjunto de vectores caracteriza al objeto multimedia). Adicionalmente, es necesario definir una función que permita comparar la similitud entre dos descriptores (globales o locales) para determinar qué tanto se parecen dos objetos. Por conveniencia, es usual definir la función de similitud como una función de distancia entre los objetos: mientras más lejanos estén, más disímiles son. La forma que se utiliza para caracterizar los objetos (descriptores) y para compararlos entre sí (función de distancia), definen el modelo de similitud para la colección de objetos.

Para realizar búsquedas por similitud, una forma usual es utilizar lo que se conoce como *query-by-example*. El usuario que necesita realizar una búsqueda dispone de un objeto del cuál desea encontrar sus similares en una colección de datos. El procedimiento consiste en calcular el descriptor asociado al objeto de consulta, compararlo con los descriptores de los objetos en la colección de datos, y retornar aquellos objetos cuyos descriptores sean los más cercanos al de consulta. Existen dos tipos básicos de consulta por similitud. El primero se denomina búsqueda

→

por rango, que toma como parámetro un radio de búsqueda y retorna aquellos objetos de la colección que se encuentren a una distancia del objeto de consulta menor o igual que el radio de búsqueda. El segundo se denomina *k* vecinos más cercanos (*k-NN*), y lo que hace es retornar los *k* objetos de la colección más cercanos a la consulta. Estas dos consultas básicas por similitud pueden ser utilizadas como una herramienta básica para realizar tareas más complejas como encontrar grupos de objetos parecidos (*clustering*), etiquetado de objetos (clasificación), reconocimiento de patrones, etc.

Hay dos aspectos relevantes a la hora de evaluar un sistema de búsqueda por similitud: la eficacia y la eficiencia. Por una parte, la eficacia corresponde a qué tan buenos son los resultados retornados por el sistema, es decir si los objetos retornados realmente se parecen al objeto de consulta (comparándolo, por ejemplo, a lo que habría escogido un ser humano). Las medidas de eficacia permiten evaluar el modelo de similitud definido. Por otra parte, la eficiencia corresponde a qué tan rápido el sistema es capaz de retornar la respuesta. En este artículo nos concentraremos en el tema de la eficiencia de la búsqueda por similitud.

Un algoritmo trivial para implementar las búsquedas por similitud (por rango o *k-NN*) es realizar una búsqueda secuencial en la colección de datos: se mide la distancia entre el descriptor del objeto de consulta y todos los descriptores en la colección de datos, reportando aquellos que sean similares a la consulta. Si bien esta es una solución simple, tiene el problema que puede ser muy lento en la práctica, ya sea porque la colección de datos es muy grande o porque el cálculo de la función de distancia es computacionalmente costoso. Para resolver esto, se puede preprocesar la colección de datos y construir un índice que permita realizar en forma eficiente las búsquedas por similitud, descartando objetos de la búsqueda sin medir su distancia a la consulta. Para esto, lo usual es utilizar una función de distancia métrica (es decir, que es simétrica, positiva, y que cumple con la desigualdad triangular) y luego utilizar sus propiedades para realizar el descarte en forma correcta, sin perder objetos relevantes. En la literatura se reportan muchas formas distintas de indexamiento, algunas basadas en índices multidimensionales [6] y otras en métodos de acceso métricos, también conocidos como MAMs [8]. En la siguiente sección describiré un MAM en particular: las tablas de pivotes.

TABLAS DE PIVOTES

Un pivote es un objeto que es utilizado para efectos de indexamiento. Suponga por ejemplo que se escogen P objetos de la colección de datos para ser utilizados como pivotes, y suponga que la colección de datos se compone de N objetos. Una tabla de pivotes es una matriz de dimensión $N \times P$ en donde se almacenan las distancias entre los pivotes escogidos y el resto de los objetos de la colección. Luego, la información guardada en la tabla de pivotes puede ser utilizada para descartar objetos durante una búsqueda por similitud. Si la función de distancia es representada por δ, los pivotes son representados por el conjunto $\{ p_1,\ldots,p_P \}$, los objetos de la colección son representados por el conjunto $\{u_1,\ldots,u_N\}$, y el objeto de consulta es representado por q, se puede demostrar que si δ es métrica entonces el valor $LB(q,u) = max \mid \delta(p_i,q) - \delta (p_i,u) \mid$ es una cota inferior de $\delta (q,u)$, la distancia entre el objeto de consulta q y el objeto u de la colección. Esto es, se cumple que $LB(q,u) \leq \delta (q,u)$. Note que para calcular $LB(q,u)$, por cada i se requiere conocer dos distancias, $\delta(p_i,q)$ (que debe calcularse) y $\delta (p_i,u)$ (que están almacenadas en la tabla de pivotes). Por último, note que para calcular $LB(q,u)$ sólo se requiere calcular P operaciones de resta y valores absolutos, lo cual implica un costo computacional bajo.

El valor LB puede ser utilizado para realizar la búsqueda por similitud en forma eficiente. Por ejemplo, si se desea realizar una búsqueda por rango con radio de búsqueda r, si se determina que para algún u se cumple que $LB(q,u) > r$, entonces $r < LB(q,u) \leq \delta (q,u)$, es decir la distancia entre q y u es necesariamente mayor que el radio de búsqueda. Por lo tanto, u no puede ser un objeto suficiente similar a q, por lo que puede descartarse. Note que el descarte se realizó sin tener que calcular directamente la distancia entre q y u, sino que se realizó en forma indirecta a través del valor obtenido para $LB(q,u)$. Por lo tanto, si la función de distancia es computacionalmente costosa, este método puede ser mucho más eficiente en tiempo que realizar la búsqueda en forma secuencial. La **Figura 1** muestra un ejemplo de cómo funciona el método basado en pivotes. Para el ejemplo, se escogieron tres objetos como pivotes. Sólo aquellos objetos que están en la intersección de los tres anillos de la Figura (en el ejemplo, uno sólo) no pudieron ser descartados, y se debe calcular su distancia a q.

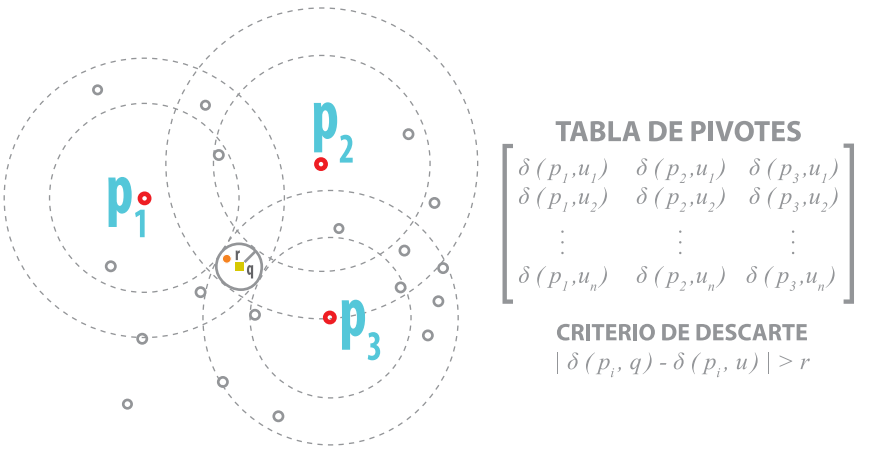


FIGURA 1.
EJEMPLO DEL USO DE LA TABLA DE PIVOTES PARA REALIZAR UNA BÚSQUEDA POR RANGO.

El método de indexamiento descrito se conoce como LAESA [5]. Hay otras técnicas de indexamiento que también utilizan tablas de pivotes, como veremos en las siguientes secciones.

STREAMS DE CONSULTAS POR SIMILITUD

Suponga que la colección de datos a indexar es estática y conocida. En las técnicas tradicionales de indexamiento para búsqueda por similitud, se realiza un preprocesamiento de la colección de datos para construir el índice. Esto tiene un costo asociado, por ejemplo construir la tabla de pivotes para el método LAESA tiene un costo de preprocesamiento asociado de $P*N$ cálculos de distancias. Todo esto se realiza en tiempo *offline*, antes que comiencen las consultas por similitud que correspondería al tiempo *online*. Si bien hay un costo que se paga *offline* para construir el índice, éste se amortiza luego durante el tiempo *online* al responder las consultas. Esto sólo vale la pena si es que efectivamente se realizarán suficientes consultas al índice, sino sería preferible no construirlo y responder las consultas usando búsquedas secuenciales.

Note que no es necesario conocer el conjunto de consulta de antemano para construir el índice. Una pregunta interesante que se puede plantear entonces es, ¿qué pasa si uno conociera a priori el conjunto de consulta? Esto es relevante, ya que desde el punto de vista de las aplicaciones prácticas se pueden identificar los siguientes tres escenarios:

1. *Completo*: El conjunto de consulta se conoce completamente de antemano.
2. *Grupos*: El conjunto de consulta no se conoce de antemano, pero el sistema de búsqueda lo va recibiendo incrementalmente en grupos de objetos de consulta.
3. *Uno-a-uno*: El conjunto de consulta no se conoce de antemano. Las consultas se realizan una a la vez, pudiendo estar correlacionadas (es decir, el resultado de una consulta puede influir en cuál será la siguiente consulta).

Otro escenario importante a considerar en bases de datos multimedia es el de *streams* de consultas, que definiremos como un conjunto de consulta de largo indeterminado a priori, cuyos objetos de consulta se van conociendo de a uno o de a grupos en forma incremental, lo cual corresponde al segundo y tercer caso descrito anteriormente (Grupos y Uno-a-uno). Más aún, dado que ahora se están considerando los conjuntos de consultas, resulta interesante preguntarse si estos pueden tener propiedades de las cuales un sistema de búsqueda podría tomar ventaja y poder mejorar la eficiencia de las búsquedas. Por ejemplo, en búsqueda por similitud en vídeos, el método estándar de búsqueda escoge *frames* consecutivos del vídeo y realiza consultas por cada uno de ellos. Dado que cada segundo de vídeo contiene usualmente 25 *frames* por segundo, dos *frames* consecutivos son muy similares entre sí (salvo que justo en ese momento se realiza un cambio de *shot*). Entonces, cabe preguntarse si, para efectos de eficiencia de la búsqueda, es posible aprovechar el hecho que los objetos asociados a consultas consecutivas son similares entre sí. La respuesta es sí, y el resto de este artículo explica en detalle la técnica de indexamiento que analiza e indexa directamente el conjunto de consulta, aprovechándose de la similitud entre los objetos de consulta. Pero antes, es necesario explicar otro algoritmo de búsqueda por similitud relevante para este trabajo que no requiere calcular explícitamente un índice de la colección de datos en el tiempo *offline*.

INDEXAMIENTO SIN ÍNDICE: D-FILE Y D-CACHE

Anteriormente se explicó que una forma sencilla pero lenta de implementar búsquedas por similitud, es realizar una búsqueda secuencial en la colección de datos. La técnica denominada D-file hace exactamente eso, pero utilizan una estructura de datos llamada D-cache [7] que le permite mejorar la eficiencia de la búsqueda. El D-cache es un cache de distancias. En esta estructura, se almacenan distancias que hayan sido calculadas durante una búsqueda por similitud, que luego pueden ser utilizadas para calcular cotas inferiores de distancia. Estas cotas pueden ayudar a descartar objetos durante la búsqueda secuencial realizada por D-file.

El algoritmo de búsqueda utilizado por D-file es el siguiente. Parte inicializando el D-cache como una estructura vacía. Esto implica que la primera búsqueda por similitud necesariamente será una búsqueda secuencial en toda la colección de datos. Cada vez que D-file calcula una distancia entre un objeto y la consulta, intenta almacenarla en el D-cache. Para esto, escoge arbitrariamente una celda utilizando una función de hash, y si está vacía guarda ahí la distancia calculada. Si no, ocupa alguna estrategia de reemplazo y decide si almacenar o no la distancia calculada. Luego, D-file utiliza el D-cache para calcular una cota de distancia entre un objeto y la consulta utilizando la información guardada en el D-cache, usando un criterio similar al utilizado en la técnica basada en pivotes. El cálculo de la cota se realiza en tiempo *O(1)*. Dicha cota puede permitir descartar el objeto que se está revisando, evitando realizar el cálculo de distancia. La **Figura 2** muestra en seudocódigo cómo funciona la búsqueda secuencial implementada en el D-file.

El método D-file tiene la ventaja con respecto a utilizar tablas de pivotes que no requiere de tiempo de preprocesamiento, ya que no necesita construir un índice para empezar a procesar las consultas. Sin embargo, D-file es eficiente sólo cuando la función de distancia utilizada es computacionalmente costosa comparada con el costo de calcular la cota inferior de distancia y los costos extras en tiempo ocupados en el cálculo de la función de hash y las estrategias de reemplazo. En caso contrario, la complejidad interna de D-file es muy grande, por lo que si bien puede evitar realizar muchos cálculos de distancia, esto no se verá reflejado en el tiempo total ocupado para realizar la consulta. Para →

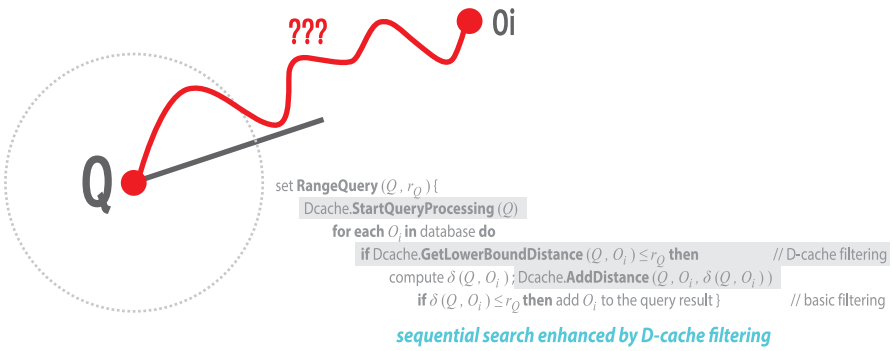


FIGURA 2.
ALGORITMO DE BÚSQUEDA UTILIZADO POR D-FILE.

muchas aplicaciones interesantes en búsqueda por similitud multimedia ésta es una gran desventaja, ya que muchos sistemas de búsqueda ocupan funciones de distancias que son simples de calcular, como la distancia Euclidiana o la distancia Manhattan. Para estas aplicaciones, el D-file no es de utilidad práctica.

Sin embargo, si el conjunto de consulta cumple con la propiedad que consultas consecutivas son similares, es posible tomar ventaja de esto en términos de eficiencia, incluso si la función de distancia es simple de calcular. Este índice se llama Snake Table y será descrito en la siguiente sección.

SNAKE TABLE: BUSCANDO SERPIENTES EN LOS CONJUNTOS DE CONSULTA

El Snake Table ("Tabla de Serpientes") es un índice basado en tablas de pivotes, diseñado específicamente para responder consultas por similitud en conjuntos de consultas que cumplen con la denominada distribución de serpiente *(snake distribution)*. Se define que un conjunto tiene una distribución de serpiente si cumple con la propiedad que la distancia entre consultas consecutivas es mucho menor que la distancia promedio entre dos objetos aleatorios de la colección de datos. Es decir, una distribución de serpiente implica que objetos consecutivos en una serie se encuentran cercanos entre sí. La **Figura 3** muestra un ejemplo de esto (R corresponde a la colección de datos). El conjunto de consulta $Q=\{q_1,\ldots,q_{12}\}$ es un *stream* de consultas que cumple con la distribución de serpiente. Note que si se dibuja un trazo que vaya uniendo consultas consecutivas forma un dibujo parecido a una serpiente. Es posible que haya consultas que no sean similares, por ejemplo en la Figura las consultas q_7 y q_8 son consecutivas, pero no están cercanas. Esto puede suceder, pero no afecta el hecho que la distribución sea de serpiente. El concepto se puede definir formalmente [2].

El índice funciona de la siguiente forma. No se realiza ningún tipo de preprocesamiento, por lo que al momento de realizar la primera consulta el Snake Table está vacío. Toda distancia que se calcule entre un objeto de la colección y una consulta se va almacenando en la Snake Table. El lugar en donde se almacene dependerá de la estrategia de reemplazo escogida. La idea es ir formando dinámicamente una tabla de pivotes, utilizando como pivotes las consultas previas realizadas. La ventaja de hacer esto es que los pivotes serán objetos cercanos a la siguiente consulta, lo cual produce buenas cotas inferiores $LB(q,u)$ dado que en el valor absoluto de la resta de distancia una de ellas es un valor pequeño. Esto es, la tabla se va modificando constantemente y se va guardando la información correspondiente a buenos pivotes para las futuras consultas. Finalmente, cuando se acaban las consultas la Snake Table se descarta. Si se inicia una nueva sesión de consultas se crea una nueva Snake Table, como muestra la **Figura 4**.

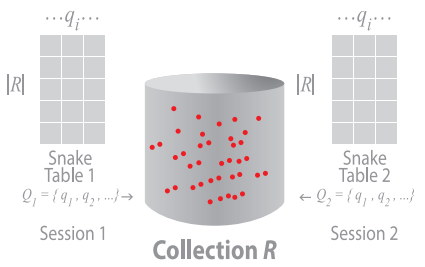


FIGURA 4.
PARA DOS SESIONES DE CONSULTA DISTINTA SE CREAN SNAKE TABLES DISTINTAS.

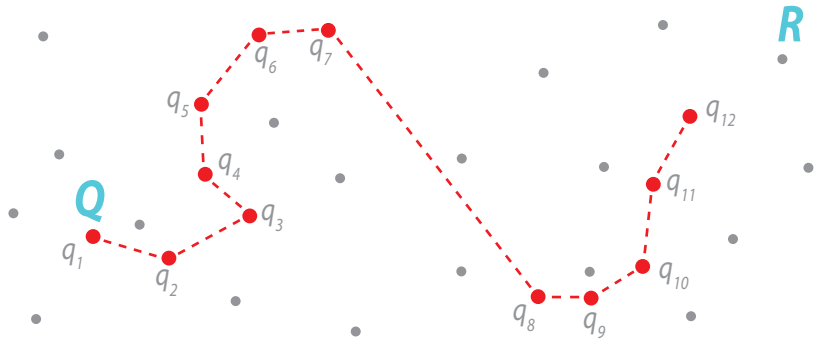


FIGURA 3.
EJEMPLO DE CONJUNTO DE CONSULTA CON DISTRIBUCIÓN DE SERPIENTE.

El tamaño de la Snake Table es fijo, y por lo tanto en algún momento no habrá más espacio para almacenar distancias nuevas. Por lo tanto, se requiere de estrategias de reemplazo para borrar distancias antiguas. En particular, se propusieron las siguientes estrategias:

1. *FIFO/Sparse*: las distancias se van almacenando consecutivamente por columnas, una por cada consulta previa, en modo round-robin. La columna más antigua de la tabla se reemplaza por la columna nueva. Si una distancia no se calcula, dicho casillero del Snake Table queda desocupado.

2. *Highest/Compact*: siempre se reemplaza la distancia más grande en la fila correspondiente. Para esta estrategia, es necesario anotar por cada celda de la tabla qué pivote es el asociado a dicha distancia.

3. *FIFO/Compact*: el reemplazo se realiza en modo round-robin, pero todos los casilleros de la tabla se utilizan. Si uno queda vacío, se utiliza con la siguiente distancia que se calcule para el objeto correspondiente. En esta estrategia también es necesario anotar por cada celda de la tabla qué pivote es el asociado a dicha distancia.

EVALUACIÓN EXPERIMENTAL

Se probó el Snake Table en tres escenarios distintos: un conjunto de consulta con distribución de serpiente, un conjunto de consulta del tipo Grupos (con un cierto grado de similitud intra-grupo), y un conjunto de consulta aleatorio.

PRIMER ESCENARIO: CONSULTAS CON DISTRIBUCIÓN DE SERPIENTE

En este primer escenario se ocupó como benchmark la colección de datos del MUSCLE-VCD-2007 [3], que es un test estándar para evaluar algoritmos de detección de copias de vídeo. Se utilizaron descriptores visuales basados en colores y bordes para caracterizar los frames de los vídeos. En particular, se mostrarán resultados con el descriptor denominado OM (cuya función de distancia es simple de calcular), con el descriptor EH (cuya función de distancia es un poco más costosa), y con el descriptor denominado EK3 (cuya función de distancia es un orden de magnitud más lento de calcular que la asociada al descriptor OM). Los detalles sobre estos descriptores puede encontrarlos en el artículo original del Snake Table [2]. Para este experimento, se compararon los índices LAESA (R: pivotes de la colección de datos, Q: pivotes de la colección de consulta), D-file, y Snake Table con sus tres estrategias de reemplazo (FS, HC, FC).

La **Figura 5** muestra los resultados obtenidos, tanto en número de distancias calculadas (izquierda) como de tiempo total de búsqueda (derecha). El valor indicado es la fracción con respecto a haber realizado una búsqueda secuencial. Se observa que Snake Table es más eficiente que LAESA usando la misma cantidad de pivotes. Además, se observa que D-file funciona mal especialmente en tiempo total de búsqueda si la función de distancia es simple, por su alta complejidad interna, pero mejora con

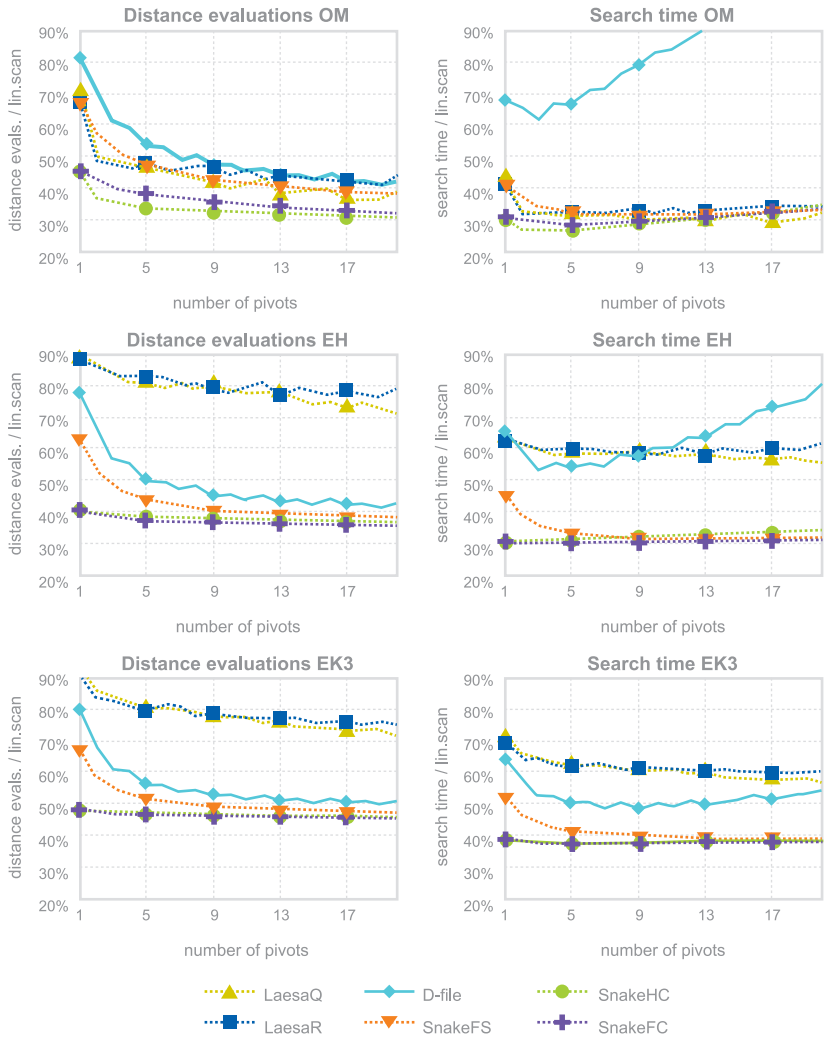


FIGURA 5.
RESULTADOS EXPERIMENTALES PARA EL PRIMER ESCENARIO.

→

funciones de distancia más costosas. De todas formas, el Snake Table es siempre más eficiente en cálculos de distancia y en tiempo. La mejor de las tres estrategias de reemplazo es la FIFO/Compact.

SEGUNDO ESCENARIO: CONSULTAS CON DESCRIPTORES LOCALES

En este segundo escenario se ocupó como benchmark la colección de datos del PISA Dataset [1]. Este benchmark consiste en una colección de fotos, a las que se les calcularon descriptores locales SIFT. Dado que ahora se utilizan descriptores locales, cada imagen está asociada a un grupo de vectores de características (aproximadamente 400 por foto). Analizando dichos grupos, se determinó que dentro de ellos existe un cierto grado de similitud, pero no tan fuerte como en los datos del primer escenario. Por esto, se probaron dos alternativas: realizando las consultas en el orden original (RND) y ordenándolas, poniendo en consultas consecutivas a vectores similares (NN). Esto último tiene un costo adicional de proceso, pero dado que los grupos de consulta son pequeños no es muy relevante este costo adicional. Se comparó el Snake Table con LAESA (pivotes sólo escogidos de la colección de datos). Ya no se considera D-file porque la función de distancia utilizada es la Euclidiana.

La **Figura 6** muestra los resultados obtenidos. Claramente, el Snake Table es más eficiente que LAESA. Además, las versiones NN en donde se ordenan primero los datos adicionales son mucho más eficientes aún, incluso considerando el costo de tener que ordenarlos. Esto implica que fue posible obtener más serpientes en los datos, lo cual beneficia al Snake Table.

TERCER ESCENARIO: CONSULTAS ALEATORIAS

En este tercer escenario se ocupó como benchmark MIRFLICKR-1M [4], que consiste en un millón de fotos bajadas de Flickr. Un subconjunto aleatorio de estas fotos se utiliza como conjunto de consulta, por lo que se comprobó que no siguen una distribución de serpiente. Para intentar forzarla, se agruparon las consultas en grupos de mil consultas y se ordenaron con el mismo método que en el segundo escenario. En este experimento se compararon LAESA con Snake Table.

La **Figura 7** muestra los resultados obtenidos. Estos muestran que incluso forzando una (leve) distribución de serpiente, Snake Table es capaz de tomar ventaja de esto. Note que el costo total en tiempo es menor que LAESA, incluso considerando el costo de ordenar los grupos de consulta. Para que esto funcione, es necesario ordenar grupos de consulta pequeños, ya que este proceso toma tiempo cuadrático en el tamaño del grupo. Aunque no se muestra en el gráfico, si no intenta inducir la distribución de serpiente entonces LAESA funciona mejor que Snake Table. Por último, cabe destacar que la mejor estrategia de reemplazo en todos los escenarios fue la estrategia FIFO/COMPACT.

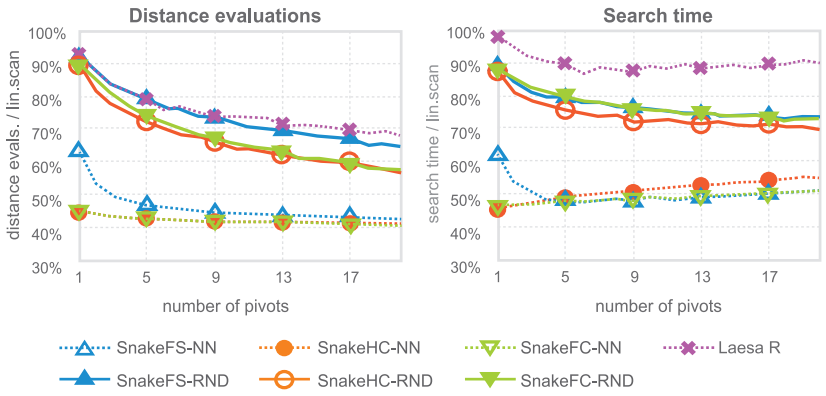


FIGURA 6.
RESULTADOS EXPERIMENTALES PARA EL SEGUNDO ESCENARIO.

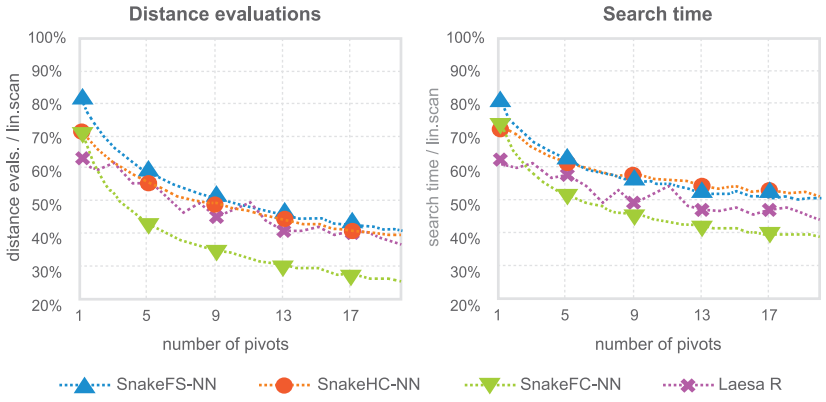


FIGURA 7.
RESULTADOS EXPERIMENTALES PARA EL TERCER ESCENARIO.

CONCLUSIONES

EN ESTE TRABAJO SE PROPUSO EL SNAKE TABLE, UN ÍNDICE DINÁMICO BASADO EN PIVOTES QUE UTILIZA AL CONJUNTO DE CONSULTA COMO PIVOTES. ANALIZANDO LAS PROPIEDADES QUE PUDIERAN TENER LOS CONJUNTOS DE CONSULTA, SE DESCUBRIÓ QUE ES POSIBLE APROVECHARSE DE LAS SIMILITUDES ENTRE LOS ELEMENTOS DE CONSULTA PARA HACER MÁS EFICIENTES LAS BÚSQUEDAS. EL SNAKE TABLE ES CAPAZ DE TOMAR VENTAJA DE LOS CONJUNTOS DE CONSULTA QUE TIENEN DISTRIBUCIÓN DE SERPIENTE (CONSULTAS CONSECUTIVAS SIMILARES ENTRE SÍ), PARA EFECTUAR EFICIENTEMENTE BÚSQUEDAS POR SIMILITUD. ■

REFERENCIAS

[1] Pisa landmarks dataset, 2011. http://www.fabriziofalchi.it/pisaDataset/.

[2] Juan Manuel Barrios, Benjamín Bustos, and Tomas Skopal. Analyzing and dynamically indexing the query set. Information Systems 45:37-47, 2014.

[3] J. Law-To, A. Joly and N. Boujemaa. Muscle-vcd-2007: a live benchmark for video copy detection, 2007. http://www-rocq.inria.fr/imedia/civrbench/.

[4] B. Thomee, Mark J. Huiskes and Michael S. Lew. New trends and ideas in visual concept detection: The mirflickr retrieval evaluation initiative. In MIR '10: Proceedings of the 2010 ACM International Conference on Multimedia Information Retrieval, pages 527{536, New York, NY, USA, 2010. ACM.

[5] María Luisa Micó, José Oncina, and Enrique Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. Pattern Recognition Letters, 15(1):9{17, January 1994.

[6] HananSamet. Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[7] Tomas Skopal, Jakub Lokoc, and Benjamín Bustos. D-cache: Universal distance cache for metric access methods. IEEE Transactions on Knowledge and Data Engineering, 24(5):868{881, May 2012.

[8] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. Similarity Search: The Metric Space Approach, volume 32 of Advances in Database Systems. Springer, 2006.