Universidad de Almería

Escuela Superior de Ingeniería

Ingeniero en Informática (Plan 1999)

PROYECTO FIN DE CARRERA

Estudio del Comportamiento del iDistance en la Recuperación de Video Basada en Contenido

Autor: ***Guillermo López González***

Director: ***Antonio Leopoldo Corral Liria***

Fecha: ***6 de Febrero de 2011***

AGRADECIMIENTOS

En primer lugar, quiero mostrar mi gratitud por el apoyo que mi familia ha mostrado en estos años de carrera y sin el cual no hubiera podido culminar mi aprendizaje con este proyecto fin de carrera.

También quiero nombrar aquí, mostrando mi agradecimiento, al que ha sido el responsable de que este proyecto sea posible, mi tutor el doctor Antonio Corral.

Dedicado a mí hermana Elena.

RESUMEN

En este proyecto se presenta el *iDistance* como método de indexación de datos altamente dimensionales utilizando la técnica *reducción de la dimensionalidad* y se estudia su comportamiento en un sistema de video basado en contenido *(Content-Based Video Retrieval, CBVR)*. Para poder crear el índice "*iDistance*" es necesario obtener los puntos de referencia del conjunto de datos *dim*-dimensional y para ello se va a utilizar una técnica de clusteirng llamada *kmeans*. Una vez creado el *iDistance,* este puede incluirse en un CBVR para probar su comportamiento en la identificación de subsecuencias de video, de manera que será el *iDistance* el que recupere los frames similares para un procesado posterior con el objetivo de la identificación de la subsecuencia de consulta. Para comparar los resultados se ha utilizado otra técnica para resolver la maldición de la dimensionalidad basada en vectores de aproximación, como es el *VA-File*. Mientras que respecto a la búsqueda de videos, se realizan mejoras para la identificación de subsecuencias de video.

En cuanto al contenido, en este proyecto se presentan las características mas importantes de los datos altamente dimensionales así como las métricas utilizadas para su clasificación en distancia. Se introduce el árbol B+ como núcleo en el que está basado el *iDistance* así como todas las operaciones asociadas a dicha estructura de datos. También se trata la teoría de grafos bipartitos y matching puesto que es imprescindible para la identificación de subsecuencias de video. Posteriormente, se estudia e implementa el *iDistance* como motor de indexación en bases de datos altamente dimensionales, prestando especial atención en la metodología de indexación y búsqueda en la consulta de los K vecinos más cercanos. Tras dicho estudio se proponen una serie de experimentos con datos de video reales con el objetivo de estudiar el rendimiento con la variación de parámetros clave en la configuración del *iDistance*. Una vez estudiado el *iDistance,* se procede a introducir dicho motor de indexación en un sistema de recuperación de video basado en contenido para la identificación de subsecuencias de video. En este proyecto, además, se propone la recuperación de las K mejores subsecuencias en ranking estudiando el comportamiento del acierto de las mismas en una batería de experimentos posterior.

PALABRAS CLAVE

Datos multidimensionales, base de datos multidimensionales, maldición de la dimensionalidad, reducción de la dimensionalidad, iDistance, VA-File, sistema de recuperación de video basado en contenido (CBVR), key-frame, identificación de subsecuencias de video, consulta de los K vecinos mas cercanos, matching, grafo bipartito, métricas de distancias, recuperación de las k mejoras subsecuencias en ranking, kmeans, algoritmo de Lloyd.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CÓDIGOS

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICAS

1

Introducción

Con el rápido auge que están teniendo tanto técnicas de almacenamiento masivo de videos, como la recuperación de recursos de video distribuidos en la Web, la investigación en el campo de recuperación de video basada en contenido (Content-Based Video Retrieval, CBVR) está teniendo una gran aceptación. Como es bien sabido, un sistema CBVR genérico [SHS+08], además de la base de datos (de videos) y de la interfaz gráfica de usuario (para interactuar con el sistema), consta principalmente de los siguientes módulos: (1) segmentación de vídeo y extracción de características, (2) organización de vectores de características, y (3) motor de búsqueda de videos.

Este proyecto se centra en los dos últimos módulos. En lo que se refiere a la *organización de vectores de características*, las representaciones de video caracterizadas se indexan utilizando eficientes métodos de acceso altamente dimensionales [Sam06], de tal manera que la recuperación de videos se pueda realizar de manera efectiva y eficiente. Respecto al *motor de búsqueda de videos*, éste realiza una búsqueda en la estructura de índices subyacente para acelerar el proceso de consulta. En general, el proceso de almacenamiento, indexación y recuperación de datos altamente dimensionales, extraídos como características de objetos complejos (videos), es una nueva funcionalidad de los sistemas de bases de datos que ha cobrado gran relevancia en muchos dominios de aplicación en la actualidad.

En un proyecto anterior se analizó, diseñó e implementó un framework utilizando *VA-file* para poder almacenar y realizar consultas de vídeo [Riv10] tal y como se ilustra en la Figura 1. En dicha gráfica se observa que a partir de una base de datos de secuencias de vectores de características (previa segmentación y caracterización del video) se realiza un proceso de indexación para la creación del *VA-file*, y sobre esta estructura de datos se realizarán las consultas de vídeo. Por un lado, la consulta *VSS* (Video Similarity Search), que a partir de un conjunto de vectores de características (secuencia) agrupados como una consulta, tratará de encontrar cuál es la secuencia de la base de datos a la que es más similar. Por otro lado, un conjunto de vectores de características para una consulta de tipo *VSI* (Video Subsequence Identification) será la entrada del proceso de consulta sobre el *VA-file* tanto para la consulta de los K vecinos más cercanos (kNNSearch) como para la consulta en rango de distancia. Una vez que se ha ejecutado dicha consulta el resultado se envía a la consulta *VSI*, la cual buscará el mapping 1:1 más similar de los *key-frames* devueltos con respecto a la secuencia de consulta para la obtención de la subsecuencia más similar. Para ello se estudian e implementan algoritmos para la identificación de subsecuencias de video similares mediante el uso de un grafo bipartito. En este proceso se realiza una transformación en dicho grafo y un proceso de matching (emparejamiento). Todo ello con la intención de identificar una posible existencia de un ordenamiento temporal, una longitud o una edición del contenido diferente.

Se va a hacer especial hincapié en la consulta de los K vecinos más cercanos puesto que es la operación más demandada en los sistemas de datos multidimensionales. Hay numerosas investigaciones para intentar optimizar dicha búsqueda, en este proyecto fin de carrera nos vamos a centrar en el *iDistance*.

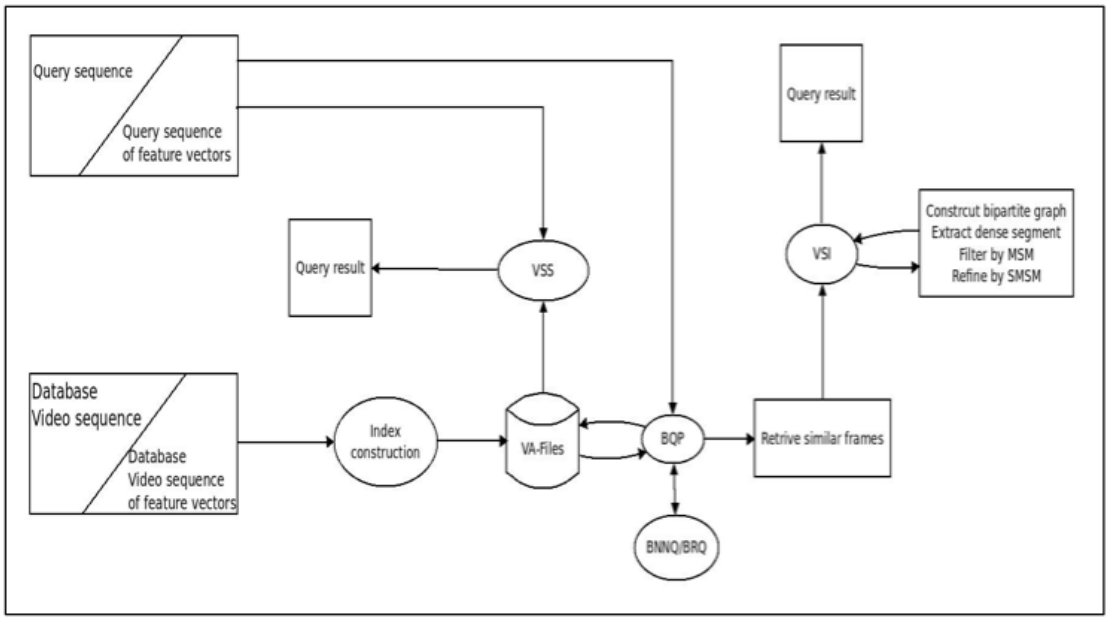


Figura 1. Framework para la indexación y procesamiento de consultas de video utilizando *VA-files*.

El *iDistance* [JOT+05] es un método de acceso multidimensional el cual transforma cada punto altamente dimensional en un valor de distancia unidimensional (respecto al punto de referencia seleccionado) para poder indexarlo en un B^+-Tree. De una forma más precisa, el *iDistance* consiste en seleccionar un centro *p* y agrupar sus *m* elementos más cercanos. Al conjunto restante se le aplica el mismo proceso hasta que todos los elementos hayan sido agrupados. Cada centro almacena un radio de cobertura, que después se utiliza para descartar elementos en una consulta.

En este proyecto se estudiará el framework existente para CBVR basado en *VA-files* [Riv10] y se realizarán extensiones representativas para mejorarlo. También se estudiará e implementará el *iDistance* [JOT+05], y se sustituirá por el *VA-file* en dicho entorno, realizándose una comparativa desde el punto de vista de la eficiencia.

1.1 Objetivos

Con la realización de este proyecto se pretenden alcanzar los siguientes objetivos:

- Estudiar el framework actual para la recuperación de video basada en contenido utilizando *VA-file* [Riv10] y realizar extensiones representativas como:
    - Obtener las K subsecuencias más similares en ranking.
    - Implementar otras métricas que reflejen la distancia entre dos vectores de características (Euclídea, Canberra, Bray-Curtis, Chord, Squared Chic-Squared, etc.), como función principal para determinar la similitud de contenido visual entre secuencias de video y comprobar su comportamiento.

- Estudiar e implementar el *iDistance*, haciéndolo variable en el factor de compactación, e incluyéndole una implementación eficiente del algoritmo *k-Means* [KMN+02] como técnica de clustering. Además, se implementarán consultas basadas en distancias sobre dicho índice.

- Cambiar, en el framework existente, el *VA-file* por un índice jerárquico con un excelente rendimiento ante datos altamente dimensionales, como es el *iDistance*.

- Comparar, desde el punto de vista de la efectividad y eficiencia, el nuevo framework (basado en *iDistance*) con respecto al que utiliza *VA-file* en base a las consultas: KNNQ (K Nearest Neighbor Query), *VSS* (Video Similarity Search) y *VSI* (Video Subsquence Identification).

- Estudiar la posibilidad de añadir nuevas consultas al framework para la recuperación de video basada en contenido. Por ejemplo, consultas basadas en matching (matching exacto y matching de similitud), u otro tipo de consulta basada en contenido de video, según la caracterización hecha de sus *key-frames*: recuperación de las K sub-secuencias en ranking en base a su similitud.

1.2 Ámbito del proyecto y motivación

Los sistemas de gestión de base de datos tradicionales ofrecen al usuario facilidades para diseñar la estructura de la información y realizar operaciones básicas de recuperación en base a unos determinados criterios de búsqueda. Dichas colecciones de datos son cada vez menos estructuradas y con datos cuya dimensionalidad crece conforme aumenta el número de características a considerar en el proceso de estudio, como por ejemplo, objetos multimedia (imágenes, videos, música, texto, etc.).

Con el auge de la gestión de información de video, tanto centralizado como distribuido, la recuperación de video basada en contenido está tomando una gran importancia en la actualidad. Para poder soportar eficientemente este tipo de aplicaciones, la indexación de video basada en contenido debe tenerse en cuenta. Como ya sabemos, un video puede representarse como una secuencia de *key-frames* caracterizados por vectores altamente dimensionales. Además, conforme aumenta la dimensionalidad de los datos tenemos el problema de que algunos índices dejan de ser eficiente y para solucionar este problema se han propuesto muchos índices, como por ejemplo el *iDistance*.

Por lo tanto, el ámbito de este proyecto se centra en el estudio e implementación del *iDistance*, y su uso en la recuperación de video (vectores de características altamente dimensionales) utilizando consultas basadas en distancias. Finalmente, se integrará en el framework de procesamiento de consultas para la recuperación de video basada en contenido, como por ejemplo la consulta de video similares, la identificación de subsecuencias de video, etc.

1.3 Conocimientos adquiridos

Tras el estudio de los fundamentos y del *VA-File* y tras la implementación tanto del *iDistance* y del sistema de recuperación de video mejorando el algoritmo de reconocimiento de subsecuencias, este proyecto me ha aportado grandes conocimientos sobre tratamiento e indexación de datos altamente dimensionales y como los sistemas de recuperación de video trabajan con las bases de datos indexadas en archivos altamente dimensionales.

Además me ha permitido reconocer en qué circunstancias es mejor utilizar el *iDistance* o el *VA-File* en la recuperación de video y que parámetros son los que presentan un mejor rendimiento a la hora de identificar subsecuencias. También me ha permitido conocer un algoritmo de clustering: Lloyd´s Algorithm junto con los arboles *KD-Tree* para el cálculo de clústeres.

Por último, este proyecto me ha enseñado a enfrentarme a diseños y códigos ajenos para extenderlos y mejorarlos, como por ejemplo el CBVR utilizado, el *iDistance*, el *VA-File* y el algoritmo de Lloyd para el cálculo del *kmeans*

1.4 Herramientas

Desde el punto de vista software, se utilizará para la implementación el compilador gcc/g++ sobre Fedora, usando Eclipse con IDE de programación C/C++.

Los recursos hardware que se utilizarán son los siguientes:

- Ordenador portátil HP Intel® Core™ i7 1,60 GHz y 4 GBs de RAM.

- Ordenador HP (Intel Core i5 660 a 3,33Ghz (TurboBost 3,47Ghz), 4 GB de RAM, Arquitectura 32 bits y SO Windows 7) para la ejecución de los experimentos.

1.5 Plan de trabajo

Para la elaboración del proyecto se realizarán las siguientes fases:

1. Revisión bibliográfica y búsqueda de información en libros y revistas científicas sobre métodos de acceso altamente dimensionales y su utilización en recuperación de video basada en contenido.

2. Estudio del framework existente para la recuperación de video basada en contenido utilizando *VA-file* [Riv10] e implementación de extensiones representativas para mejorarlo y enriquecerlo.

3. Estudiar e implementar en C/C++ el *iDistance* junto con las operaciones de consulta más comunes (los K vecinos más próximos y en rango de distancia).

4. Sustituir el *VA-file* por el *iDistance* en el framework existente para la recuperación de video basada en contenido e intentar implementar nuevas consultas de video.

5. Experimentación con el nuevo entorno basado en el *iDistance* y análisis de los resultados obtenidos. Comparación con el framework existente basado en *VA-file*, fundamentalmente desde el punto de vista de la eficiencia.

6. Elaboración de la memoria del proyecto, prestando especial atención a los resultados experimentales obtenidos.

1.6 Temporización

El proyecto ha sido desarrollado en diferentes etapas, quedando expuestas a continuación:

1. Revisión bibliográfica y búsqueda de información en libros y revistas científicas: 5 semanas.

2. Estudio del framework existente para la recuperación de video basada en contenido utilizando *VA-file* [Riv10]: 3 Semanas.

3. Implementación de extensiones en el framework: 2 semanas.

4. Estudio e implementación en C/C++ del *iDistance* junto con las operaciones de consulta más comunes (los K vecinos más próximos y en rango de distancia): 8 semanas.

5. Sustitución del *VA-file* por el *iDistance* en el framework existente: 2 semanas.

6. Experimentación con el nuevo entorno basado en el *iDistance* y análisis de los resultados obtenidos: 2 semanas

7. Elaboración de la memoria del proyecto: 12 semanas.

2

Fundamentos

2.1 Introducción

Para comenzar, es necesario exponer una serie de conceptos y estructuras básicas para continuar con la exposición del proyecto. Vamos a dividir este capítulo en siete partes: en primer lugar, es necesario definir con qué tipo de datos vamos a trabajar: datos multidimensionales, por tanto está será la primera parte del capítulo de fundamentos. En la segunda parte del capítulo se expondrán las medidas de similitud usadas para identificar sub-secuencias de video y a continuación algunas distancias utilizadas en el proyecto con los datos multidimensionales. El siguiente apartado trata de *la maldición de la dimensionalidad* (dimensionality curse) y sus efectos en los datos multidimensionales y se mencionarán algunas de las técnicas utilizadas para solventarla: en especial la reducción de la dimensionalidad con *iDistance*. A continuación, se expondrá la técnica de clustering utilizada en el proyecto con datos multidimensionales para obtener centros: *kmeans*. Para terminar, se explicara las consultas basadas en distancias utilizadas para la realización del proyecto. Se introducirá una breve descripción de grafos junto con algoritmos y teoremas utilizados en este proyecto para la recuperación de video basada en contenido y por último una descripción del *árbol B+*, en el cual está basado el *iDistance*: la estructura de datos que nos ocupa en este proyecto.

2.2 Datos multidimensionales

En el contexto del proyecto vamos a trabajar con datos multidimensionales, o bien, *dim-dimensionales*, esto es: vectores de dimensión *dim*. Para seguir una estructura definida y probada vamos a utilizar las definiciones descritas en [Riv10] sobre datos *dim-dimensionales*:

Definición. Espacio *dim*-dimensional de datos (D^{dim}).

Se define el espacio *dim*-dimensional de datos (espacio multidimensional) D^{dim} como:

$$D^{dim}=D_0 \times D_1 \times D_2 \times \ldots \times D_{dim-1}=\{(p_i) \text{ donde } 0 \leq i \leq dim-1 \text{ y } p_i \in D^{dim}\}$$

Donde cada dominio D_i está formado por un conjunto de valores, sobre los que está definida una relación de orden $\leq_i$ ($D_i \subseteq \mathbb{R}$), que salvo indicación explícita, se denotará como $\leq$ para cualquier $1 \leq i \leq n$. Además, dichos dominios no tienen que ser necesariamente distintos. Por ejemplo, $D^{dim} \subseteq \mathbb{R}^{dim}$.

Definición. Espacio métrico *dim*-dimensional (M^{dim}).

Se dice que M^{dim} es un espacio métrico *dim*-dimensional de datos, si es un espacio *dim*-dimensional de datos D^{dim}, con una métrica o distancia (dist) asociada a cada par de elementos de D^{dim}.

$$M^{dim} = (D^{dim}, dist)$$

Por ejemplo, el espacio Euclídeo *dim*-dimensional, E^{dim}, está formado por el espacio de datos $\subseteq \mathbb{R}^{dim}$, a cuyos datos se les puede aplicar la distancia Euclídea. Es decir, E^{dim} = ($\mathbb{R}^{dim}$, distEuclídea).

Una vez definidos el espacio *dim*-dimensional, vamos a ver que es un punto *dim*-dimensional que son con los que vamos a tratar en el ámbito del proyecto y por último veremos la manera de guardar dichos puntos en un archivo: el archivo *dim*-dimensional.

Definición. Punto *dim*-dimensional (p).

Se define un punto *dim*-dimensional $p \in D^{dim}$, como un conjunto finito de *dim* valores de la forma:

$$p = (p_0, p_1, p_2, \dots, p_{dim-1}) = \{(p_i)_{0\leq i\leq dim-1} \in D^{dim}(\subseteq \mathbb{R}^{dim}), \text{ donde } p_i \in D_i\ (p_i \in \mathbb{R}^{dim})$$

Definición. Archivo *dim*-dimensional de datos F^{dim}.

Un archivo *dim*-dimensional de datos (también denominado, base de datos multidimensional en [BBK+01] es una colección o conjunto de n puntos *dim*-dimensionales, caracterizados cada uno de ellos por un conjunto de *dim* atributos, asociados cada atributo a un dominio concreto (que puede ser $\mathbb{R}$.

$$F^{dim}=P = \{P_0, P_1, \cdots, P_{n-1}\} = \{(P_i)_{0\leq i\leq n-1} \in D^{dim}(\subseteq \mathbb{R}^{dim})\}.$$

Es decir, bajo una nomenclatura similar, cada punto *dim*-dimensional $P_i \in P$ es de la forma $P_i = (p_{i0}, p_{i1}, \dots, p_{ij}, \dots, p_{idim-2}, p_{idim-1})$ tal que $0\leq i\leq n-1$ (número de puntos en P) y $0\leq j\leq dim-1$ (número de dimensiones en D^{dim}).

2.3 Medidas de similitud

En los sistemas de recuperación de video, imágenes o música las bases de datos tratan con vectores de características o puntos multidimensionales en el que cada dimensión representa un atributo del objeto a representar. Cuando queremos comparar objetos o bien para indexación en la base de datos o bien para recuperar objetos más similares entre sí, como por ejemplo: en una consulta entre un objeto dado como referencia y la base de datos para encontrar los objetos más parecidos al dado. Necesitamos una medida de similitud que nos proporcione información de cuanto es de próximo en el espacio M^{dim} un vector de otro.

Podemos concluir que una buena medida de similitud es calcular la distancia entre dos vectores (X, Y) en el espacio multidimensional. Vamos entonces a exponer una serie de métricas sobre datos multidimensionales más usadas para calcular distancias tradicionales y luego se expondrá una serie de distancias relativas a la recuperación de video e imágenes en bases de datos multidimensionales.

2.3.1 Distancias comunes

En el cálculo de distancias tradicionales sobre datos multidimensionales vamos a basarnos en [KCB03] y [Riv10] que exponen las distancias más utilizadas entre vectores. La métrica de distancia se puede denominar como medida de similitud, que es el componente clave en la recuperación de imágenes basada en contenido. Es importante explorar las diferentes medidas de similitud para intentar encontrar la mejor distancia métrica para la recuperación de contenido de la imagen base. La imagen de la consulta será más similar a las imágenes de base de datos si la distancia es menor. Sean x e y dos vectores *dim*-dimensionales de características, de la base de datos y de consulta respectivamente, sea dist(X, Y) la distancia entre los dos puntos citados anteriormente y siendo sus datos reales ($D^{dim} = \mathbb{R}^{dim}$). La dist(X, Y) debe cumplir las siguientes propiedades:

1. Positividad: dist(X, Y) ≥ 0
2. Identidad: dist(X, Y) = 0 si y solo si X=Y
3. Simetría: dist(X, Y) = dist(Y, X)
4. Desigualdad Triangular: Sea Z ϵ D^{dim} dist(X, Y) = dist(X, Z) + dist(Z, Y)

Cumpliendo las cuatro reglas básicas anteriores podemos definir las siguientes distancias:

- Mediante la distancia Minkowski se pueden obtener dos de las distancias más usadas: la Euclidea y la Máxima:

$$Distancia\ Minkowski: d_t(x,y) = \sqrt[t]{\sum_{i=0}^{dim-1} |X_i - Y_i|^t}$$

$$Distancia\ Euclidea\ (t=2): d_E(x,y) = \sqrt{\sum_{i=0}^{dim-1} (|X_i - Y_i|)^2}$$

$$Distancia\ Máxima\ (t=\infty) = Max_{0 \le i < \dim} |X_i - Y_i|$$

- Existen otros tipos de distancias como la distancia Mahalanobis:

$$dist_{Mah}(X,Y) = \sqrt{(X-Y)' Cov(X)^{-1} (X-Y)}$$

- Distancia Weighted-Mean-Variance:

$$dist(X,Y)_{Wmv} = \sum_{i=0}^{m} \sum_{j=0}^{n} dist_{ij}(X,Y)$$

$$dist_{ij}(X,Y) = \left| \frac{\mu_{ij}^x - \mu_{ij}^y}{\sigma(\mu_{ij})} \right| + \left| \frac{\sigma_{ij}^x - \sigma_{ij}^y}{\sigma(\mu_{ij})} \right|$$

Donde $\sigma(\mu_{ij})$ son las desviaciones estándar de las respectivas funciones en la base de datos, y se utilizan para normalizar la función de los componentes individuales, que incrementan la resistencia y mejoran el rendimiento de recuperación de objetos ya sean imágenes o videos.

- Distancia Canberra:

$$dist_C(X,Y) = \sum_{i=0}^{dim-1} \frac{|X_i - Y_i|}{|X_i| + |Y_i|}$$

- Distancia Bray-Curtis:

$$dist_{BC}(X,Y) = \sum_{i=0}^{dim-1} \frac{|X_i - Y_i|}{X_i + Y_i}$$

- Distancia Squared Chord:

$$dist_{SC}(X,Y) = \sum_{i=0}^{dim-1} (\sqrt{X_i} - \sqrt{Y_i})^2$$

- Distancia Chi-Squared:

$$dist_{Chi}(X,Y) = \sum_{i=0}^{dim-1} \frac{(X_i - Y_i)^2}{X_i + Y_i}$$

2.3.2 Similitud entre datos de video e imágenes

A continuación ilustraremos las similitudes para la comparación de *key*-frames. Un *key*-frame es una imagen que caracteriza a una secuencia de frames de una parte de un video determinado. De tal manera que un *key*-frame representa una secuencia de determinada duración en un video. De manera formal, si consideramos dos videos Q y S definidos como Q = $\{q_0, q_1, \ldots, q_{Q-1}\}$ y S = $\{s_0, s_1, \ldots, s_{S-1}\}$, donde $q_i = \{q_{i0}, q_{i1}, \ldots, q_{idim-1}\} \in Q$ y $s_j = \{s_{j0}, s_{j1}, \ldots, s_{jdim-1}\} \in S$ son vectores de características *dim*-dimensionales, y |Q| y |S| el número total de frames (*key-frames*) de Q y S, respectivamente. Podemos identificar si S es similar a Q, normalmente por cada $q_i \in Q$, se ejecuta una operación de búsqueda en S para recuperar los vectores de características *similares* a q_i. Una medida de similitud típica entre videos es el porcentaje de vectores de características similares, compartidos por las dos secuencias de video [ChZ03]. Es decir, dados Q y S, su similitud, Sim(Q,S), se podría definir como la siguiente fórmula:

$$Sim(Q,S) = \frac{\sum_{i=0}^{|Q|} 1_{\{s_j \in S, dist(q_i,s_j) \leq \varepsilon} + \sum_{j=0}^{|S|} 1_{\{q_i \in Q, dist(q_i,s_j) \leq \varepsilon}}{|Q| + |S|}$$

Nótese que dist(q_i, s_j) es la distancia utilizada entre dos puntos *dim*-dimensionales. Se puede utilizar cualquiera de las distancias mencionadas anteriormente aunque se debe prestar especial atención que normalmente las diferentes dimensiones de los puntos multidimensionales están normalizadas entre 0 y 1 y que las distancias deben quedar también entre dicho intervalo, prestando mayor atención en el algoritmo *VSS* para la recuperación secuencial de video (capítulo 5). Dos *key-frames* son similares si las distancias de sus puntos *dim*-dimensionales no superan un ε dado, como se pude ver en la formula anterior. Dicha cota depende del tipo de secuencias a recuperar y de la base de datos: precisión y eficiencia.

En [Riv10] se exponen las diferentes medidas de similitud utilizadas en la recuperación de video en este proyecto:

1. Mean Distance (normalized pairwise distance)
2. Dynamic Time Warping (DTW)
3. Longest Common Subsequence (LCSS)
4. Edit distance y sus extensiones con pesos como Edit distance with Real Penalty (ERP), o Probability-based Edit distance (PED)

Las más utilizadas en similitud de secuencias de video (o video matching) son *Edit distance*, *Dynamic time warping* (alineación temporal dinámica) y *Longest common subsequence distance*.

2.4 Consultas basadas en distancias

Tanto en [Riv10] como en este proyecto se trabaja con bases de datos multidimensionales para realizar recuperación de datos basados en contenido. Dicha recuperación se llevará a cabo mediante comparaciones basadas en distancias, como las expuestas anteriormente. Dichas consultas (basadas en distancias) se pueden dividir en dos grupos o categorías: por un lado están las que sólo se utiliza un conjunto de puntos en E^{dim} y por otro lado están las que utilizan dos conjuntos de puntos del mismo espacio. En las consultas basadas en distancias que sólo utilizan un conjunto de puntos tenemos la consulta en rango de distancia y la consulta de los K vecinos más próximos. La consulta de los K pares más cercanos y el join de similitud (*Similarity join*) necesitan dos conjuntos de puntos en E^{dim}. En el proyecto que nos ocupa, nos centraremos en la recuperación de video basada en contenido por tanto solo se trabajarán las consultas con un conjunto de puntos E^{dim}:

- **Consulta en rango de distancia:** Dado un conjunto de puntos S de E^{dim}, un punto de consulta *q* y una distancia $\delta \geq 0$, esta consulta devolverá todos los objetos que estén dentro de la hiperesfera con centro en el punto de consulta y radio δ, tal y como muestra la Figura 2.

$$\boldsymbol{DRQ(S, q, \delta) = \{s_i \in S: dist\ (s_i, q) \leq \delta\ \}}$$

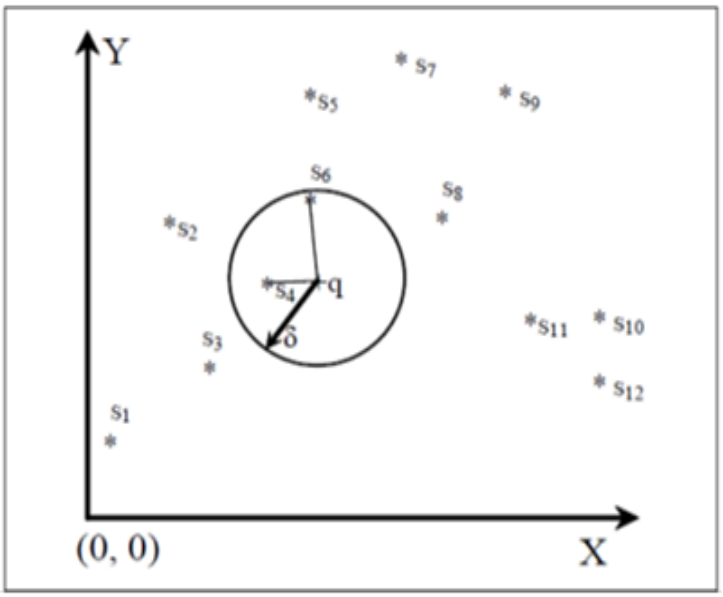


Figura 2. [Riv10] Consulta en rango de distancia en un espacio bidimensional.

- **Consulta del vecino más próximo (NN):** Dado un conjunto de puntos S de E^{dim} y un punto de consulta q, esta consulta encuentra el punto cuya distancia al punto de consulta es menor. Una extensión de esta consulta es la denominada **consulta de los K vecinos más próximos (k-NN)**, que dado un conjunto de puntos S de E^{dim}, un entero positivo K y un punto de consulta q, devolverá los K puntos más próximos a dicho punto, ordenados en orden creciente de distancias.

$$KNN(S,q,K)=\{ (s_1, s_2, ..., s_K) \in S:$$
$$dist\ (s_1, q) \leq dist\ (s_2, q) \leq \ldots \leq dist\ (s_K, q) \leq dist\ (s_i, q) : (K \leq I \leq |S|\ ) \}$$

Veamos un ejemplo en la Figura 3 de k-NN con k=3: kNN(S, q, 3)= {s_8, s_6, s_{11}}

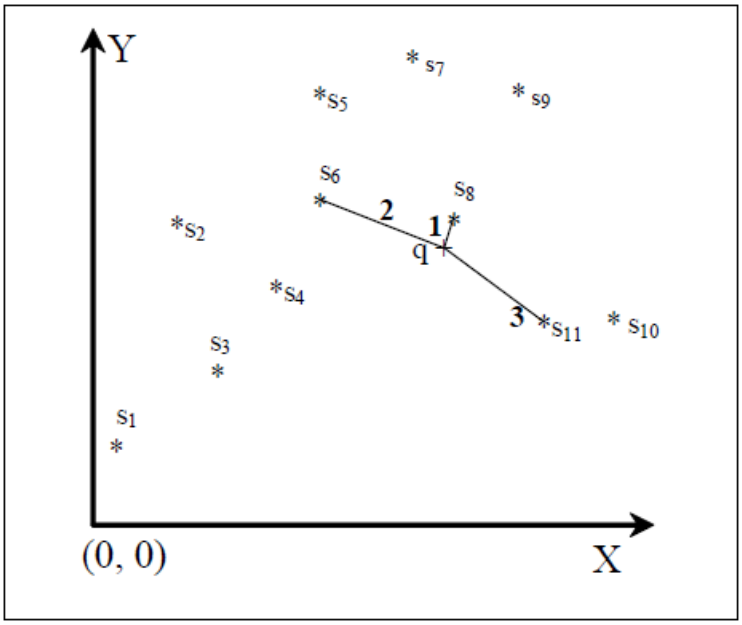


Figura 3. [Riv10] Ejemplo de consulta del vecino más próximo en un espacio bidimensional.

2.5 Técnicas para evitar la maldición de la dimensionalidad

Tal y como se expone en [Riv10] en las bases de datos multidimensionales hay un problema de rendimiento conforme se aumenta la dimensión de sus vectores y es que para dimensiones muy grandes numerosos estudios han demostrado que la indexación tiene peor comportamiento que un tratamiento secuencial de los datos. A dicho fenómeno se le denomina *maldición de la dimensionalidad (dimensionality curse)*.

Cuando se tienen bases de datos de video e imágenes basadas en contenido, es decir, en vectores de características *dim*-dimensionales, aunque estos vectores estén correctamente indexados, conforme aumenta su dimensión los puntos tienden a estar equidistantes entre ellos en una esfera *dim*-dimensional alrededor de un punto dado como ejemplo (punto de consulta), independientemente de donde se encuentre dicho punto. Es decir, el radio de la esfera aumenta mientras que su anchura permanece inalterada, y el espacio alrededor del punto de consulta está vacío. Esto hace cada vez más improbable (al aumentar *dim*) que una división previa de los datos pueda ser utilizada de manera eficiente.

En este proyecto se nos plantea este problema: como intentar solucionar la maldición de la dimensionalidad de una forma eficiente para poder realizar consultas de video basadas en contenido a posteriori. Algunas de las técnicas que tratan de atenuar dicho problema, son expuestas en [Riv10]:

1. Reducción de la dimensionalidad [AKS98]
2. Aproximación al vecino más próximo [ASR$^+$98, AMN$^+$98]
3. Múltiples curvas de llenado del espacio [MeS97]
4. Basadas en filtros o vectores de aproximación [WSB98, FTA$^+$00, CCP$^+$02]
5. Enfoque híbrido [ChC02, BBJ$^+$00]

No nos vamos a detener en detallar cada una de ellas puesto que ya se hace en [Riv10] si no que vamos a tratar las técnicas que nos ocupa en este proyecto que es: *la reducción de la dimensionalidad*: como el *iDistance* y la basada en filtros y vectores de aproximación (*VA-File*).

2.5.1 Reducción de la dimensionalidad

La técnica de la reducción de la dimensionalidad (*Dimensionality Reduction*) realiza una *condensación* de los datos en unas pocas dimensiones aplicando un valor único de descomposición. Después, los datos *condensados* en unas pocas dimensiones pueden ser indexados. Este enfoque presenta los siguientes inconvenientes:

- La reducción de la dimensionalidad está acompañada de una pérdida de precisión en los resultados de las consultas.

- No es fácil su aplicación en bases de datos dinámicas, ya que el valor único de descomposición debe de ser calculado a priori sobre la base de datos completa y dicho cálculo es muy costoso.

- Esta técnica funciona bien sólo cuando los datos están fuertemente correlacionados.

Aun así, la reducción de la dimensionalidad es muy utilizada en bases de datos multidimensionales, **centrándonos en este proyecto en la técnica *iDistance* [JOT+05].** Aunque existen otras técnicas de reducción de la dimensionalidad como el TV-Tree (*Telescopic Vector Tree*) [LJF94], P-Sphere Tree [GRa00], etc.

El (*Telescopic Vector Tree*) TV-Tree (Figura 4) fue el primero en abordar el problema de indexación de datos altamente dimensionales en el contexto de video e imágenes, en general, en bases de datos basadas en métricas temporales. Se basa en un índice especialmente diseñado para vectores multidimensionales donde la importancia de las dimensiones está ordenada. Las características de estos vectores están representadas por una varianza alta en las primeras dimensiones y una varianza baja en las últimas.

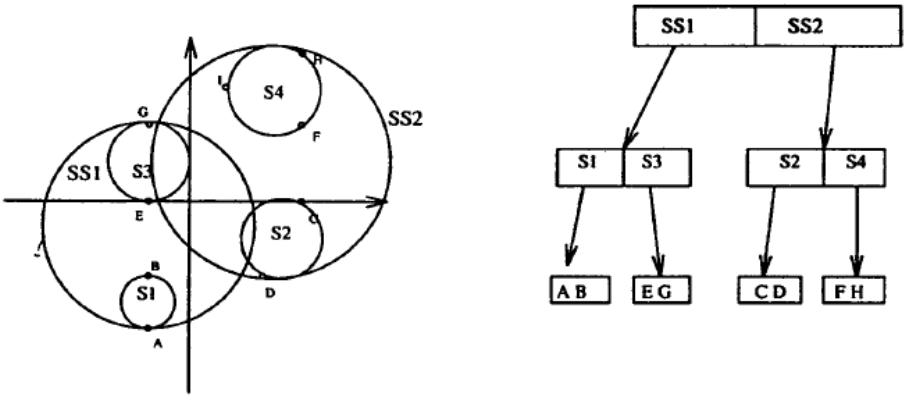


Figura 4. [LJF94] Ejemplo de un TV-2-Tree con esferas de dos dimensiones (datos bidimensionales)

El P-Sphere Tree (*Probabilistic Sphere Tree*) trabaja bien sobre bases de datos estáticas y sus respuestas tienen una exactitud asignada previa. Logra su eficiencia mediante la duplicación de puntos de datos en clústeres de datos basado en una serie de consultas de muestra. Podemos definir el P-Sphere Tree como una estructura de índices de dos niveles: el nodo raíz contiene la descripción de cada esfera en el siguiente nivel y apuntadores que apuntan a cada esfera de los nodos hoja. Cada nodo hoja contiene todos los puntos de datos que se encuentran en el ámbito descrito en el descriptor correspondiente del nodo raíz. Un punto interesante a destacar es que cada nodo hoja no sólo cubre el mismo número de puntos de datos, sino también todos los puntos que se encuentran dentro de una partición esférica del espacio de datos.

Las esferas pueden superponerse, y los puntos de datos pueden aparecer en varios nodos de la hoja. Además, no hay garantía de que todos los puntos de datos del conjunto de datos original sean indexados.

Para crear un árbol P-Sphere, la capacidad de la raíz, el centro de gravedad de cada ámbito y el tamaño de la hoja deben ser predeterminados. Los centroides se generan mediante un muestreo aleatorio del conjunto de datos y su distribución sigue la distribución de los datos. El tamaño de la hoja se determina en base a los puntos de consulta de ejemplo y el usuario deberá especificar la precisión, expansiones, centroides y el tamaño de la hoja L.
En general, esta estructura no es adaptable para según qué distribución de datos se use, por lo tanto, tienden a funcionar bien para algunos conjuntos de datos y mal para otros.

En el proyecto que nos ocupa, vamos a describir detalladamente el *iDistance* como una técnica adaptativa en bases de datos multidimensionales para la reducción de la dimensionalidad y posteriormente vamos a introducirla en un sistema de recuperación de video basada en contenido para comprobar su eficiencia. Tras comprobar su funcionalidad vamos a comparar dicho sistema con otro sistema de recuperación de video basada en contenido con otra técnica para evitar la maldición de la dimensionalidad: basada en filtros y vectores de aproximación, utilizando la estructura de datos usada en [Riv10]: el *VA-File*.

2.5.2 Técnica basada en filtros y vectores de aproximación

Esta técnica nos va a servir para comparar el *iDistance* (reducción de la dimensionalidad) con otra estructura de datos distinta como es el *VA-File* usada en [Riv10]. La técnica basada en filtros y aproximaciones está basada en el filtrado de los vectores de características (dim-dimensionales), de tal manera que solo una pequeña parte de ellos debe ser visitada durante la búsqueda. Entre los enfoques que intentan vencer la maldición de la dimensionalidad, éste y el *iDistance* son los únicos que permiten obtener los k-NNs de manera exacta.

El *VA-file* divide el espacio de datos en 2^b celdas rectangulares siendo *b* un número de bits especificado por el usuario para la codificación de las celdas. En vez de utilizar una estructura jerárquica como los índices multidimensionales convencionales, el *VA-file* asocia una cadena única de bits de longitud *b* a cada celda y realiza una aproximación de los puntos de datos contenidos en dicha celda mediante la citada cadena de bits. El *VA-file* es simplemente un array con todas estas cadenas de bits que representan (aproximan) a los puntos multidimensionales de datos. En la Figura 5, podemos observar como el *VA-file* divide el espacio de datos y como obtiene las aproximaciones (compresión) de los puntos es base a las celdas resultantes de la división.

Su funcionamiento a la hora de procesar una consulta de k-NN consiste en descartar todos los vectores que no son necesarios mediante el recorrido del array de cadenas (aproximaciones) y así centrar la búsqueda real en un número muy reducido de vectores reales.

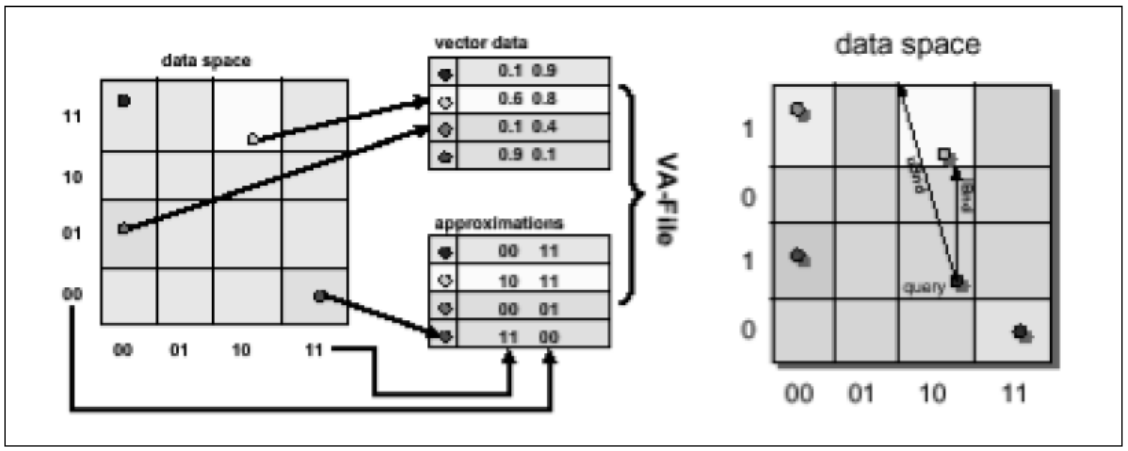


Figura 5. [Riv10] Ejemplo de aproximación del *VA-File.*

Como se puede desprender de lo comentado, el rendimiento depende en gran medida de la potencia del proceso de filtrado (asignación de las cadenas de bits) y éste a su vez en la precisión de las aproximaciones.

En [Riv10] se presenta también un procesamiento por lotes de la consulta de los K vecinos más cercanos con el *VA-File* la cual reduce considerablemente el número de accesos a disco con respecto al procesamiento de la misma consulta sin dicha mejora. En este proyecto se utilizará el *VA-File* con y sin el procesamiento por lotes *(BNN, Batch Nearest Neighbours)* para la comparación del rendimiento del *iDistance* y el *VA-File*.

2.6 Clustering de datos: Obtención de centros altamente dimensionales

Este proyecto se propone estudiar el rendimiento de una técnica de reducción de la dimensionalidad llamada *iDistance* en la consulta de video basada en contenido. Para indexar una secuencia de puntos altamente dimensional, el *iDistance* necesita saber previamente cuáles son sus centros mediante algún algoritmo de clustering, que se puede definir de la siguiente manera:

Sea S = {s_0, s_1… s_{S-1}}, donde s_j = {s_{j0}, s_{j1}… s_{jdim-1}} ∈ S, se trata de encontrar K puntos en S que representen los centros dim-dimensionales de K esferas dim-dimensionales que engloben toda la base de datos S.

Para ello se ha utilizado un algoritmo eficiente de clustering: Lloyd´s *kmeans* [KMN+02]. Que dado un numero n de puntos *dim*-dimensionales en R ($\mathbb{R}^{dim}$) y un entero K determina un conjunto de puntos Z con |Z|=K llamados centros, con el fin de minimizar el error cuadrático medio entre cada punto de datos a su centro más cercano.

Por lo tanto dicho algoritmo lo que pretende es solucionar el problema geométrico *K-centros* [KMN+02] en la que el objetivo es minimizar la distancia máxima desde todos los puntos a su centro más cercano. Por desgracia no hay soluciones eficientes conocidas para dicho problema y algunos son NP-completos. Uno de los algoritmos heurísticos más popular para resolver el problema de k-medias (*kmeans*) se basa en un sencillo esquema iterativo para encontrar una solución a nivel local mínimo. Este algoritmo se conoce como el algoritmo de k-means. Hay una serie de variantes de este algoritmo, por lo que, para aclarar la versión que estamos usando, se le llamará: *algoritmo de Lloyd.*

Algoritmo de Lloyd se basa en la simple observación de que la ubicación óptima de un centro está en el centroide[1] (centro de masas o lugar con menor error cuadrático medio) del grupo asociado. Dado cualquier conjunto de *k* centros *Z*, para cada centro $z \in Z$, vamos a llamar *V(z)* denotan su *entorno (neighborhood)*, es decir, el conjunto de puntos de datos para que el centro *z* es el vecino más cercano. Por tanto cada iteración s del algoritmo de Lloyd mueve cada punto central *z* al centroide de *V(z)* y actualiza *V(z)* volviendo a calcular la distancia de cada punto a su centro más cercano. Estos pasos se repiten hasta que alguna condición de convergencia se cumple.

Debido a su simplicidad y flexibilidad, el algoritmo de Lloyd es muy popular en el análisis estadístico. En particular, teniendo en cuenta cualquier otro algoritmo de clustering, el algoritmo de Lloyd se puede aplicar como una etapa de post-procesamiento para mejorar la distorsión final que como se demuestra en los resultados experimentales de [KMN+02] pueden darse mejoras significativas. Sin embargo, una implementación directa del algoritmo de Lloyd puede ser bastante lento. Esto se debe principalmente al costo de la computación de los vecinos más cercanos.

La implementación del algoritmo de Lloyd [KMNS_IMPL] viene dada mediante la técnica de filtrado propuesta en [KMN+02] el cual se inicia mediante el almacenamiento de los puntos *dim*-dimensionales en un árbol KD [KMN+02 y Ben75]. Hay que recordar que, en cada etapa del algoritmo de Lloyd, el centro más cercano a cada punto de datos y se recalcula para que cada centro se mueva al centro de gravedad (centroide)[1] de los vecinos asociados. La idea es mantener, para cada nodo del árbol, un subconjunto de centros candidatos. Los candidatos para cada nodo se podan (filtran), ya que se propagan a los nodos hijo de dicho nodo. Ya que el árbol KD se calcula para los puntos de datos en lugar de para centros, no hay necesidad de actualizar la estructura dentro de cada etapa del algoritmo de Lloyd. El pseudocódigo del algoritmo se puede resumir en que, en cada iteración mueve cada centro escogido al centroide (o centro de masas[2]).

---

[1] ***Centro de gravedad* o *Centroide*:** Cuando se habla del centroide de un conjunto de datos (o puntos multidimensionales), este viene referido a la media ponderada de dicho conjunto la cual obtiene un valor que es el que mejor (optimo) representa o engloba a dicho conjunto. Si quisiéramos crear una esfera que englobase de manera eficiente y con el menor radio a todo el conjunto de datos a tratar, el centro óptimo de dicha esfera será el centro de gravedad o centroide de dicho conjunto de datos.

[2] ***Centro de masas*:** Similar a centro de gravedad (centroide) de un conjunto de puntos o datos. Cuando calculamos la media ponderada (ver[1]) podemos asociar diferentes *masas* o pesos a los distintos puntos del conjunto de datos. De esta manera el centro de masas o centroide no se sitúa en el centro *geográfico* de la nube de puntos, sino que tiende a situarse cerca de los puntos con mayor *masa* o peso.

El cálculo de la distorsión solo se hace si es necesario para calcular los pesos y las sumas de cada centro *dim*-dimensional, es decir, solo es necesario hacerlo en la primera iteración. Por tanto un centroide viene definido (*weighted*) como la ponderación de las sumas de los pesos (*weight[j]*) de sus vecinos, tal que: *ctrs[j] = sums[j] / weights[j]*.

Por lo general, se permite un factor de amortiguamiento (*dampFactor*) en el movimiento de cada centro a su centroide el cual está definido entre 0 (muy amortiguado) y 1 (sin amortiguación). Teniendo en cuenta este factor de amortiguamiento *dampFactor* podemos definir la siguiente formula:

$$ctrs[j] = (1\text{-}dampFactor) * ctrs[j] + dampFactor * sums[j]/ weights[j]$$

la cual nos dará los mejores centros en *MAX_ITERACIONES*.

El pseudocódigo del algoritmo completo es el siguiente:

```
kmeans_lloyd_algoritm(MAX_ITERACIONES, DATA, Dimension, nCentros)

1.  Mientras iteracion<MAX_ITERACIONES
2.    Si es la primera iteracion
3.      CalcularDistorsion(Dimension, DATA, nCentros)
4.    Fin Si
5.
6.    Desde j=0 hasta j<nCentros hacer
7.      wgt = pesos del centro
8.      Si (wgt > 0)
9.        Desde d = 0 hasta d < Dimension hacer {
10.         Centros[j][d] = (1 - dampFactor) * ctrs[j][d] + dampFactor *
    sums[j][d]/wgt;
11.       Fin Desde
12.     Fin Si
13. Fin Desde

14. Si el cluster actual tiene menos distorsion
15.   Hacer mejor = actual
16.   Si mejoran
17.     Guardar como mejores.
18.   Fin Si
19.   Iteración=iteración+1
20. Fin Si

21. Devolver mejor
```

Código 1. Pseudocódigo del algoritmo kmeans de Lloyd.

El cálculo de la distorsión inicial se hace para calcular el total de distorsiones individuales del conjunto de puntos. Para ello se recuperarán los vecinos del conjunto de centros a tratar mediante el árbol KD (KD-Tree) comentado anteriormente el cual calculará los pesos de dichos centros para calcular la distorsión a posteriori.

Para el cálculo de la distorsión individual supongamos que algún centro se ha fijado (indexados por *j* en el código de abajo). Si Sum_i denota la suma de todos *(weight[j])* los vecinos del centro dado. Los puntos de datos *(p[i])* y los puntos del centro (*c[j]*) son vectores, y el producto de dos vectores: el producto escalar $u*v = (u.v)$, $u^2 = (u.u)$. La distorsión de un solo centro *j*, denotado *dists[j]*, se define como la suma de los cuadrados de las distancias de cada punto a su centro más cercano, es decir:

$$dists[j] = Sum_i\ (p[i] - c[j])^2 =$$

$$Sum_i\ (p[i]^2 - 2*c[j]*p[i] + c[j]^2) =$$

$$Sum_i\ p[i]^2 - 2*c[j]*Sum_i\ p[i] + wgt[j]*c[j]^2 =$$

$$sumSqs[j] - 2*(c[j].sums[j]) + wgt[j]*(c[j]^2)$$

Por lo tanto la distorsión individual puede ser calculada a partir de estas cantidades. La distorsión total es la suma de las distorsiones individuales.

El pseudocódigo del cálculo de distorsiones es el siguiente:

```
CalcularDistorsion(Dimension, DATA, nCenters) {

1.  Crear un KD-Tree con DATA.
2.  Obtener los vecinos de los centros actuales en el árbol.
3.  DistorsionTotal = 0
4.  Desde j = 0 hasta j < nCenters
5.      cDotC = 0;          // inicializo el resultado: (c[j] . c[j])
6.      cDotS = 0;          // inicializo el resultado: (c[j] . sum[j])
7.      Desde d = 0 hasta d < Dimension
8.          cDotC += ctrs[j][d] * ctrs[j][d];
9.          cDotS += ctrs[j][d] * sums[j][d];
10.     Fin Desde
11.     dists[j] = sumSqs[j] - 2*cDotS + weights[j]*cDotC;
12.     DistorsionTotal += dists[j];
13. Fin Desde
14. Guardar la distorsión total.
```

Código 2. Pseudocódigo del cálculo de la distorsión inicial de los datos.

En la Figura 6 se muestra un ejemplo del algoritmo de aplicación del algoritmo de Lloyd con filtrado, para una distribución de puntos bidimensionales y 9 centros (K=9):

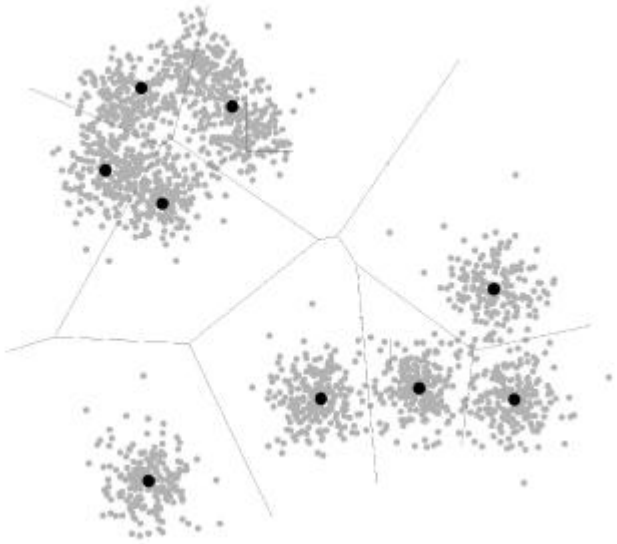


Figura 6. [KMN+02] Ejemplo de clustering de datos utilizando el algoritmo de Lloyd para calcular los k centros del espacio de datos.

Puesto que la obtención de los centros no es trivial, que en el proyecto se va a trabajar con archivos *dim*-dimensionales F^{dim} de aproximadamente 125000 vectores y mínimo clústeres de 16 centros, siendo el valor más usado el de 64 centros y el rendimiento del *iDistance* depende en gran medida del número de centros y lo buenos que sean estos. Se ha decidido en base a los resultados experimentales en [KMN+02] utilizar el algoritmo de Lloyd con filtrado para el cálculo de los centros en la indexación de datos *dim*-dimensionales y se ha seguido la implementación de [KMNS_IMPL].

En cuanto a los fundamentos del árbol KD-Tree no vamos a entrar en ellos por salirse del ámbito del proyecto tanto en cuanto a quedar fuera del estudio del comportamiento del *iDistance* en la recuperación de video basada en contenido.

2.7 Fundamentos del iDistance y del framework de recuperación de video (CVBR) utilizado.

A continuación vamos a describir el funcionamiento y estructura de un árbol B+ en el cual se basa la implementación del *iDistance* y para concluir con los fundamentos de este proyecto terminaremos con algunas consideraciones sobre grafos a tener en cuenta a la hora de estudiar el matching que realiza el sistema de video basado en contenido.

2.7.1 Fundamentos del núcleo del iDistance: Árbol B+

La implementación que nos ocupa para la reducción de la dimensionalidad hace uso en su núcleo de un árbol B+ adaptativo que se detallará en el siguiente capítulo. En esta sección vamos a describir muy brevemente los fundamentos básicos de los arboles B+ (familia de los árboles B).

Los registros de una base de datos altamente dimensional se almacenan en memoria secundaria para proteger la integridad de la base de datos frente a fallos hardware y software, dicho almacenamiento en nuestro caso será un archivo *dim*-dimensional: F^{dim}.

Si no ordenásemos el archivo F^{dim} la búsqueda en cada consulta solo podría ser secuencial en páginas de disco, puesto que normalmente las páginas que conforman a F^{dim} están contiguas en disco. Para evitar dicha búsqueda secuencial se pueden utilizar una estructura de datos la cual creen un archivo ordenado F_{ord}^{dim} en el cual se pueda realizar una búsqueda binaria en las páginas del disco siguiendo los criterios para mantener el archivo ordenado. Y puesto que el tiempo de recuperación de la memoria externa (disco, por ejemplo) es miles de veces mayor que la memoria principal. El objetivo en la búsqueda externa es reducir al mínimo el número de accesos al disco. Por esta razón, el acceso típico de un registro de memoria externa se lee una página entera o un bloque de datos a la vez. Al ir a buscar una página de disco que contiene muchos registros, nos aprovechamos de la localidad de referencia: si un registro ha sido recuperado y hemos organizado los registros en función de cómo se accede a ellos, entonces es probable que los registros que busquemos en la siguiente consulta estén en el mismo bloque. Como en un sistema de recuperación de video basada en contenido es necesario realizar búsquedas en archivos *dim*-dimensionales en los cuales se puede seguir un orden de inserción y búsqueda, el árbol B+ es una excelente apuesta para ser la base de una estructura de indexación altamente dimensional como el *iDistance* puesto que permite la búsqueda binaria y además mantiene una lista enlazada con las paginas en los que se encuentran cada bloque de registros o puntos (en nuestro caso *dim*-dimensionales).

El **árbol B+** forma parte de una familia de árboles de búsqueda multidireccional. Estos árboles fueron propuestos por primera vez por Bayer y McCreight en 1972 [BaC72] y en pocos años había remplazado a casi todos los métodos de acceso a los archivos que no sean por hashing. Las ventajas de un árbol B+ son las siguientes:

- Están siempre equilibrados en altura, con todos los nodos de la hoja en el mismo nivel o altura.
- Actualización y operaciones de búsqueda solo afectan a unas pocas páginas de disco, por lo que el rendimiento es bueno.
- Mantienen registros relacionados en la misma página de disco, por tanto se aprovecha la localidad de referencia comentada anteriormente.

- Los árboles B+ garantizan que todos los nodos en el árbol se llenan por lo menos un porcentaje mínimo. Esto mejora la eficiencia del espacio al mismo tiempo que reduce el número de accesos a disco que son necesarios durante la búsqueda o la operación de actualización con miles de registros en el árbol.

2.7.1.1 Estructura de un árbol B+

El árbol B $^{+}$ en ciertos aspectos es una generalización de un árbol binario de búsqueda *(Binary Search Tree)*. La principal diferencia es que los nodos de un árbol B+ se apuntan a los nodos de muchos hijos en lugar de estar limitado a sólo dos. Ya que nuestro objetivo es minimizar los accesos a disco cada vez que estamos tratando de localizar a los registros, queremos que la altura del árbol de búsqueda multidireccional sea lo más pequeña posible. Este objetivo se logra haciendo que un nodo pueda tener muchas claves y que la rama del árbol englobe grandes cantidades de nodos.

Por tanto podemos definir el árbol B+ como un árbol de orden m donde cada nodo interno contiene hasta *m* ramas (nodos hijos) y por lo tanto puede almacenar hasta *m-1* valores de clave de búsqueda (en un árbol binario de búsqueda, sólo es necesaria una clave puesto que solo hay dos hijos por nodo interno o la raíz). A *m* también se conoce como el factor de ramificación del árbol:

1. El árbol B+ almacena los registros (o punteros a los registros reales) sólo en los nodos hoja, que se encuentran todos en el mismo nivel en el árbol.

2. Todos los nodos internos, excepto la raíz, tienen entre *m/2* y *m* hijos.

3. La raíz es una hoja o tiene como mínimo dos hijos.

4. Los nodos internos almacenan los valores clave de búsqueda, y sólo se utilizan como marcadores de posición para guiar la búsqueda. El número de valores de clave de búsqueda en cada nodo interno es uno menos que el número de sus hijos no vacíos. Las claves se almacenan en orden no decreciente.

5. Dependiendo del tamaño de un registro, en comparación con el tamaño de la clave, un nodo hoja en un árbol B+ de orden m puede almacenar más o menos que los registros *m*. Normalmente, el almacenamiento se basa en el tamaño de un bloque de disco, el tamaño de un puntero de registro, etc. Los nodos hoja deben referenciar o almacenar páginas que por lo menos tienen que tener los registros suficientes para estar a la mitad de su capacidad.

6. Los nodos hoja están unidos entre sí para formar una lista enlazada. Esto se hace para que los registros se puedan recuperar de forma secuencial sin acceder a la estructura del árbol.

Veamos en la Figura 7, un ejemplo de árbol B+ de orden 3 (m=3):

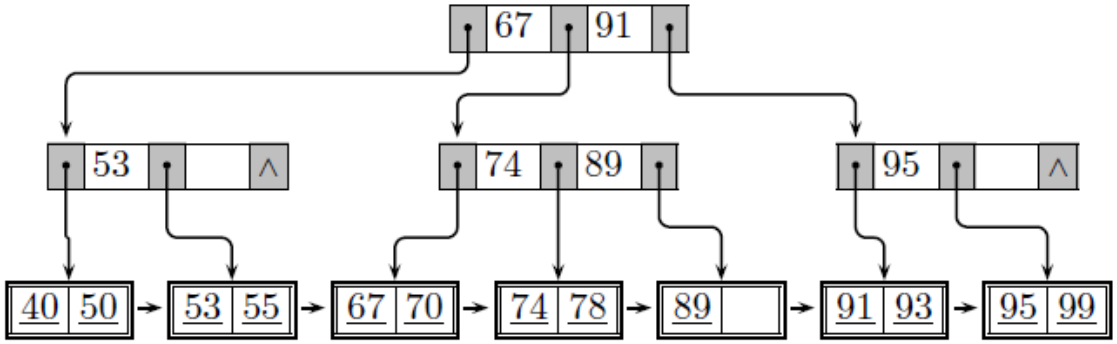


Figura 7. Ejemplo de un árbol B+ de orden 3.

Como se puede observar hay más de una clave por nodo hasta un límite de m-1 claves y m hijos por nodo (no hoja) posibles. En los nodos hoja se mantiene la lista enlazada para un posible procesamiento secuencial de los datos y un puntero al bloque o página de disco.

En la práctica, el tamaño de un nodo en el árbol B+ se elige para llenar una página de disco. En un árbol B+ de altura de 4 con un factor de cargabilidad de 67% que permita entre 100 y 200 hijos tiene un promedio de nodos secundarios de 133, por tanto puede indexar una tabla de más de 300 millones de registros (para ser exactos: $133^4 = 312\ 900\ 700$). Además, es normal que los niveles más altos del B+ se almacenen en cache para agilizar la búsqueda. Los requisitos de memoria no son tan altos: con un tamaño de página típica de 4Kbytes, los dos primeros niveles (hasta 134 páginas de disco) pueden ser cacheados en 1Mbyte. Es por esto que los arboles B+ tienen tanto éxito en la indexación de registros, puesto que en nuestro ejemplo con 4 accesos a disco (número de niveles atravesado más la búsqueda en la página del registro o punto *dim*-dimensional).

2.7.1.2 Operaciones sobre un árbol B+

Búsqueda de un registro con clave K

Buscar en un árbol B+ es una alternancia de dos pasos, comenzando con el nodo raíz. Decir que la búsqueda para el registro con valor de la clave K es única puesto que la clave para cada registro debe de ser única. Se puede definir el proceso de búsqueda en un árbol B+ para una clave de registro K como el siguiente:

1. Realice una búsqueda binaria de los valores clave de búsqueda en el nodo actual: la búsqueda se inicia con la raíz del árbol. Queremos encontrar la clave K_i de tal manera que $K_i \leq K < K_{i+1}$.

2. Si el nodo actual es un nodo interno, seguir la rama asociada con la clave K_i y repita el proceso de búsqueda en el nodo.

3. Si el nodo actual es una hoja, entonces:

- Si K = K_i , entonces el registro existe en la tabla y podemos devolver el registro asociado con K_i
- De lo contrario, K no se encuentra entre los valores clave de búsqueda en la hoja, se informa de que no hay ningún registro en la tabla con el valor de K.

Inserción de un nuevo registro con clave K

Inserción en un árbol B+ es similar a la inserción en los árboles binarios de búsqueda, el nuevo registro siempre se inserta en uno de los nodos de la hoja. La complejidad añadida es que la inserción podría desbordar un nodo hoja que ya está lleno. Los pasos para insertar en un árbol B+ son los siguientes:

1. Determinar el camino que seguiría una búsqueda centrada en la clave K del registro a insertar.
2. La página de hoja de L que se alcanza es el nodo en el que se debe insertar el nuevo registro:
    a. ***Si L no está llena***, entonces una entrada de índice se crea que incluye el valor de la clave de búsqueda de la nueva fila y una referencia al lugar donde la nueva fila se encuentra en el archivo de datos o disco.
    b. ***Si L está llena***, entonces un nuevo nodo de hoja L_{nueva} se introduce en el árbol como un hermano derecho de L. Las claves de L, junto con la entrada de un índice para el nuevo registro se distribuyen uniformemente entre L y L_{nuevo}. L_{nueva} se inserta en la lista enlazada de nodos de la hoja a la derecha de la *L*. El menor valor de la clave de la L_{nueva} se **copia** y se inserta en el padre de *L* (que también será el padre de L_{nueva}). Este paso se conoce como la **división de un nodo hoja**:

        - ***Si el padre P de L está lleno***, entonces se divide a su vez. Para la **división de un nodo interno** se procede de la siguiente manera: los valores clave de búsqueda del padre (P) y la clave recién insertada aún debe ser distribuida equitativamente en P y la nueva página se presenta como un hermano del P. En esta división, sin embargo, la **clave central se desplaza al nodo anterior.** A diferencia de la división de un nodo hoja donde se copia la clave del medio y se inserta en el padre, cuando se divide un nodo interno, la clave central se retira de la división de nodo que se va a dividir y se inserta en el nodo padre. Esta división de los nodos puede continuar hacia arriba hasta la raíz del árbol.

        - ***Cuando una clave se agrega a una raíz completa***, entonces la raíz se divide en dos y la clave central se convierte en la nueva raíz.

Eliminación de un registro con clave K

Lo más importante en la eliminación de una clave es seguir manteniendo el árbol B+ balanceado y que todos los nodos estén como mínimo a la mitad de su capacidad. La complejidad añadida radica cuando tras la eliminación un nodo hoja queda por debajo del tamaño mínimo de su capacidad. Los pasos para eliminar un registro o clave de un árbol B+ son los siguientes:

1. Realizar el proceso de búsqueda de la clave del registro que desea borrar. Esta búsqueda termina en una hoja de L.
2. ***Si la hoja contiene L más que el número mínimo de elementos*** (más de m/2 - 1), entonces la entrada de índice para el registro que se retira puede eliminarse sin problemas de la hoja sin ninguna acción adicional.
3. ***Si la hoja contiene el número mínimo de entradas***, la entrada eliminada se sustituye con otra entrada que puede ocupar su lugar mientras se mantiene el orden correcto. Para encontrar esas entradas, inspeccionamos las dos hojas hermanas: nodos $L_{izquierda}$ y $L_{derecha}$ junto a L:
   1) *Si uno de estos nodos de la hoja tiene más entradas que el número mínimo de entradas*, se transfieren los suficientes registros para que los hermanos queden con el mismo número de registros. Esta técnica heurística se usa para retrasar un posible desbordamiento a 0 del nodo que nos ocupa. Si no bastaría con transferir una entrada de algún hermano. Tras esta operación es necesario verificar que el nodo padre sigue conteniendo el valor de la clave adecuada para referenciar a sus hijos.
   2) *Si $L_{izquierda}$ y $L_{derecha}$ tienen solo el número mínimo de entradas*, los registros de L son introducidos en $L_{izquierda}$ *o* $L_{derecha}$ y L es eliminada del árbol. La nueva página no contendrá más que el número máximo de entradas permitidas. Este proceso de fusión combina dos subárboles de los padres, por lo que la entrada que separa a los padres debe ser eliminada.
   3) *Si los dos últimos hijos de la raíz se funden en un nodo*, este nodo combinado se convierte en la nueva raíz y el árbol pierde un nivel.

Rango de búsqueda en el intervalo de claves I

Sea I un intervalo de claves I= [I_0, I_1,.. I_n], un árbol B+ nos permite buscar todos los registros cuyas claves estén contenidas en dicho intervalo. Para ello por supuesto las claves deben estar definidas en un espacio comparable como por ejemplo $\mathbb{R}^{dim}$ (en nuestro caso). La búsqueda en rango con un intervalo I de claves puede definirse de la siguiente manera:

1. Realizar una búsqueda simple de la clave I_0 del intervalo I. Dicha búsqueda acabará en una página L apuntando al registro con clave I_0.
2. Una vez que el primer registro se ha encontrado utilizando el algoritmo de búsqueda, el resto de los registros incluidos en el rango de búsqueda I se pueden encontrar un procesamiento secuencial de los registros restantes en el nodo hoja, y luego continuar hacia la derecha de la hoja actual nodo ($L \rightarrow L_{derecha}$) de la lista enlazada de nodos de la hoja.

3. Una vez que el desplazamiento se encuentre que tiene un valor clave de búsqueda por encima del límite superior del rango solicitado I_n, la búsqueda finaliza.

Hay que tener en cuenta que para la realización de un rango de búsquedas, sólo podremos realizarla por el tipo clave de registro la cual se ha usado para crear el árbol y además dicha clave debe de ser comparable entre sí.

2.7.2 Fundamentos de un CBVR: Grafos

En los sistemas de video basados en contenido actuales existe la posibilidad de realizar consultas para reconocer subsecuencias las cuales pueden necesitar grafos para realizar el matching de los puntos de consulta con las subsecuencias obtenidas de la base de datos. Por ello es de relevancia destacar algunas propiedades de los grafos a la hora de realizar el matching de video.

Por tanto podemos definir de manera resumida que un grafo es una colección de vértices y de aristas que unen estos vértices. En esta sección presentaremos especial atención a las operaciones de matching [Riv10, Shi04] y grafos bipartitos [Sed02, GrY04].

2.7.2.1 Conceptos básicos sobre grafos

Definición. Un *grafo G no dirigido* se define como un par (V, E) en el que V es un conjunto finito no vacío de elementos denominados vértices o nodos, y E es una relación de V en V ($E \subseteq VxV$), que da lugar a un conjunto de pares desordenados de elementos distintos denominados aristas o arcos. Es decir, una arista e $\in$ E tiene la forma del par (u, v) de vértices u, v $\in$ V, y u $\neq$ v. (u, v) = (v, u).

Definición. Un *grafo dirigido* G es un par (V, E), donde V es un conjunto finito no vacío y E es una relación binaria en V, es decir, un conjunto de pares ordenados de elementos de V. (u $\rightarrow$ v) (u, v) $\neq$ (v, u) (v $\rightarrow$ u).

Conocemos también que dos vértices u y v son *adyacentes* si son distintos y existe una arista que los une. Si (u, v) es una arista de un grafo, decimos que el vértice v es adyacente al vértice u. En un grafo no dirigido, la relación de adyacencia es simétrica; no es así necesariamente en un grafo no dirigido.

También sabemos que una arista es *incidente* a un vértice v, si v es uno de los extremos de la arista, y ella no es un bucle. Es decir, si (u, v) es una arista de un grafo dirigido, decimos que incide desde o sale de el vértice u, que será su vértice inicial, y que incide hacia o entra en el vértice v, que será su vértice final. Si estamos en un grafo no dirigido, decimos que la arista (u, v) simplemente incide en los vértices u y v, que serán sus vértices extremos.

Definición. Sea G = (V, E) un grafo. Un *camino* en el grafo G es una secuencia finita de aristas, tal que, el extremo final de cada arista coincide con el extremo inicial de la siguiente. Es decir, es una secuencia de vértices $v_0, v_1,\ldots, v_n \in V$, con $n \geq 1$, tal que $(v_i, v_{i+1}) \in E$, con $0 \leq i \leq n-1$. El número de arcos que componen un camino se denomina longitud de dicho camino que se corresponde con el número de vértices menos uno.

Definición. Un camino es simple si todos sus vértices, excepto tal vez el primero y el último, son distintos. Es decir, un camino es simple si todas sus aristas son distintas.

Definición. Un *ciclo* es un camino en el que todos los vértices son distintos, salvo que el vértice inicial coincide con el final ($v_0 = v_n$). Es decir, se considera ciclo a un camino simple de longitud no nula que empieza y termina en el mismo vértice.

Definición. Se define el *grado* de un vértice v como el número de aristas que inciden en él.

Definición. Sea G un grafo, con un número de vértices n (|V| = n). Se dice que G es un grafo *completo* si grado(v) = n – 1 para todo $v \in V$.

En cuanto a la implementación de grafos en el proyecto se va a seguir el diseño de [Riv10] mediante matriz de adyacencias, donde dicha matriz viene definida por una matriz booleana o real $M=(m_{ij})$ cuadrada de dimensión n donde n = |V|. En el caso de matrices booleanas un 1 en una posición (i,j) de la matriz, indicará que hay una arista desde el nodo v_i al nodo v_j y un 0 que no la hay. En el caso de matrices reales, dicho el valor de (i,j) indica el peso de la arista que va desde v_i a v_j, si dicho peso es un valor nulo predefinido indica que no hay arista. Obsérvese que la matriz no será simétrica, en general, si el grafo es dirigido. En cambio, la matriz de adyacencia de un grafo no dirigido siempre será simétrica.

2.7.2.2 Matching en un grafo bipartito

Un grafo bipartito es un grafo no dirigido G = (V, E) cuyo conjunto de vértices V es la unión de dos conjuntos disjuntos V_1 y V_2 de forma que $(u, v) \in E$ implica que, o bien $u \in V_1$ y $v \in V_2$, o bien $u \in V_2$ y $v \in V_1$. Es decir, todas las aristas tienen un extremo en cada uno de los conjuntos V_1 y V_2. Además, recordemos que V_1 y V_2 forman una partición de V si son disjuntos y su unión es V. Desde un punto de vista más formal tenemos la siguiente definición.

Definición. Sea G = (V, E) un grafo. Se dice que G es un *grafo bipartito* si existen dos subconjuntos disjuntos V_1 y V_2 de V tales que $V_1 \cup V_2 = V$ y cada arista tiene por extremos un elemento de V_1 y otro de V_2.

Un grafo se dice bipartito completo si todo vértice de V_1 es adyacente con todo vértice de V_2 y viceversa.

Teorema. Un grafo G es bipartito si y sólo si no contiene ciclos de longitud impar.

Veamos pues, en la Figura 8, un ejemplo de grafo bipartito G = (V, E) donde $V=V_1 \cup V_2$ y $V_1= \{0, 1, 2, 3\}$ y $V_2 = \{a, b, c, d, e\}$:

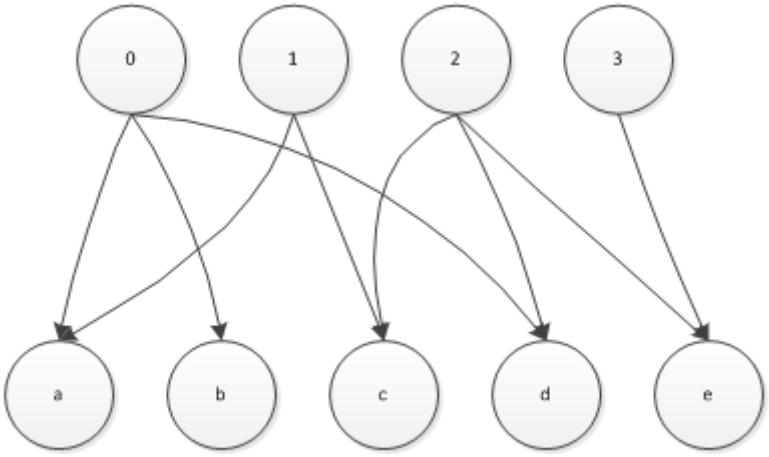


Figura 8. Ejemplo de grafo bipartito

Definición. Sea G = (V, E) un grafo bipartito con $V = V_1 \cup V_2$. Un *matching* M, de V_1 en V_2, de G es un subconjunto de E ($M \subseteq E$) tal que todo elemento de V_1 es extremo de una arista.

Es decir, un matching (correspondencia o emparejamiento) M de un grafo G = (V, E) es un subconjunto de aristas, $M \subseteq E$, tales que dos aristas no comparten el mismo vértice; por lo tanto, ningún par de aristas es adyacente. La cardinalidad o tamaño de un matching M es su número de aristas, denominado |M|.

Definición. Dado un matching M en G = (V, E), a las aristas de M ($e \subseteq M$) se les denomina *aristas emparejadas*, mientras que a las aristas e $\subseteq$ (E – M) se les denomina *aristas libres* o no emparejadas. Por otro lado, un vértice es un *vértice emparejado* si éste es incidente sobre en una arista emparejada; en otro caso se le denomina *vértice libre*.

Definición. *Un matching de tamaño máximo* es un matching que contiene un número máximo de aristas, de modo que el número de vértices libres (es decir, los vértices no incidentes con las aristas en M) es mínimo. Es decir, un matching de tamaño máximo de G, es un matching M con la cardinalidad, |M|, máxima.

Definición. *Un matching perfecto* es un matching que empareja todos los vértices del grafo G.

En general, un problema de matching consiste en hallar un matching de tamaño máximo para un cierto grafo G.
Definición. Un *camino alternativo* (alternating path) tiene aristas que son, alternativamente, libres y emparejadas. Un *camino de aumento* (augmenting path) es un camino alternativo que empieza en un vértice libre y termina en otro vértice libre.

Es decir, un *camino alternativo* para un matching M en un grafo G es una secuencia de aristas (v_1, v_2), (v_2, v_3),…, (v_{k-1}, v_k) que presenta la posibilidad de pertenecer a M y a E – M (conjunto de aristas que no están en M). Un camino de aumento para M es un camino alternativo donde los dos vértices de los extremos no son incidentes con ninguna arista en el matching M. Por tanto, un *camino de aumento* tiene un número impar de aristas, 2*k + 1, de las cuales k pertenecen a M y k + 1 no están en M (están en E – M). Si las artistas en M se

remplazan por las aristas que no están en M, entonces hay una arista más en M que antes del intercambio. De esta manera, la cardinalidad del matching M se incrementa en uno.

De todos estos conceptos relacionados con matching en grafos podemos destacar las siguientes propiedades:

1. Si M es un matching de G = (V, E), entonces el número de vértices emparejados es 2*|M|, y el número de vértices libres es |V| – 2*|M|.
2. Si M es un matching cualquiera de G = (V, E), entonces $|M| \leq \left\lfloor \frac{|V|}{2} \right\rfloor$.
3. Cualquier camino de aumento tiene un número impar de aristas.
4. (Teorema del camino de aumento) M es un matching de tamaño máximo si y sólo si no existe ningún camino de aumento con respecto a M.

Lema. Si para dos matchings M y N en un grafo G = (V, E) definimos un conjunto de aristas $M \oplus N \subseteq E$, entonces cada componente conectado del sub-grafo G' = (V, $M \oplus N$) es ya sea (a) un único vértice, (b) un ciclo con un único par de aristas tales que están en M o en N, o (c) un camino cuyas aristas están ya sea en M o en N tales que cada vértice del extremo del camino corresponde a uno de los dos matchings M o N (es decir, todo el camino debería considerarse, no solo parte del matching, para cubrir al componente conectado completo).

Lema. Si M es un matching y P es un camino de aumento para M, entonces la diferencia simétrica $M \oplus P$ es un matching de tamaño |M| + 1.

Teorema. Un matching M en un grafo G es de tamaño máximo si no hay un camino de aumento conectando dos vértice no emparejados en G.

Este teorema sugiere que un *matching máximo* puede encontrarse al principio con un matching inicial, posiblemente vacío y entonces al encontrar repetidamente nuevos caminos de aumento e incrementar la cardinalidad del matching hasta que no se pueda encontrar ningún camino. Esto requiere un algoritmo para encontrar caminos alternativos. Es mucho más fácil desarrollar una algoritmo que haga esto para grafos bipartitos que para cualquier otro tipo de grafos. Por este motivo vamos a aclarar algunos conceptos relativos a matching en grafos bipartitos.

Definición. Un matching completo de un grafo bipartito G = ($V_1 \cup V_2$, E), es un matching M en el que cada vértice de V_1 es incidente en una arista de M. Por este motivo, a un matching con esta característica se le denomina *asignación* de V_1 a V_2.

Teorema (Condición de diversidad). Sea G = (V, E) un grafo bipartito con $V = V_1 \cup V_2$. Existe un matching de V_1 en V_2 de G si y sólo si para cada subconjunto S de V_1 ($S \subseteq V_1$), la cardinalidad de dicho subconjunto S es menor o igual que la cardinalidad del subconjunto de V_2, formado por aquellos vértices de V_2 que son adyacentes con algún elemento de S ($|S| \leq |V_2|$).

Corolario. Sea G = (V, E) un grafo bipartito con $V = V_1 \cup V_2$. Existe un matching de V_1 en V_2 de G si existe un número entero K tal que grado(v_1) ≤ k ≤ grado(v_2) para todo $v_1 \in V_1$ y $v_2 \in V_2$.

Por ultimo vamos a ver el matching del grafo anterior en la Figura 9, que como hemos explicado anteriormente, no es único. En este caso existen dos matching posibles: M_1={(0,b), (1,a), (2,c), (3,e)} y M_2={(0,b), (1,a), (2,d), (3,e)}:

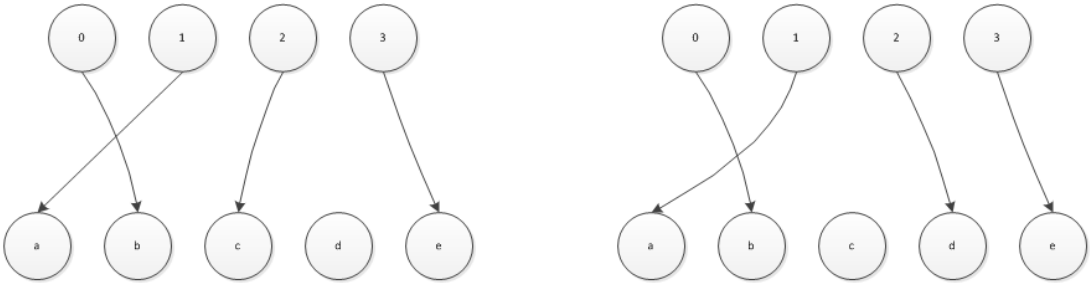


Figura 9. Matching del grafo bipartito de la Figura 8. Se exponen dos imágenes con los dos posibles matchings.

2.8 Conclusiones

En este proyecto se ha trabajado con dos grandes módulos: el *iDistance* y un sistema de recuperación video basado en contenido (CVBR). Para ello se hace imprescindible conocer las estructuras de datos en las que dichos módulos están basados. Además se debe de tener un conocimiento claro del tipo de datos a tratar así como las distancias empleadas.

Para el estudio del *iDistance* en un CVBR son necesarios los fundamentos descritos anteriormente en este capítulo, prestando especial atención en el cálculo de clústeres para datos altamente dimensionales y la estructura de datos de un árbol B+ que es la base del *iDistance*.

En cuanto al funcionamiento de un CVBR, es necesario conocer el matching sobre grafos bipartitos, puesto que es la base en la que se sustenta la recuperación de subsecuencias de video.

3

iDistance

3.1 Introducción

En esta sección vamos a presentar el *iDistance*: un árbol B+ adaptativo para indexación de datos multidimensionales optimizado para la consulta de los K vecinos más cercanos *(K-nearest neighbor query)* y consultas en rango de distancia. Para definir los datos vamos a hacer uso de bases de datos con puntos de un espacio *dim*-dimensional como se expone en el apartado de definición de datos en el capítulo anterior.

Tal y como hemos visto anteriormente, la consulta de los K vecinos más cercanos (*KNNQuery*) consiste en: dados un conjunto de puntos en la base de datos DB en un espacio *dim*-dimensional E^{dim}, un punto de consulta cualquiera q en E^{dim} encontrar un conjunto de datos *kNN* que contiene K puntos de DB ($kNN \subseteq DB$) tal que para cualquier punto $p \in kNN$ y para cualquier punto $z \in$ (DB-*kNN*) se cumple que $dist(q, p) < dist(q, z)$. Donde *dist* indica cualquier tipo de distancia definida el capítulo anterior.

El *iDistance* [JOT+05] se basa en dicha distancia para buscar los K vecinos más cercanos de un punto de consulta *q*, la distancia del K-ésimo vecino más cercano a *q* define el radio mínimo requerido para la recuperación de los puntos en la base de datos. Desafortunadamente, esta distancia no se puede determinar con un cien por cien de precisión. Por lo tanto, en un enfoque iterativo se puede utilizar para determinar en cada iteración con mayor exactitud el radio de búsqueda: La búsqueda se inicia con un ámbito de consulta sobre *q* mediante un diferencial del radio (*dr*) de búsqueda que puede ser establecido mediante registros de búsqueda e históricos.

Mantenemos por tanto el conjunto candidato encontrado que podría ser o contener los K vecinos más cercanos a *q*. Después aumentamos la esfera de consulta en (*dr*) en cada iteración ampliado así el conjunto de candidatos hasta que podamos asegurar que los K puntos recuperados son los K vecinos más cercanos de *q*.

Por tanto podemos resumir la consulta KNN en el siguiente pseudocódigo (código 3):

KNNQuery (q, DB, K)

1. *Comenzar la búsqueda con una esfera de radio mínima centrada en el punto de consulta q.*
2. *Comprobar todas las particiones de la DB que están contenidas en el espacio de consulta actual.*
3. ***Si** los K vecinos más cercanos se han encontrado*
4. *Terminar.*
5. ***Si no***
6. *Aumentar el radio de la esfera de búsqueda.*
7. *Volver a 2.*
8. ***Fin Si***

Código 3. [JOT+05] Pseudocódigo de la búsqueda de los K vecinos más cercanos a un punto de consulta dado.

Como vemos en el pseudocódigo, la búsqueda se centra en el punto de consulta *q* para crear la esfera de consulta y comprueba que partes del árbol B+ se cruzan en dicha esfera mediante las distancias definidas y utilizadas para ello. Más adelante veremos que es en este punto donde entra en juego el cálculo de centros con el algoritmo de Lloyd, puesto que la comprobación con el espacio de datos se realiza mediante la distancia del punto *q* de consulta y los centros o puntos de referencia del *iDistance*.

3.1.1 Notación

Antes de continuar y para facilitar la lectura de este proyecto vamos a exponer una tabla con los símbolos más utilizados en la definición y estructura del *iDistance*. De ahora en adelante se hará uso de esta notación y simbología:

Símbolo	**Definición**
dim, d	Dimensión del espacio de datos. (dim=d)
dist	Función de Distancia en un espacio *dim*-dimensional.
E^{dim}	Espacio de datos *dim*-dimensional.
M^{dim}	Espacio Métrico *dim*-dimensional $M^{dim} = (dist, E^{dim})$
F^{dim}	Archivo *dim*-dimensional donde se almacenarán los datos.
S, DB	Conjunto de datos (puntos) dim-dimensionales del espacio E^{dim} y distancia dist. ($S=DB \subseteq M^{dim}$)
KNNQuery	Procedimiento o consulta para recuperar los K vecinos más cercanos a un punto de referencia dado.
K	Numero K de vecinos a buscar en el espacio de datos S
kNN	Subconjunto de S que contiene los K vecinos más cercanos a un punto de consulta dado (q). ($kNN \subseteq S$ y $\|kNN\|=K$)
n	Número de puntos de referencia (O) y por consiguiente de secciones o particiones (P) del espacio de datos (S).
P_i	Partición o sección i-ésima del espacio de datos S. $\bigcup_{i=0}^{n} P_i = S$
O_i	Punto de referencia de la partición o sección i-ésima (P_i) del espacio de datos S.
r	Radio del anillo de búsqueda
$\triangle r$	Radio inicial y aumento del radio de búsqueda actual. (*dr discreto*)
C_i	Rango de indexación constante de la partición i-esima (P_i)
p	Punto *dim*-dimensional perteneciente a una partición $P_i \subseteq S$
q	Punto *dim*-dimensional de consulta.
$DistMax_i$	Radio máximo del anillo correspondiente a la partición o sección de datos P_i
Querydist(q)	Radio de consulta para q.
Sphere(q,r)	Esfera *dim*-dimensional con centro q y radio r. Anillo en el árbol.
Furthest(kNN,q)	Devuelve el punto más lejano en distancia a q del conjunto kNN.
$Dist(p_1,p_2)$	Distancia entre dos puntos del mismo espacio métrico M^{dim}.
y	Clave de indexación basada en la distancia: $y = i \times C_i + dist(p, O_i)$
i	Hace referencia a la partición o sección de datos actual.
RefSphere(Pi)	Esfera de referencia o anillo para la partición P_i centrada en el punto de referencia de la partición O_i, con un radio máximo de $DistMax_i$.

Tabla 1. Notación del *iDistance*

3.2 Definición y justificación

Podemos por tanto, definir el *iDistance* [JOT+05 y Cui02] como una manera eficiente de procesar la consulta de los K vecinos más cercanos a un punto de consulta *q* dado, utilizando un árbol B+ adaptativo a la distancia relativa entre los puntos de la base de datos y unos centros previamente dados.

La justificación de utilizar el *iDistance* como motor de búsqueda de K vecinos a un punto dado viene dada por tres observaciones principales [JOT+05]:

- ***La similitud en distancia entre los puntos dim-dimensionales puede ser aproximada por puntos de referencia (centros de las esferas que engloban el espacio de datos).***
- ***Los puntos de la base de datos pueden ser ordenados en base a la distancia a los centros anteriormente citados.***
- ***La distancia es una medida unidimensional.***

Todo esto facilita el uso de un árbol B+ para representar un espacio de datos multidimensional en un espacio unidimensional.

Si consideramos un conjunto de datos S en un espacio *dim*-dimensional E^{dim}, el cual tiene asociada una distancia *dist* tal que se define el espacio métrico M^{dim} = (dist, E^{dim}), dados $p_1=(x_0, x_1, \dots, x_{d-1})$, $p_2=(y_0,y_1,\dots,y_{d-1})$ y $p_3=(z_0,z_1,\dots,z_{d-1})$ en S, la función de distancia *dist* cumple las propiedades expuestas en el capítulo dos de espacios multidimensionales:

1. $dist(p_1, p_2) = dist(p_2, p_1)\ \forall p_1, p_2 \in S$
2. $dist(p_1, p_1) = 0\ \forall p_1 \in S$
3. $0 < dist(p_1, p_2)\ \forall p_1, p_2 \in S;\ p_1 \neq p_2$
4. $dist(p_1, p_3) \leq dist(p_1, p_2) + dist(p_2, p_3)\ \forall p_1, p_2, p_3 \in S$

De la ecuación 4, podemos tener en cuenta la desigualdad triangular en nuestro espacio métrico M^{dim} la cual nos va a permitir seleccionar los candidatos (KNN) en base a la relación de distancia *dist*. En este proyecto se van a usar como distancias: la Euclidea y la Máxima, definidas en el capítulo 2.

$$Distancia\ Euclidea: d_E(x,y) = \sqrt{(\sum_{i=0}^{dim-1} |X_i - Y_i|)^2}$$

$$Distancia\ Máxima: d_{Max}(x,y) = Max_{0 \leq i < \dim} |X_i - Y_i|$$

3.3 Estructura e indexación de datos

Para explicar el proceso que sigue el *iDistance* para recuperar los KNN supongamos que tenemos un punto O_i que es el punto de referencia (o centro) de la sección de datos P_i de S (la totalidad de la base de datos). Es decir, $S=P_0 \cup P_1 \cup P_2 \cup ... \cup P_n$ donde n es el número de puntos de referencia O.

Es importante tener en cuenta que O_i no necesita ser un punto de S o lo que es lo mismo de P_i, la sección de datos P_i y un punto de ella cualquiera $p \in P_i$ se referencia mediante el punto O_i en términos de distancia o proximidad con este, es decir, dist(O_i,p). Usando la desigualdad triangular anteriormente citada y un punto de consulta $q \in M^{dim}$ que puede o no estar en S, es fácil deducir que:

$$dist(O_i , q) - dist(p, q) \leq dist(O_i , p) \leq dist(O_i , q) + dist(p, q)$$

Para la búsqueda de los K vecinos más próximos (Figura 10), el método *iDistance* se basa en la ampliación o aumento del radio de búsqueda (*querydist(q)*) de las esferas de búsqueda centradas cada una en cada punto de referencia o centro O_i del espacio de datos. Por tanto nos interesa obtener todos los puntos *p* los cuales $dist(p,q) \neq querydist(q)$. Para todo punto *p*, teniendo en cuenta la desigualdad triangular anterior, tenemos que:

$$dist(O_i , q) - querydist(q) \leq dist(O_i , p) \leq dist(O_i , q) + querydist(q)$$

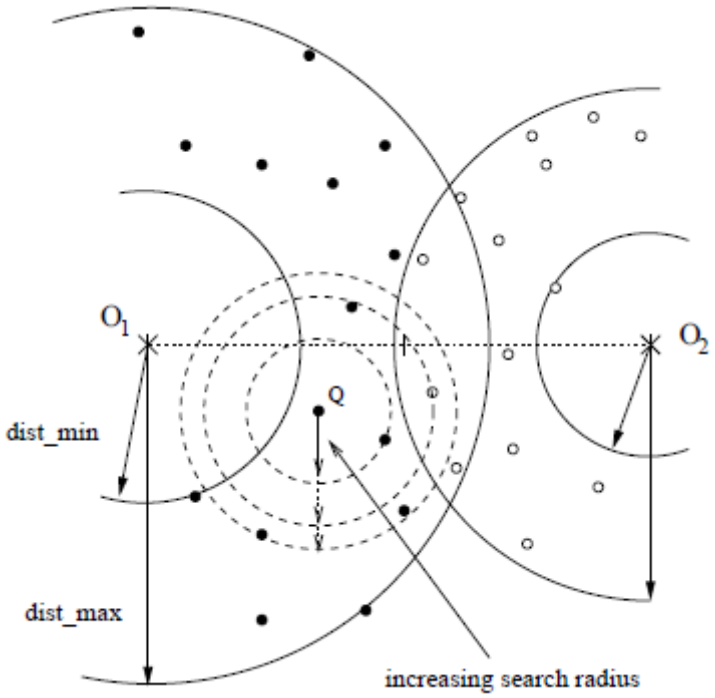


Figura 10. [Cui02] Esquema del funcionamiento del *iDistance* para la recuperación de los K vecinos a un punto de consulta dado Q.

Por lo tanto, de la sección P_i solo necesitamos examinar los puntos *p* para los que la distancia desde el punto de referencia correspondiente a la partición O_i (*dist*(O_i,*p*)) está limitada por esta desigualdad que en normas generales se reduce a un anillo alrededor del punto de referencia.

Ya sabemos que parte de cada sección (P_i) debe de ser examinada, pero no todas las secciones de S (P_i) tienen que ser examinadas, para saber cuáles debemos de consultar podemos definir en cada partición P_i: $MaxDist_i$ y $MinDist_i$ como la distancia que hay entre el punto de referencia O_i y el punto más alejando de dicho centro en P_i y el más cercano respectivamente. Por tanto $MaxDist_i$ define el radio que engloba todos los puntos de P_i con centro O_i. Con todo esto podemos determinar que sección de S tenemos que consultar si para una sección de S, P_i, con un radio $MaxDist_i$ definido, **si *dist*(O_i , *q*)−*querydist*(*q*) ≤ $MaxDist_i$, entonces la sección P_i debe de ser consultada para el cálculo de los KNN de q, si no se cumple la ecuación podemos descartar dicha partición,** esto hace que se puedan "podar" muchas secciones en la búsqueda KNN y sea mucho más eficiente en cuanto a tiempo y accesos puesto que para determinar si se debe o no consultar una partición $P_i \in S$ tan solo basta con consultar su punto de referencia o centro O_i y su radio máximo.

El rango de datos que debe de ser buscado dentro de una partición en el espacio unidimensional indexado es *dist*(O_i , *q*) − *querydist*(*q*), *min*($MaxDist_i$, *dist*(O_i , *q*) + *querydist*(*q*)). En la Figura 11 se muestra un ejemplo donde las secciones P_i están creadas en base a tres clústeres (la división de los datos en secciones se verá con más detalle en el siguiente apartado). Para el punto de consulta q y el radio de búsqueda *querydist(q)*, las secciones P_1 y P_2 deben de ser consultadas pero la sección P_3 no.

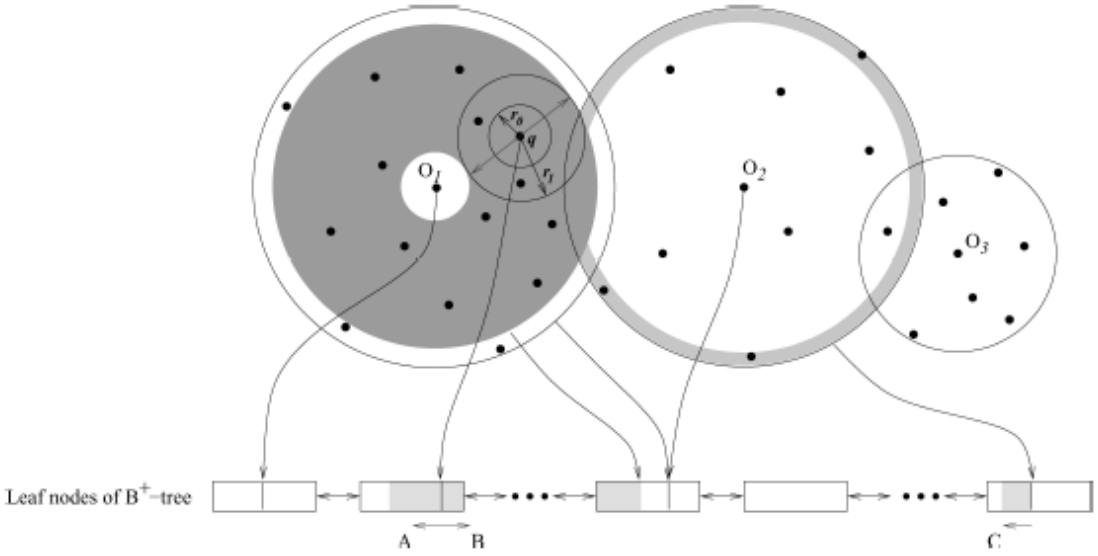


Figura 11. [JOT+05] Indexación realizada por el *iDistance* del espacio de datos *dim*-dimensional al espacio unidimensional basado en distancia.

Con motivo de la pérdida de precisión por la reducción de la dimensionalidad al indexar en un espacio unidimensional determinado por la distancia, de la Figura 11 se puede deducir que todos los puntos situados en el anillo con un mismo radio tienen el mismo valor después de la

transformación a distancias con respecto a los puntos de referencia (O). **Por lo tanto todas las regiones sombreadas en la Figura 11, son áreas que deben ser consultadas.**

Tras definir el funcionamiento del *iDistance*, vamos a detallar el tipo de estructura de datos utilizada para indexar la base de datos *dim*-dimensional que se base en métricas de distancia, que tipo de división de datos es mejor utilizar y como obtener los puntos de referencia o centros de cada sección.

Para la estructuración de los datos vamos a suponer que la base de datos ya ha sido dividida en secciones y que los puntos de referencia o centros de cada partición ya han sido calculados.

3.3.1 Creación del índice unidimensional

Partiendo de la base de que el método *iDistance* reduce la dimensionalidad de puntos *dim*-dimensionales a un espacio de datos unidimensional basado en la distancia, podemos resumir el método como un algoritmo de tres pasos:

Paso 1: Los datos multidimensionales son repartidos en un conjunto de secciones o particiones que los engloben siendo P_i una partición y $\bigcup_{i=0}^{n} P_i = S$ siendo S toda la base de datos completa.

Paso 2: Identificación de los puntos de referencia de cada sección o partición de datos. Si suponemos que tenemos *n* particiones $\bigcup_{i=0}^{n} P_i = S$ entonces tenemos los siguientes puntos de referencia correspondientes a cada partición: $O = \{O_0, O_1, O_2, \ldots, O_{n-1}\}$. Como veremos posteriormente en la división de los datos, estos puntos de referencia son muy importantes a la hora de efectuar una consulta y depende en gran medida de la efectividad del *iDistance*.

Paso 3: Todos los puntos son representados en un espacio unidimensional de la siguiente manera: Sea un punto p:$\{x_0, x_1, x_2, \ldots, x_{dim-1}) \in P_i$ y $P_i \subseteq S$ con los valores de p normalizados entre 0 y 1 y una clave y de p basada en la distancia de p con su centro de referencia más cercano O_i tal que $\boldsymbol{y = i \times c + dist(p, O_i)}$, entendiéndose que $\boldsymbol{i}$ denota la partición o sección objetivo. Donde c es una constante que se utiliza para ajustar mejor el rango de datos. Normalmente se suele utilizar valores iguales a la dimensión de los puntos que se están tratando o mayor que la diagonal mayor del hipercubo formado por el espacio de datos que se está tratando. [LIF94, Cui02 y JOT+05].

En el proceso iDistance, c sirve para dividir el espacio unidimensional a las distintas secciones o particiones y éstas en pequeñas regiones a fin de que todos los puntos de partición P_i se asignan a los valores [i * c, (i + 1) * c]. Por lo tanto c debe de ser suficientemente grande para evitar el solapamiento entre los rangos de índice de clave de las diferentes particiones.

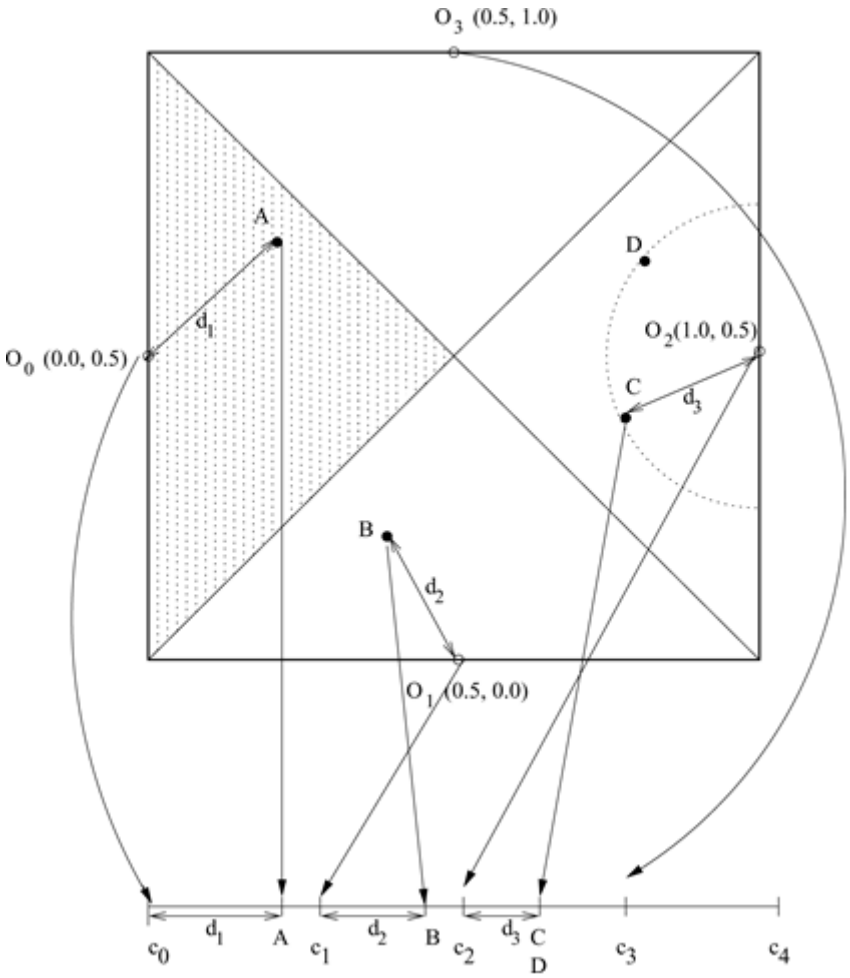


Figura 12. [JOT+05] Ejemplo de indexado con pérdida de precisión en puntos con el mismo radio hacia el punto de referencia de una misma partición.

La Figura 12 muestra un indexado de datos bidimensional, donde O_0, O_1, O_2 y O_3 son los puntos de referencia de las 4 particiones existentes. Si consideramos los puntos A, B, C y D asociados a las particiones P_0, P_1, P_2 y P_2 respectivamente y C_0, C_1, C_2, C_3 y C_4 son los valores de rango de las particiones correspondientes: O_0 está asociado con C_0 y por tanto todos los puntos que tengan de referencia a O_0 tendrán asociada la distancia con O_0 mediante C_0 *($y = i \times C_i + dist(p, O_i)$).* Si sustituimos tenemos que $y_0 = 0 \times C_0 + dist(p, O_0)$.

Siguiendo la figura anterior, podemos ver como O_0 está asociada con C_0 y todos los puntos de datos de la región sombreada tienen las distancias relativas a C_0. Podemos ver de manera evidente como el *iDistance* puede mapear con el mismo valor (unidimensional) a diferentes puntos multidimensionales, como es el caso de C y D. Esto sucede cuando dos puntos son equidistantes del mismo punto de referencia y puede dar lugar a falsos positivos durante una consulta. En [Del07] se propone una técnica para evitar dichos falsos positivos llamada *Multidimensional iDistance (MiD)* que se comentará en el capítulo 7 (*Conclusiones y Trabajo Futuro*), en el apartado *Mejoras del iDistance.*

3.3.2 Compactación del árbol B+

En ocasiones, tras la construcción del *iDistance*, el árbol B+ puede quedar con un bajo uso de la capacidad de los nodos hoja y puede suponer un gran consumo de memoria principal cuando dicho índice (*iDistance*) se carga en memoria. En gran medida, la razón por la que quedan tantos nodos hoja sin ocupar en su gran parte, es por el *offset* C_i de cada sección P_i del espacio de datos. Como se puede observar en la Figura 12, cada partición o sección de datos tiene su valor de rango (*offset),* si una vez creado el árbol se sabe a ciencia cierta que no se van a insertar un gran número de claves a posteriori, podría reorganizarse el árbol para que los nodos hoja que estén incompletos queden llenos. Esto no quiere decir que no se pueden insertar registros tras la reorganización, simplemente que al no quedar espacio, el árbol tendría que reorganizarse con lo cual tardaría más dicha inserción. Según [JOT+05], el uso de los nodos internos del árbol, en los peores casos, puede llegar a ser de un 33% de todo el índice.

En la implementación que nos ocupa, no se han realizado los métodos de eliminación e inserción en el *iDistance* ya construido puesto que el proyecto se centra en el comportamiento para la consulta de los K vecinos más cercanos. Es por eso, que no está de más, la implementación de un algoritmo de compactación que, una vez creado el *iDistance*, intente reorganizar el árbol para que el porcentaje de uso de los nodos sea más alto.

La compactación puede conllevar un tiempo de cálculo alto, por tanto se deja al usuario elegir el factor de compactación (*compactRate*) en tanto por ciento, el cual indica el porcentaje de utilización de los nodos hoja. Si *TamañoMaximo* indica el número máximo de claves en un nodo hoja y *TamañoDeseado* el número de claves que el usuario quiere que haya como mínimo en un nodo hoja, dado que *TamañoMaximo* es conocido (grado del árbol menos uno) podemos calcular el *TamañoDeseado* con la siguiente formula:

$$TamañoDeseado = TamañoMaximo * (compactRate / 100)$$

En el código 4, descrito a continuación, se puede observar como en la primera línea se calcula el tamaño de ocupación del nodo hoja del árbol:

Compactar(árbol, compactRate)

1. *TamañoDeseado=arbol→TamañoMaximoNodoHoja*(compactRate/100)*

2. *NodoHoja=Primer nodo hoja del árbol.*
3. ***Mientras** NodoHoja->Entradas ≥ TamañoDeseado Hacer*
4. *NodoHoja=NodoHoja→Siguiente*
5. ***Fin Mientras***

6. *$Nodo_1$=NodoHoja*
7. *$Nodo_2$=$Nodo_1$→Siguiente*
8. ***Mientras** queden nodos sin procesar en la lista de nodos hoja del árbol*
9. ***Si** el nodo 1 tiene espacio*
10. *Volcar claves del $Nodo_2$ al $Nodo_1$*
11. ***Fin Si***
12. ***Si** quedan nodos sin procesar*
13. ***Si** no está completo el Nodo2*
14. *$Nodo_2$=$Nodo_2$→Siguiente*

15. ***Fin Si***
16. ***Fin Si***
17. ***Si*** *el Nodo$_1$ está completo y es el ultimo*
18. *Nodo$_1$ →Siguiente=-1*
19. *Guardar en el árbol el Nodo$_1$*
20. ***Sino***
21. *Nodo$_1$ →Siguiente=Siguiente nodo sin procesar o completo.*
22. *Guardar en el árbol el Nodo$_1$*
23. ***Fin Si***
24. *Nodo$_1$=Siguiente nodo que cumpla: Nodo->Entradas < TamañoDeseado*
25. ***Fin Mientras***

26. *Reorganizar los nodos internos del árbol.*
27. *Guardar el principio de la lista de nodos hoja del árbol.*

Código 4. Compactación del índice, tras su generación.

3.3.3 Estructuras de datos del iDistance

Por último, en el *iDistance* tenemos dos estructuras de datos fundamentales:

- ***Un árbol B+ adaptativo usado para indexar los datos siguiendo la técnica mencionada anteriormente en "Creación del índice unidimensional".***

- ***Una matriz para guardar las n particiones con sus respectivos puntos de referencia y los radios (DistMax$_i$ y DistMin$_i$) de las esferas de referencia de cada partición RefSphere(P$_i$). Dicha matriz se utilizará para determinar el número de secciones (o particiones) que son necesarias consultar durante la consulta KNNQuery.***

En este sentido hablamos de un *árbol B+ adaptativo* cuando las operaciones sobre dicho árbol se realizan teniendo en cuenta la estructuración de los datos multidimensionales en una sola dimensión siguiendo el proceso del *iDistance*. En la implementación del árbol B+, los nodos hoja están conectados con sus respectivos hermanos (izquierdo y derecho) [GRG00]. Esto facilita la búsqueda de los nodos vecinos según se vaya aumentando la región a buscar.

Por último en [JOT$^+$05] se escogió el árbol B+ por ser una de las estructuras de indexación de datos más utilizada en los sistemas de bases de datos de la actualidad y más eficiente sobre datos unidimensionales.

En la Figura 13 podemos ver la correspondencia entre la división de los datos, los puntos de referencia y la distribución de los datos en el árbol B+. No se han representado los puntos de referencia en los nodos hoja puesto que estos no tienen por qué pertenecer al conjunto de datos.

Es por tanto, que son necesarias las dos estructuras de datos anteriormente citadas: el árbol para construir el índice y la matriz con los puntos de referencia y las distancias máximas y mínimas o esferas de referencia (*RefSphere*) de cada partición de los datos P_i.

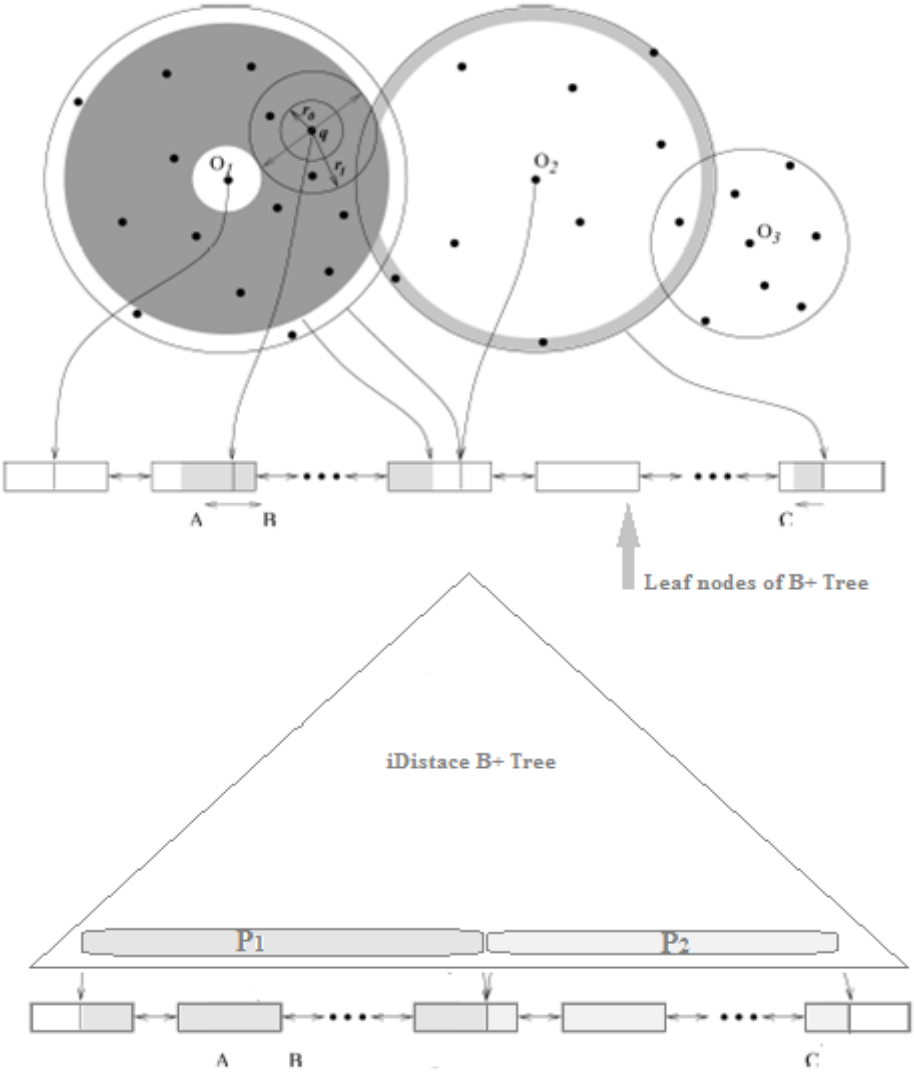


Figura 13. Estructura de datos del *iDistance: B+ Tree*

3.4 Búsqueda de los K vecinos más cercanos en el iDistance (KNN Search)

Sea $q \in E^{dim}$ un punto de consulta dado, $\triangle r$ el radio mínimo de búsqueda inicial y el aumento del radio progresivo a aplicar y K el número de vecinos a buscar, la consulta KNN utilizando el *iDistance* se define de la siguiente manera [JOT+05]:

```
iDistance_KNNQuery(q, ∆r, K)
1. r=0
2. stopFlag = FALSE
3. Inicializar lp[], rp[], oflag[]
4. Mientras stopFlag==FALSE
5.        r=r+∆r
6.        SearchO(q, r, K, stopFlag)
7. Fin Mientras
```

Código 5. Método principal para la recuperación de los K vecinos más próximos a un punto de consulta dado, mediante el método *iDistance*.

Veamos pues el pseudocódigo del método principal de búsqueda *SearchO*:

```
SearchO(q, r, K, stopFlag)

1.  p_furthest = furthest(kNN,q)
2.  Si dist(p_furthest, q) < r and |kNN| == K
3.    stopFlag=TRUE
4.  Fin Si
5.  Para i=0 hasta n-1 /* n=Particiones del espacio de datos*/
6.    dis=dist(O_i,q)
7.    Si ~oflag[i] /*Si O_i no ha sido consultada antes.. */
8.       Si sphere(O_i, dist_max_i) contiene a q
9.          oflag[i]=TRUE
10.         lnode = LocateLeaf (btree, i * c + dis)
11.         lp[i]= SearchInward (lnode, i * c + dis – r)
12.         rp[i]=SearchOutward (lnode, i * c + dis + r)
13.      Sino: si sphere(O_i, dist_max_i) interseca a sphere(q, r)
14.         oflag[i]=TRUE
15.         lnode = LocateLeaf (btree, dist_max_i)
16.         lp[i]= SearchInward (lnode, i * c + dis - r)
17.      Fin Si
18.   Sino
19.      Si lp[i] no es nulo
20.         lp[i]= SearchInward(lp[i] →leftnode, i * c + dis – r)
21.      Si rp[i] no es nulo
22.         rp[i] = SearchOutward(rp[i] →rightnode, i * c + dis + r)
23.   Fin Si
24. Fin Para
```

Código 6. Método *SearchO* encargado de buscar en una partición el conjunto de puntos candidatos para los K vecinos más próximos a *q*.

La consulta *KNNQuery* comienza con la búsqueda en una pequeña esfera y va incrementando el espacio de búsqueda hasta que los K vecinos hayan sido encontrados. La búsqueda termina cuando la distancia al vecino más lejano de *kNN* (conjunto de puntos que representa los K vecinos más próximos a *q*) sobre el punto de consulta *q* es menor o igual al radio de búsqueda actual *r*.

A continuación, es obligatorio detallar el pseudocódigo de búsqueda recursivo sobre los nodos con datos hacia el punto de referencia de la partición actual (consultada), la cual interseca con la esfera de búsqueda: *SearchInward*. Y los nodos hacia fuera del punto de referencia de la partición actual (consultada), la cual interseca con la esfera de búsqueda: *SearchOutward*.

Si nos fijamos en la Figura 12 podemos decir, como ejemplo, que ***SearchInward*** centrado en A buscará hacia O_0, es decir, puntos donde la distancia entre los puntos de la esfera de consulta y el centro O_0 sean menores que las distancias actuales. En cambio, ***SearchOutward*** buscará hacia el sentido contrario de O_0 o lo que es lo mismo los puntos de la esfera de consulta en los que la distancia con el centro O_0 sean mayores que la distancia actual.

En [JOT+05] se definen los pseudocódigos de búsqueda de la siguiente manera:

SearchInward(node, iDistValue)

1. ***Para*** *cada entrada e en el nodo* ($e = e_j, j=1, 2, \ldots, Numero\_de\_Entradas$)
2. ***Si*** $|kNN| == K$
3. $p_{furtherst} = furthest(kNN, q)$
4. ***Si*** $dist(e, q) < dist(p_{furthest}, q)$
5. $kNN = kNN - p_{furthest}$
6. $kNN = kNN \cup e$
7. ***Fin Si***
8. ***Si no***
9. $kNN = kNN \cup e$
10. ***Fin Si***
11. ***Fin Para***

12. ***Si*** $e_1.key > iDistValue$
13. $node = SearchInward(node \rightarrow leftnode, i * c + dis - r)$
14. ***Si*** *se ha alcanzado el final de la particion*
15. $node = null$

16. ***Devolver*** *(node)*

Código 7. Método de búsqueda hacia la izquierda del nodo, hacia el punto de referencia de la partición actual.

El algoritmo *SearchOutward* es similar al código 7, con la salvedad de que la búsqueda se hace hacia el exterior de la partición de datos:

```
SearchOutward (node, iDistValue)

1.  Para cada entrada e en el nodo (e = e_j, j=1, 2, ..., Numero_de_Entradas)
2.      Si |kNN| == K
3.          p_furtherst = furthest(kNN, q)
4.          Si dist(e, q) < dist(p_furthest, q)
5.              kNN = kNN – p_furthest
6.              kNN = kNN ∪ e
7.          Fin Si
8.      Si no
9.          kNN = kNN ∪ e
10.     Fin Si
11. Fin Para

12. Si e_1.key > iDistValue
13.     node = SearchOutward(node → rightnode, i * c + dis + r)
14. Si se ha alcanzado el final de la particion
15.     node = null

16. Devolver (node)
```

Código 8. Método de búsqueda hacia la derecha del nodo, alejándose del punto de referencia de la partición actual.

Dado un nodo hoja, *SearchInward* examina las entradas del nodo dado hacia la izquierda con el objetivo de encontrar los K vecinos más cercanos y actualizar el conjunto solución, *kNN*, si fuera necesario. Como la indexación del *iDistance* pierde precisión al basarse en un solo valor: la distancia; es posible que los puntos con igual valor de indexación no estén cerca unos de otros en el espacio *dim*-dimensional, unos estarán más lejos de q y otros más cerca por lo que puede haber falsos vecinos encontrados.

Centrándonos en ***SearchInward***, puesto que ***SearchOutward*** sigue el mismo pseudocódigo pero la búsqueda se hace en la dirección contraria a *SearchInward;* tenemos que el primer elemento (o el último en *SearchOutward*) del nodo está contenido en la esfera de consulta entonces es probable que su predecesor (o sucesor en *SearchOutward*) con respecto a la distancia del punto de referencia de la partición actual esté cerca del punto de consulta *q* considerándose un posible vecino de q, por tanto, el hermano a la izquierda debe de ser examinado (o derecha en el caso de *SearchOutward*).

Partiendo del ejemplo de la Figura 13, podemos concluir que los métodos de búsqueda ***SearchInward*** y ***Outward*** buscan en el espacio cercano (*Inward*) o lejano (*Outward*) del punto de referencia de la partición actual. Si para el punto de consulta q, *SearchInward* busca en la partición P_1 buscará hacia el punto de referencia el hermano de la derecha del nodo actual, dirección de la flecha del punto A. Mientras que *SearchOutward* buscará alejándose del punto de referencia actual sobre los hermanos hacia la derecha del nodo actual en dirección de la flecha del punto B. Para la partición P_2, solo es necesario buscar los hermanos de la parte izquierda del nodo con *SearchInward* como indica la flecha del punto C.

El proceso para localizar un nodo hoja (***LocateLeaf***) es un método propio del árbol B+ que devuelve la posición de un nodo hoja mediante un valor de clave, en este caso de distancia referente a un centro dado. Una vez conocidos los principales métodos de búsqueda que van a desplazarse entre las distintas particiones hacia dentro (*Inward*) o hacia fuera (*Outward*) nos encontramos en disposición de detallar el algoritmo de búsqueda para los K vecinos más cercanos en profundidad.

3.4.1 Situaciones en la búsqueda de los KNN en el iDistance

La búsqueda comienza comprobando de una matriz, los puntos de referencia O_i de la base de datos que están contenidos en el espacio de consulta del punto *q*. Para cada partición o sección que necesite ser consultada, tenemos que encontrar el punto de partida desde donde se va a comenzar la búsqueda: si *q* está contenido en la partición actual se utilizará la clave de indexación de *q* para comenzar la búsqueda, si no se utiliza el valor de $DistMax_i$ de la partición actual. Una vez obtenido el punto de partida, se comienza con un radio inicial pequeño, que puede ser el propio $\triangle r$ usado para aumentar el radio de búsqueda en cada iteración. Al aumentar el radio de búsqueda en cada paso creamos una esfera de consulta más grande para la cual tenemos que considerar los siguientes casos:

1. ***Que la partición de datos actual contenga a q***. En este caso, necesitamos buscar en la partición de datos actual lo suficiente como para obtener los K vecinos más cercanos. Primero se obtiene el nodo hoja donde se encuentra *q* referenciado (u almacenado) y se busca hacia el punto de referencia de la partición (*Inward*) y hacia a fuera (*Outward*) del mismo.

2. ***Que el punto de consulta q esté fuera de la partición actual pero que la esfera de consulta de q se cruza en parte con dicha partición de la base de datos.*** Cuando el *iDistance* se encuentra en este caso solo necesita buscar hacia el punto de referencia de la partición actual.

3. ***Que la partición actual no se cruce con la esfera de consulta.*** Es en este caso, cuando el *iDistance* descarta el examinar dicha partición agilizando así, el proceso de búsqueda.

El algoritmo de búsqueda termina cuando se han encontrado los K vecinos más cercanos en las particiones o secciones de datos las cuales se cruzaban con la esfera de búsqueda centrada en *q*. En el momento que se alargue el radio de búsqueda y no se modifique el conjunto *kNN* podemos estar seguros de que los K vecinos más cercanos a *q* han sido encontrados. Por tanto todos los puntos que estén fuera de la intersección de las particiones de la base de datos sobre la esfera de búsqueda centrada *q*, estarán a una distancia mayor de *q* que el propio radio de la esfera de búsqueda *(querydist(q))* lo que conlleva a asegurar que por más que alarguemos la esfera en busca de otros puntos no vamos a encontrar ***mejores (a menor distancia)*** vecinos de *q* de los encontrados ya en las particiones que si se intersecaban con la esfera de búsqueda.

Para finalizar, se hace evidente que la respuesta del *iDistance* en cuanto a la consulta KNNQuery es determinista y por tanto tiene una precisión del 100% con la excepción de los falsos positivos mencionados anteriormente.

3.5 Distribución del espacio dim-dimensional de datos

Una vez definida la estructuración de los datos en el *iDistance* así como su búsqueda para los KNN, es necesario argumentar las estrategias existentes para la división del espacio de datos. La división de la base de datos (*S*) en particiones requiere, por tanto, el cálculo del punto de referencia O_i asociado a la partición o sección de los datos P_i correspondiente. En la división de datos altamente dimensionales existen diversas técnicas para realizar una clasificación efectiva para cada problema a tratar. En [Cui02 y JOT^{+}05] se proponen dos técnicas para realizar la distribución de los puntos *dim*-dimensionales:

- *División basada en el espacio de datos* M^{dim}.
- *División basada en el tipo de datos en S.*

En la implementación se ha optado por la segunda técnica aunque vamos a detallar las dos estrategias en el orden que se ha establecido anteriormente.

3.5.1 División de los datos basado en el espacio métrico M^{dim}

Para la distribución del espacio de datos, una buena fórmula seria la división del espacio en particiones iguales. Por lo tanto si tenemos un espacio E^{dim}, podemos definir 2^{dim} hiperplanos de división. Como ejemplo, siguiendo esta técnica en un espacio bidimensional ***normalizado entre 0 y 1***, podemos dividir el espacio en triángulos isósceles con los centros de referencia el punto medio del lado largo del triángulo (centro del hiperplano). La Figura 14 muestra cómo quedaría el espacio de datos bidimensional anteriormente citado.

Una vez establecida la lógica de división del espacio de datos veamos cómo podemos obtener los puntos de referencia y qué tamaño puede tener cada partición. Con esto, nos referimos a que según se establezcan los puntos de referencia y el radio de referencia de cada partición, puede ser que una partición se solape con otra, es decir, que dadas dos particiones P_i y P_j de S con $i \neq j$, se tiene que $P_i \cap P_j \neq \emptyset$.

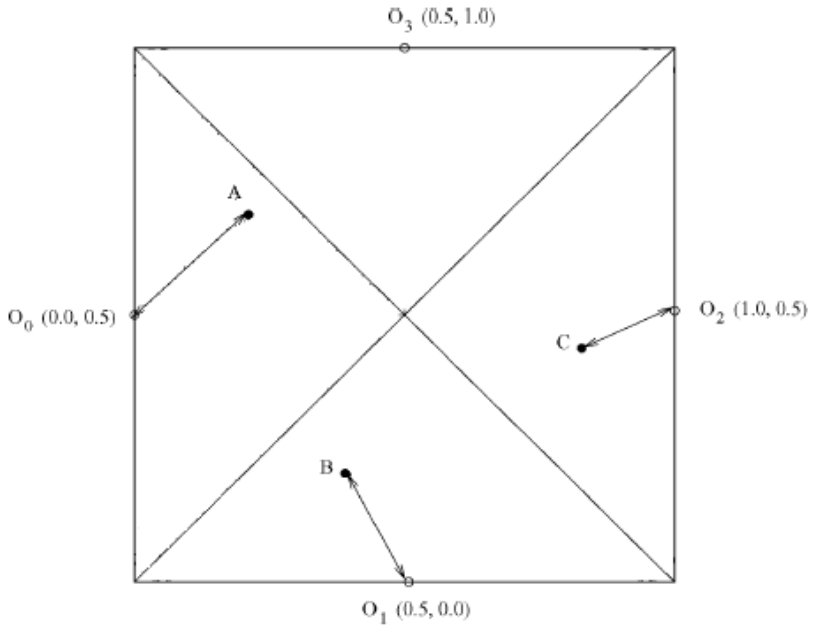


Figura 14. División del espacio bidimensional en 4 hiperplanos triangulares.

Para el tratamiento de las esferas de referencia de la partición P_i (*RefSphere(P_i)*), podemos escoger entre tres posibilidades:

1. ***Punto de referencia en el centro del hiperplano y distancia mínima (DistMax) para englobar todos los puntos de la partición.*** En este caso (Figura 15), estaríamos hablando del mismo sistema que se ilustró en la Figura 14, en la cual se establece como punto de referencia el centro del hiperplano de cada partición y el radio máximo de la esfera de referencia de cada partición *RefSphere(P_i)* es la altura del triángulo (hiperplano).

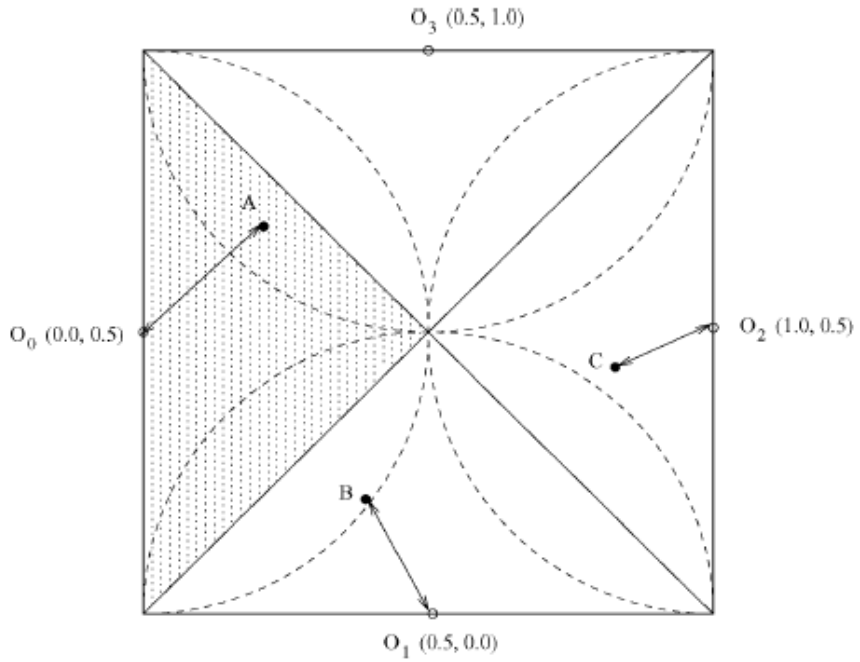


Figura 15. [JOT+05] Puntos de referencia en el centro del hiperplano con el radio de referencia mínimo.

El espacio de datos quedó dividió en secciones disjuntas aunque las esferas de referencia se superpongan, es decir, que dadas dos particiones P_i y P_j en S para todo i, j en $\{0, 1, 2, 3\}$ tal que $i \neq j$, se tiene que $P_i \cap P_j \neq \emptyset$.

En la Figura 16, se considera un punto de consulta q con un radio de búsqueda r.

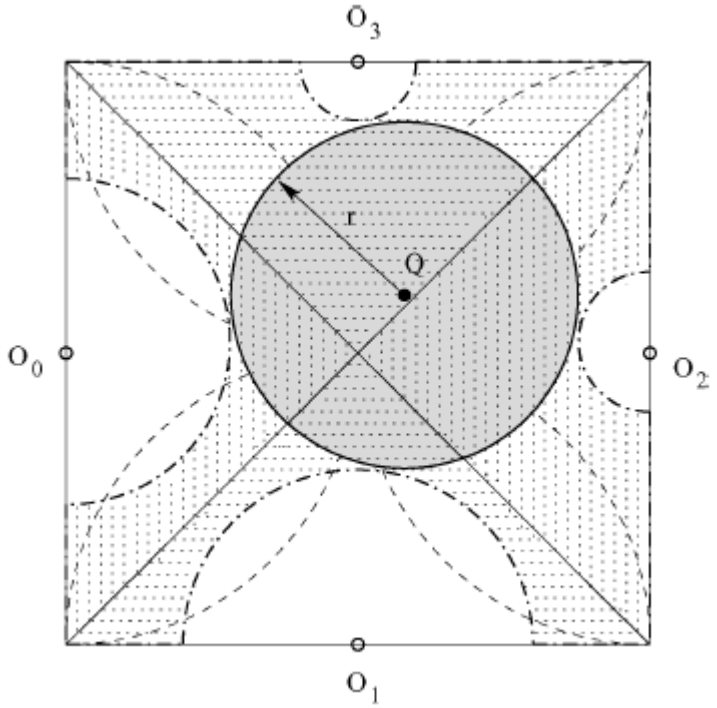


Figura 16. [JOT+05] Área de consulta para un punto Q estableciendo los puntos de referencia en el centro del hiperplano con el radio de referencia mínimo.

Toda el área marcada de cada sección debe de ser revisada para obtener el conjunto *kNN*. Sin embargo, dada la pérdida de precisión del *iDistance* al indexar datos altamente dimensionales a una sola dimensión basada en la distancia, hay puntos que quedan fuera del espacio de consulta y que deberán ser consultados en el momento de procesar los puntos con el mismo radio que sí se solapan con la esfera de consulta de q en la partición de datos correspondiente.

2. ***Punto de referencia en el centro del hiperplano y distancia máxima (DistMax) para la cual la partición engloba TODOS los puntos que estén más lejanos a dicha partición.*** En la Figura 17 podemos observar como *RefSphere*(P_1) engloba los puntos más lejanos a ella del espacio de datos.

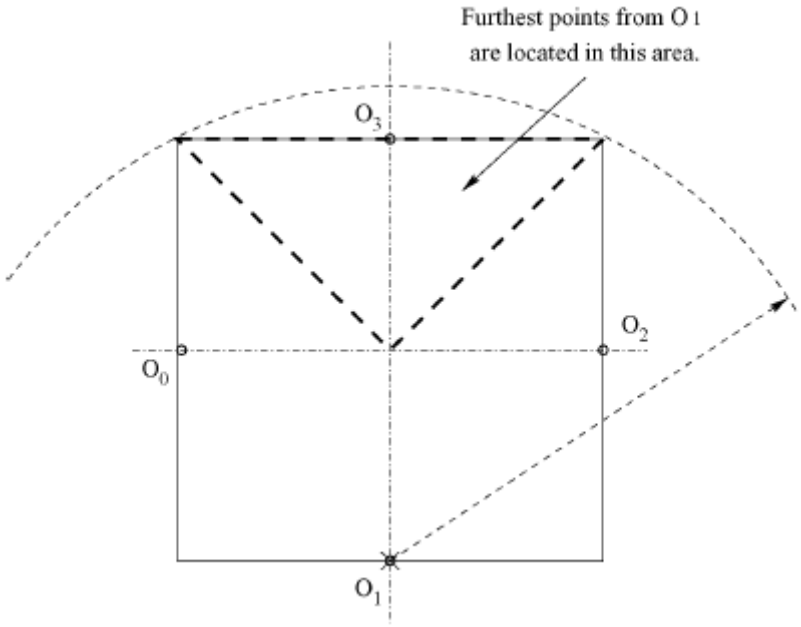


Figura 17. [JOT+05] Punto de referencia en el centro del hiperplano con el radio de referencia máximo.

Como se puede observar, al igual que el apartado (1), la línea de puntos discontinua (*RefSphere(P_1)*) engloba los puntos más lejanos de la partición 1. Si consideramos un punto de consulta q, el área sombreada en la Figura 18 es el que debía ser consultado en el sistema anterior (1), mientras que en este sistema (2) tan sólo se ha de consultar el área delimitada por los arcos de referencia de cada sección (*RefSphere(P_i)*) delimitados por los arcos en negrita de la Figura 18:

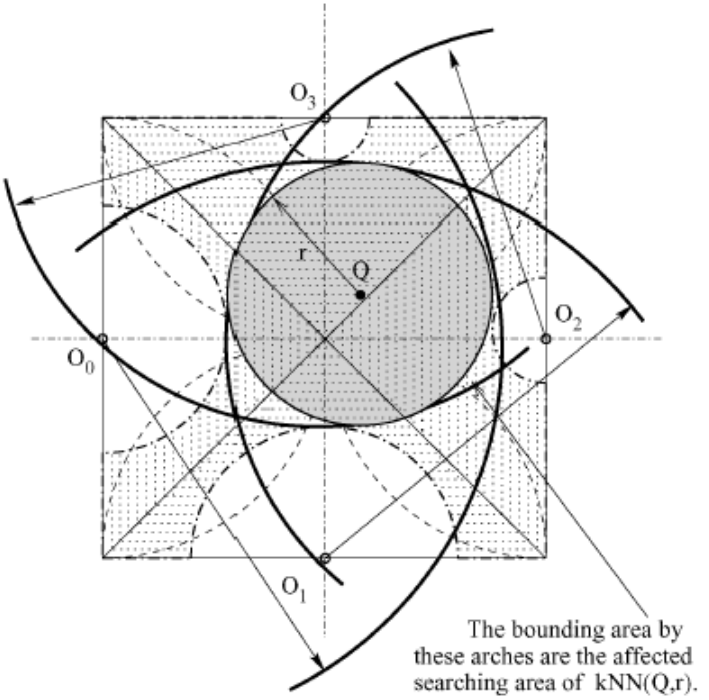


Figura 18. [JOT+05] Comparación del área de consulta de Q, seleccionando el radio máximo, contra el radio mínimos de la Figura 15.

Si observamos el área de búsqueda actual, mejora bastante en comparación con el sistema anterior aunque dicha mejora siempre está en dependencia directa con la elección correcta de los puntos de referencia y la situación del punto de consulta *q*.

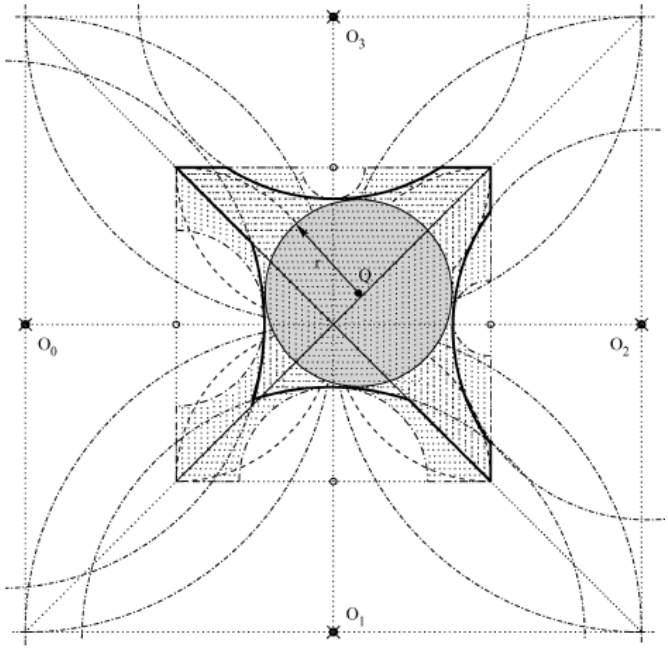


Figura 19. [JOT+05] Comparativa entre seleccionar los puntos de referencia como los centros del hiperplano o escoger puntos exteriores.

3. ***Punto Exterior. Cualquier punto que se encuentre en la línea entre el centro del hiperplano y el centro del espacio de datos***. Con esta elección, estamos considerando datos que quedan fuera del espacio de datos y por tanto esta técnica tiene buenos resultados cuando el área de búsqueda es muy grande sobre todo con datos uniformes. En la Figura 19 podemos observar una comparativa entre escoger como puntos referencia los centros de los hiperplanos que conforman el espacio de datos y coger puntos exteriores. Podemos observar como para una misma área de consulta del punto Q, la elección de puntos exteriores reduce el espacio de búsqueda frente a la elección de los centros de cada hiperplano.

3.5.2 División de los datos basado en las características de S.

Esta otra estrategia de reparto del espacio de datos en secciones se ajusta mejor a los datos de la vida real, puesto que no están uniformemente distribuidos. La división del apartado anterior (respecto del espacio E^{dim}) funciona bien cuando los datos están siguiendo una distribución uniforme en dicho espacio, pero en realidad, en raras ocasiones estos van a ser uniformes. Normalmente los datos suelen estar correlacionados, incluso cuando no existe dicha correlación en todas las dimensiones del espacio, dichos datos pueden estar asociados de manera local. Por ello, se estudia la posibilidad de introducir un algoritmo de identificación de clústeres para obtener con el error deseado cuales son los centros o puntos de referencia de los M clústeres en los que se desean dividir el espacio de datos.

Como ya se citó en el capítulo de fundamentos, en este proyecto se apuesta por un algoritmo de clustering (*kmeans*) [KMN+02], más comúnmente llamado Algoritmo de Lloyd para el cálculo de las k medias. En la implementación, se ha utilizado el cálculo de las k-medias (*kmeans*) ya que en [JOT+05] basan el cálculo de los puntos de referencia (O_i) en dicho algoritmo.

Respecto al método de clustering, este no afecta a la indexación y búsqueda del *iDistance*, pero sí que interviene en el rendimiento en las consultas de los K vecinos más próximos en cuanto a tiempo y accesos a disco. Es decir, afecta al número de recorridos de la búsqueda y el número de nodos hoja. Cuanto menor sea el error del cálculo de los centros (más iteraciones del algoritmo de clustering) más eficiencia presentara el *iDistance*. En el siguiente apartado: estudio *del comportamiento del iDistance*, veremos cómo se comporta el *iDistance* variando el número de centros y la calidad de estos.

Selección de los puntos de referencia

Una vez obtenidos los clústeres o particiones de los datos, se han de establecer los puntos de referencia del *iDistance* de cada partición. Para ello [JOT+05] propone dos posibles soluciones:

1. ***Centro del clúster*.** El centro del clúster es el candidato natural para la elección del punto de referencia de dicho clúster (sección o partición). En la Figura 20 podemos ver el resultado en un espacio bidimensional con dos secciones o clústeres.

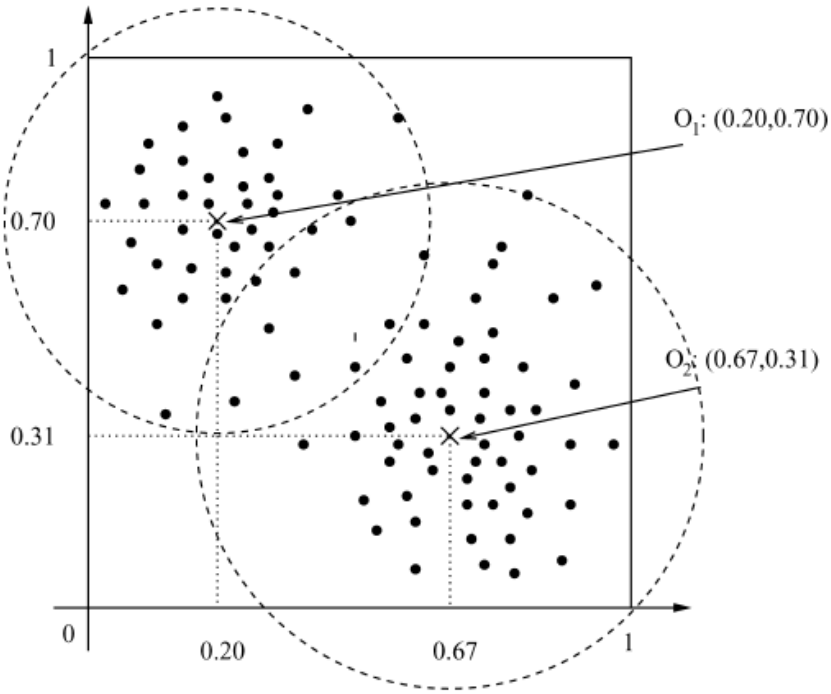


Figura 20. [JOT+05] Selección como puntos de referencia a los centros de cada clúster.

Este planteamiento es el que se ha seguido para la implementación del *iDistance* y su adaptación al sistema de video basado en contenido que trata el proyecto. Dicha estrategia, tiene la ventaja de que se pueden obtener centros con la exactitud que se desee dándole al algoritmo de clustering un error más pequeño, lo que es lo mismo,

realizar el cálculo de los centros con más iteraciones en el algoritmo de *kmeans* y con esto se consigue mayor eficiencia en la indexación y búsqueda del *iDistance*.

La principal desventaja es el tiempo de cálculo que conlleva determinar los clústeres con demasiada precisión. Aunque esto no es de extrema importancia puesto que una vez calculados los puntos de referencia la primera vez (antes de construir el árbol B+) ya no es necesario volver a calcularlos y son válidos para cualquier consulta posterior, siempre y cuando no se inserten lo suficientes puntos nuevos en el espacio de datos como para distorsionar o modificar la precisión de los centros calculados en demasía.

2. ***Esquina del clúster o partición.*** Consiste en la selección de los puntos de referencia como las esquinas o bases algebraicas de cada dimensión. Como se puede ver en la Figura 20 tenemos dos dimensiones, las esquinas o bases de cada dimensión serían {(0,1) y (1,0)}. En la Figura 18, podemos observar el solape existente entre las dos esferas de cada partición de datos. Para solucionar este solape, podemos seleccionar los bordes de cada clúster de manera que siguiendo el ejemplo de la Figura 20. En la Figura 21 podemos ver como se reduce considerablemente el cruce de las dos particiones.

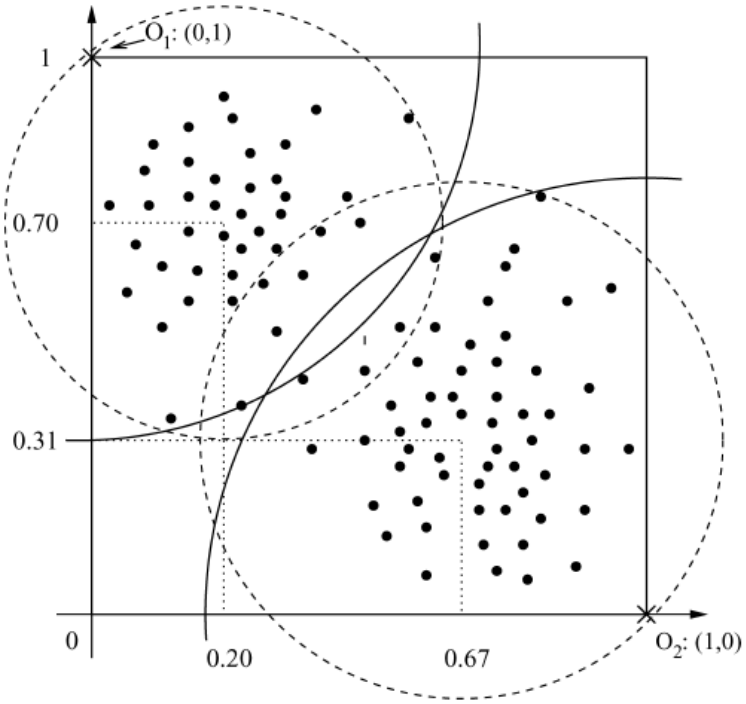


Figura 21. [JOT+05] Selección de puntos de referencia en las esquinas de cada partición.

Como se observa en la comparativa realizada en la imagen anterior, el área de solape con los puntos de referencia establecidos en los bordes de cada clúster es menor que si eligen como referencia los centros del clúster.

Por tanto, cuanto más solape haya entre las particiones, más área de datos solapada habrá en la esfera de consulta, lo que conllevará a más puntos que tengan la misma similitud (distancia) y que por consiguiente haya que examinar. En este sentido sería mejor escoger las esquinas de cada clúster, pero si nos fijamos en la Figura 21, las dos esquinas contrarias quedan sin cubrir y por tanto si hubiera puntos en dichas esquinas no estarían referenciados en el índice.

Esto se solucionaría ampliando la distancia máxima de cada partición pero entonces estaríamos volviendo a crear mucha área de solape y por tanto la elección de los centros del clúster presentaría mejores resultados.

3.6 Conclusiones

Como estrategia para vencer la maldición de la dimensionalidad, podemos terminar este capítulo, prestando atención en las características del *iDistance* para la indexación de datos altamente dimensionales y las numerosas posibilidades de configuración que presentan las diferentes estrategias de selección de puntos de referencia y división del espacio de datos, adaptándose a sistemas de datos distribuidos uniformemente o no. Como contrapunto del *iDistance*, está la perdida de precisión a la hora de transformar un espacio *dim*-dimensional a uno unidimensional basado en distancia, lo que conlleva falsos positivos a la hora de realizar la consulta KNN. Como se comentó anteriormente, para puntos con igual distancia a un mismo punto de referencia presentan el mismo valor (clave) de indexación y por supuesto dichos puntos no tienen las mismos valores en el espacio *dim*-dimensional al que pertenecen.

Si nos fijamos en la Figura 22, podemos observar en un espacio 5-dimensional, como dos puntos distintos presentan el mismo valor de indexación basado en la distancia (en este caso máxima) con el mismo punto de referencia. Sean $a= (0.1, 0.2, 0.7, 0.1, 0.4)$, $b= (0.3, 0.2, 0.3, 0.1, 0.7)$ puntos de la base de datos y $O_i= (0, 0.1, 0, 0.4, 0)$ el punto de referencia de P_i, donde están indexados los dos puntos, si y_1 e y_2 son los valores de indexación calculados por el *iDistance* mediante las siguientes formulas:

$$y_1= i \times C_i + dist(p_1, O_i) \rightarrow y_1= i \times C_i + 0.7$$

$$y_2= i \times C_i + dist(p_2, O_i) \rightarrow y_2= i \times C_i + 0.7$$

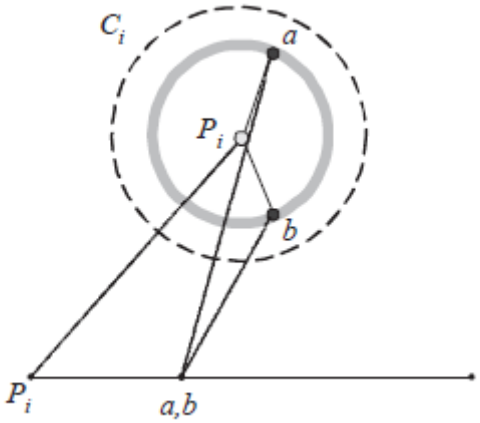


Figura 22. [Del07] Ejemplo de la pérdida de precisión del *iDistance*.

De las anteriores ecuaciones se puede observar que para la partición (i) P_i, los puntos a y b tienen el mismo valor de indexación ($y_1==y_2$) mientras que en E^5 (espacio de datos) dichos puntos están separados y por supuesto no son iguales.

En cuanto al punto de vista de sistemas de gestión de bases de datos multidimensionales comerciales, el *iDistance* presenta, una ventaja frente a otros métodos de indexación puesto que en la mayoría de estos sistemas se utiliza como índice un árbol B+, que es la base sobre la que se sustenta todo el *iDistance*.

En el siguiente capítulo, se estudiará el comportamiento del *iDistance* sobre datos de 12, 80 y 128 dimensiones.

4

Resultados experimentales con el iDistance

4.1 Introducción

En este capítulo, vamos a estudiar el comportamiento del *iDistance* con datos correlacionados de secuencias de video de distintas dimensiones. En primer lugar, estudiaremos el proceso de creación del *iDistance* para distintas dimensiones, para distinto número de centros y factor de compactación del árbol B+. Tras dicho estudio, veremos el comportamiento del *iDistance* con la variación de K, la dimensión de los datos y el número de centros o puntos de referencia del espacio de datos.

Para promediar de manera fiable los tiempos de cada consulta, se escoge un conjunto de puntos del archivo de datos Q el cual contendrá los puntos de consulta. Para cada punto de consulta $q_i \in Q$, se realizara la **búsqueda de los K vecinos más cercanos**, sacando el promedio entre las |Q| peticiones al iDistance.

En cuanto al tamaño de página para la lectura de datos desde disco, se va a mantener en ***4096*** bytes en todas las ejecuciones de este apartado, con la excepción de la comparativa con el *VA-File*, en la cual se variará el tamaño de página en las dos estructuras.

Se van a utilizar tres tipos de datos ***no uniformes*** para realizar el estudio. Dichos datos corresponden a puntos *dim*-dimensionales con las características de los *key-frames* asociados a secuencias de video [Riv10]. En concreto, se dispone de tres archivos de distintas secuencias de 12, 80 y 128 dimensiones cada uno y el número de puntos en los tres archivos es de 125378 registros o vectores. Dichos datos provienen de videos reales obtenidos desde satélite de diversos canales de televisión como ABC, CBN, CCTV, etc. Los videos fueron comprimidos en MPEG-7 para la obtención de los *key-frames* y posteriormente se generaron sus vectores de características. Comentar que el archivo de 80 dimensiones, tiene naturaleza discreta y los resultados experimentales pueden verse afectados con respecto de los demás archivos.

Por último, para el cálculo de los puntos de referencia, se va a seguir la estrategia de selección teniendo en cuenta la distribución de los datos utilizando el algoritmo de clustering citado en el capítulo 2: *el algoritmo de Lloyd* para el cálculo de las k medias. Los puntos de referencia serán los centros de cada clúster calculado. Se sigue dicha estrategia puesto que en [JOT+05] se comprueba que dicha distribución de los datos es la que mejores resultados presenta para datos no uniformes.

4.2 Parámetros influyentes en los experimentos del iDistance

En la construcción del *iDistance* intervienen cuatro factores claves, de los cuales dependerá el mejor o peor rendimiento del *iDistance*:

- *La **dimensión (Dim)** del espacio de datos.*
- ***La cantidad de datos (NP).***
- *El **número de puntos de referencia (CN)** calculados.*
- *El **factor de compactación (CompactRate)** del árbol B+.*

En la consulta de los K vecinos más cercanos influyen además de los anteriores parámetros, estos otros:

- *La **calidad de los puntos de referencia** calculados. Iteraciones (S) del algoritmo de Lloyd.*
- *El **número de puntos de consulta |Q|.** Donde Q es un subconjunto de los datos de la base de datos.*
- *El **número de vecinos a calcular K** para cada punto de consulta en Q.*
- *El **tamaño de página (PAGE_SIZE)** o buffer para la lectura en disco.*
- *La **distancia utilizada**. (Para el estudio individual del iDistance, será Euclidea [JOT+05]). En la comparación con el VA-File se utilizaran la Euclidea y la Máxima.*

4.3 Creación del índice iDistance

***Experimento 1:** Creación del iDistance variando CN y Dim*

En este primer experimento, vamos a estudiar la creación del iDistance para variaciones de la dimensión y del número de centros. Manteniendo fijo el factor de compactación.

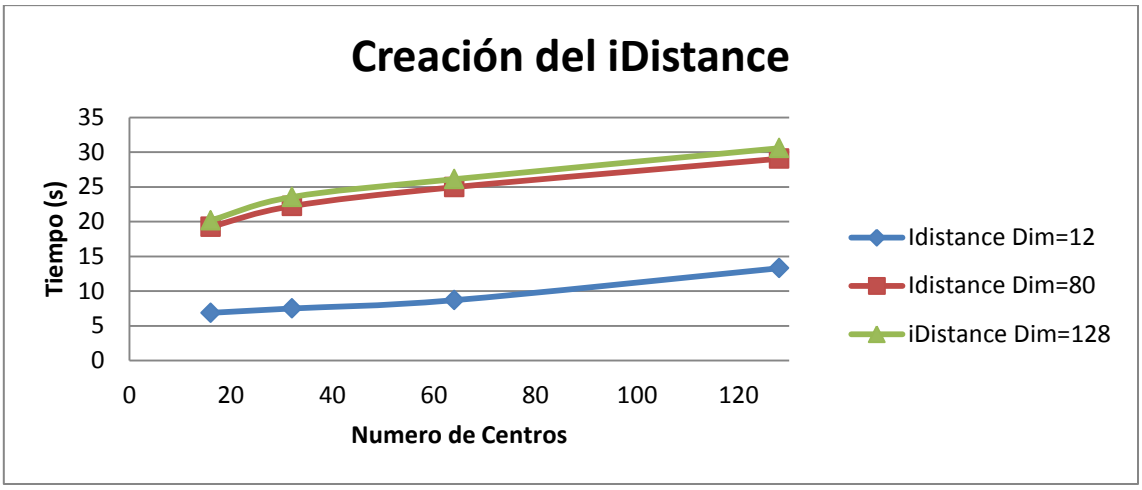
Variable	Valor
Dim	*12, 80 y 128*
CN	*16, 32, 64 y 128*
PageSize	*4096*
NP	*125378*
CompactRate	*90%*

Tabla 2. Experimento 1: estudio el tiempo de creación del *iDistance* variando el número de puntos de referencia y la dimensión de los datos.

Los resultados son los siguientes:

Dim	CN	Tiempo de Creación (s)
12	16	6,876
	32	7,505
	64	8,701
	128	13,295
80	16	19,257
	32	22,231
	64	24,965
	128	29,055
128	16	20,179
	32	23,54
	64	26,11
	128	30,57

Tabla 3. Resultados del experimento 1



Gráfica 1. Resultados del experimento 1

Según los resultados de la Gráfica 1, podemos observar que a mayor dimensión más tiempo de respuesta, esto es así por el procesado que debe hacer el *iDistance* en el cálculo de distancias de datos altamente dimensionales, como sucede en los datos con dimensión 128. También puede observarse como para dimensiones mayores o iguales a 80 el tiempo no varía demasiado. Se hace evidente, pues, que para mayor número de puntos de referencia (CN) el *iDistance* debe de comparar las distancias a cada centro.

Experimento 2: *Tiempo de creación del iDistance variando el factor de compactación*

A continuación, se propone estudiar la variación del factor de compactación con los siguientes valores:

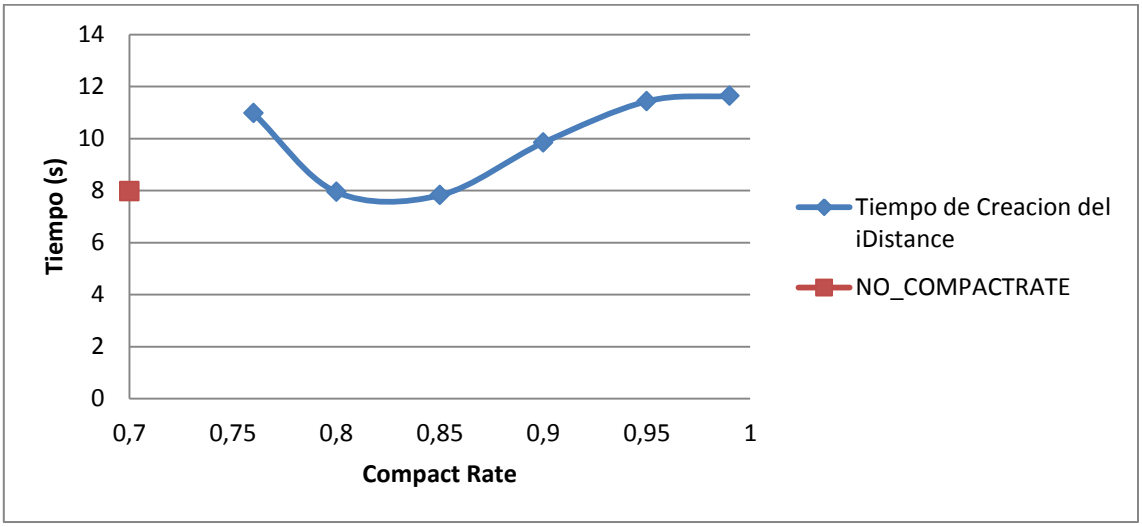
Variable	**Valor**
Dim	*12*
CN	*64*
PageSize	*4096*
NP	*125378*
CompactRate	*76%, 80%, 85%, 90%, 95%, 99% y -1*

Tabla 4. Experimento 2: estudio del tiempo de creación del *iDistance* variando el factor de compactación.

Préstese especial atención cuando el valor de compactación es -1 puesto que eso indica que NO se aplica dicha compactación. En la gráfica 2 se muestra mediante un punto rojo.

Compact Rate (%)	Tiempo de Creación (s)
-1	7,97
76	10,97
80	7,95
85	7,82
90	9,84
95	11,43
99	11,64

Tabla 5. Resultados del experimento 2.



Gráfica 2. Resultados del experimento 2.

Como se puede observar, para valores demasiado grandes de compactación en el árbol, el *iDistance* no mejora al tiempo de creación sin compactar. Un valor aceptable de compactación puede ser 0,85 (85%). Mas adelante veremos, en los siguientes experimentos, la importancia de la compactación del árbol.

A continuación, vamos a estudiar el comportamiento del iDistance en la consulta de los K vecinos más cercanos a un punto q de consulta dado.

4.4 Eficiencia del iDistance con la variación de la Dimensión y K

Experimento 3: *Eficiencia del iDistance en la consulta KNN variando Dim y K.*

Vamos a estudiar el tiempo de respuesta por consulta medio para |Q|=100 consultas con 100, 200, 500 y 800 vecinos por consulta. Todo ello para cada archivo de datos con diferentes dimensiones: 12, 80 y 128. Conservamos fijos CN, PageSize, NP, Q y el factor de compactación.

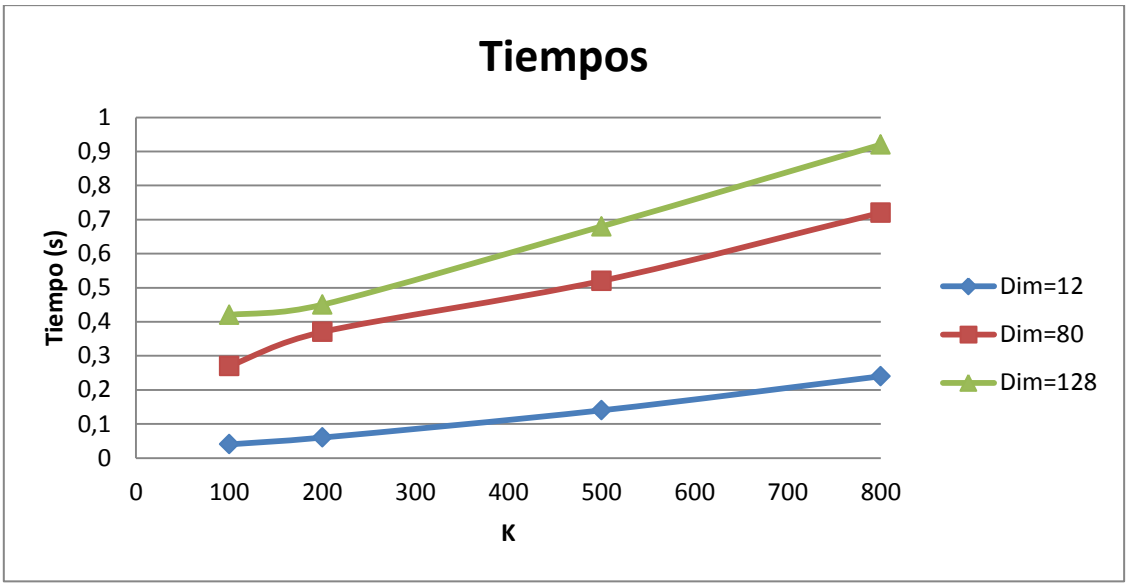
Variable	**Valor**
Dim	*12, 80 y 128*
CN	*64*
PageSize	*4096*
NP	*125378*
CompactRate	*85%*
|Q|	*100*
K	*100, 200, 500 y 800*

Tabla 6. Experimento 3: Eficiencia del *iDistance* en la consulta KNN variando Dim y K.

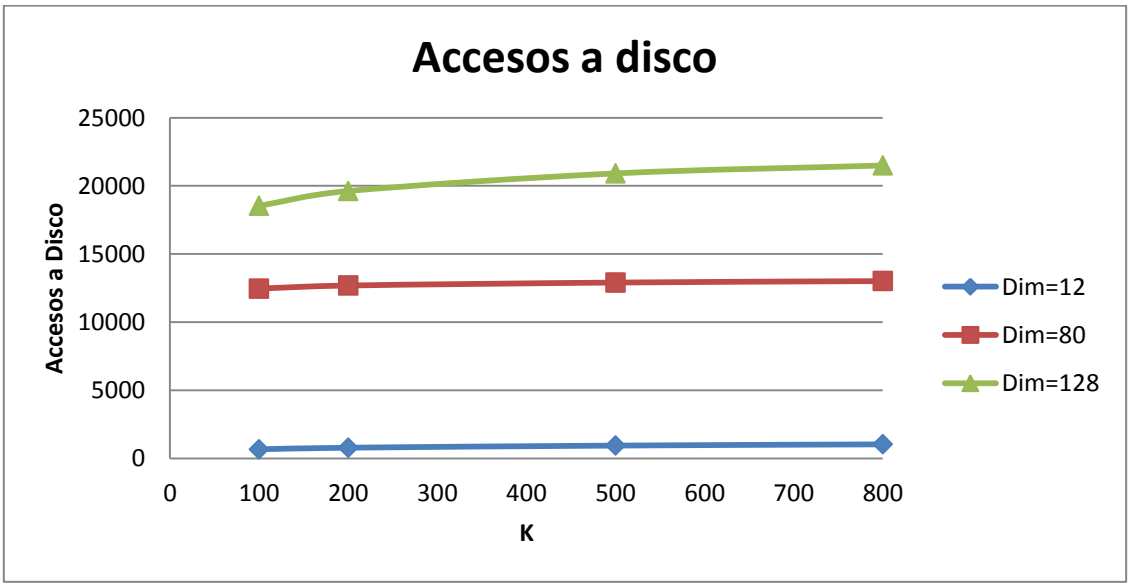
Los resultados obtenidos son los siguientes:

Dim	**K**	**Tiempo por consulta (s)**	**Accesos a disco por consulta**
12	100	0,04	667,113
	200	0,06	781,02
	500	0,14	936,011
	800	0,24	1026,651
80	100	0,27	12463,879
	200	0,37	12686,299
	500	0,52	12904,129
	800	0,72	13011,509
128	100	0,42	18531,99
	200	0,45	19624,419
	500	0,68	20915,96
	800	0,92	21495,89

Tabla 7. Resultados del experimento 3.



Gráfica 3. Resultados del experimento 3: tiempos por consulta.



Gráfica 4. Resultados del experimento 3: accesos a disco por consulta

De las Gráficas 3 y 4 podemos deducir que a mayor dimensión y número de vecinos a recuperar, el *iDistance* aumenta el tiempo de respuesta. Sin embargo, el número de accesos a disco parece no variar mucho en cada dimensión a un valor medio fijo.

Como se puede ver, poco puede optimizarse en función de estos factores. Lo único apreciable es estudiar el valor de K mínimo para que el sistema donde se integre el *iDistance* obtenga o empiece a obtener resultados aceptables.

4.5 Eficiencia del iDistance con la variación de CN y K

Experimento 4: *Eficiencia del iDistance en la consulta KNN variando CN y K*

En este experimento se persigue obtener la importancia de tener más o menos puntos de referencia en la consulta de los KNN en el *iDistance*. Por tanto, se va a dejar fija la dimensión a 12 y se variarán el número de puntos de referencia desde 32 hasta 128 y se comprobará con 100 puntos de consulta cómo se comporta el *iDistance* con K = {100, 200, 500 y 800}.

Variable	**Valor**
Dim	*12*
CN	*32, 64 y 128*
PageSize	*4096*
NP	*125378*
CompactRate	*85%*
|Q|	*100*
K	*100, 200, 500 y 800*

Tabla 8. Experimento 4: Eficiencia del *iDistance* en la consulta KNN variando CN y K

Los resultados obtenidos son los siguientes:

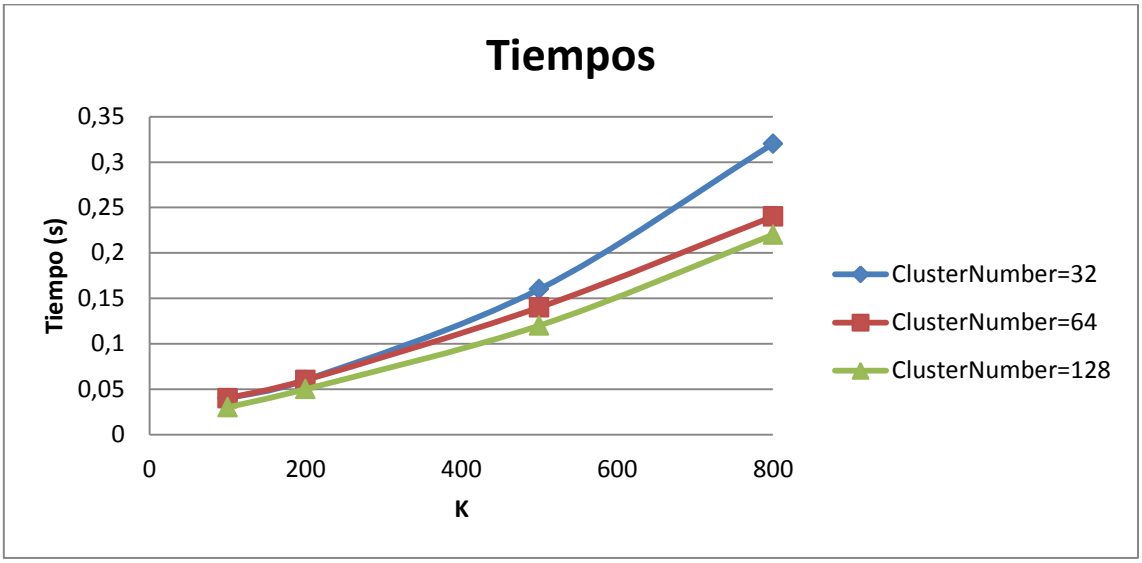
CN	**K**	**Tiempo por consulta (s)**	**Accesos a disco por consulta**
32	100	0,04	818,58
	200	0,06	936,729
	500	0,16	1113,469
	800	0,32	1211,301
64	100	0,04	677,13
	200	0,06	781,02
	500	0,14	936,32
	800	0,24	1026,679
128	100	0,03	706,969
	200	0,05	815,679
	500	0,12	991,03
	800	0,22	1092,109

Tabla 9. Resultados del experimento 4.

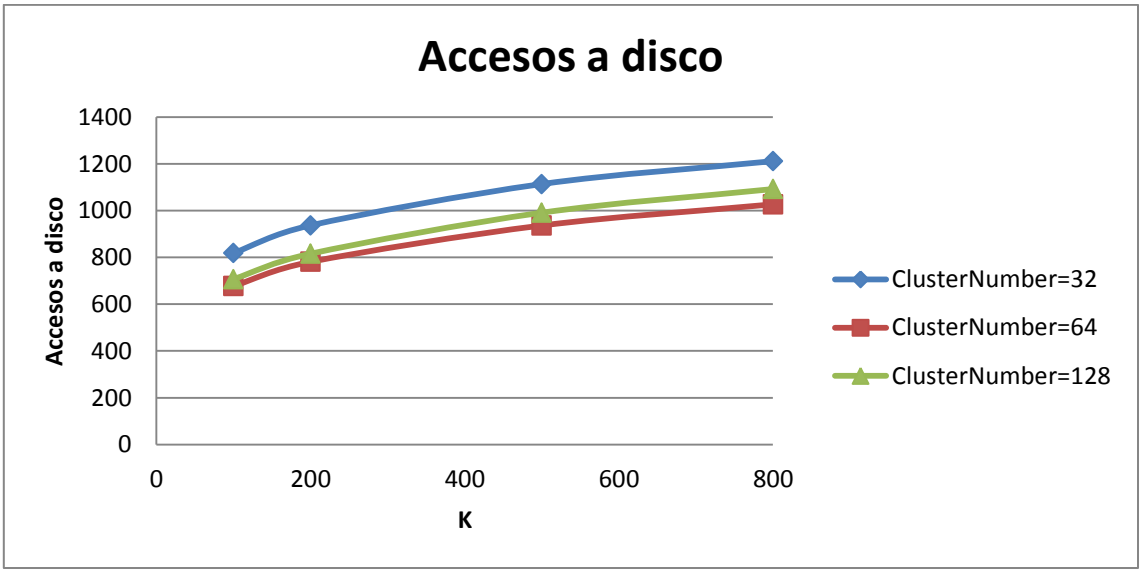
Como se puede observar en la tabla 9, cuantos más puntos de referencia se tengan mejor es la eficiencia del *iDistance* tanto en tiempo de respuesta por consulta como en los accesos a disco.

Por tanto, aunque como se vio en el experimento 1 el *iDistance* tarda más en construirse cuanto más puntos de referencia tenga, este tiempo se ve justificado a la hora de procesar las consultas de los K vecinos más cercanos puesto que el tiempo de respuesta mejora sustancialmente.

A continuación, en las Gráficas 5 y 6 podemos ver la mejora de utilizar más puntos de referencia:



Gráfica 5. Resultados del experimento 4: tiempo por consulta.



Gráfica 6. Resultados del experimento 4: accesos a disco por consulta.

Como se observa, para la misma dimensión (Dim=12) el *iDistance* con un mayor número de puntos de referencia tiene mejor eficiencia que uno con menos. Tras lo observado en los resultados, podemos concluir que un buen número de puntos de referencia podría ser 64, teniendo en cuenta el coste de calcular dichos puntos.

Experimento 5: *Eficiencia del iDistance en la consulta KNN variando Dim y CN.*

En este experimento se persigue obtener la influencia del número de puntos de referencia y la dimensionalidad de los datos en el tiempo de respuesta de la consulta KNN. Por tanto, se variará Dim y CN dejando fijos Q, K y el factor de compactación.

Variable	**Valor**
Dim	*12, 80 y 128*
CN	*16, 32, 64 y 128*
PageSize	*4096*
NP	*125378*
CompactRate	*85%*
|Q|	*100*
K	*800*

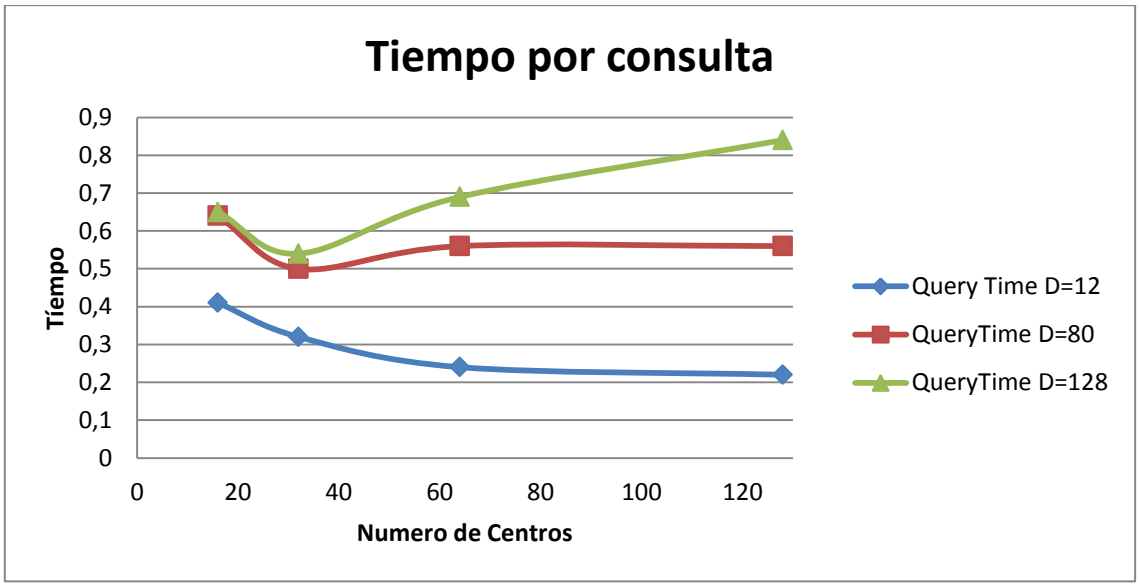
Tabla 10. Experimento 5: Eficiencia del *iDistance* en la consulta KNN variando Dim y CN.

Los resultados obtenidos son los siguientes:

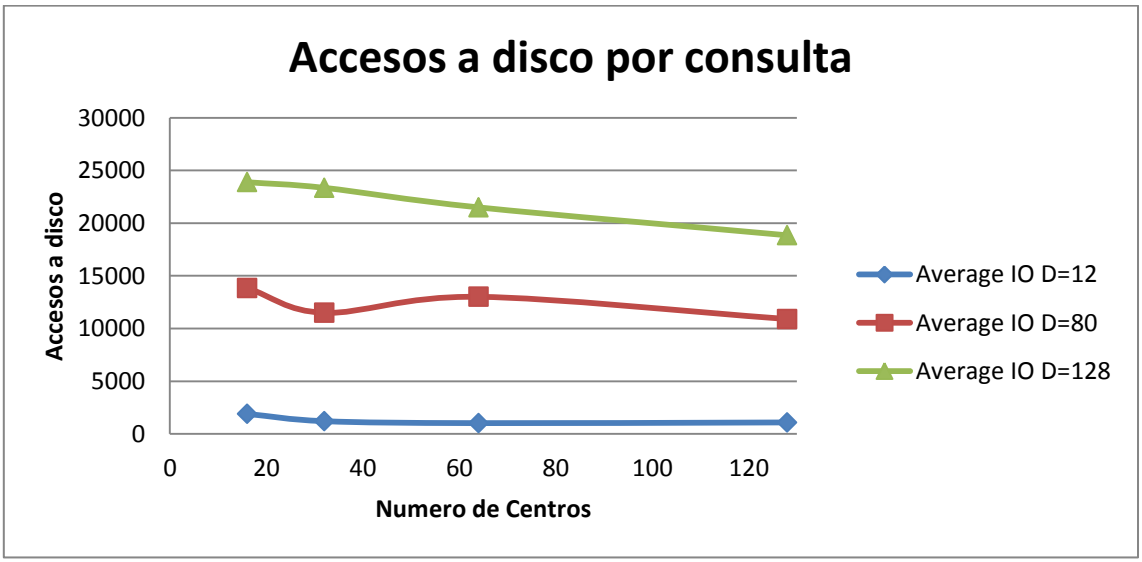
Dim	**CN**	**Tiempo por consulta (s)**	**Accesos a disco por consulta**
12	16	0,41	1902,101
	32	0,32	1211,301
	64	0,24	1026,679
	128	0,22	1092,109
80	16	0,64	13843,299
	32	0,5	11497,129
	64	0,56	13011,509
	128	0,56	10903,299
128	16	0,65	23883,929
	32	0,54	23352,83
	64	0,49	21495,89
	128	0,84	18850,23

Tabla 11. Resultados del experimento 5.

En este estudio se resalta la importancia del número de puntos de referencia a medida que aumenta la dimensión de los datos. Como se puede observar para datos de baja dimensionalidad (12) el aumento de puntos de referencia no es muy significativo en el tiempo de respuesta y en los accesos a disco. Sin embargo, para grandes dimensiones (128) se hace evidente la importancia de introducir un número suficiente de puntos de referencia para disminuir lo máximo posible los accesos a disco así como el tiempo de procesado por consulta.



Gráfica 7. Resultados del experimento 5: tiempo por consulta.



Gráfica 8. Resultados del experimento 5: accesos a disto por consulta.

De nuevo, podemos observar en las Gráficas 7 y 8, como para grandes dimensiones, el tener un número de puntos de referencia elevado (mayor que 32) mejora bastante el número de accesos a disco así como el tiempo de procesado por consulta.

4.6 Importancia de la calidad de los puntos de referencia calculados

En este apartado, vamos a estudiar como interviene el error del algoritmo de Lloyd (*kmeans*) a la hora de calcular los clústeres y los centros de los mismos. Dichos centros serán los puntos de referencia del *iDistance*. En la práctica, dicho error tiene una dependencia lineal con el número de iteraciones *S* del algoritmo de Lloyd a la hora de calcular los clústeres, por tanto vamos a estudiar como interviene la variación de *S* en la eficiencia del *iDistance*. Para dichos experimentos se va a mantener el número de centros (puntos de referencia) a 64, que es el valor que se estableció en el *experimento 4* como aceptable.

Experimento 6: *Tiempo de respuesta del algoritmo de Lloyd variando el número de iteraciones S y la dimensión de los datos.*

Parámetro	**Valor**
Dim	*12, 80 y 128*
S	*2, 10, 50 y 100*
NP (Número de Puntos)	*125378*
CN	*64*

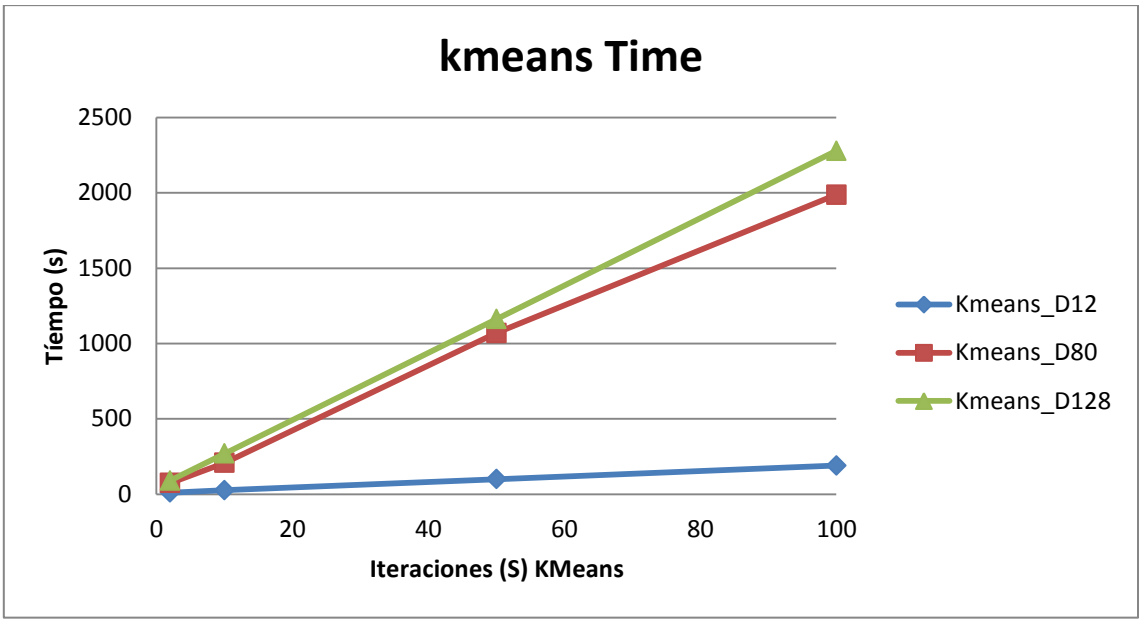
Tabla 12. Experimento 6: Tiempo de respuesta del algoritmo de Lloyd variando S y Dim.

Los resultados obtenidos son los siguientes:

Dim	**S**	**Tiempo Total (s)**
12	2	11
	10	27
	50	100
	100	190
80	2	75
	10	208
	50	1070
	100	1987
128	2	91
	10	270
	50	1162
	100	2278

Tabla 13. Resultados del experimento 6.

Como se puede apreciar en la Tabla 13 y en la Gráfica 9, el tiempo de respuesta del algoritmo de Lloyd tiene una correspondencia lineal con el número de iteraciones del algoritmo, es decir, cuanto más iteraciones el tiempo de respuesta consumido crece aproximadamente en igual proporción.



Gráfica 9. Resultados del experimento 6: tiempo de cómputo.

En este experimento se persigue denotar el coste computacional que conlleva el cálculo de unos buenos puntos de referencia mediante *kmeans*. Como se puede observar, cuanto más iteraciones más tiempo de respuesta. En los siguientes experimentos se estudiará si merece la pena dicho aceptar dicho coste computacional tras obtener el rendimiento del *iDistance*.

***Experimento 7**: Eficiencia del iDistance en la consulta KNN variando las iteraciones del algoritmo kmeans (S) y la dimensión de los datos.*

Como podemos ver en la Gráfica 9, el cálculo de buenos centros es muy costoso. Veamos en qué medida merece la pena dicho cálculo en el procesado del *iDistance* para distintos valores de S aumentando la dimensión de los datos. Se dejan fijos, por tanto, K y |Q|.

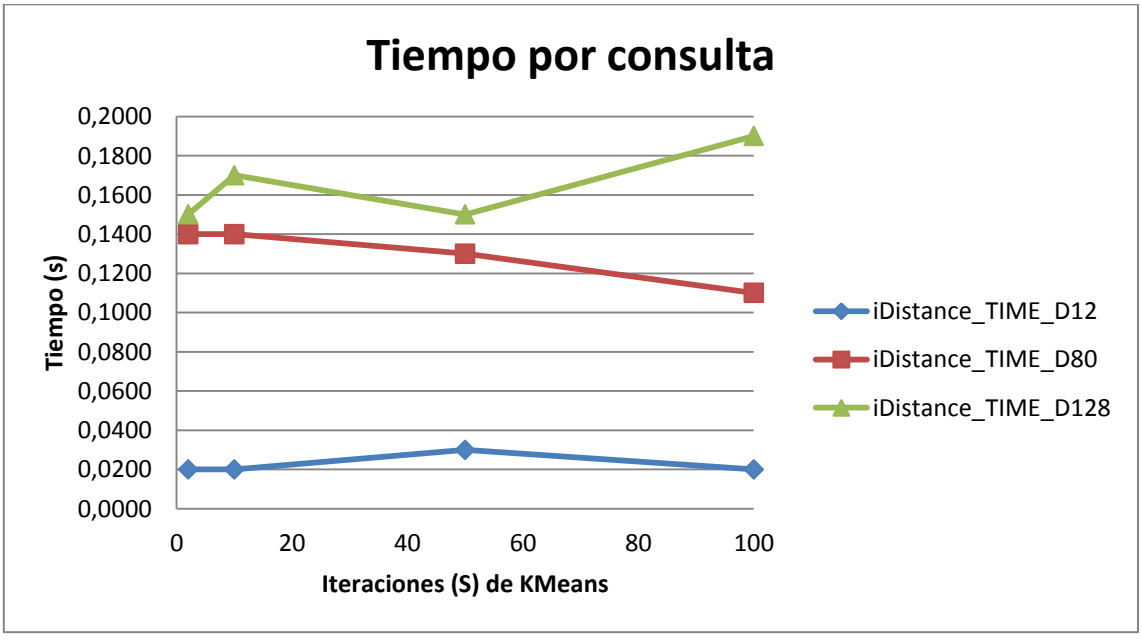
Variable	**Valor**
Dim	*12, 80 y 128*
CN	*64*
PageSize	*4096*
NP	*125378*
CompactRate	*85%*
|Q|	*100*
K	*100*
S	*2, 10, 50 y 100*

Tabla 14. Experimento 7: Eficiencia del *iDistance* en la consulta KNN variando S y Dim.

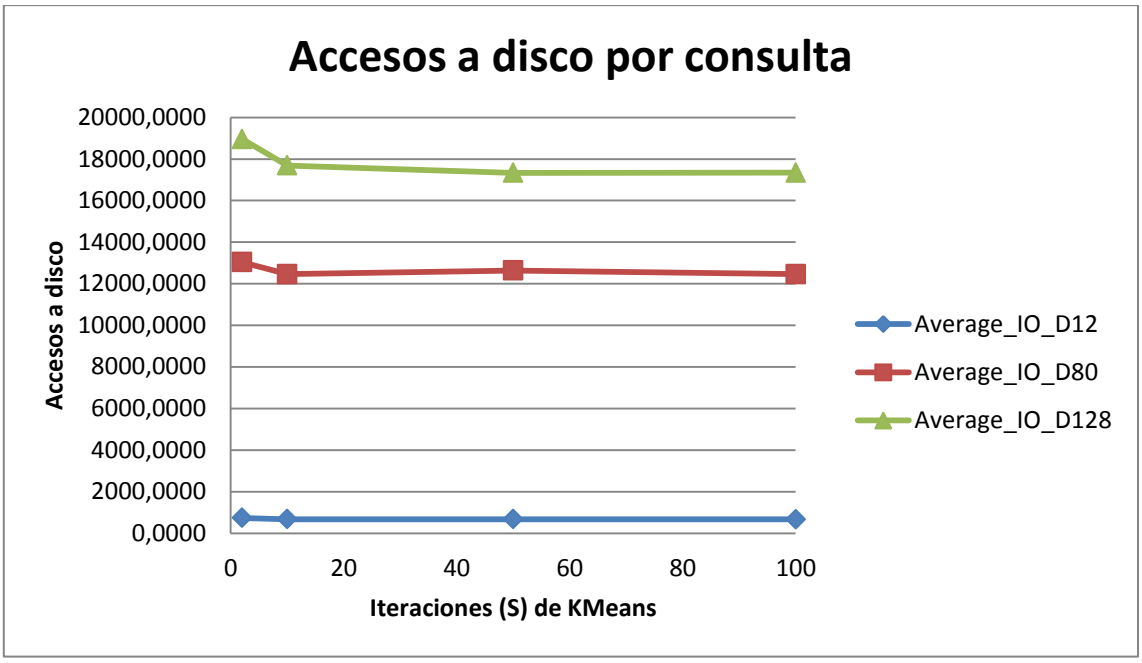
Los resultados obtenidos son los siguientes:

Dim	S	Tiempo por consulta (s)	Accesos a disco por consulta
12	*2*	0,0200	732,2700
	10	0,0200	670,6100
	50	0,0300	676,6600
	100	0,0200	666,3000
80	*2*	0,1400	13041,0801
	10	0,1400	12463,8799
	50	0,1300	12637,7500
	100	0,1100	12463,8799
128	*2*	0,1500	18959,8203
	10	0,1700	17690,2207
	50	0,1500	17333,6406
	100	0,1900	17341,0508

Tabla 15. Resultados del experimento 7.



Gráfica 10. Resultados del experimento 7: tiempo por consulta



Gráfica 11. Resultados del experimento 7: accesos a disco por consulta.

Si observamos los resultados mostrados en las Gráficas 10 y 11, así como la Tabla 15, podemos observar que para un mayor valor de *S* disminuyen el número de accesos a disco junto con el tiempo de procesado medio por consulta. Sin embargo, si observamos la evolución para valores de S superiores a 10 iteraciones, esta mejora no es apreciable. Si es cierto que hay mejora, pero dado el gran coste en tiempo que conlleva el algoritmo de Lloyd al aumentar S, no merece la pena dar valores a S superiores a 10. En nuestro caso estableceremos como valor por defecto S=10.

4.7 Estudio del factor de compactación

Experimento 8: *Eficiencia del iDistance en la consulta KNN variando el factor de compactación.*

Por último, en cuanto al estudio individual del *iDistance* se refiere, vamos a comprobar cómo afecta el factor de compactación a la hora de procesar o calcular los K vecinos más cercanos.

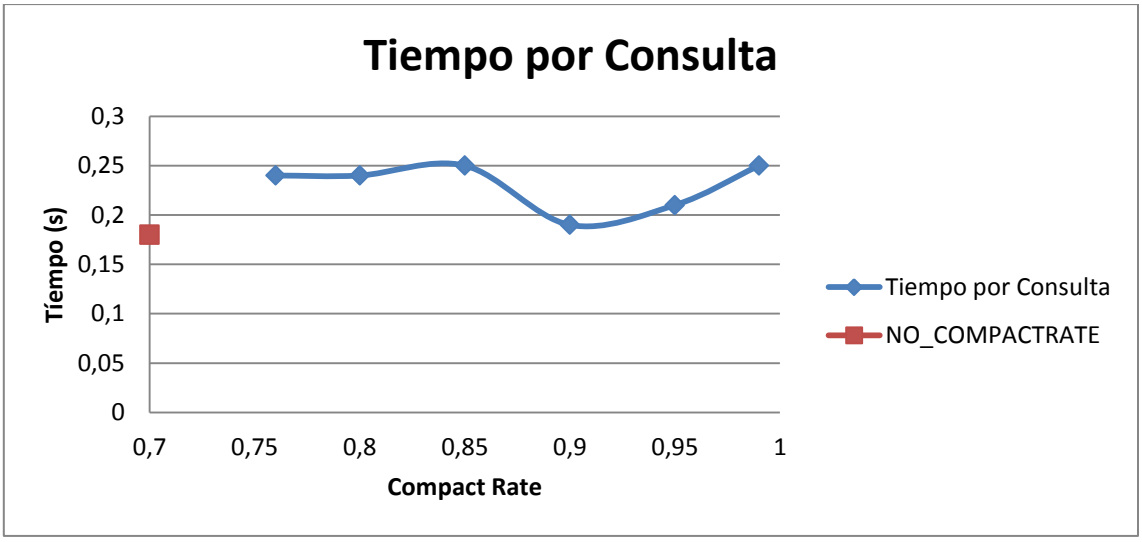
Variable	**Valor**
Dim	*12*
CN	*64*
PageSize	*4096*
NP	*125378*
CompactRate	*76%, 80%, 85%, 90%, 95%, 99% y -1*
|Q|	*100*
K	*800*
S	*10*

Tabla 16. Experimento 8: Eficiencia del *iDistance* variando el factor de compactación.

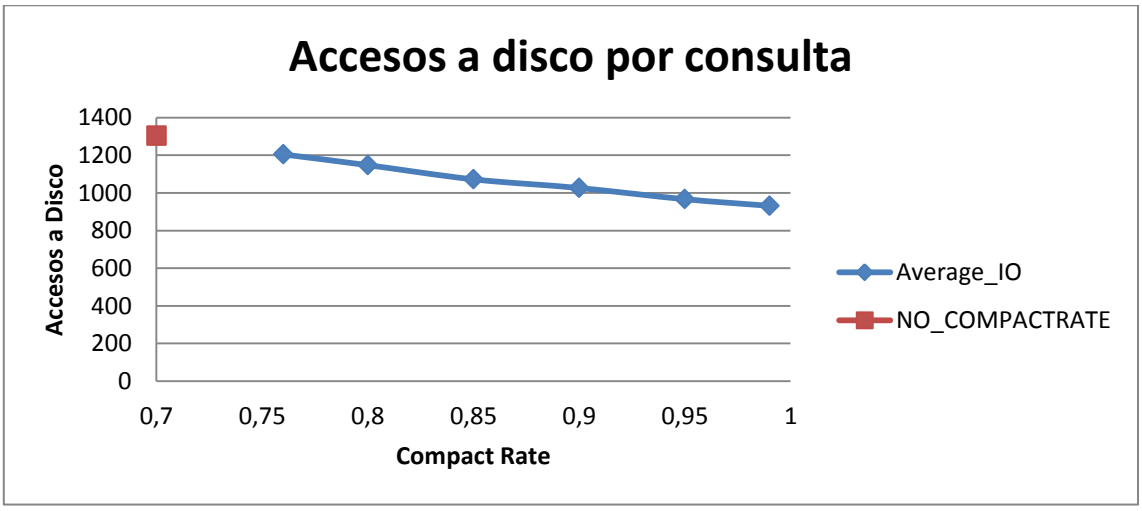
Los resultados obtenidos son los siguientes:

Compact Rate (%)	Tiempo por consulta (s)	Accesos a disco por consulta
-1	*0,18*	*1304*
76	0,24	1204,47
80	0,24	1147,41
85	0,25	1072
90	0,19	1026,09
95	0,21	966,539
99	0,25	931,37

Tabla 17. Resultados del experimento 8.



Gráfica 12. Resultados del experimento 8: tiempo de consulta



Gráfica 13. Resultados del experimento 8: accesos a disco por consulta

Si bien, el tiempo de procesado por consulta no presenta una mejora significativa e incluso es peor que el tiempo sin compactación, el número de accesos a disco recibe una mejora importante con respecto al factor de compactación (*CompactRate*) y mejora como es normal al valor obtenido sin compactar el árbol. Dicha mejora reduce al 71% los accesos a disco frente a los accesos obtenidos por el *iDistance* sin compactar. De este experimento y del experimento 2, podemos obtener un intervalo para el factor de compactación comprendido entre [85, 90] % con resultados más que aceptables.

4.8 Comparativa entre el iDistance y el VA-File

Con respecto a la indexación de datos altamente dimensionales, una parte de este proyecto se centra en el intento de vencer la maldición de la dimensionalidad mediante la reducción de la misma a la hora de crear un índice, concretamente el *iDistance*.

En este apartado se persigue comprobar la eficiencia de la reducción de la dimensionalidad frente a otras técnicas utilizadas para vencer *la maldición de la dimensionalidad* como puede ser: técnicas basadas en filtros y vectores de aproximación. Dentro de las técnicas basadas en filtros vamos a escoger la estructura de datos *VA-File* utilizada en [Riv10]. Vamos a comparar, pues, el comportamiento del *iDistance* para la consulta de los K vecinos más cercanos con el *VA-File*.

Con respecto al *VA-File* de [Riv10], este puede procesar la consulta de los K vecinos más cercanos (KNN) de dos formas posibles: con y sin procesado por lotes *(BNN, Batch Nearest Neighbors).* Dicho procesado reduce bastante el número de accesos a disco, pero puede suponer un mayor tiempo de pre-procesado. Es por tanto que se va a comparar el *iDistance* con el *VA-File* con y sin el procesado por lotes de los K vecinos más cercanos.

En los próximos tres experimentos (9, 10 y 11) se va a comprobar cómo afecta la variación del tamaño de página y la distancia utilizada para dos consultas idénticas en las dos técnicas propuestas (*iDistance* y *VA-File*). Hay que prestar especial atención en las distancias utilizadas, tanto en el *iDistance* como en el *VA-File*, para que las respuestas sean las mismas: distancias Euclidea y Máxima.

Experimento 9: *Rendimiento del iDistance frente al VA-File en la consulta KNN variando la Dimensión de los datos y K. El tamaño de página se disminuye a 1024 bytes y la distancia utilizada: Euclidea.*

Para el *iDistance* se mantiene el factor de compactación al 85% con 64 puntos de referencia, calculados para cada dimensión con 10 iteraciones del algoritmo de Lloyd.

Parámetro	**Valor**
Dim	*12, 80, 128*
NP	*125378*
|Q|	*100*
K	*100 y 200*
PageSize	*1024*
Distancia	*Euclidea*

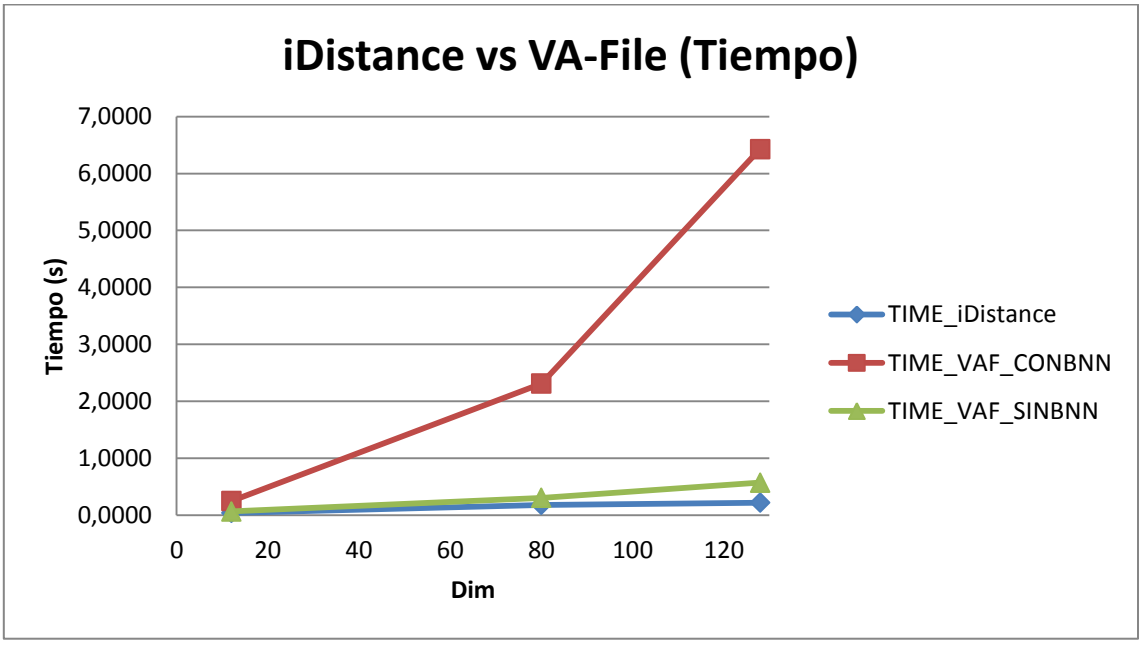
Tabla 18. Experimento 9: Rendimiento del *iDistance* frente al *VA-File* en la consulta KNN variando Dim y K.

Los resultados obtenidos (valores medios por consulta) para un tamaño de página (PS) de 1024 bytes y distancia Euclidea, son los siguientes:

KNN		VA-FILE (PS=1024)				iDistance (PS=1024)	
		CON BNN		SIN BNN		SIN BNN	
K	**D**	**Tiempo**	**Ac. Disco**	**Tiempo(s)**	**Ac. Disco**	**Tiempo(s)**	**Ac. Disco**
100	*12*	0,1948	9310,00	0,0487	36371,00	0,0200	2592,00
	80	1,8158	44020,00	0,2863	155556,00	0,1300	12463,00
	128	5,7483	39645,00	0,6395	137566,00	0,1800	49798,00
200	*12*	0,2484	13790,00	0,0649	63416,00	0,0400	2998,00
	80	2,3117	50713,00	0,3028	149218,00	0,1800	50740,00
	128	6,4300	52929,00	0,5709	144245,00	0,2200	74644,00

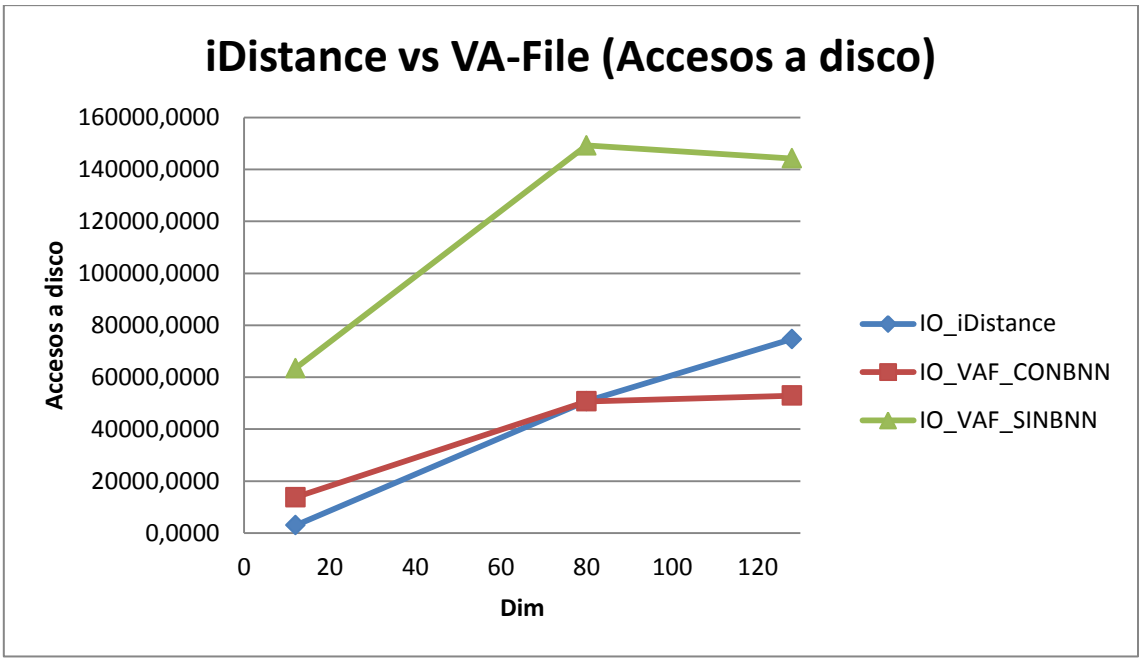
Tabla 19. Resultados del experimento 9.

Como se puede observar de los resultados obtenidos en la tabla 19, cuanto mayor sea K más tiempo y accesos a disco se recogen de las ejecuciones. Al igual que ocurre con la dimensión de los datos. Es por ello que vamos a mostrar solo las gráficas para los resultados de K = 200:



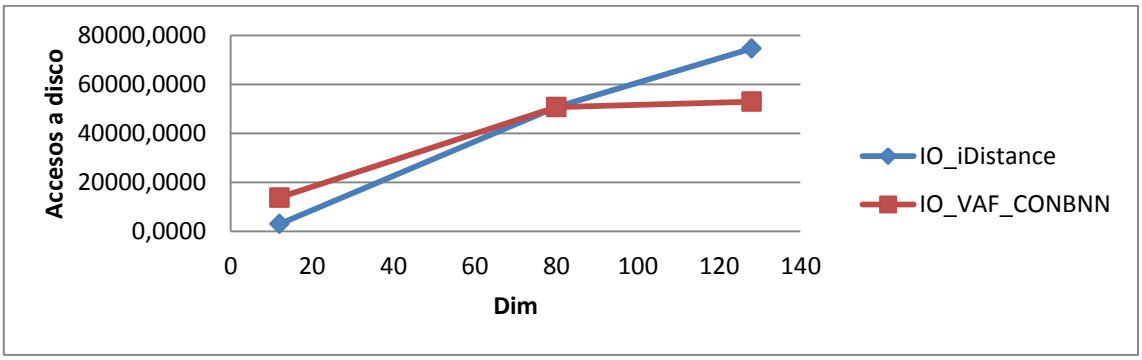
Gráfica 14. Resultados del experimento 9: tiempo de respuesta del *iDistance* contra el *VA-File* con un tamaño de página de 1024 bytes, K=200 y distancia Euclidea.

Se puede observar en la Gráfica 14 que el *iDistance* presenta un buen tiempo de respuesta por consulta con respecto al *VA-File* con BNN, mientras que el *VA-File* sin BNN se mantiene con tiempos ligeramente superiores que el *iDistance*.



Gráfica 15. Resultados del experimento 9: accesos a disco del *iDistance* contra el *VA-File* con un tamaño de página de 1024 bytes, K=200 y distancia Euclidea.

Veamos con más detalle los accesos a disco del *iDistance* con el VA-File con el procesamiento por lotes:



Gráfica 16. Resultados del experimento 9: accesos a disco del *iDistance* contra el *VA-File* con procesamiento por lotes. Tamaño de página = 1024 bytes, K=200 y distancia Euclidea.

Los accesos a disco del *iDistance* se ven afectados con respecto a los estudios anteriores por la disminución del tamaño de página, pero aun así presenta mejor rendimiento que el *VA-File* sin BNN en cuanto a accesos a disco y tiempo de procesado por consulta. El *VA-File* con BNN presenta un número menor de accesos a disco que el *iDistance* para grandes dimensiones como se puede observar en las Gráficas 15 y 16.

Experimento 10: *Rendimiento del iDistance frente al VA-File en la consulta KNN variando la Dimensión de los datos y K. El tamaño de página se mantiene a 4096 bytes y la distancia utilizada: Euclidea.*

Veamos ahora como afecta la variación del tamaño de página realizando el mismo experimento que el anterior (9) aumentando el tamaño de página a 4096 bytes.

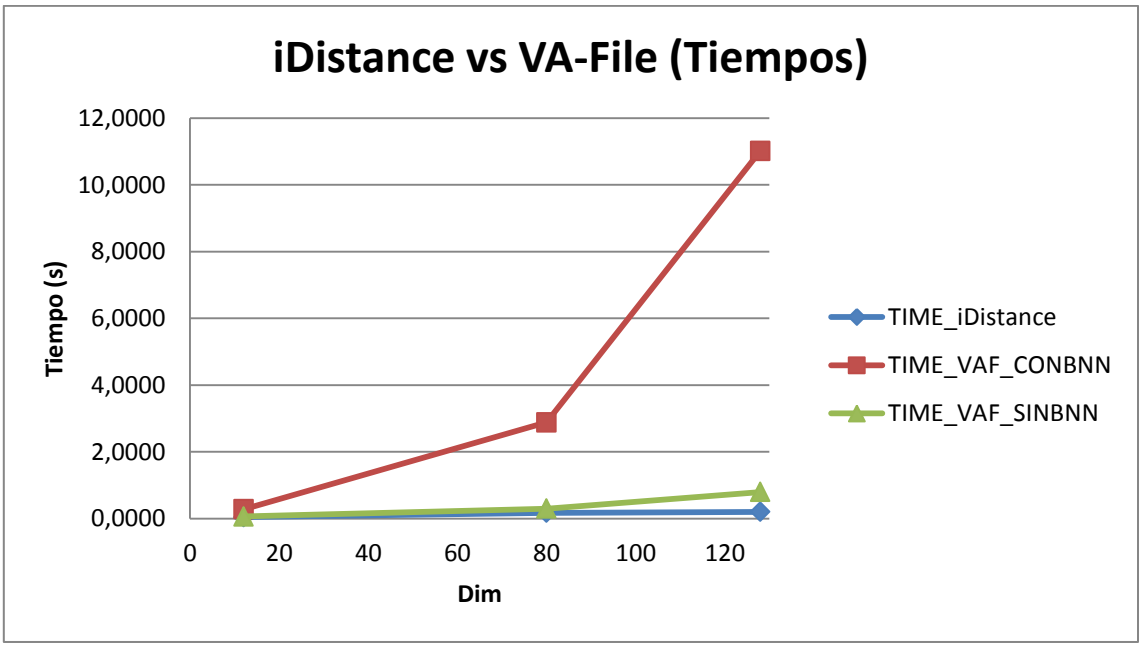
Parámetro	Valor
Dim	*12, 80, 128*
NP	*125378*
|Q|	*100*
K	*100 y 200*
PageSize	*4096*
Distancia	*Euclidea*

Tabla 20. Experimento 10: Rendimiento del *iDistance* frente al *VA-File* en la consulta KNN variando Dim y K.

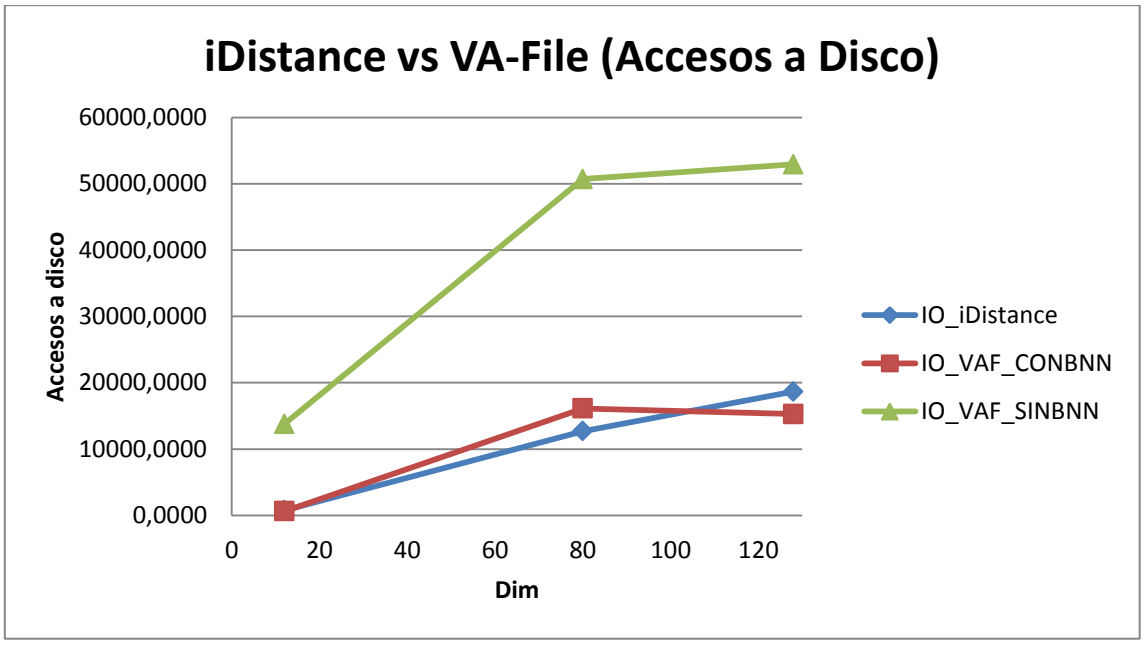
Los resultados obtenidos (por consulta) para un tamaño de página (PS) de 4096 bytes y distancia Euclidea, son los siguientes:

KNN		**VA-FILE (PS=4096)**				**iDistance (PS=4096)**	
		CON BNN		**SIN BNN**		**SIN BNN**	
K	**D**	**Tiempo**	**Ac. Disco**	**Tiempo(s)**	**Ac. Disco**	**Tiempo(s)**	**Ac. Disco**
100	*12*	0,2192	363,00	0,0451	9310,00	0,0200	666,00
	80	2,3636	14445,00	0,2625	44020,00	0,1100	12463,00
	128	9,6468	15580,00	0,4930	39645,00	0,1700	17528,00
200	*12*	0,2761	634,00	0,0621	13790,00	0,0400	770,00
	80	2,8794	16109,00	0,2946	50713,00	0,1700	12686,00
	128	11,0168	15264,00	0,7962	52929,00	0,2000	18661,00

Tabla 21. Resultados del experimento 10



Grafica 17. Resultados del experimento 10: tiempo del *iDistance* contra el *VA-File*. Tamaño de página = 4096 bytes, K=200 y distancia Euclidea.



Gráfica 18. Resultados del experimento 10: accesos a disco del *iDistance* contra el *VA-File*. Tamaño de página = 4096 bytes, K=200 y distancia Euclidea

Como era de esperar, tanto en el *iDistance* como en el *VA-File* se reduce en gran medida el número de accesos a disco tras la ampliación del tamaño de página a 4096 bytes. En cambio, el tiempo de procesado por consulta del *VA-File* con procesamiento por lotes sí que se ve afectado por la ampliación del tamaño de página.

En general, el comportamiento de las dos estructuras es el mismo que en el experimento 9, teniendo mejor prestaciones el *iDistance* que el *VA-File*, a excepción del número de accesos a disco del *VA-File* con procesamiento por lotes para grandes dimensiones.

Experimento 11: *Rendimiento del iDistance frente al VA-File en la consulta KNN variando la Dimensión de los datos y K. El tamaño de página se mantiene a 4096 bytes y la distancia utilizada: Máxima.*

Para comprobar cómo influye el tipo de distancia empleada en la consulta KNN del *iDistance* y el *VA-File* vamos a cambiar el tipo de distancia de Euclidea a **Máxima**. En este caso, vamos a utilizar K=200 directamente puesto que de los experimentos anteriores se desprende que la relación entre K y el tiempo de respuesta así como del número de accesos es proporcional.

Para el *iDistance*, el número de centros se sigue manteniendo a 64 calculados con 10 iteraciones del *kmeans*, el factor de compactación es de 85%.

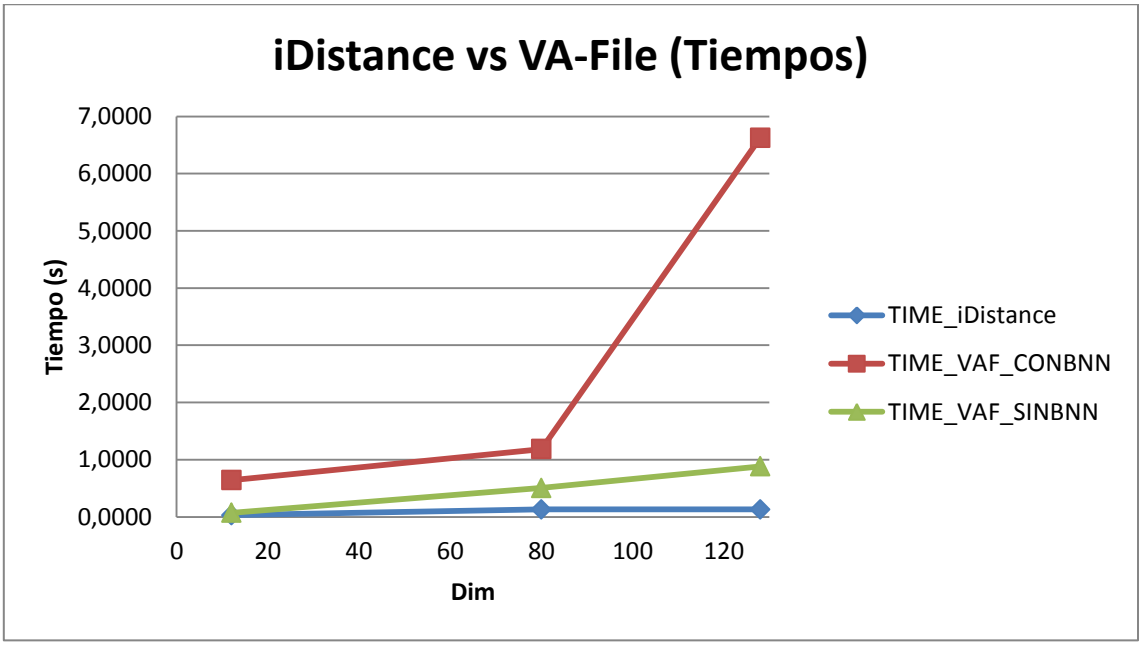
Parámetro	Valor
Dim	*12, 80, 128*
NP	*125378*
|Q|	*100*
K	*200*
PageSize	*4096*
Distancia	*Máxima*

Tabla 22. Experimento 11: Rendimiento del *iDistance* frente al *VA-File* en la consulta KNN variando Dim. El tamaño de página se mantiene a 4096 bytes y la distancia utilizada: Máxima.

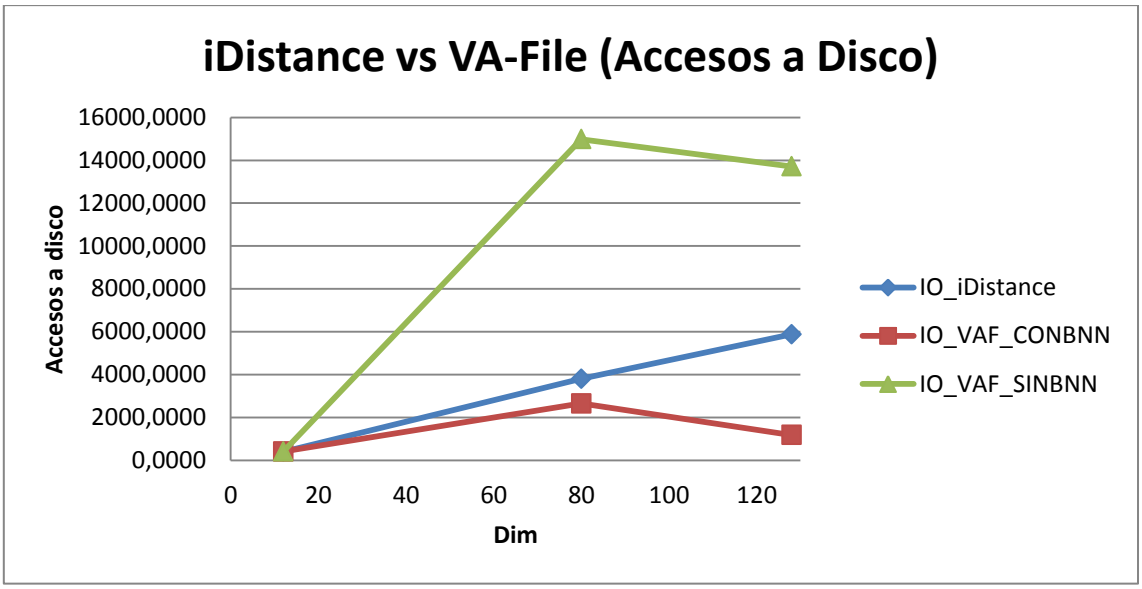
Los resultados obtenidos (por consulta) para un tamaño de página (PS) de 4096 bytes y distancia Máxima, son los siguientes:

KNN		**VA-FILE (PS=4096)**				**iDistance (PS=4096)**	
		CON BNN		**SIN BNN**		**SIN BNN**	
K	**D**	**Tiempo**	**Ac. Disco**	**Tiempo(s)**	**Ac. Disco**	**Tiempo(s)**	**Ac. Disco**
200	*12*	0,6439	411,00	0,0726	412,00	0,0300	398,00
	80	1,1848	2648,00	0,5060	14983,00	0,1300	3800,00
	128	6,6240	1179,00	0,8833	13709,00	0,1300	5877,00

Tabla 23. Resultados del experimento 11.



Gráfica 19. Resultados del experimento 11: Tiempo de procesado del *iDistance* y del *VA-File* con distancia máxima.



Gráfica 20. Resultados del experimento 11: Accesos a disco del *iDistance* y del *VA-File* con distancia máxima.

De los experimentos 9, 10 y 11 se puede deducir que el tiempo individual para el *iDistance* con los tipos de distancias, así como el del *VA-File*, presentan mejor rendimiento con distancia Máxima. Si nos fijamos en los resultados para el *iDistance* con |Q|=100 y K=200 con un tamaño de página de 4096 bytes y distancia Euclidea, tenemos para la dimensión 128 un tiempo de 0,20 segundos, mientras que para el mismo experimento con distancia máxima tenemos un tiempo de respuesta de 0,13 segundos por consulta. Obtenemos el mismo comportamiento con el tiempo de procesado por consulta del *VA-File* presentando mejores resultados utilizando la distancia máxima. Esta mejora se debe en gran medida a que el computo de la distancia máxima entre dos puntos es menor que el de la distancia Euclidea.

Ocurre igual con los accesos a disco, con la distancia máxima, tanto el *iDistance* como el *VA-File* presentan mejoras significativas en dicho valor, esto es debido a que dicha distancia presenta mejores valores a la hora de indexar y recuperar registros. Además, el VA-File con el procesamiento por lotes mejora los resultados del experimento 10 para K=200, obteniendo para datos de dimensión 128: 1179 accesos a disco. Este valor aumenta la diferencia de manera positiva con el resultado del *iDistance* que llega a 5877 accesos a disco.

4.9 Conclusiones

Para los estudios realizados, solo con el *iDistance*, podemos afirmar que es un buen candidato para introducirlo como núcleo de la recuperación de video en un framework de búsqueda de video basada en contenido (CVBR). Las pruebas realizadas con el *iDistance* para la consulta KNN se han realizado con valores de K muy altos obteniéndose muy buenos resultados en cuanto a tiempo de cómputo y accesos a disco.

En los sistemas CBVR, las peticiones a lo sumo suelen ser de 50 vecinos (K=50) por tanto queda más que demostrada la fiabilidad del *iDistance* tras los resultados experimentales.

Si consideramos, como ejemplo, un CBVR con los datos probados en este proyecto, se podría establecer una configuración idónea del *iDistance* con los siguientes valores:

Parámetro	**Valor**
Tamaño de página (PageSize)	*4096*
Distancia (dist)	*Máxima*
Puntos de referencia (CN)	*64 (Mínimo)*
Iteraciones de kmeans (S)	*10 (Mínimo)*
Factor de compactación	*85% - 90%*

Tabla 24. Ejemplo de configuración del *iDistance*

En cuanto al estudio del *iDistance* en comparación con el *VA-File*, podemos concluir que el *iDistance* presenta mejores resultados, superando al *VA-File* en todos los aspectos, excepto en los accesos a disco con altas dimensiones para el procesado por lotes de la consulta del vecino más próximo (BNN) del *VA-File*. Si bien, el *VA-File* es una estructura más que eficiente en la consulta KNN para un sistema de recuperación de video basado en contenido (CBVR), como muestra el estudio realizado en [Riv10], el *iDistance* es una alternativa lo suficientemente eficiente para sistemas CBVR con datos altamente dimensionales.

En el siguiente capítulo, se estudiara la estructura de un CBVR y su diseño con el *iDistance* como núcleo de indexación.

5

Sistema de Recuperación de Video Basado en Contenido

5.1 Introducción

Con el rápido crecimiento que ha experimentado en los últimos años el uso de sistemas tanto centralizados como distribuidos de gestión de recursos de video, la recuperación de video basada en contenido va ganando importancia día a día. Para que el rendimiento de estas aplicaciones sea eficiente ha de abordarse una mejora en la indexación y recuperación de este tipo de información. Para ello los esfuerzos actuales se centran en procesar el contenido de los videos en sí mismos y no en las percepciones de un operador humano. Debido a que en estas líneas de investigación la información de video suele representarse mediante una sucesión de fotogramas (frames), la información contenida en un video se convierte en una secuencia de vectores de características que transforman la información en una base de datos altamente dimensional en la que tanto la creación de los vectores de características como su indexación y recuperación necesitan unos recursos importantes y eficientes.

En la Figura 23, podemos observar la arquitectura de un sistema de recuperación de video genérico, el cual consta de los siguientes módulos:

1. *Interfaz grafica.*
2. *Segmentación de video y extracción de los vectores de características (dim-dimensionales).*
3. *Organización e indexación de vectores de características.*
4. *Motor de búsqueda de video.*

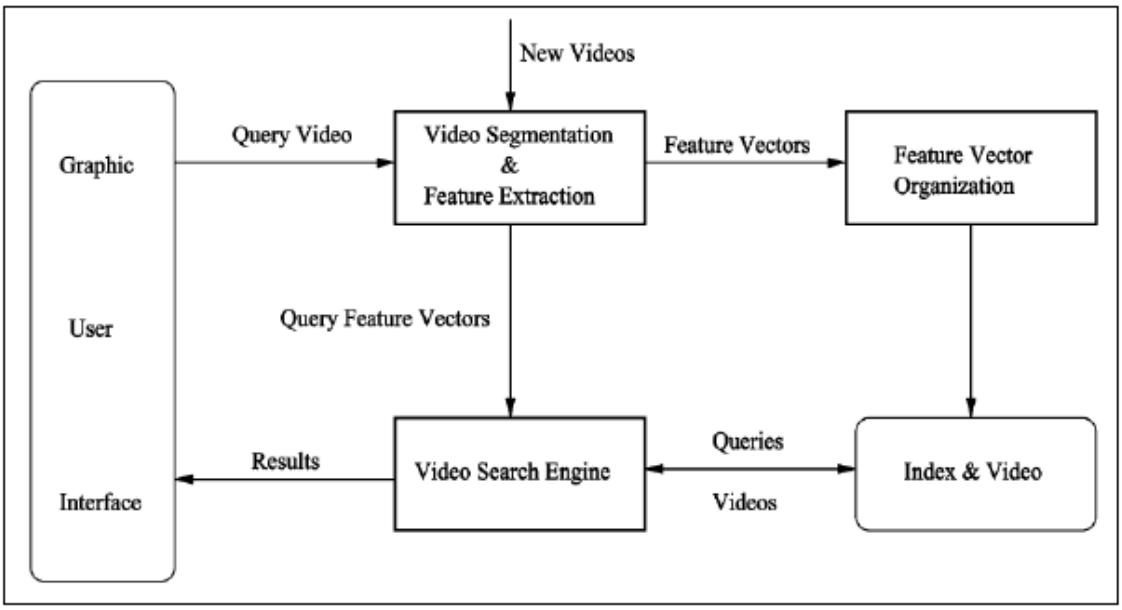


Figura 23. Arquitectura de un sistema de recuperación de video basado en contenido. [SHS+08]

Este capítulo se centra en la inclusión del *iDistance* como motor de indexación y en el estudio de mejoras en las consultas de identificación de subsecuencias en la recuperación de video. Por tanto, vamos a prestar especial atención en los módulos de organización e indexación y de búsqueda de video. Como base, se utilizará el framework de recuperación de video CBVR implementado en [Riv10]. En dicho framework se cambiará el motor de indexación utilizado (*VA-File*) por el *iDistance* y se enriquecerá la consulta de identificación de subsecuencias de video.

5.2 Módulos de un sistema CBVR

A continuación vamos a analizar cada modulo, prestando especial atención a los módulos de organización e indexación y búsqueda de subsecuencias. Comenzando por el módulo de *segmentación de video y extracción de características*, cada video se divide en un número de segmentos, para extraer de cada uno los fotogramas clave de los que se obtienen sus características espaciales (color, forma, textura, esquema, relaciones espaciales, etc.) y temporales (detección de movimiento, operaciones de cámara, composición secuencial, y relaciones entre las distintas imágenes o fotogramas). Todas estas características se representan por una secuencia de vectores de características altamente dimensionales.

Por otro lado, el proceso de *organización e indexación de video* consiste en almacenar e indexar las representaciones caracterizadas del video para poder procesar consultas posteriormente. Por tanto, podemos pensar que una base de datos de video es una base de datos de secuencias de imágenes caracterizadas (por *key-frames*) con dos dificultades añadidas. En primer lugar, el número de vectores almacenados e indexados es mucho mayor (para cada video tendremos múltiples vectores de características, cada uno de ellos representando un *key-frame*). Y en segundo lugar, los vectores de características pueden ser más complejos puesto que han de representar toda la información temporal que añade un video respecto a una imagen.

Por tanto, una *base de datos de video* consiste en el almacenamiento de los videos por una parte y por otra el índice que permite acceder a ellos de manera rápida, ya que el índice consistirá en el conjunto de los vectores de características extraídos de los videos (de los *key-frames* que lo forman). Si cada vector de características del *key-frame* correspondiente tiene *dim* dimensiones, se dice que la base de datos es *dim*-dimensional.

Finalmente, el *motor de búsqueda de videos* realiza una búsqueda en la estructura de índices subyacente para acelerar el proceso de consulta. En general, el proceso de almacenamiento, indexación y recuperación de datos altamente dimensionales, extraídos como características de los videos, es una nueva funcionalidad de los sistemas de bases de datos que ha cobrado gran relevancia en muchos dominios de aplicación en la actualidad.

5.2.1 Segmentación de video

Para el tratamiento de la información de video se suele aprovechar el hecho de que el flujo del video es una sucesión de imágenes interrelacionadas durante un determinado período de tiempo, a lo que se llama *escena*. Ya que las imágenes de una misma *escena* están interrelacionadas, es conveniente para el tratamiento del video completo su segmentación en dichas *escenas*. Y cada *escena* representa una acción continuada en el tiempo y en el espacio, que permitirá darle un significado y gestión conjunta. Una *escena* será la unidad básica para la extracción de características de la información de video.

Las técnicas más importantes para la segmentación de video en *escenas*, descritas en [Riv10] son las siguientes:

- *Técnicas para información no comprimida.* Este tipo de segmentación puede utilizar a su vez otros métodos como: concordancia de plantillas, basados en histogramas, basados en bloques, comparación de gemelos, basada en modelos, etc. Como ejemplo, podemos describir la concordancia de plantillas como la similitud espacial entre dos fotogramas, de manera que si el número de pixeles distintos entre dos fotogramas es mayor de un umbral establecido se decide tener dos *escenas* distintas.

- *Técnicas para información comprimida.* Estas técnicas representan una mejora en cuanto a los requisitos hardware necesarios para su tratamiento, debido a la gran cantidad de información necesaria para los datos de video. Estos normalmente no se encuentran sin comprimir sino que se encuentran almacenados en algún formato comprimido (por ejemplo: MPEG, Moving Picture Experts Group). Este tipo de segmentación puede utilizar otras técnicas como: la basada en coeficientes DCT (Discrete Cosine Tranformation), la basada en vectores de movimiento, híbridas, basadas en la descomposición en sub-bandas, etc.

Una vez realizada la segmentación, cada *escena* en las que se ha segmentado un video ha de ser procesada para obtener sus características en forma de vector y de esta manera ser indexadas. Éstas se dividen en dos grupos: espaciales (se centran en los atributos que pueden ser obtenidos de los fotogramas más importantes como si de imágenes se tratase) y temporales (permiten obtener información del video para representar su dinamismo, atribuyéndolo bien a movimientos de objetos o bien a movimientos de la cámara).

- *Las características espaciales* se basan en seleccionar los fotogramas más relevantes dentro de una secuencia de video y aplicar sobre ellos las técnicas vistas para la indexación de imágenes, mediante estas técnicas se pierde mucha información, puesto que se obvian las características temporales.

- *Las características temporales* de un video se pueden atribuir tanto a movimientos de objetos contenidos en el video, como a movimientos de la cámara. Por tanto, se pueden hacer enfoques en ambos ámbitos para extraer la máxima información para la indexación.

5.2.2 Organización e indexación

Si consideramos que los vectores de características de un video son altamente dimensionales y el número de vectores de características es grande, entonces el procesamiento de consultas puede consumir mucho tiempo para procesar todos los vectores almacenados con el objetivo de determinar videos similares. Todo esto, indica que son necesarias técnicas y estructuras de datos avanzadas para organizar e indexar los vectores de características y gestionar el proceso de búsqueda para poder localizar los resultados de manera rápida y eficiente.

Las técnicas de indexación de video se pueden dividir en dos categorías [Tjo05]: *indexación de video basado en características* (feature-based video indexing) e *indexación de video basado en anotaciones* (annotation-based video indexing). La primera categoría es a bajo nivel, mientras que la segunda es a un nivel más alto. Las dos grandes ventajas que tiene la indexación de video basado en características es que en primer lugar, puede automatizarse totalmente haciendo uso técnicas de extracción de características, tales como análisis de imágenes y sonido. En segundo lugar, los usuarios pueden utilizar búsquedas de similitud con ciertas características como la forma y el color de un *key-frame*. Sin embargo, la indexación basada en características ignora contenidos semánticos que pueden hacer las consultas más ricas, naturales y flexibles.

La indexación de video basada en características a su vez puede hacer uso de tres técnicas:

1. *Técnicas de indexación basada en segmentos.*
2. *Técnicas de indexación basadas en objetos.*
3. *Técnicas de indexación basadas en eventos.*

Este proyecto utiliza un framework de recuperación de video existente [Riv10], el cual ya se basa en técnicas de indexación basada en segmentos, que tiene a su vez ventajas e inconvenientes. Las principales ***ventajas*** son: (1) se puede automatizar fácilmente, (2) los segmentos proporcionan un orden temporal aceptable, (3) soporta consultas de similitud de imágenes y, (4) permite visualizar los frames que forman la escena. Por el contrario, los ***inconvenientes*** que presenta esta técnica son: (1) a menudo, el *key-frame* no es suficiente para representar de forma correcta la información que hay en la *escena* que representa, (2) la información de movimiento y temporal no se representa realmente y, (3) es bastante difícil determinar manualmente el mejor *key-frame* que caracteriza a una *escena*.

En el capítulo 2, hemos considerado múltiples opciones de indexación para datos altamente dimensionales. Entre todas las estrategias, vamos a utilizar el *iDistance* como método de indexación de vectores de características. Es por tanto, que tendremos un archivo con todas las secuencias de vectores de características sin indexar y otro archivo con el árbol B+ creado siguiendo la técnica *iDistance*. Además, tendremos un archivo con los puntos de referencia de cada partición de datos y un archivo con la información de las distancias máximas y mínimas de cada partición. En la Figura 24, podemos observar la arquitectura de un sistema CBVR con

el *iDistance* como motor de indexación. Dicha arquitectura es la que se ha seguido para la realización de este proyecto.

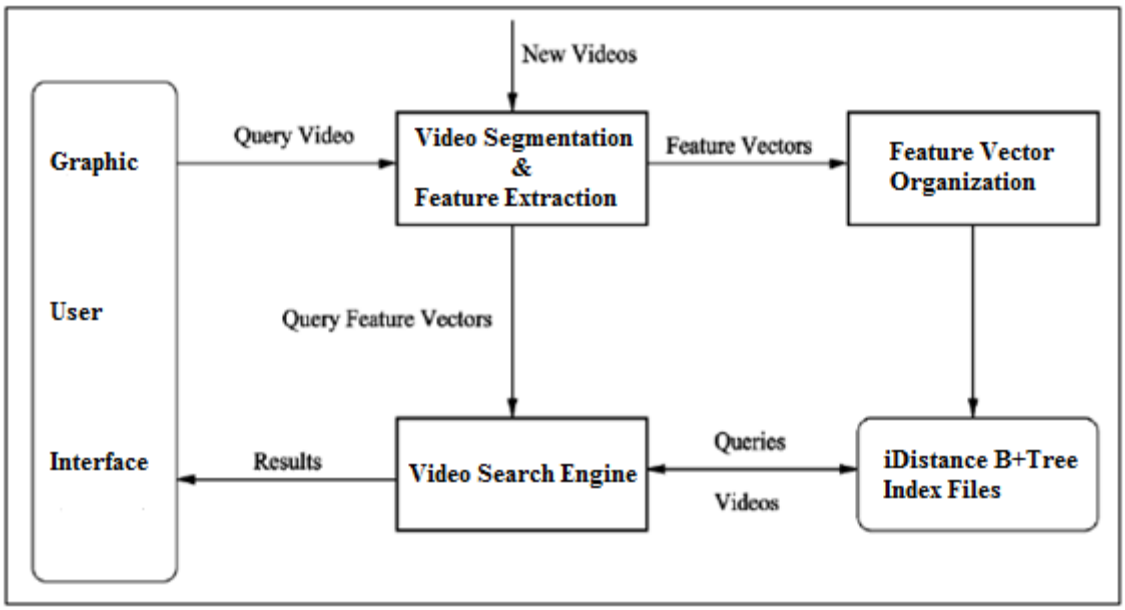


Figura 24. Framework CBVR con el *iDistance* como motor de indexación de vectores de características.

5.2.2.1 Tipos de consultas de video indexadas con vectores de características

Tras la indexación, el *iDistance*, nos permite realizar consultas de video (vectores de características) las cuales pueden considerarse, de manera abstracta, como una combinación entre una consulta a una base de datos de vectores de características (*iDistance Index B+ Tree*) y una consulta a un sistema de información espacio-temporal (en el que podemos especificar objetos y sus relaciones espacio-temporales). Según las Figuras 23 y 24, una vez que las características de los *key-frames* se extraen, se pueden utilizar para facilitar la recuperación. Normalmente, un sistema de base de datos se utiliza para manejar los datos de video y audio, esto proporciona facilidad para definir las características para los *key-frames* de video y la relación entre ellos.

El proceso de recuperación se refiere normalmente como búsqueda por similitud. Para una búsqueda por similitud, hay tres tipos comunes de consultas:

1. *Consulta de punto (o consulta exacta).* Para esta consulta, el parámetro de entrada es un vector de características de consulta y el resultado que devuelve serán todos los vectores de características que coinciden exactamente con el vector de consulta.

2. *Consulta en rango de distancia.* Para esta consulta, el usuario proporciona como entrada un vector de consulta y un rango de distancia. Todos los vectores de características que tienen distancias con el vector de consulta menores o iguales que el rango de distancia especificado se devuelven como resultado.

3. *Consulta de los K vecinos más próximos (KNN).* En esta consulta, el usuario proporciona un vector de consulta y un entero K positivo (K > 0). Los K vectores

de características que tienen las distancias más pequeñas respecto al vector de consulta se devuelven como resultado.

Cuando realizamos una búsqueda por similitud en una base de datos video, el parámetro de la consulta es normalmente un video clip que consiste una secuencia de *key-frames*. Durante el proceso de búsqueda, cada *key-frame* del video-clip consulta supone una búsqueda por similitud, seguida por la integración de los resultados de todos los *key-frames*. Se han sugerido muchas propuestas para la medida de similitud entre dos video clips. Una medida para similitud de video, ampliamente utilizada y aceptada, es el porcentaje de frames similares compartidos por dos secuencias sin considerar el orden temporal [ChZ03, SOZ+05].

5.2.2.2 iDistance como motor de indexación

El *iDistance* es una excelente opción para ejercer de motor de indexación para un sistema de búsqueda de video organizado en *key-frames* los cuales se transforman en vectores de características altamente dimensionales.

Tal y como muestran los experimentos realizados en el capítulo 4, se van a utilizar 64 puntos de referencia obtenidos mediante clustering (*kmeans*) seleccionando los centros de cada clúster como el punto de referencia de cada partición (o clúster).

Los algoritmos de búsqueda y construcción del índice son idénticos a los descritos en el capítulo 3. El *iDistance* recibe una subsecuencia de puntos de consulta Q, donde Q está formada por |Q| vectores de consulta (o vectores de características). Para cada punto $q_i \in Q$, el *iDistance* buscara los K vecinos más cercanos a dicho punto, obteniéndose entonces una estructura *(kNN)* con K vecinos por cada punto de Q. El número de puntos recuperados es entonces |Q|*K.

Una vez recuperados todos los K vecinos de cada punto de consulta de Q, estos se devuelven al motor de búsqueda de video del CBVR para su procesamiento y obtención de video, cuyas búsquedas serán explicadas en el siguiente apartado. En la Figura 25 podemos observar una situación de búsqueda para |Q|=4 y K=4, es decir, para cuatro puntos de consulta Q = {q_1, q_2, q_3, q_4} se quieren recuperar los 4 vecinos mas cercanos de cada punto. Por tanto los puntos recuperados serian 4*4=16 puntos del *iDistance*.

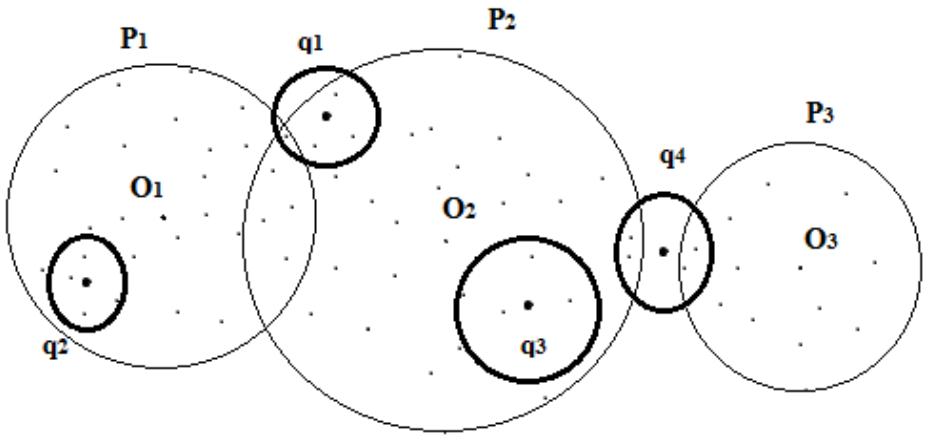


Figura 25. Consulta para una subsecuencia de 4 *key-frames*. |Q|=4. (K=4)

5.2.3 Motor de búsqueda de videos

Como se ha explicado anteriormente, un sistema de base de datos de video es un sistema software que gestiona una gran colección de datos de video y proporciona a los usuarios acceso basado en contenido. Por este motivo, uno de los objetivos principales de este tipo de sistemas es proporcionar a los usuarios un acceso eficiente y cómodo a una gran colección de datos de video. En este contexto, el proceso de recuperación de video basado en contenido (CBVR) juega un papel fundamental, y dentro de éste el motor de búsqueda de videos se encarga de la búsqueda en la estructura de índices subyacente (métodos de indexación altamente dimensionales) para acelerar el proceso de consulta.

El proceso de recuperación de video (CBVR) consta de los siguientes pasos [EJH+97]:

1. El usuario especifica una consulta utilizando las facilidades proporcionadas por la interfaz de usuario.

2. La consulta se procesa y se obtienen los resultados apropiados. En este punto entra la consulta KNN procesada por el *iDistance* para cada punto de consulta seleccionado por el usuario en el paso anterior.

3. Los resultados obtenidos (en términos de frames) se utilizan para realizar un matching y recuperar el video correspondiente de la base de datos de videos.

4. El video o videos resultantes de la recuperación se visualizan a través de la interfaz de usuario en el dispositivo de salida apropiado.

En cuanto al paso 3, en [YDT+04] se han presentado una lista de los métodos desarrollados en consultas por video clip. Teniendo en cuenta una serie de parámetros como requerimientos del usuario y características de la búsqueda de video, estos métodos de consulta se pueden clasificar en tres categorías: (1) detección de instancias de copias, (2) detección en base a similitud de duplicados y (3) recuperación a alto nivel de entradas en base a similitud. En este contexto (CBVR), las dos principales operaciones de consulta que se van a tratar y mejorar en este proyecto son: ***búsquedas de videos similares (Video Similarity Search, VSS) e identificación de subsecuencias de video (Video Subsequence Indentification, VSI).*** La consulta *VSS* consiste en encontrar secuencias de video de una base de datos de secuencias de videos, cuya similitud con una secuencia de consulta dada es menor de una cota dada ($\varepsilon > 0$), como por ejemplo dado una secuencia de gol (Q_{gol}) en un partido de futbol, encontrar en las *n* subsecuencias de S (ya conocidas) si existe algún gol similar. Mientras que la consulta *VSI* encuentra una subsecuencia, la cual es la parte más similar a una secuencia de consulta dada bajo una determinada función de similitud, como por ejemplo dada una base de datos con multitud de videos de distintas categorías, identificar subsecuencias similares a la secuencia de gol, Q_{gol}, antes mencionada.

En el proyecto que nos ocupa, vamos a prestar especial atención en la inclusión del *iDistance* como motor de indexación y en la identificación de subsecuencias de videos implementada en [Riv10] en la cual se implementarán mejoras en los algoritmos de recuperación de subsecuencias similares. Para ello se hará uso de grafos y heaps para mejorar la similitud entre secuencias de video, probándose experimentalmente la efectividad y eficiencia de dicho entorno para datos reales y sintéticos altamente dimensionales.

5.3 Framework para identificación de subsecuencias de video

Existe una gran cantidad de bibliografía relacionada con la recuperación de video basada en contenido (Content-Based Video Retrieval, CBVR), sin embargo no se ha prestado demasiada atención a las consultas relacionadas con la identificación de subsecuencias en video, que consisten en encontrar en una larga secuencia de video, la subsecuencia más similar a un video-clip (secuencia de video) de consulta dado. Se persigue entonces la identificación de secuencias de video con ordenamientos diferentes de sus frames, una longitud diferente o una edición del contenido distinta.

A partir de la recuperación de los K vecinos más cercanos (frames más similares) realizada por el *iDistance*, podemos establecer la relación (mapping) entre el video-clip de consulta y para ello, la secuencia de video se representa como un grafo bipartito. En [Riv10], las partes emparejadas (matched) más densas a lo largo de la secuencia de video se extraen, y a continuación se aplica una estrategia de búsqueda por filtrado y refinamiento para podar la mayoría de subsecuencias irrelevantes. Durante la fase de filtrado, se calcula el matching de tamaño máximo (*Maximum Size Matching*, *MSM*) para cada sub-grafo construido por la secuencia de consulta y la secuencia de candidatos, para obtener un conjunto de secuencias candidatas más pequeño. Durante la fase de refinamiento, se genera el matching de similitud sub-máximo (*Sub-Maximum Similarity Matching*, *SMSM*) para identificar la subsecuencia con la mayor puntación (de similitud) de todas las candidatas, de acuerdo con un modelo de similitud de video robusto que incorpora contenido visual, orden temporal e información de alineación de frames.

En este proyecto se propone, además, la recuperación de las K subsecuencias (*kSS*) más similares en el algoritmo SMSM, de manera similar a como se realiza la consulta KNN. Se compara la similitud entre la subsecuencia recuperada y la de consulta para organizar las K subsecuencias en orden de similitud.

La Figura 26 representa la arquitectura del framework implementado para la recuperación de video basada en contenido, utilizando el *iDistance* para proporcionar salida a la consulta de identificación de subsecuencias, *VSI* y además soporte para la consulta de búsqueda de secuencias mas similares, *VSS*.

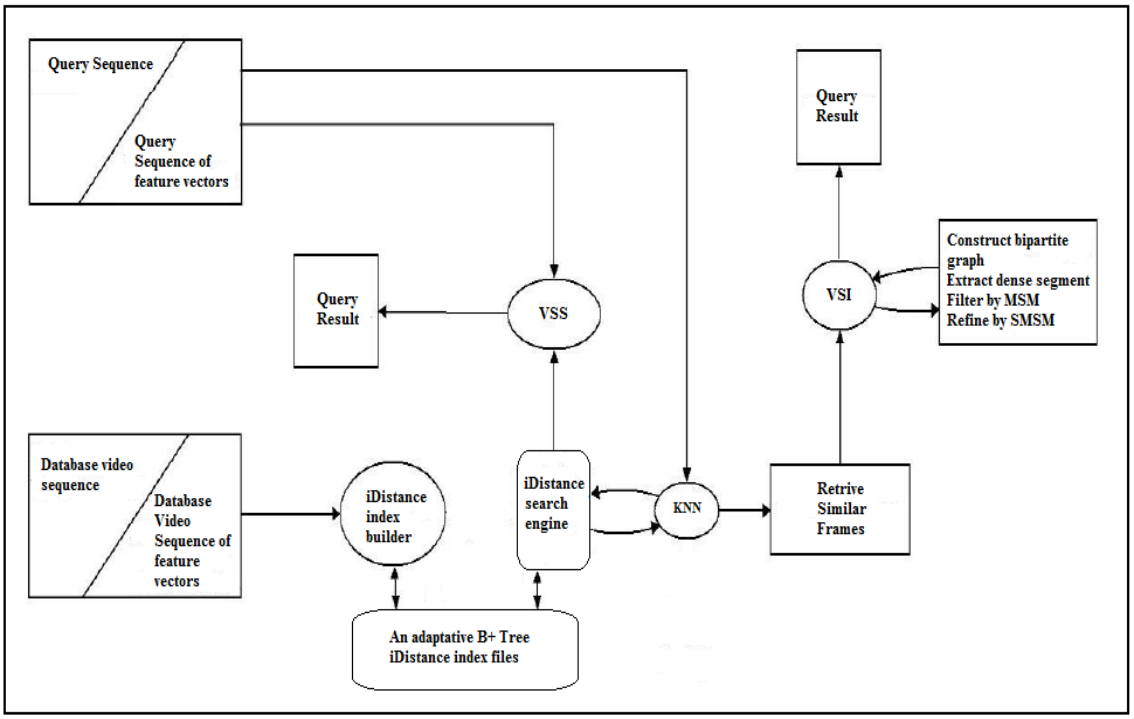


Figura 26. Framework para la recuperación de video basada en contenido en las consultas *VSI* y *VSS* utilizando el *iDistance* como motor de indexación.

5.3.1 Fundamentos

Una *secuencia de video* es un conjunto ordenado de un gran número de *key-frames*. Normalmente, cada *key-frame* es considerado y representado por un vector de características *dim*-dimensional, en el que han sido extraídas algunas características de contenido de bajo nivel, como la distribución del color, patrones de textura, estructuras de forma, etc. Los matching (emparejamientos) de videos a menudo se traducen en búsquedas entre estos vectores de características. En la práctica, a menudo se desea comprobar manualmente si un video es parte de una larga secuencia, navegando por todos los *key-frames* que representan a dicha secuencia. Por ello se hace necesaria una solución fiable para automatizar la búsqueda de contenido similar. La identificación de subsecuencias de video implica la localización de la posición de las partes más similares entre un video-clip de consulta dado *Q* y una larga secuencia de video *S*. Lo ideal sería identificar secuencias de video relevantes, incluso si existen algunas transformaciones de distorsión, reordenamiento parcial del contenido, inserción, eliminación o modificación.

La principal diferencia entre *recuperación de video* (Video Retrieval, VR) y la *identificación de subsecuencias de video* (Video Subsequence Identification, *VSI*) es que, mientras la tarea de recuperación de video consiste en devolver video-clips similares (a un video-clip de consulta dado) de una gran colección de videos que han sido o bien cortados en partes similares o con contenidos parecidos; la tarea de la identificación de subsecuencias aspira a encontrar si existe alguna subsecuencia en una gran secuencia de videos la cual comparte un contenido similar al del video-clip de consulta dado. En otras palabras, mientras que en la recuperación de video, los video-clips a buscar han sido procesados y están listos para realizar una similitud por ranking; la identificación de subsecuencias de video está relacionada con el

problema de matching de secuencias. Dado que el límite y la longitud de la subsecuencia objetivo no están disponibles inicialmente, no se puede conocer de antemano qué fragmentos deben evaluarse para la similitud. Por lo tanto, la mayoría de los métodos existentes para la tarea de recuperación en colecciones de video no son aplicables para este problema relativamente complicado.

A diferencia de la detección de copia de video, la que normalmente considera sólo transformaciones de distorsión, un video visualmente similar se puede cambiar tranquilamente con edición del contenido del frame o a nivel de la escena (intercambio, inserción, eliminación o substitución), lo que podría llevar a un orden o longitud diferente respecto a la fuente original. En este proyecto se va a continuar mejorando la identificación de subsecuencias de video de [Riv10] mediante el *iDistance* como motor de recuperación de frames similares y la recuperación de las K subsecuencias mas similares (*kSS*) a una subsecuencia de consulta dada (Q). En resumen, el proceso seguido comienza por la adquisición de frames similares mediante el algoritmo KNN del *iDistance* visto en el capítulo 3, tras ello, se construye un grafo bipartito que asocie los frames similares de cada punto de consulta (q_i) de la subsecuencia de consulta Q dada con cada frame s_i similar recuperado de la base de datos de video S. Entonces, para identificar de manera efectiva y eficiente la mejor o mejores subsecuencia/as similar/es (*kSS*), el proceso de consulta propuesto se lleva a cabo la imposición de una restricción de mapping 1:1. El matching de tamaño máximo (MSM) se emplea para filtrar rápidamente la mayoría de la subsecuencias no similares con un coste computacional bajo. Una vez obtenido un número de subsecuencias más reducido, entonces se puede evaluar su similitud con un coste computacional pequeño, para asegurar la identificación de las k-subsecuencias más similares. Dado que la medida de la similitud de video para todos los mappings 1:1 en un *sub-grafo* es computacionalmente intratable, se empleará un método heurístico denominado Sub-Maximum Similarity Matching (SMSM) para identificar rápidamente las subsecuencias correspondientes mediante un heap.

En general, la *identificación de subsecuencias de video* intenta averiguar una o varias subsecuencias de S que se parezcan lo suficiente a Q. *Dado $Q = \{q_1, q_2, ..., q_{|Q|}\}$ un video-clip de consulta corto (una secuencia de consulta) y $S = \{s_1, s_2, ..., s_{|S|}\}$ una gran secuencia de video, donde d es la dimensión de los vectores de características y $q_i = \{q_{i1}, q_{i2}, ..., q_{id}\} \in Q$ y $s_j = \{s_{j1}, s_{j2}, ..., s_{jd}\} \in S$ son vectores de características d-dimensionales que representan frames de video y $|Q|$ y $|S|$ denotan el número total de frames de Q y S respectivamente (normalmente $|Q| << |S|$).* ***La identificación de subsecuencias de video intenta encontrar $\vec{S} = \{s_m, s_{m+1}, \dots, s_n\}$ en S donde $1 \leq m \leq n \leq |S|$, que es la parte de S más similar a Q bajo la definición de una función de puntuación (medida de similitud).***

5.3.2 Tipos de consultas en la identificación de subsecuencias

En esta sección se va a detallar el proceso de identificación de subsecuencias de video implementadas dentro del framework de recuperación de video (CBVR). Las dos consultas principales implementadas son la identificación de subsecuencias (*VSI*) y la búsqueda de subsecuencias más similares (*VSS*).

Para comprender mejor los algoritmos y la notación utilizada de ahora en adelante, vamos a exponer en la Tabla 25 los símbolos utilizados en esta sección:

Símbolo	Definición
d	Dimensión de los vectores de características.
Q	Secuencia de consulta.
S	Secuencia de video, base de datos de video.
q_i	Vector i-ésimo de características (frame) de la secuencia de consulta Q.
s_j	Vector i-ésimo de características (frame) de S.
$F(q_i)$	Conjunto de vectores de características similares q_i. En este proyecto $F(q_i) = KNN(q_i, S, K)$
$\vec{S}$	Una subsecuencia de S.
$Dist(q_i, s_j)$	Distancia entre dos puntos.
$\overline{S}$	Conjunto de s_j con contador diferente de 0.
$\overrightarrow{S^*}$	Un segmento denso en S.
$\overrightarrow{\mathbb{S}^*}$	Conjunto de segmentos densos $\overrightarrow{S^*}$.
$\overrightarrow{\mathbb{S}^*}'$	Conjunto de candidatos similares después de filtrar $\overrightarrow{\mathbb{S}^*}$ con MSM.
M_k	Un mapping 1:1 en un sub-grafo.
$\vec{M}_k$	Un sub-mapping de M_k (mapping 1:1).
M'	Mapping 1:1 más apropiado después de aplicar SMSM.
$kSS_{M'}$	Heap de tamaño *k* que almacena las *kSS* mejores subsecuencias recuperadas.
kSS	Numero de subsecuencias a recuperar en la mejora del algoritmo SMSM propuesta en este proyecto. También indica el tamaño del heap $kSS_{M'}$

Tabla 25. Notación para la identificación de subsecuencias de video.

Primero vamos a estudiar la consulta *VSS* y a continuación la consulta *VSI* con las mejoras implementadas.

5.3.2.2 Video Similarity Search (VSS)

Esta consulta realiza una búsqueda entre todas las subsecuencias de S (ya conocidas), la subsecuencia $\vec{S}$ más similar a una subsecuencia de consulta dada Q. Para ello compara la distancia de cada punto de $\vec{S}$ con cada punto de Q, la subsecuencia la cual sus puntos presenten menos distancia con los puntos correspondientes de la subsecuencia de consulta que un valor determinado (*delta*) se considera una posible respuesta. Tras identificar las subsecuencias con menor distancia que *delta*, se procede a devolver la subsecuencia mas similar. Para el cálculo de la distancia entre los puntos de cada subsecuencia $\vec{S}$ de S y Q pueden utilizarse numerosas distancias, la única restricción es que estas estén *normalizadas*, preferiblemente entre 0 y 1 incluidos. En el capítulo 2, se introducen varias distancias, de las cuales se han implementado las siguientes:

$$Distancia\ Euclidea\ (X,Y) = \sqrt{(\sum_{i=0}^{dim-1} |X_i - Y_i|)^2}$$

$$Distancia\ Máxima\ (X,Y) = Max_{0\leq i<\dim}|X_i - Y_i|$$

$$Distancia\ Camberra\ (X,Y) = \sum_{i=0}^{dim-1} \frac{|X_i - Y_i|}{|X_i| + |Y_i|}$$

$$Distancia\ Bray\ Curtis\ (X,Y) = \sum_{i=0}^{dim-1} \frac{|X_i - Y_i|}{X_i + Y_i}$$

$$Squared\ Chord\ (X,Y) = \sum_{i=0}^{dim-1} (\sqrt{X_i} - \sqrt{Y_i})^2$$

$$Distancia\ Chi\ Cuadrado\ (X,Y) = \sum_{i=0}^{dim-1} \frac{(X_i - Y_i)^2}{X_i + Y_i}$$

A la hora de realizar los experimentos sólo se pueden utilizar las distancias normalizadas y la única que SIEMPRE nos devuelve un valor entre 0 y 1 (rango del espacio *dim*-dimensional trabajado) es la ***distancia máxima*** y por tanto es la que vamos a utilizar para la consulta *VSS*.

Para comprobar que sólo la distancia máxima es la única que está normalizada, sean cuales sean los puntos de entrada, vamos a considerar los puntos *a = {0.9, 0.9, 0.1}* y *b = {0.1, 0.1, 0.9}* las distintas distancias:

$$Distancia\ Eculidea = \sqrt{(0.9-0.1)^2 + (0.9-0.1)^2 + (0.1-0.9)^2} = \sqrt{1.92} > 1$$

***Distancia Máxima* = Max {0.8, 0.8, 0.8} = 0.8**

$$Distancia\ Camberra = \frac{0.8}{1} + \frac{0.8}{1} + \frac{0.8}{1} = 2.4 > 1$$

$$Distancia\ Bray\ Curtis = \frac{0.8}{1} + \frac{0.8}{1} + \frac{0.8}{1} = 2.4 > 1$$

$$Distancia\ Chi\ Cuadrado = \frac{0.8^2}{1} + \frac{0.8^2}{1} + \frac{-0.8^2}{1} = 1.92 > 1$$

$$Distancia\ Scuared\ Chord = (\sqrt{0.9} - \sqrt{0.1})^2 + (\sqrt{0.9} - \sqrt{0.1})^2 + (\sqrt{0.1} - \sqrt{0.9})^2 = 1.19 > 1$$

Queda por tanto demostrado que la única distancia de las propuestas que esta normalizada dentro del espacio de datos es la distancia máxima. Como trabajo futuro, se dejará abierta la posibilidad de normalizar las distancias para comprobar la eficiencia de las mismas en esta consulta.

Definida la distancia a utilizar, vamos a proponer formalmente la consulta *VSS*: ***dado un espacio de datos de video S indexado mediante vectores de características dim-dimensionales, subsecuencias de S ($\vec{S}_i$) y una subsecuencia de consulta Q dada, para cada punto de consulta $q_i \in Q$ y cada punto $s_j \in \vec{S}$ calcular la distancia, mediante la distancia máxima y comprobar si dicha distancia es inferior a un valor delta prestablecido por el usuario. Si lo es almacenar la subsecuencia $\vec{S}$ como una posible candidata. Tras recuperar todas las subsecuencias candidatas, devolver la que menos distancia presente con respecto a la subsecuencia de consulta.***

El pseudocódigo implementado en la consulta es el siguiente (Código 9):

VSSQuery (S, indiceSecuecias, Q, delta)

1. *Establecer los limites 'posINI' y 'posFIN' de las subsecuencias de S mediante el archivo de índices 'indiceSecuencias'.*
2. *Enumerar las secuencias existentes en S.*
3. ***Para** cada subsecuencia $\vec{S}_k$ en S*
4. ***Para** cada punto q_i de Q hacer*
5. ***Para** cada punto s_j de $\vec{S}_k$ hacer*
6. ***Si** la Dist(q_i, s_j) < delta*
7. *Añadir $\vec{S}$ al conjunto de subsecuencias candidatas*
8. *Continuar con otra subsecuencia*
9. ***Fin Si***
10. ***Fin Para***
11. ***Fin Para***
12. ***Fin Para***
13. ***Devolver** la $\vec{S}$ encontrada (o su identificador) que haya presentado menor distancia.*

Código 9. [Riv10] Pseudocódigo de la consulta de búsqueda de video similar *VSS*.

Como se puede observar, la búsqueda de subsecuencias similares no considera la desalineación de frames ni el cambio de orden. Simplemente se basa en la distancia calculada entre cada *key-frame* de la subsecuencia de consulta y el *key-fame* de la subsecuencia $\vec{S}$ consultada, además las subsecuencias de S deben de ser conocidas. Ésta es la principal diferencia con la consulta de identificación de subsecuencias, que lo que persigue es encontrar secuencias de video teniendo en cuenta el espacio temporal, el orden de los *key-frames*, existencia de huecos y ruido.

5.3.2.3 Video Subsequence Indentification (VSI)

Esta es la consulta principal del framework utilizado [Riv10], encargada de la identificación de subsecuencias de video. Como ya se introdujo anteriormente dicho algoritmo tiene como entradas $Q = \{q_1, q_2, ..., q_{|Q|}\}$ un video-clip de consulta corto (una secuencia de consulta) y $S = \{s_1, s_2, \ldots, s_{|S|}\}$ una gran secuencia de video, donde *d* es la dimensión de los vectores de características y $q_i = \{q_{i1}, q_{i2}, \ldots, q_{id}\} \in Q$ y $s_j = \{s_{j1}, s_{j2}, \ldots, s_{jd}\} \in S$ son vectores de características *d*-dimensionales que representan frames de video y |Q| y |S| denotan el número total de frames de Q y S respectivamente. La identificación de subsecuencias de video intenta encontrar como salida, $\vec{S} = \{s_m, s_{m+1}, \ldots, s_n\}$ en S, donde $1 \leq m \leq n \leq |S|$, que es la parte de S más similar a Q bajo la definición de una medida de similitud.

El proceso de identificación de subsecuencias consta de 5 pasos: (1) obtención de los frames mas similares a los puntos de consulta de la subsecuencia dada, (2) transformación de cada correspondencia entre $q_i \in Q$ y $s_i \in S$ en un grafo y hacer el matching propuesto, (3) extracción de segmentos densos, (4) filtrado por matching de tamaño máximo y (5) refinamiento por Sub-Maximum Similarity Matching.

5.3.2.3.1 Obtención de frames similares

El proceso de identificación de subsecuencias de video comienza con el algoritmo de obtención de los frames más similares (Código 10). Normalmente, dos frames se consideran similares si su distancia está por debajo de un umbral de distancia, así la búsqueda en rango (definida en el capítulo 2) se puede utilizar sin problemas. En [Riv10] y en este proyecto se ha decidido utilizar la búsqueda KNN para que cada frame de consulta tenga los mismos frames similares asociados, K.

La razón por la cual se usa la consulta KNN es porque a priori no sabemos a que distancia van a estar los vecinos del punto de consulta, pero si sabemos cuantos (o estimamos) cuantos puntos mas similares vamos a necesitar para el matching, es por ello que se prefiere la consulta de los K vecinos mas cercanos a la de rango de distancia. Si no se dispusiera de la KNN habría que llamar a la consulta de rango ampliando la distancia de consulta hasta que se recuperen los puntos necesarios para el procesamiento posterior. El siguiente pseudocódigo supone que el *iDistance* está ya creado y que la consulta se realiza sobre la base de datos S.

ObtenerFramesSimilares(Q, K)

1. ***Para*** *cada punto de consulta q_i en Q hacer:*
2. ***Si*** *esta definida la búsqueda KNN para el motor de indexación:*
3. *$F(q_i)$ = iDistance_KNNQuery(q_i, K)*
4. ***Sino***
5. *$F(q_i)$ = rango(q_i)*
6. ***Fin Si***
7. ***Fin Para***
8. ***Devolver*** *F*

Código 10. Obtención de los frames más similares en la identificación de subsecuencias de video (*VSI*)

Como resultado de la ejecución del algoritmo de *ObtencionFramesSimilares*, tenemos los correspondientes conjuntos de frames $F(q_i) \in S$ para cada $q_i \in Q$.

5.3.2.3.2 Transformación en grafo bipartito

Cada frame se puede corresponder como un nodo del grafo a lo largo de la línea temporal del video. Dada una secuencia de consulta (video-clip corto) Q y una secuencia (video largo) S, se pueden abstraer una línea temporal corta y una línea temporal larga, respectivamente. En lo sucesivo cada frame ya no es modelado como un punto altamente dimensional como en el algoritmo anterior, sino que simplemente se trata como a un nodo. Q y S que son dos conjuntos finitos de nodos ordenados a lo largo de la línea temporal, se tratan como los dos lados de un grafo bipartito. Formalmente G = {V, E} es un grafo bipartito que representa los matchings de frames entre Q y S. Por tanto, V = Q ∪ S y es el conjunto de vértices que representa los frames, mientras que E ⊆ Q x S es el conjunto de aristas que representa los matchings de frames similares. En la Figura 27, podemos observar la alineación temporal en el grafo bipartito.

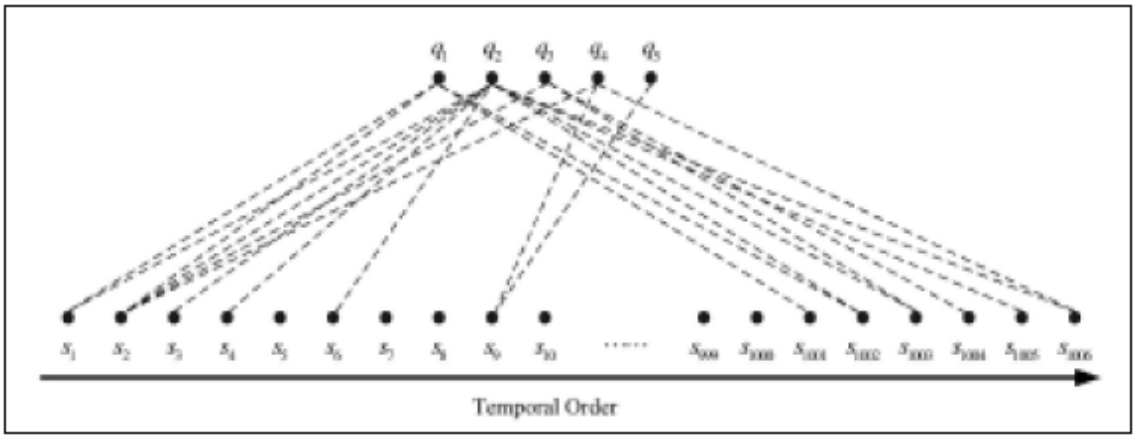


Figura 27. [Riv10] Grafo bipartito generado a partir de los frames obtenidos en del algoritmo de Obtención de Frames Similares (Código 10).

A partir de la obtención de frames similares (Código 10) podemos convertir cada par (q_i, $F(q_i)$) en nodos y aristas del grafo bipartito. El código 11 muestra como se genera dicho grafo bipartito: por cada $s_j \in F(q_i)$, tenemos un contador que indica el número de frames similares de la consulta, estando inicializado a 0. Por cada $q_i \in Q$ si $s_j \in F(q_i)$ entonces esto genera una arista entre q_i y s_j en G, y el contador de s_j se incrementa en 1. Al mismo tiempo, el conjunto de s_j con el contador distinto de cero, denotado como $\overline{S}$, se actualiza. La Figura 27 muestra el ejemplo de construcción de un grafo bipartito. El algoritmo de *GenerarGrafoBipartito* (Código 11) necesita procesar cada $F(q_i)$, por tanto el número total de aristas del grafo es $\sum_{i=1}^{|Q|} |F(q_i)|$ y el algoritmo tiene una complejidad de orden O(|Q|). En el caso de que la búsqueda KNN sea la establecida para recuperar $F(q_i)$, entonces el grafo tendrá K*|Q| aristas (|E|).

GenerarGrafoBipartito (F, Q)

1. ***Para*** *cada* q_i *en Q*
2. ***Para*** *cada* s_j *que este en* $F(q_i)$
3. *Inicializar el contador de* s_j *a 0*
4. ***Fin Para***
5. ***Fin Para***

6. ***Para*** *cada* q_i *en Q*
7. ***Para*** *cada* s_j *que este en* $F(q_i)$
8. *Añadir la arista* $\{q_i, s_j\}$ *a E*
9. *Incrementar el contador de* s_j
10. *Actualizar con* $<s_j$, *contador de* $s_j>$
11. ***Fin Para***
12. ***Fin Para***

13. ***Devolver*** *E y* $\overline{S}$

Código 11. [Riv10] Generación de un grafo bipartito con los frames más similares a una secuencia de consulta dada.

Observando los *mappings* de frames similares a lo largo de la línea temporal (unidimensional) de S, sólo una pequeña parte está densamente emparejada (matched), mientras que la mayor parte está no emparejada (unmatched) o dispersamente emparejada. Intuitivamente, estas últimas dos partes se pueden descartar directamente, puesto que no es posible que tengan una subsecuencia similar con Q, porque la condición necesaria para que una subsecuencia sea similar a Q es que compartan un número suficiente de frames similares. Gracias a ello, se evita que se comparen todas las posibles subsecuencias en S, puesto que se filtran una gran cantidad de partes irrelevantes para la evaluación de similitud.

Para ello, se deben identificar los segmentos más densos de S que contienen todas las posibles subsecuencias de video similares. Nótese que para esto no es necesario mantener todo el grafo. En su lugar, se realiza un proceso para reducir el tamaño de E y para las siguientes etapas.

5.3.2.3.3 Extracción de segmentos densos

A lo largo del lado S de G (grafo bipartito) con contadores de enteros {0, 1, …, |Q|}, consideramos que la presencia de ceros consecutivos no sea relativamente grande. Considerando las potenciales alineaciones de frames y los huecos, los segmentos que no tengan estrictamente un contador sin ceros, como por ejemplo $\{s_1, \ldots, s_6\}$ con sus respectivos contadores "241101", se podrán aceptar. Para representar la frecuencia de mappings de frames similares, se introduce el concepto de *densidad de un segmento*.

Densidad de segmento: Para un segmento $\vec{S} = \{s_m, s_{m+1}, \ldots, s_n\}$ en S, su densidad den($\vec{S}$) es el número total de frames de consulta similares para todo s_j, es decir, den($\vec{S}$)=($\sum_{j=m}^{n} contador(s_j)$)/(n-m+1).

Con la comprobación de la densidad de los segmentos, se pueden averiguar las partes vacías o con escasos elementos. El algoritmo *CalcularSegmentosDensos* (Código 12) formaliza la extracción del conjunto $\overrightarrow{\mathbb{S}^*}$ de segmentos densos. Se utiliza el conjunto $\overline{S}$ como entrada, el cual contiene las tuplas de la forma <s_j, contador (s_j)> devueltas por el algoritmo *GenerarGrafoBipartito* (Código 11), donde s_j es el identificador del frame y contador(s_j) ≠ 0 refleja el número frames de la consulta similares con s_j. Adicionalmente, se utilizará un umbral que determina el número de contadores a cero consecutivos requerido para separar dos segmentos altamente densos. Primero $\overline{S}$ se ordena por el identificador del frame y se inicializa $\overrightarrow{S_1^*}$. Empezando desde el primer elemento se comprueba la distancia δ entre los identificadores de los dos frames adyacentes, $s_{j'}$ y s_j en $\overline{S}$. Todos los s_j adyacentes con δ < α se agrupan en un segmento $\overrightarrow{S_k^*}$. Si no, el elemento actual $s_j \in S$, formará un nuevo segmento. Todos esos segmentos densos se añaden dentro de $\overrightarrow{\mathbb{S}^*}$. Procesando el conjunto ordenado de esta manera hasta el último elemento en $\overline{S}$, tenemos el conjunto con k segmentos densos.

En la Figura 27 cuando α se establece a 3, los segmentos $\overrightarrow{S_1^*}$ = {s_1,…, s_6}, $\overrightarrow{S_2^*}$= {s_9},…,$\overrightarrow{S_n^*}$ = {s_{1001},…, s_{1006}} se extraen. Claramente la complejidad de este algoritmo es lineal en $|\overline{S}|$.

CalcularSegmentosDensos($\overline{S}$,α)

1. *Ordenar $\overline{S}$ por sj*
2. *k=1*
3. *Añadir el primer elemento de $\overline{S}$ en $\overrightarrow{S_k^*}$*
4. *Añadir $\overrightarrow{S_k^*}$ en $\overrightarrow{\mathbb{S}^*}$*
5. ***Mientras** el primer elemento de $\overline{S}$ sea distinto del ultimo*
6. *δ = Siguiente elemento de $\overline{S}$.sj – Primer elemento $\overline{S}$.sj*
7. ***Si** δ < α*
8. *Añadir el siguiente elemento de $\overline{S}$ en $\overrightarrow{S_k^*}$*
9. ***Si no***
10. *Incrementar k en 1*
11. *Añadir el siguiente elemento de$\overline{S}$ en $\overrightarrow{S_k^*}$*
12. *Añadir $\overrightarrow{S_k^*}$ en $\overrightarrow{\mathbb{S}^*}$*
13. ***Fin Si***
14. *Asignar al primer elemento de $\overline{S}$ el siguiente elemento de $\overline{S}$*
15. ***Fin Mientras***
16. ***Devolver** $\overrightarrow{\mathbb{S}^*}$*

Código 12. [Riv10] Calculo de segmentos densos a partir de un grafo bipartito.

El algoritmo del cálculo de segmentos densos (Código 12) asegura que cualquier segmento denso no se solapará con otro, y no hay riesgo de cortar una subsecuencia similar en dos segmentos, debido a que dicho algoritmo ayuda a reducir el número de segmentos densos. En [Riv10] queda demostrado que con un α apropiado, no hay solape de segmentos densos con la misma secuencia. Sin embargo, es probable que dos subsecuencias sucesivas o solapadas que realmente tengan algo similar con Q, se agrupen en un segmento largo. Para filtrar segmentos densos aunque no sean similares, y extraer las subsecuencias más similares con precisión en

un segmento relativamente largo, se aplicará una estrategia de búsqueda y filtrado que se explica a continuación.

5.3.2.3.4 Filtrado por matching de tamaño máximo (Maximum Size Matching, MSM)

Después de localizar los segmentos densos, tenemos k sub-grafos separados de la forma G_k = {V_k, E_k} donde $V_k = Q \cup \overrightarrow{S_k^*}$ es el conjunto de vértices mientras E_k es el conjunto de aristas que representan los mappings de frames similares entre Q y $\overrightarrow{S_k^*}$. Sin embargo, la alta densidad de los segmentos no es suficiente para indicar una alta similitud con la consulta debido a no prestar la suficiente atención al número actual de frames similares, el orden temporal o la alineación de frames. Por ejemplo, en la Figura 29, $\overrightarrow{S_2^*}$ es un frame aislado cuyo contador (s_9) = 2 pero tiene al menos α – 1 ceros consecutivos antes y después de él, también se tratará como un segmento denso (den($\overrightarrow{S_2^*}$) = 2). Sin embargo, se debe descalificar considerando la longitud de Q. Por otra parte, un q_i individualmente puede ser similar a múltiples frames en un único segmento $\overrightarrow{S_k^*}$ (y probablemente con algunos frames de otros segmentos densos $\overrightarrow{S_k^*}$). Y viceversa, ya que la consulta Q está compuesta por una serie de frames, un frame s_j en $\overrightarrow{S_k^*}$ puede tener múltiples frames similares con Q. Esto se puede observar en la Figura 27, la relación original del mapping de frames similares entre Q y $\overrightarrow{S_k^*}$ es normalmente múltiple de la forma M:M. Incluso cuando los videos se representan por escenas en lugar de por frames, este emparejamiento M:M es aún más común, aunque no es significativo a nivel de frame. De hecho, no todos los segmentos densos con longitudes aceptables de media son realmente relevantes. Un contraejemplo es que cuando un segmento $\overrightarrow{S_k^*}$ muestra contenido repetido, los frames consecutivos pero no repetidos corresponden a dicho contenido pueden ser todos similares con q_i pero no son similares con cualquier otro frame, es decir, q_{i-1} o q_{i+1} de Q.

La Figura 28 sirve de ejemplo de esta relación 1:M. Claramente, éste no debería considerarse como similar.

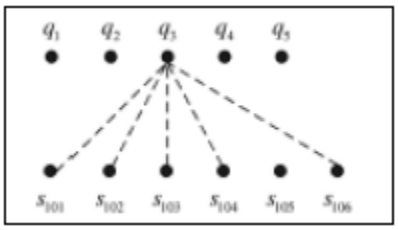


Figura 28. [Riv10] Mapping 1:M.

Para hacer que el número de pares similares sea suficientemente completo, es decir obtener el mayor número de frames que estén llenos lo máximo posible, y considerando que el coste de edición de las operaciones de alineación para un mapping 1:M es muy alto, se restringen las relaciones de mapping de frames en el sub-grafo G_k de la forma 1 a 1 (1:1). El grafo bipartito divide los vértices en dos conjuntos, y esta restricción puede expresar que cada vértice de ambas caras es incidente con al menos una arista de E_k. Ya que queremos saber cómo las aristas pueden combinar los dos conjuntos para lograr el mayor número de pares, entonces se adopta el clásico algoritmo de caminos de aumento (*Augmenting Path Algorithm*) sobre grafos bipartitos no ponderados para encontrar el mapping 1:1 entre Q y S con el tamaño máximo.

Maximum Size Matching, MSM. Un matching M en G = {V, U} es un subconjunto de E, en el que ningún par de aristas tienen nodos comunes (con pares de aristas no adyacentes). El tamaño de un matching M es el número de aristas en M y se describe como |M|. El MSM de G es un matching M_{MSM} con el mayor tamaño de $|M_{MSM}|$.

El algoritmo utilizado en este proyecto para realizar el MSM es el que aparece en el framework CBVR de [Riv10]. El pseudocódigo de dicho algoritmo esta compuesto por los métodos *Aumentar*, *Matching y FiltradoPorMatching* (Códigos 13, 14 y 15):

```
Aumentar(vértice v)
1. Si label[v] es 0
2.    mate[exposed[v]]=v
3. Si no
4.    exposed[label[v]]=mate[v]
5.    mate[v]=exposed[v]
6.    mate[exposed[v]]=v
7.    Aumentar(label[v])
8. Fin Si
```

Código 13. [Riv10] Método recurrente, encargado de aumentar el contador de cada arista.

```
Matching(Grafo G={V, U, E})

1.  Para cada v ∈ V ∪ U
2.      mate[v]=0
3.  Fin Para

4.  Mientras Finalizado sea falso
5.      Para cada v ∈ V
6.          exposed[v]=0
7.      Fin para
8.      Vaciar A
9.
10.     Para cada arista [v, u] de E
11.         Si mate[u] es 0
12.             Si mate[u] es distinto de v
13.                 Añadir a A la arista (v, mate[u])
14.             Fin Si
15.         Fin Si
16.     Fin Para
17.
18.     Vaciar Q
19.     Para cada v ∈ V
20.         Si mate[v] es 0
21.             Añadir a Q el vértice v
22.             Asignar label[v] a 0
23.         Fin Si
24.     Fin Para
25.
26.     Mientras Q no esté vacío
27.         Obtener primer v de Q y eliminarlo
28.         Si exposed[v] es distinto de 0
29.             Aumentar(v)
30.             Salir del mientras
31.         Si no
32.             Para cada v' que: label[v'] es 0 y que (v, v') es una arista de A
33.                 label[v']=v
34.                 Añadir v' a Q
35.             Fin Para
36.         Fin Si
37.     Fin mientras
38.     Si Q esta vacío
39.         Asignar verdadero a Finalizado
40.     Fin Si
41. Fin Mientras

42. Devolver vector mate
```

Código 14. [Riv10] Matching.

El algoritmo usa dos vectores, *mate* y *exposed*, además del array *label* usado para la búsqueda. El array *mate* tiene un tamaño de |V| + |U| y es donde se irá representando el matching. Para cada v ∈ V, *exposed[v]* es un nodo de U el cual está conectado con v, si no existe entonces *exposed[v]=0*. Si un nodo v ∈ V con *exposed[v]≠0* se encuentra en la búsqueda, entonces se ha encontrado un camino de aumento. El algoritmo termina si no existen caminos en el grafo auxiliar que pueda aumentar un nodo de V.

Por último, veamos el algoritmo de filtrado que aparece en [Riv10] (Código 15):

FiltradoPorMatching($\overrightarrow{\mathbb{S}^*}$, β)

1. ***Para*** *cada segmento en* $\overrightarrow{\mathbb{S}^*}$
2. M_k=*Matching*(G_k)
3. ***Si*** $|M_k| > \beta$
4. *Añadir* $\overrightarrow{S_k^*}$ *en* $\overrightarrow{\mathbb{S}^*}'$
5. ***Si no***
6. *Descartar* $\overrightarrow{S_k^*}$
7. ***Fin Si***
8. ***Fin Para***

9. ***Devolver*** $\overrightarrow{\mathbb{S}^*}'$

Código 15. Filtrado por matching de tamaño máximo.

El parámetro β indica el tamaño mínimo de un matching para no descartar un segmento denso, en función del valor de β el filtrado descartará más o menos segmentos densos y por lo tanto habrá menos o más posibles subsecuencias candidatas. Si tomamos como ejemplo la Figura 28 y los subgrafos (Q, $\overrightarrow{S_1^*}$) y (Q, $\overrightarrow{S_n^*}$) podemos obtener como ejemplo el mapping 1:1 por MSM de la Figura 29. Cabe destacar que la salida del algoritmo MSM puede no ser única como se demuestra [Riv10].

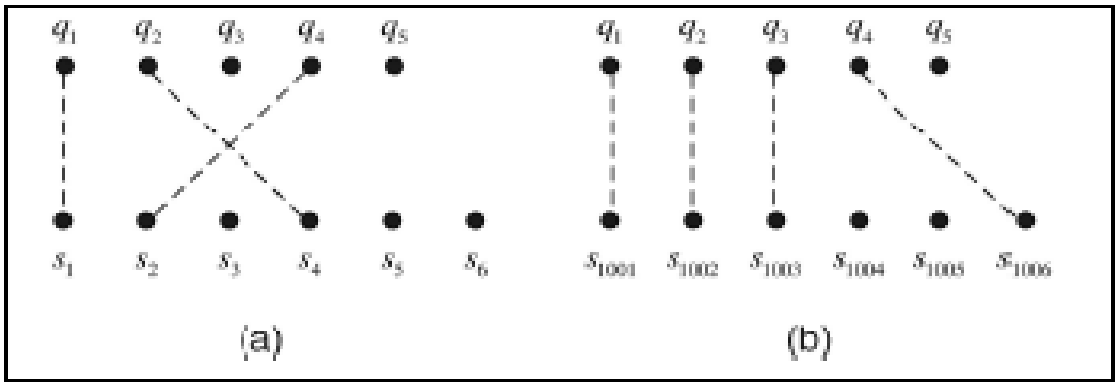


Figura 29. [Riv10] Mapping 1:1 por MSM.

5.3.2.3.5 Refinamiento por Sub-Maximum Similarity Matching (SMSM)

La anterior etapa de filtrado se puede ver como una evaluación de similitud en bruto, sin tener en cuenta la información temporal. Observando que un segmento $\overrightarrow{S_k^*}$ en $\overrightarrow{\mathbb{S}^{*\prime}}$ puede tener múltiples mappings 1:1, y la subsecuencia más adecuada en S puede ser sólo una parte de $\overrightarrow{S_k^*}$, a continuación, se expondrá el algoritmo de refinamiento de $\overrightarrow{\mathbb{S}^{*\prime}}$ planteado en [Riv10] para encontrar el mapping 1:1 más adecuado para una identificación (o ranking) más precisa, mediante la consideración del contenido visual, orden temporal y alineación de frames simultáneamente. Por ultimo se propone, en este proyecto, una mejora del algoritmo de refinamiento para recuperar las K mejores subsecuencias en ranking en función a su similitud con la secuencia de consulta, Q.

Medidas de Similitud

Para las medidas de similitud de video se van a usar las definidas en [Riv10], y son las siguientes:

- *Similitud de contenido visual (Sim_{VC}).* Para localizar la mejor subsecuencia similar visualmente, recurrimos a la distancia entre dos frames similares, en lugar de simplemente juzgar si son similares o no. En consecuencia, se modifica la definición de arista del grafo bipartito no ponderado. Denotemos el peso de una arista w_{ij} como la similitud detallada entre los frames q_i y s_j de M_k. Donde w_{ij}=1- dist(q_i,s_j)/$\sqrt{d}$ (distancia ***normalizada***, por ejemplo máxima) donde su dominio se demuestra que esta entre [0,1] en [Riv10].

$$Sim_{VC}\left(Q, \overrightarrow{S_k^*}, M_k\right) = \frac{\sum_{ij}^{n} w_{ij}}{|Q|} \text{ , } donde\ w_{ij} \epsilon M_k$$

- *Similitud del orden temporal (Sim_{TO}).* Podemos preferir reordenar algunos frames para lograr una mejor Sim_{VC} con un sacrificio aceptable sobre preservar el orden temporal, para una mayor similitud total desde el punto de vista de la percepción humana. Dado un mapping 1:1 M_k de G_k correspondiente a $\overrightarrow{S_k^*}$, el número de pares de frames que han mantenido el orden temporal, es decir emparejados en la misma dirección, en [Riv10] se define la LCSS (Longest Common Subsequence), la cual se denota LCSS (|Q|,|$\overrightarrow{S_k^*}$|) como un indicador del número de pares de frames de M_k emparejados a lo largo del orden temporal.

$$Sim_{TO}(Q, \overrightarrow{S_k^*}) = \frac{LCSS(|Q|, |\overrightarrow{S_k^*}|)}{|Q|}$$

- *Similitud de alineación de frames (Sim_{FA}).* Considerar sólo los factores del contenido similar y el orden temporal todavía puede ser problemático. Dado que el LCSS de dos secuencias permite que muchos elementos se salten sin coste ni reordenamiento, las dos secuencias "ACBD" y "ACCBBD" tiene el mismo LCSS con "ABCD". En el caso que también comparta el mismo contenido visual, no se podrán diferenciar. Por ello, se incorpora un factor de alineación de frames para solucionar este problema y conseguir una mejor discriminación. El número de frames que necesitan operaciones de alineación se consideran como el número de vértices no saturados (libres) en G_k, porque esos vértices no saturados sugieren que no tienen contrapartidas similares bajo una restricción de mapping 1:1. Dado Q y un mapping 1:1 M_k para $\overrightarrow{S_k^*}$, el grado de similitud de alineación de frames se puede calcular por el número relativo de vértices saturados como se muestra en la siguiente fórmula:

$$Sim_{FA}\left(Q, \overrightarrow{S_k^*}, M_k\right) = \frac{2\,|M_k|}{|Q| + |\overrightarrow{S_k^*}|}$$

- ***Similitud de video (Sim).*** Juntando los tres factores de similitud descritos anteriormente, podemos definir la similitud entre dos subsecuencias de video Q y $\overrightarrow{S_k^*}$ bajo un mapping M_k como:

$$Sim\left(Q, \overrightarrow{S_k^*}, M_k\right) = \sum_{i \in \{VC,TO,FA\}} Sim_i\left(Q, \overrightarrow{S_k^*}, M_k\right)/3$$

Sub-Maximum Similarity Matching (SMSM)

Dado un subgrafo G_k, el algoritmo SMSM procesa todas las aristas de E_k comenzando por la que más conectividad[1] presente y las inserta en un nuevo mapping vacío $\vec{M}_k$ hasta que todas se hayan procesado. Cuando múltiples aristas tienen la misma conectividad[1], cada una de ellas formará un $\vec{M}_k{}'$ ampliado de $\vec{M}_k$ y el que maximice la similitud total bajo $\vec{M}_k{}'$, se añade definitivamente en $\vec{M}_k$.

---

[1]***Conectividad de una arista:*** Para una arista E_{ij} en el grafo G_k correspondiente a un $\overrightarrow{S_k^*}$, su conectividad de arista se define como $Conect(E_{ij}) = |F_k(q_i) + F_k(s_j)| - 2$, donde $F_k(q_i)$ y $F_k(s_j)$ son el número de frames similares obtenidos. Por ejemplo, por el algoritmo KNN, de q_i y de s_j en G_k, respectivamente.

En esta primera propuesta de SMSM, el algoritmo *GetBestSubsequece* (código 16) refina los segmentos similares para encontrar *la* mejor secuencia similar.

GetBestSubSequence ($\overrightarrow{\mathbb{S}^*}'$)

1. ***Para*** *cada segmento en* $\overrightarrow{\mathbb{S}^*}'$
2. $\vec{M}_k$=0
3. ***Mientras*** E_k>0
4. *Insertar las aristas con mayor Conect de* E_k *en* E'_k
5. ***Si*** $|E'_k|$=1
6. *Añadir* $E'_k(1)$ *en* $\vec{M}_k$
7. ***Si no***
8. *sub_max=0*
9. *sub_max_index=0*
10. ***Para*** *cada arista en* E'_k
11. $\vec{M}_k' = \vec{M}_k$
12. *Añadir* $E'_k(i)$ *en* $\vec{M}_k'$
13. $\overrightarrow{sub\_S_k^*}$ =*getSubsequence*($\overrightarrow{sub\_S_k^*}$, $\vec{M}_k'$)
14. *tmp=Sim(Q,* $\overrightarrow{sub\_S_k^*}$, $\vec{M}_k'$)
15. ***Si*** *tmp>sub_max*
16. *sub_max=tmp*
17. *sub_max_index=i*
18. ***Fin Si***
19. ***Fin Para***
20. *Añadir* E'_k*(sub_max_index) en* $\vec{M}_k$
21. ***Fin Si***
22. *Actualizar* E_k
23. ***Fin Mientras***
24. $\overrightarrow{sub\_S_k^*}$ =*getSubsequence*($\overrightarrow{S_k^*}$, $\vec{M}_k$)
25. ***Fin Para***

26. *M' = mapping 1:1,* $\vec{M}_k$ *procesado con mejor similitud. Donde* $\vec{M}_k \subseteq E_k$.
27. ***Devolver*** *M'*

Código 16. [Riv10] Refinamiento por SMSM.

Para ello, lo primero que se hace es inicializar un mapping vacío $\vec{M}_k$. Iterativamente se añade una o más aristas en el submapping actual $\vec{M}_k$. Se cogen las aristas con mayor *'Conect'* disponibles en E_k y se añaden en E'_k. Si sólo hay una arista, ésta se añade directamente a $\vec{M}_k$. Sino, cada arista candidata y el actual $\vec{M}_k$ generan un nuevo mapping $\vec{M}_k'$ de tamaño $|\vec{M}_k|$ + 1. Por tanto, $\vec{M}_k$ es un mapping parcial que se corresponde con la subsecuencia $\overrightarrow{S_k^*}$ que ha sido extraída. Entonces se calcula la similitud de $\vec{M}_k'$. El mapping se genera progresivamente, añadiendo la arista que maximice la similitud del submapping actual.

Una vez que una arista se determina y se añade en $\vec{M}_k$, entonces E_k se actualiza, eliminando todas las aristas que sean incidentes con cualquier vértice de la arista seleccionada. $\vec{M}_k$ irá

creciendo hasta que todas las aristas en E_k hayan sido procesadas. Nótese que el $\vec{M}_k$ final será sólo una subsecuencia de $\overrightarrow{S_k^*}$. La subsecuencia se extrae entonces de $\overrightarrow{S_k^*}$. El mapping 1:1 más adecuado de todos los $\overrightarrow{S_k^*}$ en $\overrightarrow{\mathbb{S}^{*\prime}}$ se comparan y finalmente se devuelve el mejor como resultado del algoritmo.

5.3.2.3.6 Recuperación de las K-Subsecuencias en Ranking

En el anterior algoritmo, hemos podido recuperar la mejor subsecuencia según el algoritmo SMSM. Pero en ocasiones, el usuario puede necesitar más de una subsecuencia *más similar*. Para ello, se propone una modificación del algoritmo de refinamiento: *GetKBestSubSecuences*($\overrightarrow{\mathbb{S}^{*\prime}}, kSS$). En esta *mejora* se propone recuperar las *kSS* mejores subsecuencias en ranking de similitud con la secuencia de consulta dada, Q. Para ello, se utiliza un heap máximo como estructura básica para ir almacenando los mejores mappings 1:1 con respecto a la similitud. El heap implementado ($kSS_{M'}$), tiene un tamaño máximo de *kSS*, manteniendo en memoria los *kSS* mejores mappings encontrados hasta terminar el procesamiento de $\overrightarrow{\mathbb{S}^{*\prime}}$. Al terminar el procesado, el heap contendrá las *kSS* mejores subsecuencias encontradas.

El código 17, muestra como quedaría el algoritmo SMSM para la recuperación de las k mejores subsecuencias:

GetKBestSubSequences ($\overrightarrow{\mathbb{S}^{*\prime}}$, kSS)

```
1.  Inicializar kSS_M' con un tamaño máximo de kSS.

2.  Para cada segmento S*→ en S*'→
3.     M_k→=0
4.     Mientras E_k>0
5.        Insertar las aristas con mayor Conect de E_k en E'_k
6.        Si |E'_k|=1
7.           Añadir E'_k(1) en M_k→
8.        Si no
9.           sub_max=0
10.          sub_max_index=0
11.          Para cada arista en E'_k
12.             M_k'→ = M_k→
13.             Añadir E'_k(i) en M_k'→
14.             sub_S_k*→ =getSubsequence(sub_S_k*→, M_k'→)
15.             tmp=Sim(Q, sub_S_k*→, M_k'→ )
16.             Si tmp>sub_max
17.                sub_max=tmp
18.                sub_max_index=i
19.             Fin Si
20.          Fin Para
21.          Añadir E'_k(sub_max_index) en  M_k→
```

```
22.       Fin Si
23.       Actualizar E_k
24.    Fin Mientras
25.    Añadir a kSS_M' el segmento denso tratado, S*
26. Fin Para

27. Devolver kSS_M'
```

Código 17. Recuperación de las K mejoras subsecuencias en ranking de similitud.

Con el motivo de comparar el acierto o no del algoritmo SMSM, la clase implementada para almacenar las *kSS* subsecuencias con mayor similitud calcula el *Hit Ratio* de cada subsecuencia recuperada para comprobar que las que tienen mayor similitud sean las que mejor acierto tienen.

Podemos definir el *Hit Ratio* como el número de frames coincidentes de cada subsecuencia con la secuencia de consulta Q. El código 18 muestra como se calcula dicho ratio:

```
CalcHitRatio(Q, kSS_M')

1.  Para cada subsecuencia S_k en kSS_M'
2.      equals=0;
3.      Para cada frame s_j en S_k
4.          Para cada frame q_i en Q
5.              Si los frames s_j y q_i son iguales
6.                  equals++;
7.              Fin Si
8.          Fin Para
9.      Fin Para
10.     kHitRatios[k]= equals / |Q|;
11. Fin Para

12. hitRatioTotal=0.0;
13. Para cada hitRatio en kHitRatios
14.     hitRatioTotal+=kHitRatios[i];
15. Fin Para
16. hitRatioTotal = hitRatioTotal / |kHitRatios|;

17. Devolver  hitRatioTotal
```

Código 18. Calculo de Hit Ratios para las k mejores subsecuencias.

5.4 Conclusiones

Tras concluir el capítulo, podemos advertir que la recuperación de video así como la identificación de subsecuencias están sujetos a numerosos factores que deben quedar configurados a la hora de implantarlo en un sistema o en otro.

Si prestamos atención a la consulta *VSS*, podemos observar que la eficacia del algoritmo, depende en gran medida el factor *delta* para acotar la distancia mínima entre dos frames. Por supuesto es importante tener en cuenta que dicha distancia es fundamental a la hora del procesado de los frames, advirtiendo que aunque en este proyecto se han implementado varias distancias, anteriormente mencionadas, éstas deben estar normalizadas y por lo tanto solo se usará la distancia máxima. Es importante tener en cuenta dichas distancias para el procesado de la consulta *VSS* puesto que una distancia puede presentar mejores resultados que otra dependiendo de la naturaleza de los datos.

En cuanto a la consulta *VSI*, tenemos cuatro factores fundamentales en cuanto a la identificación de subsecuencias:

- La distancia con la cual se procesa la consulta KNN en la recuperación de frames similares. En este proyecto se han considerado dos distancias, la euclidea o la máxima.

- La distancia con la cual se procesan los pesos de las aristas en el cálculo del grafo bipartito. Dicha distancia debe estar normalizada entre 0 y 1.

- El parámetro α para la división de segmentos. Si α es demasiado grande, se agruparán en menos segmentos y viceversa.

- El parámetro β que indica el tamaño mínimo del matching para no descartar un segmento. Si β es demasiado pequeño el filtro puede ser muy pobre pero si es demasiado grande puede provocar el descarte de segmentos importantes.

- Las funciones de similitud de video. Existen varios artículos con mejoras en dichas funciones de similitud. En el apartado de trabajo futuro se expondrá uno de ellos.

6

Resultados Experimentales en un CBVR

6.1 Introducción

En este capítulo se persigue validar el *iDistance* como motor de indexación de datos de video altamente dimensionales, así como probar la eficiencia de la recuperación de las K mejores subsecuencias en ranking del algoritmo SMSM.

Para realizar todos los experimentos se van a utilizar los mismos archivos de video usados en el capítulo 4, para realizar las pruebas con el *iDistance*. De dichos archivos se obtendrá la secuencia de consulta (Q), para todos los test realizados. Para crear dicha secuencia de consulta, se van a seguir los cuatro modelos propuestos en [SSH+09], que son los siguientes:

- *Group Reorder*: En este modelo se obtienen |Q| vectores consecutivos del archivo de datos, posteriormente el 20% de los vectores se barajan sus posiciones, es decir hay un reordenamiento parcial de la consulta.

- *Group Shorten*: Al igual que en el caso anterior, se obtienen |Q| vectores consecutivos del archivo de datos, y posteriormente se eliminan el 20% de los vectores.

- *Group Modify*: En este tipo de reorganización se obtienen |Q| vectores consecutivos del archivo de datos, posteriormente se divide en 5 partes (cada parte es el 20%) y aleatoriamente cada una de estas 5 partes pueden ser reordenados sus elementos, suprimir (poner a 0) el 20% de sus dimensiones o dejarla intacta.

- *Group Direct.* Por último, en este modelo se obtienen |Q| vectores consecutivos y se dejan intactos.

Este capítulo se divide en dos partes, una primera parte se centra en el comportamiento del *iDistance* dentro del framework CBVR para la consulta *VSI* y la segunda parte trata del estudio del ratio de acierto de la recuperación de las K subsecuencias mas similares.

6.2 Parámetros de estudio

Como ya se comentó en las conclusiones del capítulo 5, existen varias variables a tener en cuenta para realizar el estudio del framework CBVR. Algunas de ellas, ya se estudiaron en [Riv10] utilizando el *VA-File* como motor de indexación en un CBVR y otras las estudiaremos en este proyecto. En cuanto a la consulta *VSS*, no será objeto de estudio en este proyecto puesto que no tiene ninguna relación con el *iDistance* ni con la recuperación de las *kSS* subsecuencias más similares en ranking.

Para la consulta *VSI*:

- El correcto funcionamiento del *iDistance* y la comparación de resultados con el *VA-File* para el mismo framework y los mismos videos.
- La *distancia (Dist)* con la cual se procesa la consulta KNN en la recuperación de frames similares.
- El parámetro α para la división de segmentos.

- El parámetro β que indica el tamaño mínimo del matching para no descartar un segmento.
- Las funciones de similitud *(Sim)* de video.
- El numero de subsecuencias a devolver *kSS.*

A continuación, se detallan los parámetros ya estudiados en [Riv10] y que no serán objeto de estudio en este proyecto.

6.2.1 Parámetros fijos en la consulta VSI

En cuanto a la identificación de subsecuencias de video, podemos fijar los parámetros α y β ya estudiados en [Riv10].

En dicho proyecto, se observó que para consultas de tipo *Group Shorten* valores de α demasiado pequeños hacen decaer la efectividad del algoritmo de cálculo de segmentos densos (Código 12, capítulo 5). Por tanto el valor de α establecido como *apropiado* con los datos de video de los que se disponen es de 0.2. Puesto que un valor más pequeño hace que el número de segmentos densos sea más alto y por tanto de menor tamaño, disminuyendo así el número de probabilidades de contener una subsecuencia completa del tamaño de Q. Un valor más elevado de α podría provocar que el número de segmentos densos sea muy bajo y de longitud elevada, lo que conllevaría según [SSH+09] a un número de mappings 1:1 elevado con lo que habría puntuaciones muy parecidas y por tanto decaería la eficiencia del algoritmo.

En cuanto al valor β, se propone un valor de 0.5 puesto que asegura que los segmentos analizados tienen un matching con un número elevado de aristas.

Otro factor a tener en cuenta son las *funciones de similitud*. En este proyecto se persigue comprobar el comportamiento del *iDistance* en la identificación de subsecuencias de video. Por tanto en este ámbito las funciones de similitud para recuperar el mejor mapping 1:1 no repercute en el rendimiento del *iDistance*. Es por eso que de todas las funciones de similitud expuestas en [Riv10] se va a utilizar la descrita en el capítulo 5.

Por último, el valor de *K* para la recuperación de frames similares se establece a 100 para obtener aciertos superiores del 90% en la mayoría de los casos.

Por tanto, quedan establecidos, para los posteriores experimentos del capítulo, los valores siguientes como fijos:

Parámetro	Valor
Similitud de video (Sim)	$Sim(Q, \overrightarrow{S_k^*}, M_k) = \sum_{i \in \{VC,TO,FA\}} Sim_i(Q, \overrightarrow{S_k^*}, M_k)/3$
Parámetro α	*0.2*
Parámetro β	*0.5*
Parámetro K (kNN)	*100*

Tabla 26. Valores fijados para la identificación de subsecuencias de video.

6.3 Funcionamiento del iDistance en un CBVR

Para poder realizar una comparativa objetiva entre un CBVR con *VA-File* y un CBVR con *iDistance* es necesario en primer lugar comprobar que los resultados del algoritmo de recuperación de frames similares (capítulo 5, Código 10) con *iDistance* son idénticos en posición de similitud con respecto al del *VA-File*, para la misma distancia.

Experimento 12. *Comparativa de resultados entre el iDistance y el VA-File (sin BNN) en la recuperación de frames similares con una subsecuencia del conjunto de vectores de características.*

Parámetro	**Valor**
Dim	*12, 80, 128*
NP	*125378*
|Q|	*10*
K	*10*
PageSize	*4096*
Distancia	*Máxima*
IDistance Centers	*64*

Tabla 27. Experimento 12: Comparativa de resultados entre el *iDistance* y el *VA-File* (sin BNN) en la recuperación de frames similares.

Los resultados mostrados indican para cada $q_i \in Q$ los K vecinos más cercanos. En la columna de la izquierda se muestran los resultados del *iDistance (*ID y Distancia*)* y en los de la derecha los del *VA-File* sin BNN (ID y Distancia). La elección de ejecutar el *VA-File* sin BNN es simplemente por rapidez, puesto que los resultados son idénticos.

DIMENSIÓN 12	
KNNQuery iDistance	**KNNQuery VA-File (sin BNN)**
Punto de Consulta 0	Consulta 0
0 0.000000	0 0
12 0.038832	12 0.0388323
159 0.068448	159 0.0684479
2298 0.076261	2298 0.0762608
34 0.076698	34 0.0766983
161 0.076770	161 0.0767703
160 0.082176	160 0.0821756
162 0.092732	162 0.0927315
2301 0.095147	2301 0.0951472
56 0.097892	56 0.0978924
Punto de Consulta 1	Consulta 1
1 0.000000	1 0
3 0.147559	3 0.147559
97081 0.176942	97081 0.176942
75445 0.203279	75445 0.203279
97079 0.207354	97079 0.207354
75446 0.210774	75446 0.210774
75444 0.211769	75444 0.211769
97074 0.212404	97074 0.212404

7838	0.217686
97075	0.226016
Punto de Consulta 2	
2	0.000000
4499	0.052015
4498	0.079602
97	0.084566
4500	0.086886
96	0.092274
99	0.095600
2379	0.098722
98	0.101245
8985	0.103209
Punto de Consulta 3	
3	0.000000
7838	0.128318
1	0.147559
2316	0.158327
147	0.160066
86820	0.164805
66806	0.166943
66871	0.167717
6766	0.168622
9	0.170088
Punto de Consulta 4	
4	0.000000
7837	0.098454
2311	0.098506
7877	0.099914
36793	0.102586
36794	0.102586
1279	0.104515
36792	**0.108625**
36790	**0.108625**
36788	0.110303
Punto de Consulta 5	
5	0.000000
6	0.088811
36798	0.092544
36796	**0.098364**
36795	**0.098364**
36797	0.104572
36794	**0.106826**
36793	**0.106826**
4	0.115660
1279	0.118122
Punto de Consulta 6	
6	0.000000
5	0.088811
2312	0.106838

7838	0.217686
97075	0.226016
Consulta 2	
2	0
4499	0.052015
4498	0.0796015
97	0.0845659
4500	0.0868862
96	0.0922736
99	0.0956002
2379	0.0987218
98	0.101245
8985	0.103209
Consulta 3	
3	0
7838	0.128318
1	0.147559
2316	0.158327
147	0.160066
86820	0.164805
66806	0.166943
66871	0.167717
6766	0.168622
9	0.170088
Consulta 4	
4	0
7837	0.0984542
2311	0.0985064
7877	0.099914
36793	0.102586
36794	0.102586
1279	0.104515
36790	**0.108625**
36792	**0.108625**
36788	0.110303
Consulta 5	
5	0
6	0.0888114
36798	0.0925445
36795	**0.0983644**
36796	**0.0983644**
36797	0.104572
36793	**0.106826**
36794	**0.106826**
4	0.11566
1279	0.118122
Consulta 6	
6	0
5	0.0888114
2312	0.106838

1279 0.115544 36795 0.129733 36796 0.129733 84357 0.131870 84359 0.133073 2311 0.133472 36798 0.133947 Punto de Consulta 7 7 0.000000 22094 0.098552 22092 0.099949 22093 0.109246 22091 0.113644 **67122 0.116180** **67125 0.116180** **67120 0.116180** 22095 0.120579 67118 0.120868 Punto de Consulta 8 8 0.000000 7874 0.098194 9 0.106185 4498 0.119179 1200 0.134944 4546 0.136341 4544 0.136724 4545 0.137262 4500 0.137528 3497 0.138543 Punto de Consulta 9 9 0.000000 8 0.106185 2310 0.130206 4500 0.133461 1249 0.138841 53840 0.144010 53841 0.144010 38952 0.145433 4498 0.146048 4546 0.146453 10 queries TIME PER QUERY: 0.300000	1279 0.115544 36795 0.129733 36796 0.129733 84357 0.13187 84359 0.133073 2311 0.133472 36798 0.133947 Consulta 7 7 0 22094 0.0985518 22092 0.0999485 22093 0.109246 22091 0.113644 **67120 0.11618** **67122 0.11618** **67125 0.11618** 22095 0.120579 66983 0.120868 Consulta 8 8 0 7874 0.0981938 9 0.106185 4498 0.119179 1200 0.134944 4546 0.136341 4544 0.136724 4545 0.137262 4500 0.137528 3497 0.138543 Consulta 9 9 0 8 0.106185 2310 0.130206 4500 0.133461 1249 0.138841 53840 0.14401 53841 0.14401 38952 0.145433 4498 0.146048 4546 0.146453 10 queries TIME PER QUERY: 0.563781

Tabla 28. Resultados del experimento 12. Dimensión 12.

DIMENSIÓN 80	
KNNQuery iDistance	**KNNQuery VA-File (sin BNN)**
Punto de Consulta 0	Consulta 0
0 0.000000	0 0
44758 0.150159	44758 0.150159
44850 0.158754	44850 0.158754
44826 0.160763	**44825 0.160763**
44825 0.160763	**44826 0.160763**
44828 0.160763	**44827 0.160763**
44830 0.160763	**44828 0.160763**
44827 0.160763	**44830 0.160763**
44759 0.163492	44759 0.163492
43075 0.166122	43075 0.166122
Punto de Consulta 1	Consulta 1
1 0.000000	1 0
3445 0.292655	3445 0.292655
9744 0.312291	9744 0.312291
6789 0.317525	6789 0.317525
78 0.322306	78 0.322306
4538 0.324540	4538 0.32454
41628 0.324614	41628 0.324614
8971 0.325750	8971 0.32575
6790 0.326647	6790 0.326647
9746 0.327912	9746 0.327912
Punto de Consulta 2	Consulta 2
2 0.000000	2 0
104 0.281675	104 0.281675
8970 0.288115	8970 0.288115
7876 0.288318	7876 0.288318
8943 0.297025	8943 0.297025
5626 0.297291	5626 0.297291
3406 0.300210	3406 0.300209
3407 0.305219	3407 0.305219
6730 0.306466	6730 0.306466
90142 0.308546	90142 0.308546
Punto de Consulta 3	Consulta 3
3 0.000000	3 0
9190 0.252854	9190 0.252854
9191 0.253501	9191 0.253501
115502 0.267275	115502 0.267275
4527 0.268256	4527 0.268256
35041 0.270622	35041 0.270622
6790 0.271083	6790 0.271083
2322 0.272518	2322 0.272518
4005 0.277316	4005 0.277316
115530 0.278314	115530 0.278314
Punto de Consulta 4	Consulta 4
4 0.000000	4 0
7866 0.188165	7866 0.188165
6756 0.211202	6756 0.211202

8985 0.229818 2361 0.236223 6754 0.244407 102068 0.246965 8929 0.247438 7867 0.248019 102066 0.248130	8985 0.229818 2361 0.236223 6754 0.244407 102068 0.246965 8929 0.247438 7867 0.248019 102066 0.24813
Punto de Consulta 5 5 0.000000 1279 0.228101 187 0.231909 1278 0.233798 3433 0.258381 8975 0.259081 4536 0.259917 6 0.260704 7871 0.266038 7870 0.266113	Consulta 5 5 0 1279 0.228101 187 0.231909 1278 0.233798 3433 0.258381 8975 0.259081 4536 0.259917 6 0.260704 7871 0.266038 7870 0.266113
Punto de Consulta 6 6 0.000000 8975 0.249937 7801 0.250166 2333 0.258013 7870 0.258235 5 0.260704 7840 0.265506 4533 0.272357 38 0.274776 4529 0.277278	Consulta 6 6 0 8975 0.249937 7801 0.250166 2333 0.258013 7870 0.258235 5 0.260704 7840 0.265506 4533 0.272357 38 0.274776 4529 0.277278
Punto de Consulta 7 7 0.000000 46 0.227756 21 0.243222 106658 0.244798 8954 0.251666 2331 0.256818 4515 0.257543 1273 0.259302 3428 0.265763 138 0.271175	Consulta 7 7 0 46 0.227756 21 0.243222 106658 0.244798 8954 0.251666 2331 0.256818 4515 0.257543 1273 0.259302 3428 0.265763 138 0.271175
Punto de Consulta 8 8 0.000000 4515 0.268352 7 0.279507 4508 0.314388 21 0.327192 8954 0.330303 138 0.335068 7874 0.335200 100963 0.335218 4500 0.337796	Consulta 8 8 0 4515 0.268352 7 0.279507 4508 0.314388 21 0.327192 8954 0.330303 138 0.335068 7874 0.3352 100963 0.335218 4500 0.337796

Punto de Consulta 9 9 0.000000 7857 0.305655 316 0.310253 33438 0.315445 327 0.319698 8305 0.322893 35212 0.324201 329 0.325856 345 0.326306 326 0.331139 10 queries TIME PER QUERY: 0.200000	Consulta 9 9 0 7857 0.305655 316 0.310253 33438 0.315445 327 0.319698 8305 0.322893 35212 0.324201 329 0.325856 345 0.326306 326 0.331139 10 queries TIME PER QUERY: 1.15603

Tabla 29. Resultados del experimento 12. Dimensión 80.

DIMENSIÓN 128	
KNNQuery iDistance	**KNNQuery VA-File (sin BNN)**
Punto de Consulta 0 0 0.000000 67 0.298034 23 0.352855 56 0.448509 1253 0.494132 160 0.556929 12 0.576921 3439 0.588207 161 0.607194 171 0.619627 Punto de Consulta 1 1 0.000000 3 0.695032 89 0.756711 7873 0.779677 78 0.786755 145 0.804933 144 0.836115 6761 0.844474 4536 0.845803 4532 0.848281 Punto de Consulta 2 2 0.000000 99 0.296612 98 0.396444 96 0.461899 4524 0.478585 97 0.488871 4523 0.516414 4499 0.554600	Consulta 0 0 0 67 0.298034 23 0.352855 56 0.448509 1253 0.494132 160 0.556929 12 0.576921 3439 0.588207 161 0.607194 171 0.619627 Consulta 1 1 0 3 0.695032 89 0.756711 7873 0.779677 78 0.786755 145 0.804933 144 0.836115 6761 0.844474 4536 0.845803 4532 0.848281 Consulta 2 2 0 99 0.296612 98 0.396444 96 0.461899 4524 0.478585 97 0.488871 4523 0.516414 4499 0.554600

4498	0.572486
3497	0.573286
Punto de Consulta 3	
3	0.000000
78	0.553907
1	0.695032
5685	0.734759
5686	0.738760
89	0.767259
384	0.781334
8981	0.793380
8983	0.795497
5688	0.819687
Punto de Consulta 4	
4	0.000000
129	0.525041
130	0.571905
3413	0.597723
3410	0.609531
2317	0.627616
3412	0.631826
144	0.647322
46	0.664648
7819	0.670891
Punto de Consulta 5	
5	0.000000
6	0.665437
4535	0.670643
2318	0.688056
2312	0.695558
142	0.695759
7873	0.717488
6724	0.736925
2317	0.741524
7884	0.741898
Punto de Consulta 6	
6	0.000000
3420	0.608220
3416	0.611651
14	0.643693
7830	0.654787
7832	0.660518
5	0.665437
7834	0.667350
7831	0.668949
2322	0.669856
Punto de Consulta 7	
7	0.000000
129	0.731707
7799	0.742980

4498	0.572486
3497	0.573286
Consulta 3	
3	0
78	0.553907
1	0.695032
5685	0.734759
5686	0.738760
89	0.767259
384	0.781334
8981	0.793380
8983	0.795497
5688	0.819687
Consulta 4	
4	0
129	0.525041
130	0.571905
3413	0.597723
3410	0.609531
2317	0.627616
3412	0.631826
144	0.647322
46	0.664648
7819	0.670891
Consulta 5	
5	0
6	0.665437
4535	0.670643
2318	0.688056
2312	0.695558
142	0.695759
7873	0.717488
6724	0.736925
2317	0.741524
7884	0.741898
Consulta 6	
6	0
3420	0.608220
3416	0.611651
14	0.643693
7830	0.654787
7832	0.660518
5	0.665437
7834	0.667350
7831	0.668949
2322	0.669856
Consulta 7	
7	0
129	0.731707
7799	0.742980

2315 0.745130 7831 0.777678 4 0.802326 7808 0.806621 128 0.807153 7819 0.808211 104 0.825657 Punto de Consulta 8 8 0.000000 2316 0.506814 2315 0.703102 45999 0.720188 9 0.809733 2 0.816729 8901 0.825430 386 0.829261 40651 0.831283 8902 0.834897 Punto de Consulta 9 9 0.000000 1203 0.538765 6760 0.588406 7874 0.614150 4537 0.628354 2316 0.667461 1199 0.726033 4525 0.742461 4526 0.744664 1201 0.754415 10 queries TIME PER QUERY: 4.300000	2315 0.745130 7831 0.777678 4 0.802326 7808 0.806621 128 0.807153 7819 0.808211 104 0.825657 Consulta 8 8 0.000000 2316 0.506814 2315 0.703102 45999 0.720188 9 0.809733 2 0.816729 8901 0.825430 386 0.829261 40651 0.831283 8902 0.834897 Consulta 9 9 0.000000 1203 0.538765 6760 0.588406 7874 0.614150 4537 0.628354 2316 0.667461 1199 0.726033 4525 0.742461 4526 0.744664 1201 0.754415 10 queries TIME PER QUERY: 5.04017

Tabla 30. Resultados del experimento 12. Dimensión 128.

Como podemos ver, excepto algunos errores de precisión en la distancia (**en negrita**), el resultado de obtener los K vecinos de cada punto de consulta es exactamente igual tanto el *iDistance* como en el *VA-File*. Puesto que los demás algoritmos trabajan con el conjunto de frames similares recuperados del Código 10, los resultados son idénticos en los sucesivos pasos de la identificación de subsecuencias.

Tras comprobar que presentan los mismos resultados, vamos a estudiar como responde dentro de un CBVR el *iDistance* en la consulta *VSI*.

Experimento 13: *Estudio del rendimiento del CBVR para la consulta VSI usando el iDistance.*

En dicho experimento, se utilizaran los cuatro tipos de secuencias de consulta: *Group Reorder, Group Shorten, Group Modify y Group Direct.* Se variaran los valores de K y la dimensión de los datos.

Parámetro	Valor
Dim	*12, 80, 128*
NP	*125378*
Q	*Reorder, Modify, Shorten y Directa*
|Q|	*20*
K	*100*
PageSize	*4096*
Distancia	*Máxima*
iDistance Centers	*64*
iDistance CompactRate	*90%*
K Best Subsequences (kSS)	*1 (Basic)*

Tabla 31. Experimento 13, estudio del rendimiento del CBVR para la consulta *VSI* usando el *iDistance*.

Para calcular el acierto (*Hit Ratio*) de la identificación de subsecuencias, se compara la subsecuencia identificada con la original para ver que puntos coinciden con la consulta original, Q, tal y como se muestra el Código 18.

Primero se expondrán los resultados de todos los tipos de consultas para la dimensión 12 y así sucesivamente hasta 128 dimensiones.

Dimensión 12: Group Reorder	
Obtener Frames Similares	*KNN iDistance* *2.71356 s*
Generar Grafo Bipartito	*0.000190039 s*
Calcular Segmentos Densos	*Segmentos Densos = 88* *0.000225988 s*
Filtrado MSM	*Segmentos Candidatos = 2* *0.0213651 s*
Refinado SMSM **Secuencia identificada**	***0.00487408 s*** ***Segmentos Coincidentes con Q = 18*** ***HitRario = 90%***
Dimensión 12: Group Modify	
Obtener Frames Similares	*KNN iDistance* *0.308212 s*
Generar Grafo Bipartito	*0.000243924 s*
Calcular Segmentos Densos	*Segmentos Densos = 142* *0.00050505 s*
Filtrado MSM	*Segmentos Candidatos = 4* *0.0143609 s*
Refinado SMSM **Secuencia identificada**	***0.00400194 s*** ***Segmentos Coincidentes con Q = 20*** ***Hit Ratio = 100%***
Dimensión 12: Group Shorten	
Obtener Frames Similares	*KNN iDistance* *0.633175 s*

Generar Grafo Bipartito	*0.000246969 s*
Calcular Segmentos Densos	*Segmentos Densos = 343* *0.000927957 s*
Filtrado MSM	*Segmentos Candidatos = 1* *0.013483 s*
Refinado SMSM **Secuencia identificada**	***0.00142791 s*** ***Segmentos Coincidentes con Q = 20*** ***Hit Ratio = 100%***
Dimensión 12: Group Direct	
Obtener Frames Similares	*KNN iDistance* *0.366307 s*
Generar Grafo Bipartito	*0.000218929 s*
Calcular Segmentos Densos	*Segmentos Densos = 60* *0.000177903 s*
Filtrado MSM	*Segmentos Candidatos = 1* *0.089628 s*
Refinado SMSM, **Secuencia identificada**	***0.00448606 s*** ***Segmentos Coincidentes con Q = 20*** ***Hit Ratio = 100%***

Tabla32. Resultados del experimento 13, dimensión 12.

Dimensión 80: Group Reorder	
Obtener Frames Similares	*KNN iDistance* *5.93546 s*
Generar Grafo Bipartito	*0.000150073 s*
Calcular Segmentos Densos	*Segmentos Densos = 9* *6.30008e-05 s*
Filtrado MSM	*Segmentos Candidatos = 3* *0.0686089 s*
Refinado SMSM **Secuencia identificada**	***0.101203 s*** ***Segmentos Coincidentes con Q = 0*** ***Hit Ratio = 0%***
Dimensión 80: Group Modify	
Obtener Frames Similares	*KNN iDistance* *4.42618 s*
Generar Grafo Bipartito	*0.000353029 s*
Calcular Segmentos Densos	*Segmentos Densos = 382* *0.000942005 s*
Filtrado MSM	*Segmentos Candidatos = 1* *0.0168631 s*
Refinado SMSM **Secuencia identificada**	***0.000971036 s*** ***Segmentos Coincidentes con Q = 20*** ***Hit Ratio = 100%***

Dimensión 80: Group Shorten	
Obtener Frames Similares	*KNN iDistance* *4.12737 s*
Generar Grafo Bipartito	*0.000206939 s*
Calcular Segmentos Densos	*Segmentos Densos = 314* *0.000780956 s*
Filtrado MSM	*Segmentos Candidatos = 1* *0.0120139 s*
Refinado SMSM **Secuencia identificada**	***0.00238297*** ***Segmentos Coincidentes con Q = 20*** ***Hit Ratio = 100%***
Dimensión 80: Group Direct	
Obtener Frames Similares	*KNN iDistance* *3.96597 s*
Generar Grafo Bipartito	*0.000392033 s*
Calcular Segmentos Densos	*Segmentos Densos = 155* *0.000494888 s*
Filtrado MSM	*Segmentos Candidatos = 2* *0.0189699 s*
Refinado SMSM **Secuencia identificada**	***0.0030511s*** ***Segmentos Coincidentes con Q = 20*** ***Hit Ratio = 100%***

Tabla 33. Resultados del experimento 13, dimensión 80.

Dimensión 128: Group Reorder	
Obtener Frames Similares	*KNN iDistance* *29.7026 s*
Generar Grafo Bipartito	*0.000211917 s*
Calcular Segmentos Densos	*Segmentos Densos = 54* *0.000186116 s*
Filtrado MSM	*Segmentos Candidatos = 2* *0.013184 s*
Refinado SMSM **Secuencia identificada**	***0.00704405 s*** ***Segmentos Coincidentes con Q = 20*** ***Hit Ratio = 100%***
Dimensión 128: Group Modify	
Obtener Frames Similares	*KNN iDistance* *6.1611 s*
Generar Grafo Bipartito	*0.000398888 s*
Calcular Segmentos Densos	*Segmentos Densos = 300* *0.000783936 s*
Filtrado MSM	*Segmentos Candidatos = 4* *0.0162131 s*

Refinado SMSM **Secuencia identificada**	***0.00168795 s*** ***Segmentos Coincidentes con Q = 20*** ***Hit Ratio = 100%***
Dimensión 128: Group Shorten	
Obtener Frames Similares	*KNN iDistance* *2.13468 s*
Generar Grafo Bipartito	*0.000256066 s*
Calcular Segmentos Densos	*Segmentos Densos = 36* *0.000158004 s*
Filtrado MSM	*Segmentos Candidatos = 3* *0.097924 s*
Refinado SMSM **Secuencia identificada**	***0.0448189 s*** ***Segmentos Coincidentes con Q = 0*** ***Hit Ratio = 0%***
Dimensión 128: Group Direct	
Obtener Frames Similares	*KNN iDistance* *4.77635 s*
Generar Grafo Bipartito	*0.000331043 s*
Calcular Segmentos Densos	*Segmentos Densos = 202* *0.000496998 s*
Filtrado MSM	*Segmentos Candidatos = 2* *0.0203229 s*
Refinado SMSM **Secuencia identificada**	***0.00194009 s*** ***Segmentos Coincidentes con Q = 20*** ***Hit Ratio = 100%***

Tabla 34. Resultados del experimento 13, dimensión 128.

Como se puede observar, de los resultados en las tablas 32, 33 y 34. Las consultas que mejores resultados presentan son: *Direct y Modify* presentando aciertos (hit ratios) del 100% en todas las dimensiones. *Group Reorder y Modify* sin embargo presentan bajos hit ratios en algunas consultas. Esto es debido a que en ocasiones la subsecuencia elegida como mas apropiada en el algoritmo SMSM (código 17 del capítulo 5) basada en la función de similitud propuesta (tabla 26) no presenta buenos resultados para la recuperación de UNA sola secuencia mas apropiada. Es por eso que en el siguiente apartado, se va a realizar el estudio para la recuperación de las K mejores subsecuencias en ranking del algoritmo SMSM.

Aun así, los resultados de la identificación de subsecuencias son muy buenos obteniéndose un hit ratio medio de 82% para todas las pruebas realizadas en el experimento 13. Dichos resultados coinciden con los obtenidos en el CBVR de [Riv10] usando el *VA-File* como motor de indexación.

6.4 Recuperación de las K Subsecuencias más similares

Tras los resultados del *experimento 13*, se hace evidente estudiar la variación del hit ratio en la identificación de subsecuencias con la obtención de las K mejores subsecuencias en ranking. Con esto, se pretende aumentar el hit ratio en las ocasiones que la función de similitud usada (Tabla 26) no de los resultados deseados: es posible que si la que mejor similitud presente no sea igual que la secuencia de consulta Q, las K-1 siguientes subsecuencias recuperadas si que tengan mejor puntuación.

Con el experimento 14, se pretende analizar la mejora del hit ratio con el uso del algoritmo *GetKBestSubsecuences* expuesto en el Código 17 variando *kSS*.

Experimento 14. *Estudio del hit ratio con la variación de kSS y la dimensión en el algoritmo GetKBestSubsecuences con los cuatro tipos de consultas propuestos: Group Reorder, Modify, Shorten y Direct.*

Parámetro	**Valor**
Dim	*12, 80, 128*
NP	*125378*
Q	*Reorder, Modify, Shorten y Directa*
|Q|	*20*
K	*100*
PageSize	*4096*
Distancia	*Máxima*
iDistance Centers	*64*
iDistance CompactRate	*90%*
K Best Subsequences (kSS)	*1, 5, 10 y 15*

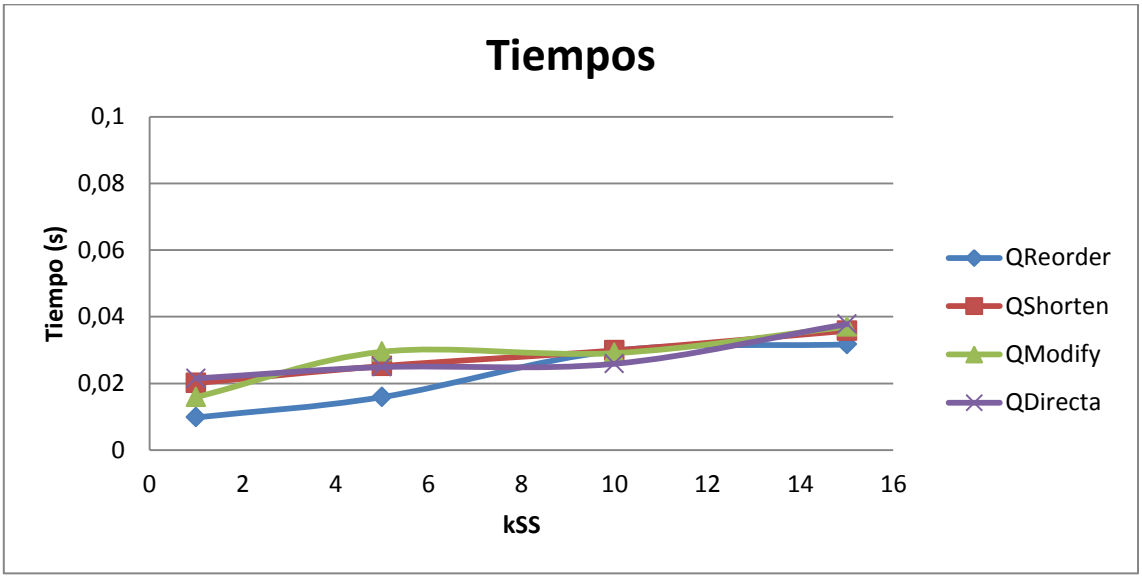
Tabla 35. Experimento 14: Estudio del hit ratio con la variación de *kSS*.

Por cada dimensión, se expondrán los resultados de cada consulta con los distintos valores de *kSS*. Tras los resultados de cada dimensión, se expondrán las gráficas de tiempo e hit ratio en relación con la variación de *kSS*.

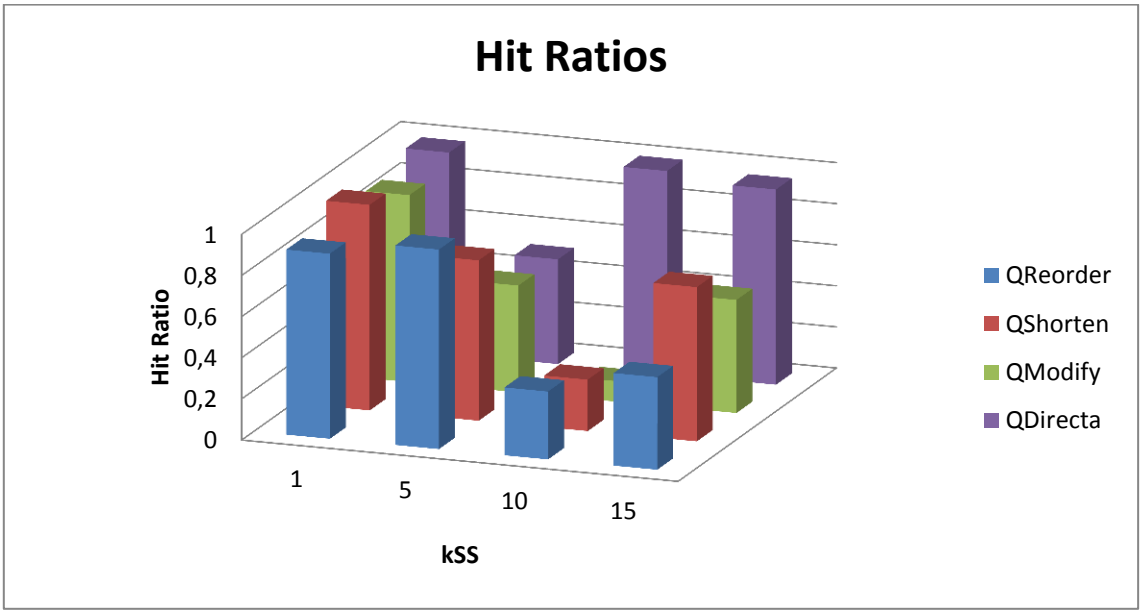
DIMESNION 12		
kSS	**Tiempo (s)**	**Hit Ratio (%)**
Group Reorder		
1	*0,0098*	*90*
5	*0,0159*	***97***
10	*0,0298*	*33*
15	*0,0317*	*45*
Group Shorten		
1	*0,0201*	***100***
5	*0,0252*	*78*
10	*0,0299*	*25*
15	*0,0358*	*75*
Group Modify		
1	*0,0159*	***91***
5	*0,0295*	*52*
10	*0,0291*	*10*

15	*0,037*	*55*
Group Direct		
1	*0,0215*	*98*
5	*0,0249*	*51*
10	*0,0259*	***99***
15	*0,0378*	*95*

Tabla 36. Resultados del experimento 14, dimensión 12.



Gráfica 21. Evolución del tiempo del algoritmo *GetKBestSubsequences* con la variación de *kSS*. Dimensión 12.

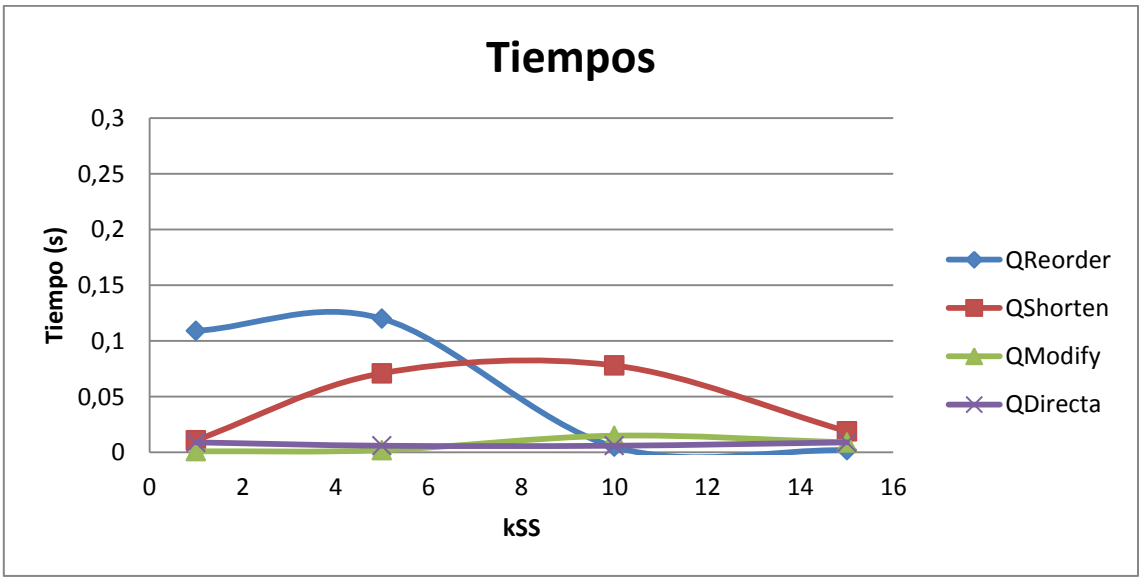


Gráfica 22. Variación del hit ratio en el algoritmo *GetKBestSubsequences* con la variación de *kSS*. Dimensión 12.

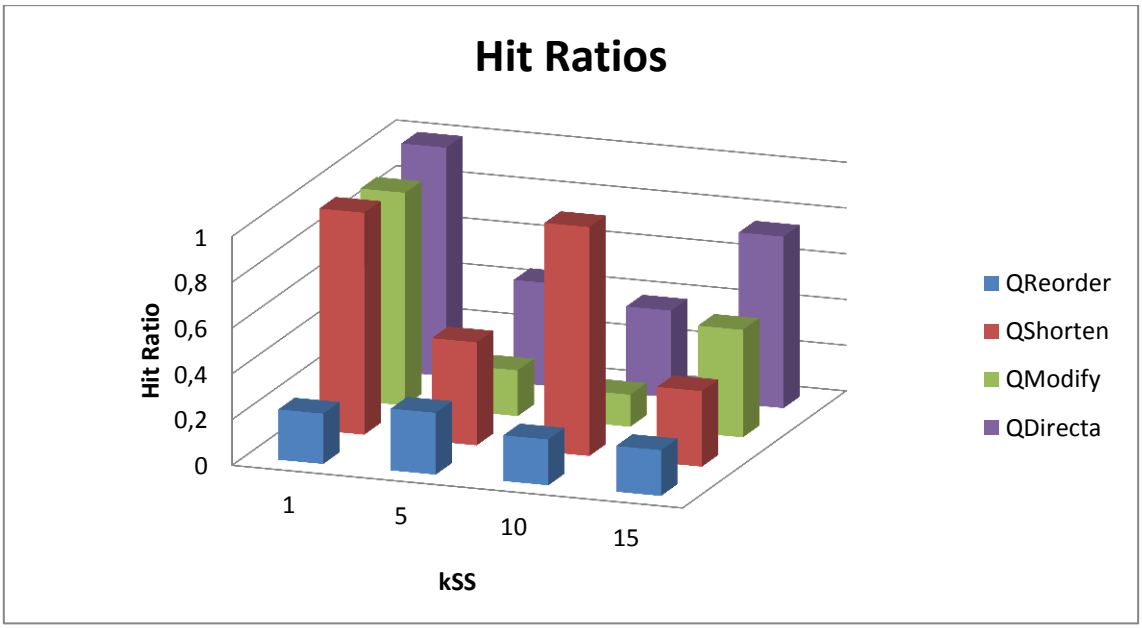
Se prosigue con los datos de dimensión 80:

DIMENSION 80		
kSS	**Tiempo (s)**	**Hit Ratio (%)**
Group Reorder		
1	*0,109*	*22*
5	*0,120*	***27***
10	*0,005*	*20*
15	*0,002*	*20*
Group Shorten		
1	*0,011*	*97*
5	*0,071*	*45*
10	*0,078*	***100***
15	*0,019*	*33*
Group Modify		
1	*0,001*	***93***
5	*0,002*	*20*
10	*0,015*	*14*
15	*0,009*	*47*
Group Direct		
1	*0,009*	***100***
5	*0,006*	*45*
10	*0,006*	*38*
15	*0,009*	*75*

Tabla 37. Resultados del experimento 14, dimensión 80.



Gráfica 23. Evolución del tiempo del algoritmo *GetKBestSubsequences* con la variación de *kSS*. Dimensión 80.

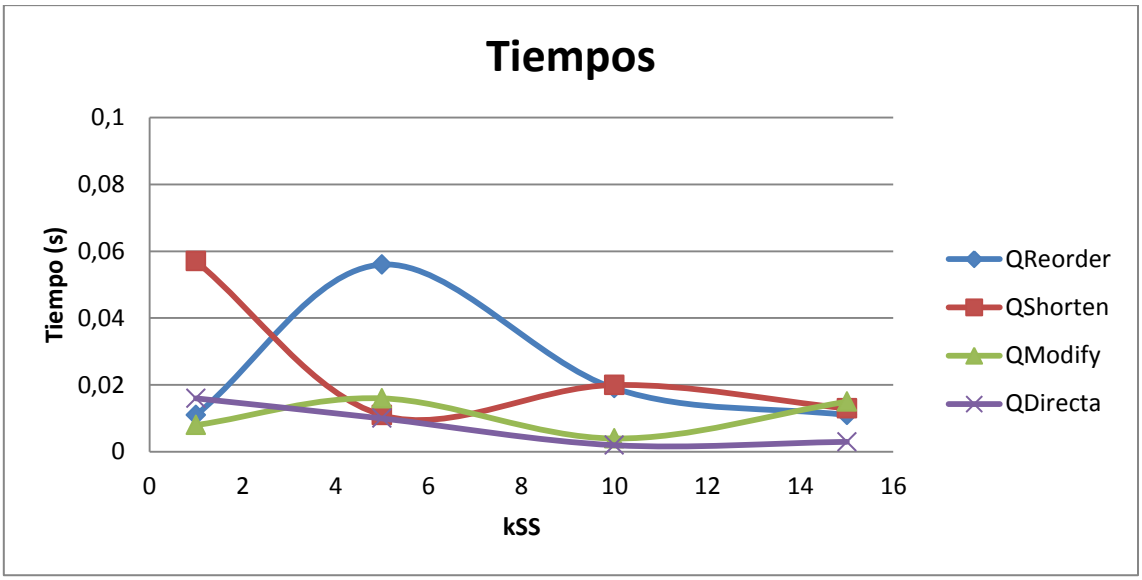


Gráfica 24. Variación del hit ratio en el algoritmo *GetKBestSubsequences* con la variación de *kSS*. Dimensión 80.

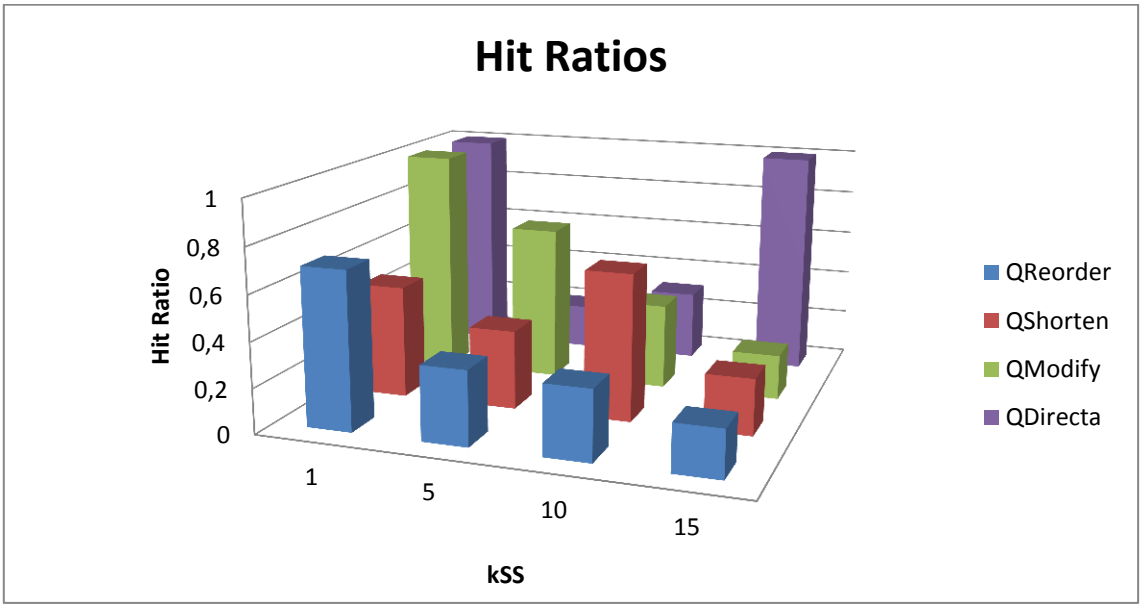
Por último, datos de dimensión 128:

DIMENSION 128		
kSS	**Tiempo (s)**	**Hit Ratio (%)**
Group Reorder		
1	*0,011*	***70***
5	*0,056*	*33*
10	*0,019*	*31*
15	*0,011*	*21*
Group Shorten		
1	*0,057*	*50*
5	*0,011*	*35*
10	*0,020*	***65***
15	*0,013*	*25*
Group Modify		
1	*0,008*	***100***
5	*0,016*	*69*
10	*0,004*	*38*
15	*0,015*	*20*
Group Direct		
1	*0,016*	***100***
5	*0,010*	*20*
10	*0,002*	*31*
15	*0,003*	***100***

Tabla 38. Resultados del experimento 14, dimensión 128.



Gráfica 25. Evolución del tiempo del algoritmo *GetKBestSubsequences* con la variación de *kSS*. Dimensión 128.



Gráfica 26. Variación del hit ratio en el algoritmo *GetKBestSubsequences* con la variación de *kSS*. Dimensión 128.

Si observamos el tiempo, éste varía según el *kSS* elegido y los matching (1:1) refinados por el algoritmo SMSM: si hay menos matchings que *kSS* entonces el tiempo se vera reducido, aunque dicha variación es mínima. Por lo tanto, el tiempo depende en mayor medida del número de segmentos candidatos con los que trabaja el refinamiento SMSM del algoritmo *GetKBestSubsequences*.

En cuanto al hit ratio, vemos que en la mayoría de las ocasiones cuando el ratio para la mejor subsecuencia (kSS=1) es de 0%, el aumentar el numero de secuencias a recuperar (kSS) puede proporcionar una leve mejora en el acierto con la secuencia de consulta Q.

Aun así, el comportamiento del hit ratio depende más de la secuencia de consulta Q que del propio algoritmo: si la secuencia de consulta se basa en frames muy repetidos en la base de datos, valores de kSS superiores a 1 puede mejorar el hit ratio puesto que es muy posible que se encuentren varias secuencias candidatas. En cambio, si la secuencia de consulta esta basada en frames muy singulares (poco repetidos en características) dentro de la base de datos, el valor de kSS debe de ser 1 o muy cercano a 1, puesto que es muy posible que no haya demasiadas subsecuencias que pueden ser identificadas como similares.

6.5 Conclusiones

Podemos concluir este capítulo, afirmando que el *iDistance* tiene un comportamiento ejemplar en un sistema de recuperación de video basado en contenido altamente dimensional, a la vista de los estudios realizados. Además puede competir con el rendimiento del *VA-File* llegando incluso a mejorarlo.

En cuanto a la recuperación de las K Subsecuencias en ranking, este añadido al CBVR puede ser muy útil cuando se persigue recuperar varias secuencias de la base de datos muy parecidas en cuanto a sus características.

Un uso lógico de dicho añadido seria en las ocasiones en las que se dispone de frames donde aparece alguna escena similar a la que pueda aparecer repetida en una misma película. Si intentamos recuperar secuencias muy repetidas en similitud en la base de datos es mejor utilizar un valor de *kSS* alto, sin embargo si la secuencia de consulta es muy singular con respecto a los datos de la base de datos es mejor proporcionar un valor de *kSS* bajo, para no bajar en exceso el acierto.

También cabe, usar el parámetro *kSS* para el estudio del acierto de la función de similitud usada puesto que, si sabemos que la subsecuencia de consulta es fácilmente identificable y con un *kSS* bajo obtenemos un bajo acierto y aumentando dicho parámetro obtenemos mejor acierto, quiere decir que la función de similitud no está actuando lo mejor posible y esta valorando peor subsecuencias con mas acierto que la que dicha función de similitud coloca como la mejor subsecuencia (o subsecuencia mas similar a la subsecuencia de consulta).

7

Conclusiones y Trabajo Futuro

7.1 Conclusiones

Las contribuciones más importantes de este proyecto han sido las siguientes. La implementación del *iDistance* como motor de indexación de vectores de características altamente dimensionales. Junto con el *iDistance*, se ha estudiado y usado un algoritmo de clustering como es el *kmeans* de Lloyd [KMN+02]. Una vez realizada la implementación estable del *iDistance*, se ha incluido en un framework de recuperación de video basada en contenido como el de [Riv10] y se ha comprobado el rendimiento del mismo frente a la recuperación de frames similares en la identificación de subsecuencias de video.

En dicho framework, se ha realizado una mejora del algoritmo de refinamiento SMSM (Código 17) para la recuperación de las *k* subsecuencias más similares en ranking.

En cuanto a los resultados experimentales, el *iDistance* presenta un comportamiento bastante alto en comparación con el *VA-File* para un número de centros superior a 32 y un factor de compactación alto, tal y como se concluye en el capítulo 4. Es evidente, que a cuanto mayor dimensionalidad y mayor es el parámetro K, el rendimiento decae, pero éste se mantiene con un crecimiento estable (lineal) para dimensiones mayores de 80. La configuración del *iDistance* con la cual se han obtenido los mejores resultados viene dada en las conclusiones del capítulo 4 con: 64 puntos de referencia calculados como mínimo con 10 iteraciones del *kmeans*, un factor de compactación del 90% y un tamaño de página igual al tamaño de bloque del sistema operativo en disco, que en nuestro caso son 4096 bytes. La distancia que menor tiempo de computo presenta es la máxima.

En cuanto al CBVR usado, podemos decir que se han obtenido los mismos resultados con el *iDistance* que con el *VA-File* en la recuperación de frames similares, presentando el *iDistance* un mejor rendimiento en la mayoría de las ocasiones. Como ya se comentó anteriormente, dicho rendimiento es inversamente proporcional a la dimensión de los datos, a la duración de la secuencia de consulta |Q| y al numero de vecinos a recuperar K.

Para la recuperación de las *kSS* mejores subsecuencias en la identificación de video, se ha demostrado que el *hit ratio* depende en mayor medida de la función de similitud usada y de si la subsecuencia de consulta esta repetida o aparece casi igual en varias ocasiones en la base de datos S. No es posible fijar un valor de *kSS* para todas las consultas de identificación de video. Si prevemos que la secuencia de consulta Q va a ser identificada en numerosas subsecuencias de la base de datos podemos establecer un valor de *kSS* alto (10 o 15) y obtendremos un *hit ratio* mas que aceptable (entre el 80% y 100%), pero si por el contrario sabemos que la secuencia de consulta es poco probable que sea bien identificada por nuestra función de similitud se propone utilizar un valor para *kSS* bajo (entre 1 y 3) para obtener mejores aciertos.

No obstante, si no sabemos cómo es nuestra secuencia de consulta (Q), podemos establecer un valor alto de *kSS*, si el *hit ratio* medio es muy bajo podemos asegurar que Q no va a ser identificada con facilidad en nuestra base de datos puesto que no hay subsecuencias que se parezcan a ella lo suficiente y por tanto es aconsejable establecer el valor de *kSS* a 1, puesto que las demás subsecuencias identificadas pueden resultar demasiado distintas.

7.2 Trabajo futuro

En este apartado, se proponen posibles trabajos futuros como continuación de este proyecto fin de carrera en base al *iDistance* implementado y la identificación de subsecuencias de video en el CBVR usado. Primero se harán referencias a posibles mejoras en el *iDistance* para terminar con el CBVR estudiado.

7.2.1 Mejoras en el iDistance

Dadas las conclusiones obtenidas en el capítulo 4, podemos proponer varias mejoras y trabajo futuro en las distintas partes del diseño del *iDistance*, tanto en la selección de puntos de referencia, como en la consulta de los K vecinos más próximos y una posible solución para los falsos positivos del *iDistance* para puntos con el mismo valor de distancia en una misma división de los datos.

7.2.1.1 Selección de puntos de referencia

Tras el estudio e implementación realizados en los capítulos 3 y 4, se hace evidente, la importancia que tiene en el rendimiento del *iDistance* la selección de distintos tipos de puntos de referencia. Como ya se expuso en el capítulo 3, existen dos estrategias básicas de selección de puntos de referencia:

- Basados en la división del espacio métrico M^{dim}.
- Basados en las características de la distribución de puntos de la base de datos *(S)*.

En este proyecto, dicho estudio se centra en puntos de referencia basados en clústeres. De manera teórica se ha demostrado cómo puede afectar la selección de distintas estrategias de construcción de dichos puntos de referencia en la consulta de los K vecinos más cercanos, pero se hace visible la posibilidad de estudiar de manera experimental la selección de otras estrategias de generación de puntos de referencia como pueden ser:

- Basadas en el espacio de datos *dim*-dimensional (para datos uniformes):
    - Centro del hiperplano con la distancia mínima para cubrir la partición.
    - Centro del hiperplano con la distancia máxima al punto más lejano.
    - Punto exterior.

- Basadas en las características de los datos:
    - Puntos de referencia como las esquinas de cada dimensión.

En el estudio de la selección de puntos de referencia basados en las características o distribución de los datos, cabe la posibilidad de introducir diferentes algoritmos de clustering para ver el comportamiento en la generación de centros intentando disminuir el tiempo de respuesta del *kmeans* sin perder efectividad en las ejecuciones del *iDistance*.

7.2.1.2 Modificación del conjunto de datos

El estudio del *iDistance* realizado en este proyecto comprueba la eficiencia del mismo con respecto a conjuntos de datos que no han sido modificados una vez creado el árbol B+. En los resultados obtenidos, el *iDistance* presenta un comportamiento ejemplar para datos altamente dimensionales.

Una posible ampliación del trabajo realizado, sería el estudio de la validez de los puntos de referencia del *iDistance* tras la inserción y eliminación de puntos en el índice. Para obtener un factor el cual indicase en que momento deberían recalcularse los puntos de referencia del índice para su correcto funcionamiento.

7.2.1.3 Procesamiento por lotes en la KNN Query

Como hemos visto en la comparación entre el *iDistance* y el *VA-File*, un procesamiento por lotes de la consulta KNN (Batch Nearest Neighbours, BNN) [Riv10] muestra una eficiencia superior a la del *iDistance* en cuanto al número de accesos a disco. Por ello, en un principio se planteó la posibilidad de diseñar un procesamiento por lotes para el *iDistance*, pero lo cierto es que a la vista de los resultados obtenidos en la comparación del *VA-File* con BNN y el *iDistance* sin BNN el tiempo de respuesta del procesamiento por lotes del *VA-File* es mucho mayor que el del *iDistance* y sin embargo, aunque el *VA-File* con BNN mejora en el número de accesos a disco al *iDistance*, esta mejora no es significativa o por lo menos no tanto como la diferencia de tiempos de respuesta.

Es entonces necesario preguntar si ¿realmente se obtendría una mejora sustancial y en dicho caso, si merecería la pena implementar un procesamiento por lotes para el *iDistance* teniendo en cuenta los resultados obtenidos por el *VA-File*?.

En un posible trabajo futuro, en el que se decidiese optar por investigar dicho procesamiento para el *iDistance*, debería intentarse reducir en gran medida el tiempo de respuesta para que dicho procesamiento sea productivo, es decir, que mejore las prestaciones del *iDistance* sin el procesamiento por lotes.

7.2.1.4 Multidimensional iDistance (MiD)

Como ya se detalló en el apartado de conclusiones del capítulo 3, el *iDistance* tiene una pérdida de precisión en la indexación debido a la naturaleza del algoritmo: reducción de la dimensionalidad. Debido a este efecto, puede haber puntos alejados en el espacio de datos E^{dim}, que presenten el mismo valor de indexación basado en la distancia a un mismo punto de referencia (de la misma partición *i*). Es por tanto que en este apartado se intenta proponer como trabajo futuro, la mejora estudiada en [Del07] como solución a este problema.

El *iDistance* es un excelente método de indexación de datos multidimensionales, pero tiene un defecto a la hora de reducir la dimensionalidad de los datos en la indexación. Si consideramos dos puntos *a* y *b* del espacio de datos y la partición *i* (P_i) con su centro de referencia O_i. Podemos observar, en las siguientes Figuras (30 y 31), que pueden darse las siguientes situaciones [Del07]:

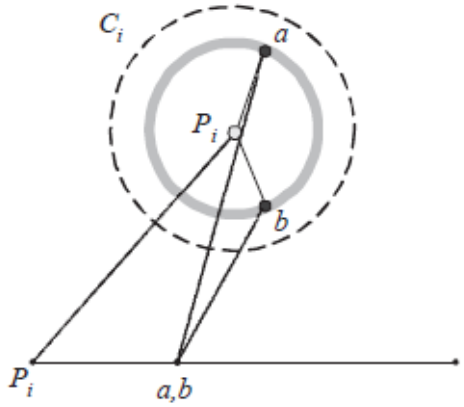


Figura 30. Pérdida de precisión A

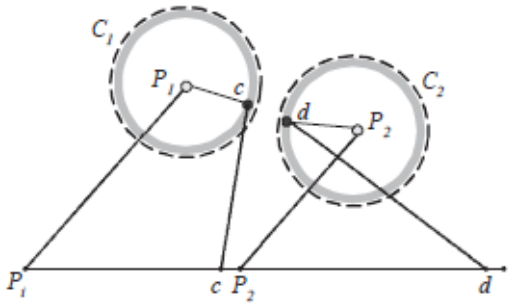


Figura 31. Pérdida de precisión B

En [Del07] se exponen dos tipos de pérdidas de precisión:

- ***Perdida de precisión A:*** *Iguales radios en la mima partición* (Figura 30): obtenemos el mismo valor de indexación para dos puntos que no son iguales en la misma partición, esto nos puede llevar a falsos positivos en la búsqueda KNN.

- ***Perdida de precisión B:*** *Dados dos puntos, incoherencia entre la distancia en el espacio dim-dimensional y la distancia en la indexación unidimensional* (Figura 31): no tenemos coincidencias erróneas, pero en cambio, dos puntos, c y d, que están relativamente cerca en el espacio de datos, el *iDistance* los indexa a una distancia muy superior.

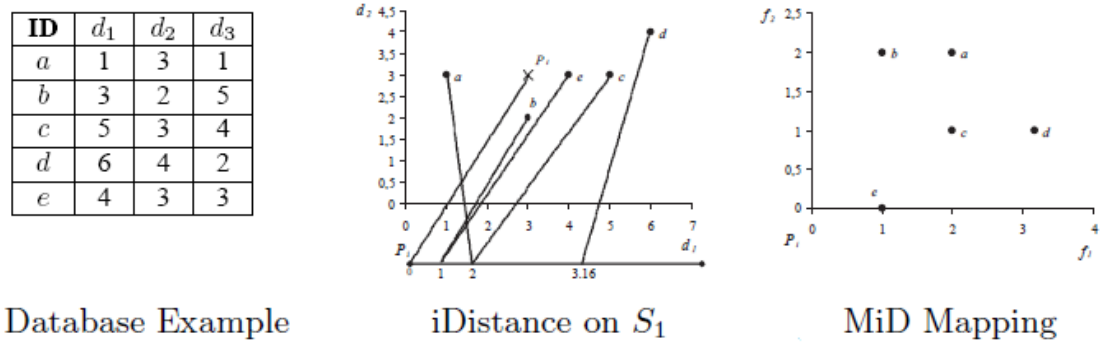
En ambas figuras (30 y 31) se representa una pérdida de precisión a la hora de indexar puntos con idéntico radio hacia su punto de referencia, es decir, $dist(a,O_i)=dist(b,O_i)$ y además $a \neq b$ (Tipo A). O bien, a la hora de indexar puntos relativamente cercanos en el espacio de datos, que quedan extremadamente separados en el árbol B+ (Tipo B).

La mayoría de errores de indexación del *iDistance* son de tipo A y por tanto son los que más afectan al rendimiento del mismo, puesto que si en alguna partición de los datos se da dicho error, hay que examinar todos los puntos candidatos de la región.

El proceso *MiD*, es en esencia, similar al *iDistance*. La diferencia fundamental radica en que en vez de tener un valor de distancia unidimensional para todas las particiones, ahora se consideran múltiples distancias para cada clúster de datos al cual se le creará una clave dimensional, por lo tanto habrá una clave dimensional por cada clúster P_i y cada P_i tendrá su propia distancia interna.

Al igual que el *iDistance*, cada clúster tendrá su punto de referencia. Pero antes del cálculo de la distancia entre un punto y su punto de referencia, dividimos las dimensiones en *m subespacios* (*m* corresponde con la dimensión deseada del espacio transformado al indexar). El segundo paso consiste en calcular las *m* distancias parciales a los puntos de referencia de todos los puntos de cada partición. Como resultado, obtenemos que las *m* distancias corresponden con las *m* claves dimensionales del conjunto de datos. Dichas claves se

nombran como las *iValues* [Del05] del conjunto de datos. En la Figura 32 podemos observar la diferencia entre la indexación del *iDistance* y la mejora del MiD:



ID	d_1	d_2	d_3
a	1	3	1
b	3	2	5
c	5	3	4
d	6	4	2
e	4	3	3

Figura 32. [Del07] Comparativa en la indexación entre el *iDistance* y el *Multidimensional iDistance* (MiD)

Como se puede observar, en la figura anterior, el *iDistance* indexa los puntos *a* y *c* con el mismo valor '*iDistance'* mientras que en el espacio de datos están en posiciones distintas. Al igual pasa con los puntos *b* y *e*. Si nos fijamos en el *MiD*, este recurre una indexación basada en una distancia para cada partición, lo cual consigue solucionar la pérdida de precisión del *iDistance*, indexando de manera *correcta* los distintos puntos *(a, e, c y b).*

7.2.2 Trabajo sobre el CBVR

En este último apartado, vamos a centrarnos en las posibles mejoras sobre el framework de recuperación de video. Entre todas las posibilidades del CBVR, vamos a centrar las mejoras en la identificación de subsecuencias de video (*VSI*) y la búsqueda de video similar (*VSS*).

7.2.2.1 Mejora en el rendimiento del CBVR

Como se ha visto en el capítulo 5, tanto en los algoritmos de la identificación de subsecuencias como en la búsqueda de video similar hay varios procesos de computo independientes susceptibles de paralelización. Dichos procesos pueden tener la suficiente carga de computo como para conseguir mejorar el rendimiento del CBVR utilizando más hilos repartidos entre el numero disponible de cores en los algoritmos expuestos.

Como posible interfaz de multiprocesamiento podría usarse la librería MPI (Message Passing Interface) para dicha paralelización del proceso de recuperación de video.

7.2.2.2 Normalización de distancias

Tanto en el capítulo 2 como en el capítulo 5 se proponen distintas distancias con las que medir la *similitud* entre frames. Pero exceptuando la distancia máxima, todas las demás no están normalizadas dentro del espacio de datos E^{dim}, que en nuestro caso es un hipercubo $[0, 1]^{dim}$. Para poder usar las distancias propuestas es necesario normalizarlas en este intervalo de valores y así poder usarlas por ejemplo para estudiar el comportamiento en la búsqueda de video similar (*VSS*) y en la distancia usada en la identificación de subsecuencias (*VSI*) para calcular los pesos del grafo bipartito.

Es de esperar que con distintas distancias normalizadas, el comportamiento de las dos consultas propuestas en este CBVR, varíen su comportamiento. En este apartado se propone como trabajo futuro la normalización de dichas distancias propuestas en este proyecto fin de carrera y posteriormente el estudio del comportamiento del framework.

7.2.2.3 Mejora en la función de similitud de video

Una vez obtenidos los segmentos candidatos, $\overrightarrow{\mathbb{S}^{*\prime}}$, se hace necesario refinar por SMSM tal y como se detalla en el capítulo 5. Dicho refinamiento exige una función de similitud entre subsecuencias de video, en este proyecto se ha utilizado una sola función de similitud, la cual ya proporciona buenos resultados a la vista de los experimentos del capítulo 6 y los realizados en [Riv10]. Pero a la vista de los resultados obtenidos, podemos observar que para consultas del tipo *Group Reorder y Modify* y un valor incorrecto de *kSS* pueden provocar un bajo acierto *(Hit Ratio)* en la identificación de subsecuencias de video. Se propone por tanto como trabajo futuro utilizar otra función de similitud descrita en [ZMP+09]: *Hybrid Similarity Model.*

En [ZMP+09] se expone la importancia de tener en cuenta tanto la similitud temporal (Sim_{TS}) como no temporal (Sim_{NT}) en los segmentos candidatos $\overrightarrow{\mathbb{S}^{*\prime}}$ obtenidos del filtrado en MSM. Para ello, en el matching resultante, la similitud no temporal viene definida por la distancia utilizada entre cada mapping o asignación. Por tanto, la ***Non-temporal similarity*** (Sim_{NT}) queda definida como se vio en el capítulo 5 como:

$$Sim_{NT} = \sum (1 - dist(q_i, s_i))/(\sqrt{d}\,|Q|)$$

Por otro lado, para calcular la Sim_{TS}, es necesario apoyarse en otras dos funciones de similitud. Es por ello, que se define la ***Temporal Order Similarity*** como:

$$Sim_{TO} = \frac{\max\left(LCSS(M), LCSS\left(\overleftarrow{M}\right)\right)}{|Q|}$$

donde, M y $\overleftarrow{M}$ denotan el matching utilizado en orden normal e inverso respectivamente. Otra función de similitud usada es la ***Temporal alignment similarity*** definida como:

$$Sim_{TA} = \frac{2\ |M|}{(|Q| + |S|)}$$

Por último, es necesario calcular la ***Temporal concentration similarity*** la cual viene definida en [ZMP+09]. Para exponer esta última similitud, vamos a denotar al grado de concentración en similitud como CD (*Concentration similarity Degree*) y el grado de similitud discreto como DD (*Discrete similarity Degree*). Además CMS y CUS denotan *Continuous Matched Subsequence* y *Continuous Unmatched Subsequence* respectivamente:

$$CD = Dur(CMS_1) + \frac{\sum_{i=2} Dur(CMS_i)}{\sum_{t=1}^{i} Dur(CMS_t)}$$

$$DD = \frac{\sum_i \gamma_i \, Dur(CUS_i)}{\sum Dur(CUS)}$$

Donde *Dur*(*CMS*) y *Dur*(*CUS*) denotan la duración de cada subsecuencia continua CMS y CUS respectivamente. γ representa la distancia interna temporal con respecto a cada lado adyacente de cada subsecuencias continuas asociadas (*matched*) y sin asociar (*unmatched*). Por tanto, si $CUS_i = [s_t,\ldots, s_p]$, γ queda definida de la siguiente manera:

$$\gamma = \begin{cases} \sum_{i=t}^{p} \frac{dist(s_{p+1}, s_i)}{|CUS_i|} & si\ t = 1 \\ \sum_{i=t}^{p} \frac{dist(s_{t-1}, s_i)}{|CUS_i|} & si\ p = lenght(S) \\ \sum_{i=t}^{p} \frac{dist(s_{p+1}, s_i) + dist(s_{p+1}, s_i)}{2|CUS_i|} & en\ otro\ caso \end{cases}$$

Una vez expuesto todas las funciones de similitud anteriores podemos definir la similitud temporal como:

$$Sim_{TC} = (CD - DD)/|Q|$$

A continuación, se propone como trabajo futuro la implementación del enfoque hibrido de similitud, el cual combina todas las funciones de similitud mencionadas anteriormente:

Hybrid similarity model

Paso 1*: Reordenar M y* $\overleftarrow{M}$ *mediante el valor de:* Sim_{TO}, Sim_{TA}, *y* Sim_{TC}*:*
Primero, reordenar mediante Sim_{TO}*, para matchings con el mismo* Sim_{TO} *reordenar estos mediante* Sim_{TA} *y por ultimo, hacer lo mismo para* Sim_{TC} *si existen matchings con igual* Sim_{TA}*.*

Paso 2*: Incluir la similitud temporal:* Sim_{TS} *=Adjust (*Sim_{TO}*).*
Para cada $\mu = j\text{-}i+1$ *matchings* $[M'_i,\ldots,M'_j]$ *con la misma* Sim_{TO}*, calcular sus similitudes en* $[Sim_{TO}+\sigma/2, Sim_{TO}-\sigma/2]$ *donde* σ *es el parámetro de ajuste, definido como sigue:*

$$\sigma = \begin{cases} (Sim_{TO_{i-1}} - Sim_{TO_{j-1}})\ /3\mu & si\ i \neq 1\ y\ j \neq x \\ \dfrac{Sim_{TO_{j-1}}}{2\mu} & si\ i = 1 \\ \dfrac{Sim_{TO_{i-1}}}{2\mu} & si\ j = x \end{cases}$$

***Paso 3**: reordenar el resultado del paso 1 mediante la similitud hibrida (Sim) que viene dada por la media entre la similitud no temporal y la similitud temporal calculada:*

$$Sim = \frac{(Sim_{NT} + Sim_{TS})}{2}$$

Código 19. [ZMP+09] Modelo de similitud hibrida, temporal y no temporal.

Una vez definida la similitud hibrida *Sim* se pueden realizar todos los estudios hechos tanto en este proyecto como en [Riv10] para estudiar el comportamiento de dicha función de similitud en la identificación de subsecuencias de video.

8

Referencias

[AKS98] D. Agrawal, K.V.R. Kanth, A. Singh. *Dimensionality reduction for similarity searching in dynamic databases*. ACM SIGMOD Conference, páginas 166–176, 1998.

[AMN+98] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, A.Y. Wu. *An optimal algorithm for approximate nearest neighbor searching in fixed dimensions*. Journal of ACM, 45(6), páginas 891–923, 1998.

[ASR+98] G. Amato, P. Savino, F. Rabitti, P. Zezula. *Approximate similarity retrieval with M-trees*. VLDB Journal 7(4), páginas 294–307, 1998.

[BaC72] Organization and Maintenance of Large Ordered Indexes *by R. Bayer and E. McCreight*, Vol. 1, Fasc. 3, 1972, páginas: 173-189.

[BBJ+00] S. Berchtold, C. Böhm, H. Jagadish, H.P. Kriegel, J. Sander. *Independent Quantization: an Index Compression Technique for High-Dimensional Data Spaces*, ICDE Conference, páginas 577–588, 2000.

[BBK+01] C. Bohm, B. Braunm ller, F. Krebs, H.P. Kriegel. *Epsilon grid order: An algorithm for the similarity join on massive high-dimensional data*. ACM SIGMOD Conference, páginas 379–388, 2001.

[Ben75] J.L. Bentley, 'Multidimensional Binary Search Trees Used for Associative Searching,' Comm. ACM, vol. 18, páginas. 509-517, 1975.

[CCP+02] G.H. Cha,C.W. Chung, D. Petkovic, X. Zhu. *An Efficient Indexing Method for Nearest Neighbor Searches in High-Dimensional Image Databases*. IEEE Transactions On Multimedia, 4(1), páginas 76–87, 2002.

[ChC02] G.H. Cha, C.W. Chung. *The GC-Tree: A High-Dimensional Index Structure for Similarity Search in Image Databases*. IEEE Transactions on Multimedia, 4(2), páginas 235–247, 2002.

[ChZ03] S. Cheung, A. Zakhor. *Efficient Video Similarity Measurement with Video Signature*. IEEE Transactions on Circuits and Systems for Video Technology, 5(3), páginas 59-74, 2003.

[Cui02] Cui Yu. *High-Dimensional Indexing: Transformational Approaches to High-Dimensional Range and Similarity Searches.* Monmouth University, Department of Computer Science,West Long Branch, NJ 07764, USA. National University of Singapore: Department of Computer Science Kent Ridge, Singapore 117543, Singapore. Páginas 85-116. 2002

[Del07] Evangelos Dellis aus Trikala. *Advanced Indexing and Query Processing for Multidimensional Databases*, Vom Fachbereich Mathematik und Informatik der Philipps-Universität Marburg als Dissertation am 3. September 2007 angenommen. Páginas 89 – 107. 2007

[EJH+97] A.K. Elmagarmi, H. Jiang, A.A. Helal, A. Joshi, M. Ahmed. *Video Database Systems. Issues, Products, and Applications*. Kluwer Academic Publishers, 1997.

[FTA+00] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, A.E. Abbadi. *Vector Approximation Based Indexing for Non-Uniform High Dimensional Data Sets*, CIKM Conference, páginas 202–209, 2000.

[GRG00] J. E. Gehrke, Raghu Ramakrishnan, and Venkatesh Ganti. RAINFOREST - *A Framework for Fast Decision Tree Construction of Large Datasets*. In *Data Mining and Knowledge Discovery*, Volume 4, Issue 2/3, July 2000, páginas 127-162. 2000.

[GRa00] Goldstein, J. and Ramakrishnan, R. *Contrast plots and p-sphere trees: Space vs. time in nearest neighbour searches.* In Proceedings of the International Conference on Very Large Data Bases. Cairo, Egypt, páginas 429-440, 2000.

[GrY04] J.L. Gross, J. Yellen. *Handbook of Graph Theory*. CRC Press, 2004.

[JOT+05] H.V. Jagadish, B.C. Ooi, K.L. Tan, C. Yu, R. Zhang. *iDistance: An Adaptive B^+-tree Based Indexing Method for Nearest Neighbor Search*. ACM Transactions on Database Systems, 30(2), páginas 364-397, 2005.

[KMN+02] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, A.Y. Wu. *An Efficient k-Means Clustering Algorithm: Analysis and Implementation*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(7), páginas 881-892, 2002.

[KMNS_IMPL] *Implementación de kmeans Lloyd´s Algotihm:* David M. Monte, Departamento de Ciencias de la Computación, Instituto de Estudios de Informática Avanzada Universidad de Maryland. College Park, Maryland 20742, mount@cs.umd.edu. Web: http://www.cs.umd.edu/~mount/Projects/KMeans/

[KCB03] Manesh Kokare, B.N. Chatterji and P.K. Biswas. *Comparison of Similarity Metrics for Texture Image Retrieval.* Electronics and Electrical Communication Engineering Department, Indian Institute of Technology, páginas 572-574, 2002

[LJF94] King-lp Lin, H.V. Jagadish, and Christos Faloutsos. *The TV-Tree: An Index Structure for High-Dimensional Data.* VLDB Journal,3, páginas 517-542 (1994).

[MeS97] N. Megiddo, U. Shaft. *Efficient nearest neighbor indexing based on a collection of space-filling curves*. IBM Almaden Research Center, San Jose, CA, Tech. Rep. RJ 10093, 1997.

[Riv10] R. Rivas. *Procesamiento de Consultas para la Recuperación de Video basada en Contenido*. PFC de la EPS de la Universidad de Almería, Julio 2010.

[Sam06] H. Samet. Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers. 2006.

[Sed02] R. Sedgewick. *Algorithms in C++: Graph Algorithms*. Addison-Wesley, 2002.

[SHS+08] J. Shao, Z. Huang, H.T. Shen, X. Zhou, E.P. Lim, Y. Li. *Batch Nearest Neighbor Search for Video Retrieval*, IEEE Transaction on Multimedia, 10(3), páginas 409-420, 2008.

[Shi04] D.R. Shier. *Matchings and Assignments*. Handbook of Graph Theory, J.L. Gross and J. Yellen, editors, páginas 1103–1116 , CRC Press, 2004.

[SSH+09] H.T. Shen, J. Shao, Z. Huang, X. Zhou. *Effective and Efficient Query Processing for Video Subsequence Identification*. IEEE Transactions on Knowledge and Data Engineering, 21(3), páginas 321-334, 2009.

[SOZ+05] H. T. Shen, B. C. Ooi, X. Zhou, Z. Huang. *Towards Effective Indexing for Very Large Video Sequence Database*, ACM SIGMOD Conference, páginas 730-741, 2005.

[Tjo05] D. W. Tjondronegorost. *Content-based Video Indexing for Sports Application susing Integrated Multi-Modal Approach*. PhD thesis, University of Deakin, 2005.

[WSB98] R. Weber, H.J. Schek y S. Blott. *A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces*. Very Large Data Bases Conference, páginas 194-205. 1998.

[YDT+04] J. Yuan, L.Y. Duan, Q. Tian, S. Ranganath, C. Xu. *Fast and Robust Short Video Clip Search for Copy Detection*. PCM Conference (PCM'04), volumen 2, páginas 479–488, 2004.

[ZMP+09] Aihua Zheng, Jixin Ma, Miltos Petridis, Jin Tang, and Bin Luo. *A Robust Approach to Subsequence Matching*. Anhui University, Hefei, 230039, People's Republic of China and The University of Greenwich, Greenwich, London, SE10 9LS, United Kingdom. Páginas 44-46. 2009.