Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Av. Pellegrini 250, Rosario, República Argentina

Licenciatura en Ciencias de la Computación

Tesina de Grado

Selección dinámica de pivotes que se adaptan a las búsquedas en Espacios Métricos

Mariano Salvetti

salvettimariano@hotmail.com

Septiembre 2009

Directoras: Claudia Deco, Cristina Bender
{deco, bender}@fceia.unr.edu.ar

Dedicado a mis padres, Carlos y Cristina,
quienes me enseñaron que
El Secreto del éxito es
la constancia en el propósito.

Resumen

Una base de datos métrica es una colección de objetos (de cualquier tipo) con una similitud y una manera formal de calcular esta similitud como una métrica. Las consultas por similitud extienden la búsqueda por exactitud en el sentido de que el resultado obtenido es un conjunto de elementos cercanos al objeto de búsqueda, utilizando estructuras índice para acelerar los resultados.

Basándonos en el método de indexación Sparse Spatial Selection (SSS), en este trabajo se presenta una estructura de indexación y búsqueda por similitud que periódicamente intenta adaptar los pivotes del índice al uso particular de la Base de Datos y no relegarlo a un estado inicial estático. El objetivo de esta idea es poder mejorar la cantidad de discriminaciones hechas por los pivotes a modo de lograr el primordial objetivo de los índices: bajar la cantidad de evaluaciones de la función distancia. Aquí combinamos las buenas propiedades de trabajos previos, como SSS, con algunas ideas fomentadas recientemente para la selección dinámica de pivotes.

Índice

Introducción

La llegada de la era digital genera un creciente interés en buscar información en grandes repositorios desestructurados de información contenedores de datos textuales, multimedia, fotografías, objetos en 3 dimensiones y cadenas de ADN, entre otros. La búsqueda en estos repositorios requiere de lenguajes de consulta más poderosos que los actuales para poder lograr su objetivo. No sólo se consultan nuevos tipos de datos, sino que además, en algunos casos, ya no se puede estructurar más la información en tablas con claves y registros.

Las bases de datos relacionales actuales están construidas sobre el concepto de búsqueda exacta: dividimos en registros, y cada registro tiene un identificador único comparable. Las consultas realizadas retornan todos los registros en los cuales la clave coincide exactamente con la clave que buscamos. Sin embargo, estas consultas pierden sentido desde el momento en que trabajamos con objetos difícilmente comparables por igualdad. Las bases de datos actuales y estos escenarios no permiten realizar consultas por similitud, por lo cual, es necesario enriquecer el lenguaje de consulta para abarcar estos casos.
El modelo de espacio métrico brinda una base teórica para definir de una forma eficaz la búsqueda de estos nuevos tipos de datos.

Las aplicaciones de las bases de datos métricas difieren de las bases de datos tradicionales, como las relacionales, en el sentido de que en estas últimas los datos se buscan por igualdad exacta con un valor de clave dado. Esta forma de búsqueda resulta inútil en varios campos de la computación, en donde las bases de datos métricas son más apropiadas, como por ejemplo, consultar sobre contenidos en objetos multimedia, recuperación de información, biología computacional, compresión de audio y video, renderizado de objetos en 3D, etc.

Al representar datos multimedia en formatos digitales, como lo hacen las computadoras, dos "vistas" de la misma realidad tienen poca probabilidad de ser representada de la misma forma, razón por la cual es más útil una búsqueda por similitud que por igualdad exacta. Dentro de este campo de aplicación se encuentran los problemas de reconocimiento de imágenes, reconocimiento de rostros y visión por computadora, entre otras áreas. La dificultad aquí radica en encontrar una forma de evaluar la similitud entre estos objetos. Para este fin son muy utilizadas algunas funciones que obtienen vectores de características (entre ellas encontramos la transformada de Fourier, la transformada de Wavelet y el Histograma de colores) para obtener una representación de algunas características del objeto y comparar sus atributos y emitir un juicio sobre la similitud de éstos.
La función que calcula la similitud entre estos objetos es provista por un experto del dominio, por lo cual habrá que comprobar que cumple con ciertas restricciones que veremos más adelante (propiedades métricas).
Luego, una vez representados estos objetos y obtenida una función que calcule la similitud entre ellos, nos interesa poder recuperarlos o realizar consultas a estos objetos almacenados dentro de una base de datos "no tradicional".
Para acelerar esta tarea de recuperación de información, existen estructuras llamadas índices que son una estructura de datos que mejoran la velocidad de las operaciones, permitiendo un rápido acceso a los objetos.
El índice tiene un funcionamiento similar al índice de un libro, guardando pares de elementos: el elemento que se desea indexar y su posición en la base de datos. Entonces, para buscar un elemento que esté indexado, sólo hay que buscar en el índice dicho elemento para, una vez encontrado, devolver el registro que se encuentre en la posición marcada por el índice.
En la actualidad, y como veremos más adelante, existen varias estructuras de índice que se han propuesto para espacios métricos y vectoriales, las cuales se diferencian entre sí, por la forma en que se los puede recorrer y por la forma en que almacenan la información o complejidad, entre otras diferencias que más adelante enumeraremos.

Si de índices hablamos, tenemos que los métodos de acceso métrico [Chávez, 2001a] son estructuras índices que utilizan propiedades métricas de la función distancia (sobre todo la desigualdad triangular) para filtrar las zonas del espacio. Hay dos clases principales de métodos de acceso métricos: índices basados en selección de pivotes e índices basados en particiones compactas (clustering). Ambas clases de índices tienen como objetivo dividir el espacio en clases de equivalencia, y luego el índice se usa para ir filtrando algunas clases en tiempo de búsqueda. Las clases de equivalencia que no son filtradas ni

desechadas deben ser comprobadas exhaustivamente ya que son consideradas como objetos relevantes para responder a la consulta.

Los métodos de acceso espacial [Bohm, 2001] son estructuras de índice sobre todo diseñadas para espacios vectoriales. Junto con las propiedades métricas, los métodos de acceso espacial usan la información geométrica para desechar puntos del espacio. Por lo general, estos índices son estructuras de datos jerárquicas que usan una estructura equilibrada para indexar la base de datos.

Casi todas estas estructuras de índice (métodos de acceso métricos y espaciales) han sido diseñadas para indexar sólo vectores de propiedades, y no soportan el uso de esos conjuntos de propiedades en los vectores.

En esta tesina se presenta una nueva estructura de indexación y búsqueda por similitud basada en el Sparse Spatial Selection (SSS) [Pedreira, 2007] junto con dos nuevas políticas de selección de pivotes. Este método propuesto basado en la selección dinámica de pivotes, tiene como característica que utiliza al SSS para la selección inicial de pivotes bien distribuidos en el espacio métrico, lo que da lugar a una mejora en el rendimiento de la búsqueda con respecto a una selección inicial aleatoria de los pivotes.
Además, tiene otras interesantes características:

- Es dinámico, ya que permite que la base de datos esté inicialmente vacía y crezca en el tiempo.
- Es adaptativo, ya que no es necesario establecer de antemano el número de pivotes a utilizar y el propio algoritmo los selecciona según son necesarios para adaptarse a la complejidad del espacio.
- Mejora los resultados de búsqueda, ya que las políticas de selección propuestas adapta el índice según los resultados de búsqueda anteriores, utilizando esta información 'histórica' para mejorar futuras búsquedas.

Luego, con el pasar del tiempo y de las búsquedas, se aplican unas nuevas políticas de selección de pivotes. La primera es para eliminar del índice a aquellos pivotes que discriminen pocos elementos, y la segunda política es para la selección de pivotes candidatos a formar parte del índice, y así lograr adaptar dinámicamente el índice a las búsquedas realizadas durante un determinado tiempo.

Nuestra hipótesis de trabajo es que, al aplicar SSS para seleccionar los pivotes iniciales del índice, la partición del espacio será más eficiente y dará lugar a un resultado mejor en las búsquedas.
Entonces, acumulando la información de dichas búsquedas y al poder adaptar el índice según los resultados de búsqueda anteriores, se logra el objetivo de mejorar la eficiencia del índice y acelerar el procesamiento de los resultados de búsqueda.

Organización de la Tesina

Este documento está organizado en tres partes: conceptos básicos, la propuesta de esta tesina con la experimentación y sus conclusiones, y por último, en los apéndices, los espacios métricos utilizados. En la primera parte se comienza con una introducción y presentación de los conceptos básicos utilizados a lo largo de esta tesina, explicando los problemas, los objetivos y alcances. Además, se hace una revisión del estado del arte de la búsqueda en espacios métricos.
En la segunda parte se presenta la propuesta de esta tesina, los algoritmos derivados con sus pruebas experimentales y el análisis de dichos resultados.
Por último, en la tercera parte se presentan los dos espacios métricos experimentales en los que fueron usados los algoritmos propuestos en esta tesina, junto con la experimentación y cuyos resultados muestran una importante contribución en esos problemas.

Capítulo 2: Marco Teórico

En este capítulo se explican algunos conceptos básicos utilizados a lo largo de la tesina junto con teorías que respaldan la experimentación y por último se describen los tipos de espacios que fueron usados en los experimentos posteriores para mostrar el desempeño de la propuesta presentada.
Una base de datos métrica es una colección de objetos (de cualquier tipo) con una similitud y una manera formal de calcular esta similitud como una métrica. Las consultas por similitud extienden la búsqueda por exactitud en el sentido de que el resultado obtenido es un conjunto de elementos cercanos al objeto de búsqueda. Este objeto de referencia que estamos buscando puede no pertenecer a la base de datos. Este concepto puede ser formalizado usando el modelo de espacio métrico. Éste consta de un espacio (conjunto de datos) y una medida de semejanza entre los elementos de ese espacio. No existe una definición general de esta medida dado que está estrechamente ligada a la aplicación y a las características que se quieran evaluar.

2.1. Espacios Métricos

Un *espacio métrico* (*X, d*) está formado por un universo de objetos válidos *X* y una *función distancia* $d:X\times X\rightarrow\Re^+$ entre sus elementos. Un subconjunto finito $U\subset X$, con $|U|=n$, es el conjunto de elementos en el que se realizan las búsquedas. La función $d:X\times X\rightarrow\Re^+$ medirá la distancia entre elementos y cuanto más pequeño es el valor de *d*, más parecidos entre sí son los elementos. Las funciones distancias deben cumplir estas propiedades:

Positividad estricta: $d(x, y) > 0$ si *x distinto de y*

Simetría: $d(x, y) = d(y, x)$

Reflexividad: $d(x, x) = 0$

Estas propiedades enumeradas sólo aseguran una definición consistente de la función distancia y no pueden ser usadas para evitar comparaciones en una búsqueda por proximidad. La *Desigualdad Triangular* es una propiedad que permite descartar elementos sin ser comparados directamente contra la consulta: $d(x, y) \leq d(x, z) + d(z, y)$.

En algunos casos tenemos espacio cuasi-métricos, donde no se conserva la propiedad de simetría. Por ejemplo, si los elementos son esquinas de una ciudad y la distancia corresponde a cuánto debe trasladarse un auto de una esquina a la otra, entonces la existencia de calles de un solo sentido hace la distancia asimétrica. Para estos casos existen técnicas para derivar una nueva distancia simétrica:

$$d_0(x, y) = d(x, y) + d(y, x).$$

Sin embargo, para poder acotar el radio de búsqueda a esta nueva función distancia es necesario tener conocimiento específico del dominio de aplicación.

2.1.1. Espacios Vectoriales

Un espacio vectorial R_d es un tipo especial de espacio métrico. El espacio R_d está compuesto por d-tuplas de números reales, que se llaman vectores. Esto significa que, si *x* pertenece a R_d entonces $x = (x_1, x_2 ..., x_d)$, $x_i \in R\ 1 \leq i \leq d$.
Existen muchas métricas para los espacios vectoriales. Una utilizada ampliamente es la familia de las distancias Minkowski (Dp), que se define como:

$$D_p(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{M}(x_i - y_i)^p\right)^{1/p}$$

Algunos ejemplos de las métricas de Minkowski son:

Distancia Manhattan: *p=1*, $D_1 = \sum_{i=1}^{M} | x_i - y_i |$

Distancia Euclidiana: *p=2*, $D_2 = \sqrt{(\sum_{i=1}^{M} (x_i - y_i)^2)}$

Distancia Chebychev: $p=\infty$, $D_\infty = \max_{1 \le i \le M} | x_i - y_i |$

Para modelar datos multimedia como un espacio vectorial, aquí se debe utilizar una función de transformación, que es sumamente dependiente del tipo de datos de multimedia. Esta función extrae rasgos importantes de los objetos multimedia y traza un mapa de estos rasgos en el espacio dimensional de vectores de d-dimensiones (también se los llama en la literatura como descriptores). Entonces, la consulta original por similitud se reduce a buscar los puntos cercanos a ese vector de d-dimensiones.

Aquí nos referiremos a "vector de propiedades" (FV, de Feature Vector) como cualquier método de transformación de propiedades bien definidas.

Por lo general, la dimensionalidad *d* del FV es un parámetro de la función de transformación: usando los valores más altos de *d* uno puede obtener una mejor representación (más fina) del objeto de los multimedia. Sin embargo, en usos prácticos hay por lo general un punto de saturación, donde si agregamos más dimensiones sólo añadimos ruido a la descripción. También, tenemos que destacar, que para la mayoría de las aplicaciones, la transformación es irreversible, es decir, no podemos reconstruir el objeto multimedia original a partir de esta descripción con el FV.

2.1.2. Tipos de Consultas

En las bases de datos métricas, las consultas difieren de la manera habitual en que se realizan en las bases de datos relacionales comunes. En las bases de datos métricas se pueden realizar 3 clases o tipos de consultas, pero siempre por similitud, graficadas en la Figura 1:

- Consultas de rango
- Vecino más cercano
- K-Vecinos más cercanos

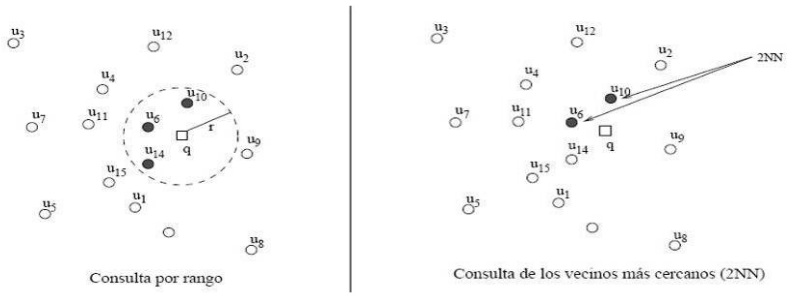


Figura 1: Ejemplo gráfico de los tipos de consultas

1. Consultas de rango

En esta clase de consultas, se buscan todos los elementos a una distancia menor a *r* dado un elemento *q* consulta, es decir todos los elementos incluidos en la bola con centro *q* y radio *r*. Expresado matemáticamente,

$$\{ u \in DB \mid d(u,q) \le r \}$$

donde DB es la base de datos.

Llevado a un lenguaje de consulta similar al SQL, la consulta se realizaría mediante la siguiente sentencia,

*SELECT * FROM DB WHERE distanciaCon(q) < r*

definiendo correctamente la función *distanciaCon()* para cada base de datos.

En la Figura 2 vemos un ejemplo de una consulta de rango con datos en $\Re_2$ tomando como distancia la distancia de Euclides.

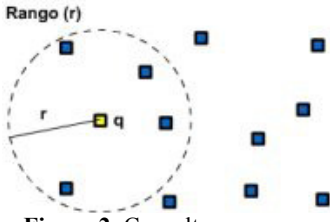


Figura 2: Consulta por rango

2. Vecino más cercano

Esta consulta busca el elemento más cercano en la base de datos a un elemento *q* dado. Tanto esta clase de consulta, como la siguiente (k-vecinos más cercanos) pueden ser reducidas a consultas de rangos, por lo cual muchos algoritmos se focalizan en resolver bien consultas de rango, sin prestar atención a este tipo de consultas. Se busca el elemento con distancia mínima a *q*. Expresado matemáticamente,

$$\{ u,v \in DB \mid d(q,u) \leq d(q,v) \}$$

En la Figura 3 vemos un ejemplo en $\Re_2$ de una consulta de este tipo. Si quisiéramos expresar la consulta en un lenguaje similar al SQL tendríamos la sentencia:

*SELECT * FROM DB ORDER BY distanciaCon(q) LIMIT 1*

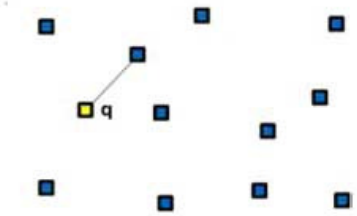


Figura 3: Vecino más cercano

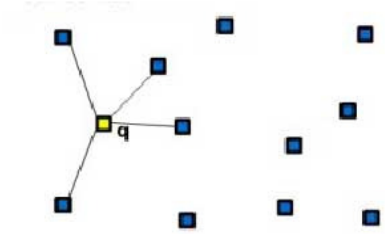


Figura 4: K-Vecinos más cercanos

3. K-Vecinos más cercanos

Esta consulta es un caso más general que la anterior, en la cual se buscan los *k* elementos más cercanos a un elemento *q* dado. Esta clase de consulta es muy utilizada en algoritmos de inteligencia artificial, machine learning y predicción de funciones. Como antes, utilizando notación matemática, se busca:

$$\{ u_i, v \in DB \mid d(q,u_i) \leq d(q,v) \ \ and \ \ \#u = K \}$$

Llevado al lenguaje de consulta sería similar al caso anterior, excepto que ahora buscamos k elementos,

SELECT * FROM DB ORDER BY *distanceto(q)* LIMIT k

2.1.3. Métodos de acceso métricos (MAMs)

Estos métodos sólo usan las propiedades métricas de la función distancia *d*, sobre todo la desigualdad triangular, filtrando objetos o regiones enteras del espacio durante la búsqueda, y así evitando la búsqueda lineal por todos los elementos.

En general, la función distancia *d* es muy cara de calcular, y en muchos casos prácticos es tan costosa que el tiempo de CPU o los gastos de tiempo de entrada-salida pueden ser muy altos.
Es por esto que, la complejidad de MAMs generalmente es medida como el número de veces que se calcula la función distancia.

2.1.4. Concepto de Índice Métrico

En espacios métricos los algoritmos que resuelven consultas por proximidad generalmente emplean índices para conocer la respuesta sin comparar toda la base de datos como lo haría una búsqueda secuencial. Un índice puede verse como una estructura de datos que facilita la búsqueda en un conjunto de elementos (base de datos).
Generalmente, un algoritmo que resuelve consultas por proximidad tiene dos fases: la de pre-procesamiento y la de consulta.
Inicialmente tenemos la base de datos, ésta se *proyecta* en un nuevo espacio que permite crear un índice (fase de pre-procesamiento).
El índice se crea una vez y se almacena, por lo tanto, el costo de su construcción no se considera al momento de resolver consultas.
En la fase de consulta se recorre el índice para conocer la *lista de candidatos* que deberán ser comparados directamente contra la consulta (comparaciones externas) y los elementos que pueden ser descartados de manera segura. En la lista de candidatos hay elementos que no son relevantes para la consulta, pero que no pudieron ser distinguidos por el índice.
El caso ideal es que todos los elementos en la lista de candidatos sean relevantes para la consulta.

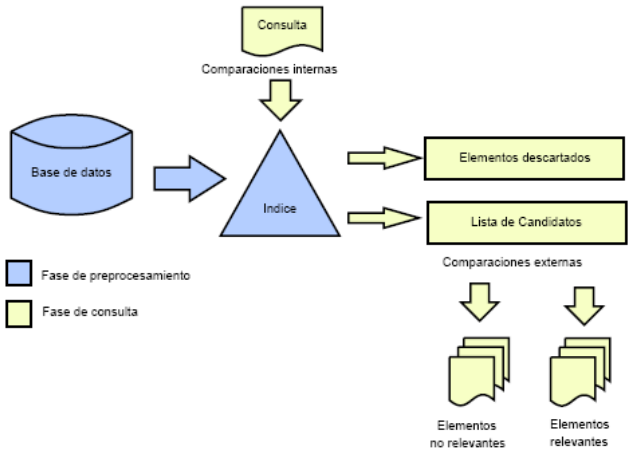


Figura 5: Tratamiento de la base de datos para resolver consultas usando un índice

Todas las estructuras de índice dividen los datos en subconjuntos, cada uno de ellos representa una clase de equivalencia, por ejemplo, un subconjunto de objetos relacionados de algún modo. Realizando una búsqueda por semejanza, el algoritmo de búsqueda elimina algunas clases filtrando.
Aquellas clases que no son eliminadas por el índice deben ser comparadas con el objeto consulta, ya que forman parte de la colección de objetos relevantes.

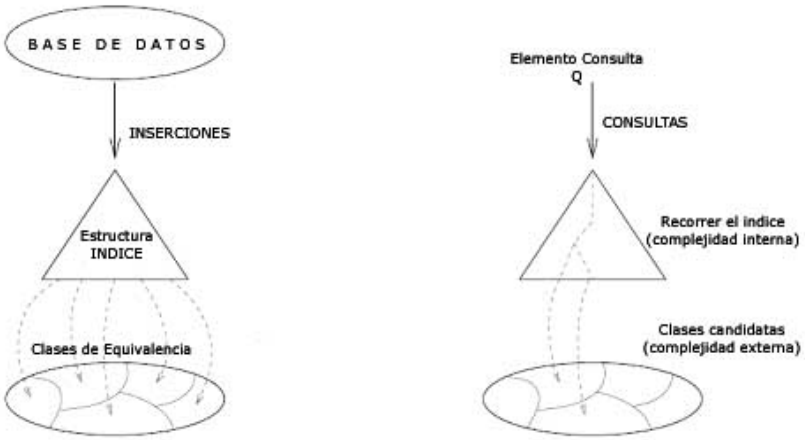


Figura 6: Estructura para Métodos de acceso métrico y su uso

El costo asociado para recorrer el índice de forma transversal es la *complejidad interna* de la búsqueda, y el costo de comparar las clases de equivalencia no filtradas contra el objeto consulta es la *complejidad externa* de la búsqueda.
Veamos a continuación los dos tipos principales de indexaciones que tenemos en este tipo de método:

a) Indexación basada en Pivotes

Son Algoritmos que seleccionan un conjunto de elementos llamados 'pivotes' y definen una relación de equivalencia basados en la distancia de estos con los demás elementos.

Sea $\{ p_1, p_2, \ldots, p_k \}$ el conjunto de pivotes. Dos elementos son equivalentes si y solo si están a la misma distancia de todos los pivotes:

$$x \sim \{p_i\} y \Leftrightarrow d(x, p_i) = d(y, p_i), \forall i = 1 \ldots k$$

Durante la búsqueda, utilizando la desigualdad triangular filtraremos los objetos de la base de datos sin medir su distancia a la consulta *q*. Dado *(q,r)*, calculamos el vector distancia a q a cada pivote de la colección, $Dist(q) = [d(q, p_1), d(q, p_2), \ldots, d(q, p_k)]$ y luego se descartan todos los elementos '*a*' tales que para algún pivote p_i se cumple

$| d(q, p_i) - d(a, p_i) | > r$,

es decir $\max_{1 \le i \le k} \{ | d(a, p_i) - d(q, p_i) | \} = L_\infty(Dist(a), Dist(q)) \le r$, esto se demuestra en el siguiente lema que garantiza que puede acotarse la distancia entre *q* y cualquier $a \in U$.

Lema 1 Dados tres objetos $q \in X$, $u \in U$, $p \in P$, sabemos que

$$| d(q,p) - d(p,u) | \le d(q,u) \le d(q,p) + d(p,u)$$

Demostración
El límite superior se obtiene directamente de la desigualdad triangular,

$$d(q,u) \le d(q,p) + d(p,u).$$

En el caso del límite inferior, de acuerdo a la desigualdad triangular, se tiene que:

$$d(p,u) \le d(p,q) + d(q,u)$$
$$d(p,q) \le d(p,u) + d(u,q)$$

Estas desigualdades implican

$$d(p,u) - d(p,q) \le d(q,u),$$
$$d(p,q) - d(p,u) \le d(u,q),$$

combinando estas desigualdades y usando la simetría, obtenemos que

$$| d(p,u) - d(p,q) | \le d(q,u).\text{--}$$

En la figura 7 se muestra un ejemplo del uso de esta técnica, donde *p* es un pivote. El anillo formado por los círculos centrados en *p*, a distancias $d(p,q)+r$ y $d(p,q)-r$, contiene a los elementos que no podrían ser descartados por *p,* estos son: $u_2, u_3, u_4, u_5, u_6, u_7$ .El resto de la base de datos sí es descartada, por ejemplo, en el caso de u_1, se cumple que $|d(p,q) - d(p,u_1)| \geq r$, lo mismo sucede con el resto.

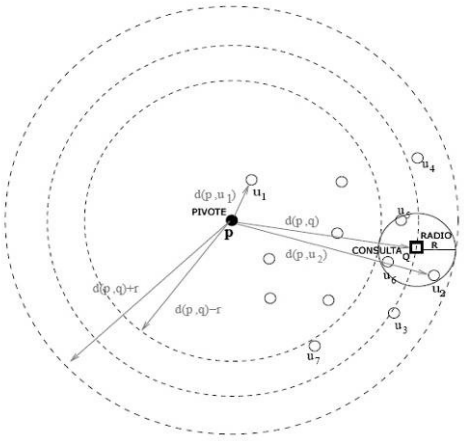


Figura 7: Algoritmo basado en pivotes para una consulta de rango (q, r)

Los elementos no descartados me permiten definir la lista de candidatos, que luego utilizaré para comparar directamente con la consulta *q*. Esto implica que la cantidad total de veces que evaluamos la función distancia queda determinada por la cantidad de pivotes *k* más la cantidad de elementos que obtenemos en la lista de candidatos. Como ya dijimos anteriormente, en estos métodos de acceso medimos la complejidad en dos partes: calcular la distancia entre el objeto consulta *q* y la lista de candidatos, es lo que llamamos '*complejidad externa'*. Mientras que calcular las distancias entre la consulta q y los k pivotes es lo que llamamos la '*complejidad interna*' del algoritmo.

b) Indexación basada en Particiones Compactas (Clustering)

Los *métodos basados en clustering* particionan el espacio métrico en un conjunto de regiones de equivalencia, cada una de ellas representada por un centro de cluster, tratando de generar la mayor cantidad de cluster posibles. Cada zona almacena un punto representativo, llamado el centro de cluster, y los datos que pueden ser usados para descartar el cluster entero al realizar una búsqueda, sin medir la distancia de los objetos del cluster al objeto consulta *q*.
Cuando realizo una búsqueda, se eliminan varias particiones enteras según la distancia de su centro de *cluster* a la consulta *q*.
Cada partición a su vez, puede ser dividida recursivamente en más zonas, formando una jerarquía de búsqueda.
Este tipo de algoritmos se dividen de acuerdo a su criterio de búsqueda y de división del espacio: Hiperplanos con particiones de Voronoi y radio de cobertura r_c .
Los hiperplanos con particiones de Voronoi de una colección de objetos es una partición del espacio en células. Cada célula contiene los objetos más cerca a un centro particular que a cualquier otro. Un conjunto de *M* centros es seleccionado y el resto de los objetos es asignado a la zona de su centro más cercano.
Considerando una consulta por radio (*q*, *r*), las distancias entre la consulta *q* y los *M* centros son calculadas. Sea *C* el centro más cercano a la *q*. Cada zona de centro c_i = c que satisface $(q, c_i) > (q, c) + 2r$ puede ser desechada, porque su hiperplano no puede cruzarse con la esfera de la consulta

En el caso del radio de cobertura r_c , el espacio se divide en varias esferas que pueden intersecarse, y una consulta puede pertenecer a mas de una esfera. El radio cobertor r_c es la distancia desde el centro del cluster hasta el objeto del cluster más alejado de él. Conociendo la distancia de la consulta *q* al centro del

cluster, el radio de búsqueda y el radio cobertor r_c , podemos decidir si ese cluster no tiene objetos en el resultado.

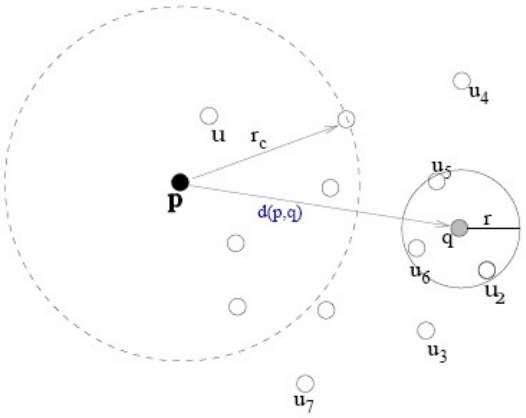


Figura 8: Algoritmo de clustering (particiones compactas), consulta de rango (*q, r*)

En la figura 8 se muestra un ejemplo del uso de esta técnica. El radio de cobertura del centro *p* está representado por r_c . En la figura toda la zona del centro *p* puede ser descartada puesto que todos los elementos en ella cumplen con $d(q,p)-r_c \le d(q,u)$ y $d(q,p)-r_c \ge r$.

2.1.5. Métodos de Acceso Espacial (MAE)

Los métodos de acceso Espacial (MEA o SAM por si sigla en inglés) son estructuras índice especialmente diseñadas para el modelado de índices de espacios vectoriales. Estas estructuras índice utilizan más información (por ejemplo, las propiedades geométricas de los puntos de datos), proporcionando así mejores estructuras para el manejo eficiente de consultas similitud en estos espacios.
Las estructuras de los MAE mantienen los datos jerárquicamente como clusters de páginas, generalmente por medio de un directorio balanceado.
La Figura 9 muestra la estructura básica de un MAE. El nodo raíz es un nodo directorio y se corresponde con el primer nivel jerárquico.
Todos menos el último nivel de la búsqueda está formado por directorios de páginas.
Cada nodo del índice incluye espacialmente todos los puntos de datos de todas las páginas de su subárbol.
El último nivel de la jerarquía corresponde a los datos de las páginas, que contienen los datos reales de puntos.

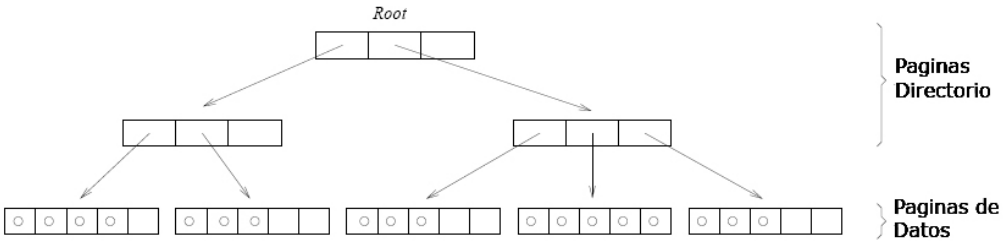


Figura 9: Estructura de Acceso Espacial

Como ejemplos de estos métodos de acceso espacial podemos enumerar:
R-Tree y sus variantes, *X-Tree* como una extensión al *R*-Tree* y *VA-File* entre los mas importantes. Para mayor detalle, ver [Bustos 2006] ya que a lo largo de este trabajo no nos enfocaremos en este tipo de método de acceso.

2.1.6. Dimensionalidad

Uno de los principales obstáculos en el diseño de buenas técnicas de indexación es lo que se conoce con el nombre de *maldición de la dimensionalidad*.
El concepto de dimensionalidad está relacionado a la dificultad o facilidad de buscar en un determinado espacio métrico. La dimensión intrínseca de un espacio métrico se define en [Chávez ,2001a] como

$$\rho = \frac{\mu^2}{2\sigma^2}$$

siendo μ y σ^2 la media y la varianza respectivamente de su histograma de distancias. Es decir que, a medida que la dimensionalidad intrínseca crece, la media crece y su varianza se reducen. Esto significa que el histograma de distancia se concentra más alrededor de su media, lo que influye negativamente en los algoritmos de indexación.

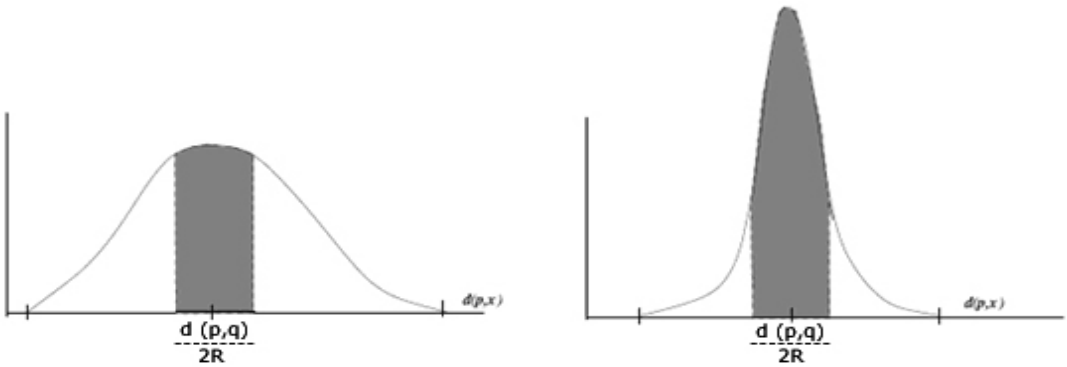


Figura 10: Problema de dimensionalidad y búsqueda

La Figura 10 muestra una idea intuitiva de por qué el problema de búsqueda se torna más difícil cuando el histograma es más concentrado.
Consideremos una búsqueda *(q, r)*, y un índice basado en pivotes elegidos de forma aleatoria. Los histogramas de la figura representan posibles distribuciones de distancias entre *q* y algún pivote p_i .
La regla de eliminación dice que podemos descartar aquellos puntos *y* tales que

$$y \geq [\, d(p_i, q) - R, d(p_i, q) + r \,]$$

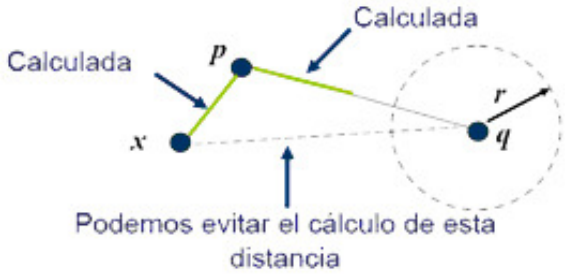


Figura 11: Desigualdad Triangular aplicada en la búsqueda

Para ilustrar el problema con la dimensión intrínseca alta, veamos la Figura 12, donde se muestran dos histogramas de distancias de dos bases de datos distintas, con respecto a un elemento distinguido $p \in X$.
El histograma del lado derecho (dimensión alta) está mucho más concentrado que el del lado izquierdo (dimensión baja).
Piense que una consulta de rango *q* con radio *r* se aplica a estos dos espacios; y que la distancia entre *p* y *q*, representada por *d* (*p*, *q*), se ubica al centro del histograma, que por cierto es su posición con mayor probabilidad.
En ambos espacios las zonas sombreadas corresponden a los elementos que podrían ser descartados por la desigualdad triangular, sin compararse contra la consulta usando un índice.

Notemos que el número de elementos que podrían ser descartados es mayor en dimensión baja que en dimensión alta. Son las zonas sombreadas las que corresponden a estos elementos, que podrían ser descartados sin compararse contra la consulta usando un índice y la desigualdad triangular.

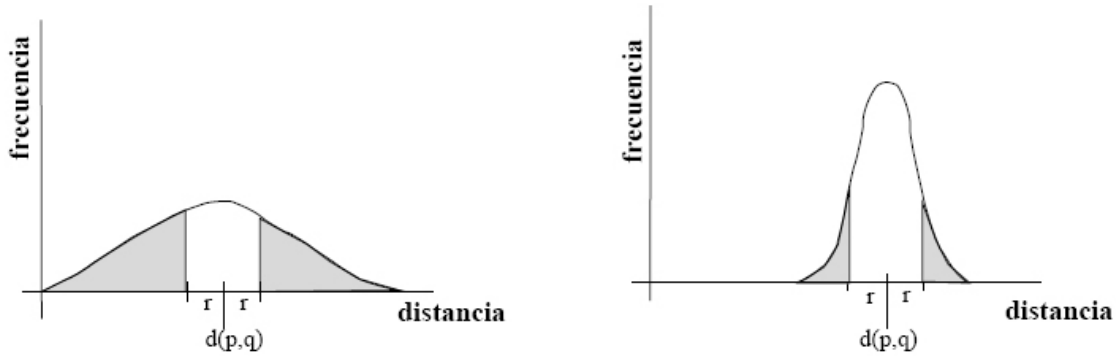


Figura 12: Histogramas de distancias de una consulta *(q, r)* hacia un elemento *p* en dimensión baja y alta (izquierda y derecha, respectivamente).

2.1.7. Función distancia

El problema central en las base de datos métricas es que generalmente la función distancia es muy costosa y compleja de calcular. Entonces, estas funciones son brindadas por expertos del dominio de aplicación y su funcionamiento interno es totalmente desconocido para el sistema de base de datos. Por esta razón una implementación razonable trataría de realizar la menor cantidad de evaluaciones de esta función, utilizando índices como veremos más adelante.

Ejemplos de funciones distancias en algunos dominios de aplicación:

Trivial: *d(x, y) = 1 si x = y , 0 en otro caso*. Esta función nos lleva al caso de las bases de datos relacionales en donde los atributos son comparados por igualdad.

Imágenes: para el caso de las imágenes se pueden definir varias funciones distancias, como la comparación de su vector TVS (Transformated Vector of Symetry), comparación de histogramas de colores, descomposición espectral por la transformada de Fourier. Todas estas funciones implican mucho cálculo numérico.

Reconocimiento de voz: aquí se pueden utilizar funciones basadas en el análisis espectral del espectro de voz, análisis de frecuencias promedio, etc.
Como en el caso anterior todas necesitan mucho calculo.

Similitud textual: En este ámbito hay infinidades de funciones distancias.
Entre ellas se encuentran:

- Mapeo a un vector de similitud: en el cual el texto es representado por un vector donde cada componente representa la cantidad de veces que aparece una palabra en el texto.
- Distancia de edición: Esta distancia se define como la cantidad de inserciones, eliminaciones y modificaciones que tengo que hacer de una palabra para llegar a otra.

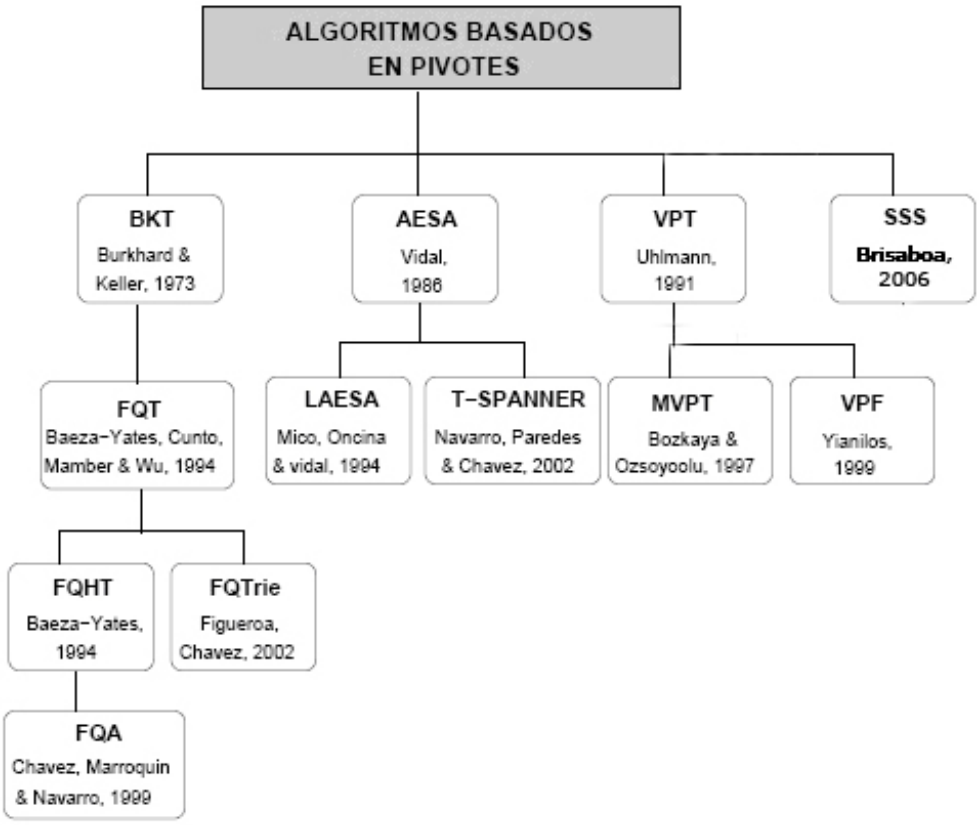
En todos los casos, tendremos que probar que la función distancia propuesta cumple con las condiciones necesarias para definir un espacio métrico. Como vimos antes si no cumplen algunas de las condiciones, la función puede ser modificada para estar en un espacio métrico.

2.2. Estado del arte

2.2.1 Algoritmos Basados en Pivotes

Sabemos que los pivotes seleccionados impactan en la eficiencia del método en la búsqueda dentro del espacio métrico, y la “ubicación" de cada uno de ellos con respecto a los demás pivotes determinan la capacidad del índice para descartar elementos sin compararlos directamente con la consulta.

La mayoría de los métodos de búsqueda basados en pivotes seleccionan los pivotes de forma aleatoria. Además, no se conoce ninguna forma de determinar el número óptimo de pivotes, un parámetro que depende de la colección concreta con la que estamos trabajando.



BKT: Burkhard-Keller Tree

Es una estructura de datos diseñada para guardar valores discretos de la función distancia. Cada nodo del árbol es un pivote *p*, y todos los elementos a distancia *i* de *p* son puestos en el i-ésimo sub-árbol de ese nodo correspondiente.
Y así se construye recursivamente hasta que tenemos b elementos (o menos) que son simplemente guardados en un 'bucket' dentro de una hoja del árbol.
A la hora de buscar por una consulta *q* y su radio *r*, recorremos el árbol en forma de 'back-tracking' solo en los sub-árboles desde $d(q, p) - r$ hasta $d(q, p) + r$, ya que el resto se descarta usando la desigualdad triangular.

FQT y FQHT

Esta estructura de datos se crea como una solución al "problema" de los BK − TREES y KD − TREES: eran lentos. Los FQ-TREES son similares a los recién mencionados, usando mucho la desigualdad triangular para minimizar el número de elementos a comparar. En esta estructura de datos, el costo de filtrar elementos depende de la distribución de las distancias y no del algoritmo (es decir, de la función distancia). Además, tiene orden logarítmico para búsquedas exactas y sub-lineal para las búsquedas de los k-vecinos más cercanos.
Un FQ-TREE difiere en dos cosas de los árboles comunes:

1. Las claves NO son números de un conjunto ordenado, sino que son elementos de *U*, donde $X \subset U$ (*X* está contenido en *U*, ya que *X* es mucho más grande que *U*)
2. Todos los nodos internos de un mismo nivel están asociados con la misma clave, es decir, en cada nivel del árbol, uso un pivote que no está en los nodos. Nos referimos como "claves" a los elementos que son comparados cuando recorremos el árbol, y hablamos de "elementos" cuando referimos a los miembros de *X*.

Además, un FQ-TREE de un conjunto de elementos *X* satisface las siguientes 4 propiedades:

- Todos los elementos de *X* (base de datos) se asocian con las hojas.
- Si tenemos solo *b* elementos en *X*, el FQ-TREE es una sola hoja con los *b* elementos.
- Todos los nodos internos a profundidad *r* están asociados a una clave K_r. Las K_r pueden ser aleatorias, miembros de *X* o tomadas especialmente de *X* para optimizar.
- Cada nodo interno v es raíz de un FQ-TREE valido asociado a un sub-conjunto X_V contenido en *X* (X_V contenido dentro del 'gran' *X*).Un nodo interno v con clave K_r tiene un sub-árbol para cada sub-conjunto X_i no vacío definido como:

$$X_i = x \in X_v \quad d(K_r, x) = d_i > 0$$

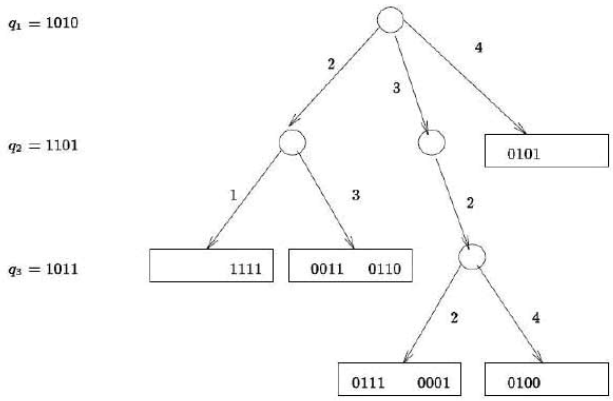


Figura 13: FQ-Tree con b= 2 y distancia de Hamming

A la hora de buscar por una consulta q, vamos bajando en el árbol y comparando con las claves K_i de cada nivel. Si la búsqueda es exacta, en cada nivel el hijo que corresponda a la distancia entre *q* y K_i es seleccionado hasta encontrar una hoja. Ahora bien, si la búsqueda es por rango o vecinos más cercanos es un poco más complicado. Supongamos que tenemos que buscar todos los elementos de *X* que están a una distancia *d,* entonces usando la desigualdad triangular filtraremos los elementos. Si estamos en el nodo interno *v* con clave K_i, y *si d(q,* K_i*) = di* , entonces TODOS LOS HIJOS asociados a v con distancia *w* tal que *(di −d)* $\leq w \leq$ *(di +d)* contienen los elementos que están a distancia *D* de la consulta *q*. Así seguimos buscando recursivamente, aprovechando el hecho de tener una misma clave para todos los elementos en el mismo nivel, además realizamos una sola comparación por nivel.
El costo de las búsquedas está muy relacionado con la altura del árbol, y con la cantidad de comparaciones que tenemos que realizar dentro de un bucket de b elementos en la hoja ya encontrada.
Ahora bien, como los pivotes no residen en los nodos podemos promover mejoras incrementando el número de pivotes, o lo que es lo mismo incrementar su altura, logrando así los llamados: "Fixed Height Fixed Queries Trees" (FH-FQT, Baeza - Yates, 1997), donde se observaron mejores resultados experimentales que su predecesor FQ-TREE

FQA: Fixed Queries Array [Chavez, 2001b]

Es una nueva estructura de datos que tiene dos propiedades interesante:

- Es la primera en lograr un orden sub-lineal de cómputos sin usar espacio extra.
- Es capaz de "negociar" el número de pivotes con precisión para optimizar el uso del espacio disponible.

Es considerada una opción simple y eficiente para consultas de proximidad en espacios métricos, ya que para búsquedas exactas usamos búsqueda binaria con un vector o un árbol.

Estructura:
Suponemos que trabajamos con un conjunto de distancias posibles discreto, entonces primero tomamos *k* pivotes y obtengo la distancia de todos los elementos de la base de datos a esos *k* pivotes. En el FQA estamos almacenando una lista de *k* distancias enteras. Luego todos los elementos de la base de datos

simplemente son ordenados en orden lexicográfico, es decir, 1ero ordenamos por la distancia al pivote 1, luego ordenamos las igualdades por la distancia al pivote 2, y así continuamos ya que el vector se va ordenando mas y mas. Lo que obtenemos como resultado es algo muy similar a un FH-FQT de altura k, ya que si recorremos las hojas de esta ultima estructura en orden, obtenemos el FQA. Es más, cada nodo en el FH-FQT se corresponde con un rango de celdas en el FQA, es decir, los primeros h valores se corresponden con el camino de los primeros valores en los nodos de un FH-FQT de altura *h*.

Para que la idea quede más clara, veamos rápidamente el algoritmo de búsqueda. Dada un elemento consulta *q*, un radio de tolerancia *r* y los *k* pivotes $p_1, p_2, \ldots, p_k$ calculamos $d_1 = d(q, p_1)$. Ahora para cada i entre $(d_1 - r)$ y $(d_1 + r)$ realizamos una búsqueda binaria de rangos clasificando los i elementos.
Luego, para cada uno de ellos, realizamos una búsqueda recursiva en el sub-array encontrado, pero desde el pivote *p2*. La búsqueda termina cuando utilizamos los k pivotes, y los sub-arrays obtenidos hasta este momento son evaluados secuencialmente. La búsqueda de los vecinos más cercanos se realiza de forma similar.
La búsqueda en el FQA es equivalente al ir entrando recursivamente en los
Sub-árboles i-ésimos de los FH-FQT.

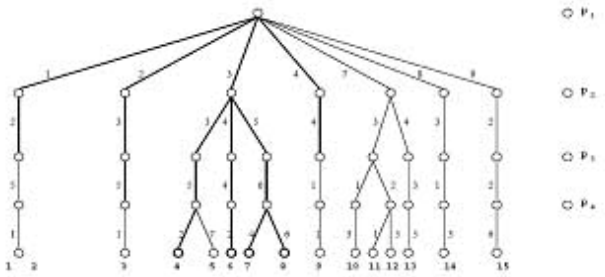


Figura 14: Implementación de FQA

Costos:

CONSTRUCCION: $\Theta(kn \log n)$ cubre la complejidad de construcción para evaluar las distancias a los k pivotes y luego obtener un orden lexicográfico.

Función DISTANCIA: Usando $\Theta(\log n)$ pivotes evaluamos $\Theta(\log n)$ veces la función distancia.

CPU: $\Theta(n^{\alpha} \log n)$ con α entre 0 y 1, recordando que FH-FQT tiene $\Theta(n^{\alpha})$

AESA: Approximating and Eliminating Search Algorithm

AESA es una técnica de indexación que se usa desde 1986. Este algoritmo calcula y guarda en una matriz, donde cada entrada guarda la distancia $d(u,v) \mid \forall u, v \in U$, esto es calcular $\Theta(n \times n)$ distancias.

AESA trabaja eligiendo pivotes del conjunto de los candidatos, y los usa para sacar más candidatos futuros, y el más cercano de los pivotes es el corte más efectivo. Puede parecer que $\Theta(n \times n)$ distancias es algo muy costoso para tener en memoria, pero para pequeñas aplicaciones es un número razonable y se puede implementar, es decir, sigue siendo considerada una opción viable.
Ahora bien, otras aplicaciones para bases de datos con gran volumen de datos optan, por ejemplo, en fraccionar sus bases de datos, y en cada una de esas partes utilizar AESA.

BUSQUEDAS:
AESA, al igual que la mayoría de los algoritmos de búsquedas indexadas, comienza tomando un pivote *u* Perteneciente a *U* para comparar con la consulta *q* y luego filtrar la mayor cantidad de elementos de *U,* y luego va repitiendo esa idea hasta comparar (o descartar) una cantidad suficiente de elementos.
AESA propone un método específico para elegir pivotes: el próximo pivote para comparar con q es el candidato u que minimiza:

$$D(u) = \sum_{p \in P} | d(u,p) - d(p,q) |$$

Y el algoritmo es el siguiente:

AESA

1. Let $\mathbb{P} \leftarrow \emptyset$ set of pivots
2. Let $\mathbb{F} \leftarrow \emptyset$ set of filtered elements
3. $r \leftarrow \infty$
4. For $u \in \mathbb{U}$, $D(u) \leftarrow 0$, $D_{max_u} \leftarrow 0$
5. **while** $\mathbb{U} \neq \mathbb{P} \cup \mathbb{F}$ **do**
6. $p \leftarrow \text{argmin}_{u \in \mathbb{U}-\mathbb{P}-\mathbb{F}} D(u)$
7. $\mathbb{P} \leftarrow \mathbb{P} \cup \{p\}$
8. **if** $d(q,p) < r$ **then**
9. $r \leftarrow d(p,q)$
10. $p^* \leftarrow p$
11. **for** $u \in \mathbb{U} - \mathbb{P} - \mathbb{F}$ **do**
12. $D_{max_u} \leftarrow \max(D_{max_u}, |d(q,p) - d(u,p)|)$
13. **if** $D_{max_u} > r$ **then**
14. $\mathbb{F} \leftarrow \mathbb{F} \cup \{u\}$
15. **else**
16. $D(u) \leftarrow D(u) + |d(q,p) - d(u,p)|$
17. **return** p^*

COSTOS:
ESPACIO: $\Theta(n \times n)$ la complejidad de espacio de almacenamiento de la matriz. Aunque el número de cálculo de la función distancia es constante.
CPU: $\Theta(h \times n)$, donde *h* es la cantidad de iteraciones sobre los elementos que aun no han sido descartados.

MEJORAS:
Una importante para realizarle es la forma de elegir los pivotes, aunque se han publicado muchos documentos donde, partiendo de AESA, se construyeron nuevos algoritmos junto con estructuras de datos que aceleran las búsquedas y bajan la complejidad de espacio.
En el año 1994, E.Vidal junto a Oncina presentaron Linear-AESA (LAESA, 1994) que tiene un orden lineal en espacio y la misma performance de búsqueda que AESA. Pero la principal desventaja que tiene es que no es práctico para grandes bases de datos, ya que se incrementa mucho la cantidad de veces calculamos la función distancia.
A continuación veremos algunas de las mejoras más importantes al AESA original.

Mejorando AESA: iAESA y TLAESA

- *iAESA*: El algoritmo consiste básicamente en modificar el criterio de aproximación, el cual es reemplazado por la similitud entre dos permutaciones entre los pivotes ya utilizados, y definido así:
- Definición de similitud entre 2 permutaciones:

$$F(u) = F(\prod_u , \prod_q) = \sum_{i=1}^{|P|} | \prod_u^{-1} p_i - \prod_q^{-1} (p_i) |$$

Entonces, en lugar de utilizar D(u) en el algoritmo, usamos F(u) recién definida, y el nuevo pivote es el que tenga el menos F(u).

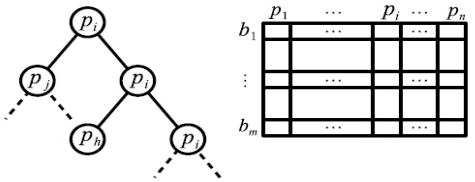
iAESA

```
1.  Let ℙ ← ∅ set of pivots
2.  Let 𝔽 ← ∅ set of filtered elements
3.  r ← ∞, Π_q ←<>
4.  For u ∈ 𝕌, F(u) ← 0, Π_u ←<>
5.  For u ∈ 𝕌, D_max_u ← 0
6.  while 𝕌 ≠ ℙ ∪ 𝔽 do
7.    p ← argmin_{u∈U−P−𝔽} F(u)
8.    ℙ ← ℙ ∪ {p}
9.    insert p in Π_q
10.   if d(q,p) < r then
11.     r ← d(p,q)
12.     p* ← p
13.   for u ∈ 𝕌 − ℙ − 𝔽 do
14.    D_max_u ← max(D_max_u, |d(q,p) − d(u,p)|)
15.    if D_max_u > r then
16.      𝔽 ← 𝔽 ∪ {u}
17.    else
18.      insert p in Π_u
19.      F(u) ← F(Π_q, Π_u)
20. return p*
```

- *Tree Linear Approximating and Eliminating Search Algorithm* (TLAESA,1996)

TLAESA redujo la complejidad a sublineal usando 2 estructuras: una matriz de distancias (Distance matrix) y un árbol binario de búsqueda (Search Tree).

ESTRUCTURA:



En el árbol mantengo la jerarquía de todos los objetos, usando la idea de los "Disjoint Subset" con el árbol, según los nodos izquierdo y derecho.

Cada nodo t se corresponde con un subconjunto $S_t \subset P$ (P son todos los objetos). Cada nodo t tiene un puntero a su pivote p_t a S_t , también tiene un puntero a su hijo izquierdo *l*, derecho *r* y guarda su radio de cobertura R_t definido como: $r_t = \max_{p \in S_t} D(p, p_t)$

Luego, S_T es particionado en dos subconjuntos disjuntos S_L y S_R definidos así: $S_R = \{ P \in S_t \mid d(p, p_r) < d(p, p_t) \}$ $S_L = S_T - S_R$

Mostremos que si *t* es una hoja, $S_T = \{ p_t \}$ y $R_t = 0$. Aplicamos recursivamente esta idea para construir el árbol binario, partiendo de una elección aleatoria del pivote raíz *p*, hasta que S_T tenga un solo elemento.

En la matriz de distancia guardo las distancias de cada objeto con los pivotes elegidos. Para armar dicha matriz de distancias tenemos que elegir los pivotes considerando:

1-Cuantos, en función de la dimensionalidad

2-Cuales, según la distribución, nosotros elegimos los objetos que maximizan la distancia mínima a los otros pivotes ya seleccionados.

BUSQUEDAS:

Vamos recorriendo el árbol en profundidad y solo calculamos la función distancia en las hojas, ya que las restantes que necesito están en la matriz distancia.

1. En el arreglo *D[ ]* guardo las distancias de los pivotes P_i a q.
2. Entre ellos, tomamos el más cercano a q y lo guardamos como "el vecino más cercano"
3. Comienzo a recorrer el árbol y a comparar las distancias con cada uno de los nodos, actualizando el pivote más cercano $p_{\min}$ y su distancia $D_{\min}$ si es necesario al momento de encontrarnos con un nodo hoja.
4. Al seguir recursivamente obtengo los n-vecinos más cercanos, según el pivote que termine dejando. Veamos esto más claro en el pseudocódigo a continuación,

procedure NN search(q)

```
1:  t ← root of T
2:  d_min = ∞, g_{p_t} = 0
3:  for b ∈ B do
4:     D[b] = d(q, b)
5:     if D[b] < d_min then
6:        p_min = b, d_min = D[b]
7:     end if
8:  end for
9:  g_{p_t} = max_{b∈B} |(D[b] − M[b, p_t])|
10: search(t, g_{p_t}, q, p_min, d_min)
11: return p_min
```

Y las búsquedas recursivas (línea 10) se realizan en el pseudocódigo siguiente:

procedure search$(t, g_{p_t}, q, p_{min}, d_{min})$

```
1:  if t is a leaf then
2:     if g_{p_t} < d_min then
3:        d = d(q, p_t) {distance computation}
4:        if d < d_min then
5:           p_min = p_t, d_min = d
6:        end if
7:     end if
8:  else
9:     r is a right child of t
10:    l is a left child of t
11:    g_{p_r} = g_{p_t}
12:    g_{p_l} = max_{b∈B} |(D[b] − M[b, p_t])|
13:    if g_{p_l} < g_{p_r} then
14:       if d_min + r_l > g_{p_l} then
15:          search(l, g_{p_l}, p_min, d_min)
16:       end if
17:       if d_min + r_r > g_{p_r} then
18:          search(r, g_{p_r}, p_min, d_min)
19:       end if
20:    else
21:       if d_min + r_r > g_{p_r} then
22:          search(r, g_{p_r}, p_min, d_min)
23:       end if
24:       if d_min + r_l > g_{p_l} then
25:          search(l, g_{p_l}, p_min, d_min)
26:       end if
27:    end if
28: end if
```

MEJORAS:
Se propone usar un árbol de M-Vías y la búsqueda Best First Order en lugar de un árbol binario y búsqueda en profundidad. Con este orden de búsqueda vamos eligiendo "qué nodo nos conviene más" para encontrar el objeto (o los objetos) más similares a nuestra consulta q.

VPT: Vantage Point Tree - [Uhlmann, 1991]

Este índice fue diseñado para trabajar en espacios métricos con funciones distancia continuas. El VPT es un árbol binario, donde los elementos en cada nodo interno se seleccionan de forma aleatoria entre los elementos del subárbol, y el criterio de división es la mediana del conjunto de todas las distancias en el subárbol.

Inicialmente, sea *p* el elemento en el nodo raíz, $M = mediana\{u \in U \mid d(p,u)\}$ los elementos con $d(p,u) = M$ serán insertados en el subárbol izquierdo, el resto en el subárbol derecho.
Lo que realiza entonces este índice es algo similar al *BKT* donde sólo existen dos etiquetas *i = M* e *i > M*.
Se toma como criterio de división la mediana, lo que genera un árbol balanceado.
A nivel de complejidad, el espacio que ocupa este índice es $\Theta(n)$.
Una propiedad interesante que remarcan en la documentación de esta estructura, es que cada elemento sólo almacena información para saber si la distancia es mayor que *M* o menor igual a *M*.
Pero la desventaja que tiene este índice es que existen elementos que no fueron descartados tempranamente por no tener la información exacta.
También se explica en los documentos de referencia que en espacios de alta dimensión, la mayoría de los elementos y la consulta, se encontrarán muy cerca de la mediana, por lo que será muy probable el descenso por ambas ramas de esta estructura índice.

MVPT: Multi Vantage-Point Tree - [Bozkaya, 1997]

Este índice es una extensión al *VPT*. Se construye un árbol m-ario usando *m-1* percentiles uniformes en lugar de sólo la mediana.
Los autores encontraron algunas mejoras al *VPT* no muy importantes, ya que el espacio de memoria utilizado sigue siendo $\Theta(n)$, porque los elementos son almacenados en las hojas.
También nos encontramos con que los pivotes de este algoritmo para generar el índice indexan sólo a un subconjunto de elementos, pues son tomados del conjunto de objetos en el subárbol al que pertenecen.
Las mejoras encontradas necesitan de muchos elementos por nodo, lo que hace muy costoso el recorrido en el árbol por más de una rama.

VPF: Excluded Middle Vantage Point Forest - [Yianilos, 1999]

Este índice es otra generalización de los *VPT*, y está diseñado para responder consultas con un radio limitado (búsquedas de los vecinos más cercanos dentro un radio).
El método consiste en excluir, para cada árbol, los elementos que se encuentren en las zonas concentradas. Con los elementos no excluidos se construye un *VPT*.
Con los elementos que fueron excluidos se repite el proceso y se forma un segundo árbol, y así sucesivamente, hasta formar un bosque.
Cuando se tiene una consulta se busca sólo en aquellos árboles que tienen la intersección con la esfera de la consulta.

T-Spanners [Paredes, 2002]

En teoría de grafos se tiene el concepto de t-spanner, que consiste en un subgrafo G` de un grafo G tal que aproxima las distancias en G con un factor de precisión *t*.
Se considera un t-spanner G` (V, E) como la representación de la base de datos métrica. En G` el conjunto de vértices V corresponde a los objetos del espacio métrico y el conjunto de aristas E corresponde a una selección reducida de las distancias entre pares de objetos.
Se proponen varios algoritmos de construcción de t-spanners, de inserción y borrado de objetos, de reconstrucción de t-spanners, y de recuperación de objetos utilizando la existencia de clusters de elemento para dar mejoras en el ámbito de los espacios métricos.

SSS: Sparse Spatial Selection [Pedreira , 2007]

Sparse Spatial Selection (SSS) es una técnica que selecciona de forma dinámica un conjunto de pivotes bien distribuidos en todo el espacio métrico. Se basa en la idea de que, si los pivotes están dispersos "espacialmente" en el espacio, serán capaces de descartar más objetos durante la búsqueda. Para lograr este objetivo, cuando un objeto se inserta en la base de datos, se selecciona como un nuevo pivote si está lo bastante lejos de los pivotes ya seleccionados. Se considera que esta lo bastante lejos de otro pivote si esta a una distancia mayor o igual que $M \times \alpha$de este, siendo *M* la distancia máxima entre dos objetos cualesquiera y α un parámetro cuyos valores óptimos están cerca de 0,4 (es decir, el objeto se selecciona

como pivote si esta a poco menos de la mitad de la distancia máxima de todos los demás pivotes). El parámetro α influye en el número de pivotes que se seleccionan. La Figura 15 muestra con varias colecciones de datos que los valores óptimos de este parámetro siempre están entre 0,35 y 0,40, y que la eficiencia de la búsqueda es prácticamente la misma para todos los valores dentro de este intervalo.

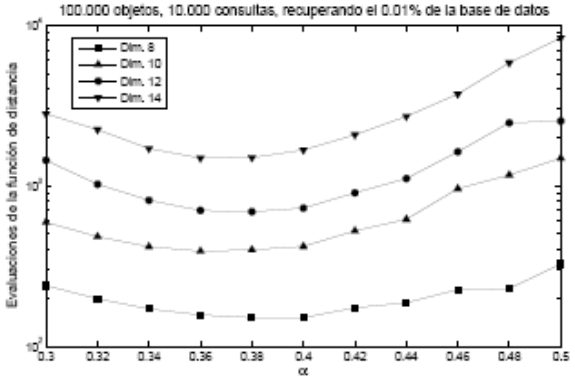


Figura 15: Eficiencia de la búsqueda en función del parámetro α

Los resultados que encontramos en la bibliografía nos demuestran que esta técnica es más eficiente que las anteriores en la mayoría de los casos. Además de ser eficiente, SSS posee otras características muy importantes. SSS es una técnica dinámica.
Es decir, siguiendo el procedimiento de selección anterior explicado, la base de datos puede estar inicialmente vacía, y los pivotes se seleccionaran cuando sea necesario según crece la base de datos. También es una técnica adaptativa, es decir, en SSS no es necesario establecer de antemano el número de pivotes a seleccionar. Cuando la base de datos crece, el algoritmo determina si la colección se ha vuelto lo bastante compleja como para seleccionar otro pivote. Así, SSS se adapta a la dimensionalidad intrínseca del espacio métrico.
Además esta técnica de selección de pivotes se evaluó con distintas colecciones de prueba, dando lugar a una eficiencia un poco mejor que técnicas anteriores, a lo que suma las propiedades de ser una estrategia dinámica y adaptativa.
Aunque originalmente se pensó como una técnica para la selección de pivotes, en este trabajo se la ha utilizado para la selección de los pivotes iniciales del índice, basándose en la hipótesis de que, si los pivotes están bien distribuidos en el espacio métrico, la partición del espacio será mejor y las búsqueda más eficiente, para luego ir adaptando el índice a las búsquedas aplicando las políticas de selección aquí propuestas.

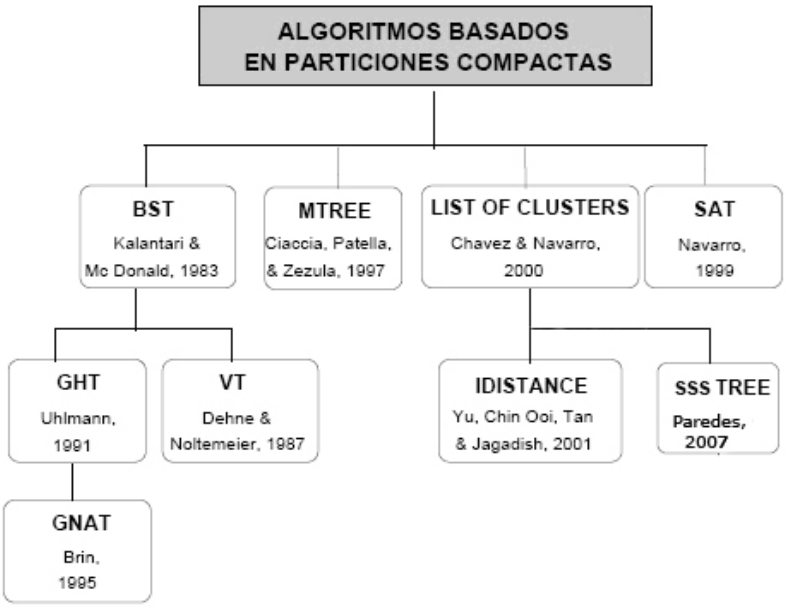
Heurísticas de selección de pivotes

En trabajos anteriores se han propuesto varias heurísticas para la selección de pivotes, según si la función distancia es continua o discreta. Veamos un poco la situación…
En [Micó, 1994] se propone seleccionar como pivotes los objetos que maximicen la suma de las distancias a los pivotes ya seleccionados. En [Yianilos, 1993] aplican heurísticas para obtener pivotes lejanos entre sí. [Bustos, 2001] presenta varias estrategias de selección basadas en un criterio de eficiencia capaz de determinar si un conjunto dado de pivotes es más eficiente que otro en una colección de datos.
La primera, denominada *Selection*, consiste en seleccionar *N* grupos aleatorios de pivotes y quedarse con el que presente un mayor valor en esta medida de eficiencia.
La siguiente, *Incremental*, consiste en realizar una selección incremental en la que el siguiente pivote seleccionado es el objeto que maximice el valor de la medida de eficiencia al unirlo a los pivotes ya seleccionados. Por último, proponen una estrategia iterativa denominada *Local Optimum*, en la que se parte de un conjunto aleatorio de pivotes y en cada paso se reemplaza por otro el pivote que menos aporte a la medida de eficiencia.
La conclusión de estos trabajos sobre las distintas políticas de selección de pivotes que formen el índice es que, los buenos pivotes deben estar distantes entre sí y también del resto de objetos de la base de datos, aunque esta condición no asegura siempre que sean buenos pivotes.

2.2.2 Algoritmos Basados en Clustering

Recordemos que estos algoritmos particionan el espacio métrico en varias regiones o clusters, cada uno de ellos representado por un centro de cluster. Así, el índice guarda información para que, al momento de procesar una consulta, puedan descartarse los clusters completos comparando la consulta con los centros de cada cluster. En aquellos que no puedan descartarse, la consulta se compara con todos los objetos que pertenecen al cluster, y que forman la lista de candidatos.

Veamos brevemente los métodos basados en *clustering* más importantes.



M-Tree - [Ciaccia, 1997]

Los autores buscaban reducir el número de cálculos de distancia para responder las búsquedas por proximidad, obtener una estructura dinámica y con un buen desempeño en operaciones de entrada/salida.
Nuevamente se realiza la construcción de un árbol, donde se toman objetos representantes formando un conjunto, y cada uno de estos forma un subárbol con los elementos cercanos a él.
Este índice con su estructura tiene una cierta semejanza con un *GNAT* donde los elementos cercanos a un representante son colocados en un subárbol. La mayor diferencia de este algoritmo está al momento de realizar la inserción de los elementos: en el *M-Tree* se insertan siempre en el "mejor" subárbol, no siempre en el más cercano como es el caso del *GNAT*.
Tengo que aclarar que se considera "mejor subárbol" a aquel con menor expansión en su radio de cobertura al insertar un elemento. Los elementos son almacenados en las hojas y cuando éstas llenan su capacidad se realiza una separación en dos nodos y un elemento es promovido hacia el padre, como en el *B-tree* o un *R-tree* [Guttman, 1984].
Se utiliza el radio de cobertura para ir descartando ramas en la estructura.
A nivel de complejidad, tengo que destacar que cada elemento guarda la distancia a su padre, lo que hace que este algoritmo use $\Theta(n)$ espacio.

List of Clusters [Chavez, 2000, 2005] y iDistance: Indexing the Distance [YU, 2001]

Estos dos tipos de índices tienen muy buen desempeño en dimensiones altas.
En los dos, la construcción del índice consiste en seleccionar un centro *'p'* y agrupar sus *'m'* elementos más cercanos. Para los elementos restantes se aplica el mismo proceso hasta que todos los elementos hayan sido agrupados.

Cada centro almacena un radio de cobertura, que después se usa para descartar elementos en una consulta. La complejidad de espacio de la estructura es $\Theta(n)$.

La diferencia entre estos dos algoritmos es que *List of Clusters (LC)* recorre la lista secuencialmente mientras que *iDistance* asocia una "clave" y a cada elemento dependiendo de la i-ésima zona (i) en que fue agrupado.
Se calcula así: $y = i \times c + d\ (u, pi)$, donde c es una constante (grande) que trata de evitar el solapamiento entre las claves. Luego, las claves son indexadas con un *B+-tree* para su localización.
Este índice muestra mejores tiempos de respuesta que el *M-Tree*, pero empeora cuando se recuperan muchos vecinos.
Los autores en *LC* utilizan dos criterios para agrupar a los elementos: fijando en *'m'* el número de elementos más cercanos, o usando un radio de tolerancia.
Los mejores resultados los obtuvieron cuando limitan el número de elementos más cercanos a *'m'*.

BST: Bisector Tree

Es un árbol binario que se construye recursivamente. Para cada nodo del árbol se eligen dos centros *p1* y *p2*. Los objetos más cercanos a *p1* que a *p2* se insertan en el subárbol izquierdo, y los objetos más cercanos a *p2* se colocan en el subárbol derecho. Para ambos centros se almacena la información de su radio de cobertura respectivo, al que representaremos por *rc1* y *rc2*.
A nivel de complejidad es $\Theta(n)$ en memoria utilizada, ya que cada elemento guarda la información de la distancia a su centro. Durante una búsqueda de un objeto *'q'*, se pueden descartar todos los subárboles donde se cumpla que $d\ (q, pi) - rci > r$.

GHT: Generalized-Hyperplane Tree [Uhlmann, 1991]

Es idéntico en construcción a un *BST*. Pero tenemos la diferencia que, el algoritmo utiliza hiperplanos como condición para descartar ramas del árbol, en lugar del radio de cobertura.
A nivel de complejidad, el espacio requerido por esta estructura es $\Theta(n)$.
El criterio para recorrer el subárbol izquierdo es $d\ (q, p1) - r < d\ (q, p2) + r$ y para entrar al subárbol derecho es $d\ (q, p2) - r = d\ (q, p1) + r$.

VT: Voronoi Tree

Este algoritmo es otra mejora al *BST*. El *VT* tiene 2 ó 3 elementos (hijos) por nodo.
Cuando se crea un nuevo nodo, se insertará en él aquél elemento más cercano que pertenece al padre.
Este algoritmo tiene la propiedad de que el radio de cobertura se reduce a medida que se desciende por el árbol.
Los autores muestran que el *VT* es superior y mejor balanceado que el *BST*.
Y la complejidad del espacio requerido es $\Theta(n)$ ya que los pivotes indexan localmente la información.

SAT: Spatial Approximation Tree [Navarro 1999, Navarro 2002a]

Como su nombre lo sugiere, este algoritmo no usa centros para separar la base de datos, sino la aproximación espacial.
La creación de este árbol es de forma recursiva, donde se selecciona un elemento *'p'* como raíz del árbol y a éste se conectan todos los vecinos en la base datos que están más cerca a *'p'* que a otro vecino. Los demás se insertan en su vecino más cercano, de forma recursiva a medida que se van insertando en cada vecino.
Al momento de realizar las búsquedas se utiliza como criterio de descarte los hiperplanos y el radio de cobertura.
Su complejidad de memoria es de es $\Theta(n)$ ya que cada elemento guarda cuáles son sus vecinos y a qué distancia se encuentran.

SSS-Tree [Paredes, 2007]

Este algoritmo es otra mejora al *SSS* en conjunto con las estructuras de datos árboles y las mejores propiedades de las técnicas de clustering.
Su principal característica es que los centros de cluster se seleccionan utilizando *Sparse Spatial Selection*, una técnica desarrollada originalmente para la selección de pivotes, capaz de adaptarse a la dimensionalidad intrínseca del espacio, como ya antes mencionamos. Gracias a esto, el número de clusters en cada nodo depende de la complejidad del subespacio que tiene asociado.
Por otro lado, al seleccionar los centros de cluster de forma adaptativa, no tiene por qué tener en cada nivel el mismo número de divisiones cada cluster, si no las que se consideren necesarias en función del espacio métrico, una diferencia muy importante con todas las estructuras propuestas anteriormente.

2.2.3 Comparación del Estado del arte

De forma inmediata observamos que los algoritmos enumerados anteriormente utilizan las distancias precalculadas, y guardadas en el índice, para procesar las consultas por proximidad. Entonces, algunos elementos de la base de datos serán descartados sin que se calcule su distancia a la consulta y así se reducirán los cálculos de distancia para responder las consultas.

Al realizar una comparación entre todos estos algoritmos, y analizar las principales diferencias se observa que mientras AESA usa toda la matriz de distancias, el resto de los algoritmos tratan de acercarse lo más posible al desempeño de AESA pero almacenando mucha menos información en memoria.

Al hablar de los algoritmos de pivoteo, lo primero que realizan para disminuir el uso de memoria, es almacenar sólo algunas columnas de esa matriz. Mientras que en los algoritmos basados en clustering, se guardan algunas áreas ó se mantienen cotas inferiores y/o superiores.

Un detalle interesante que aquí se encuentra es que en el algoritmo AESA no existen comparaciones externas, a diferencia de los restantes algoritmos que si realizan estas comparaciones, pues el próximo pivote siempre es seleccionado de la lista de candidatos.

Existen trabajos para los algoritmos basados en pivotes, donde se ha demostrado que existe un número óptimo de pivotes para usar en el índice.
La desventaja es que ese número puede no ser obtenido por falta de memoria, incluso en bases de datos de tamaño moderado. También existen algoritmos que generan de forma dinámica un número óptimo de pivotes en función de la dimensionalidad del espacio, y no en función del número de elementos dentro de la base de datos.

Entonces nos encontramos con que, en los algoritmos que no son dinámicos, para optimizar la memoria que ocupan los índice y acercarse al número óptimo de pivotes y tener un rendimiento lo mejor posible, se utilizan algunas técnicas de optimización como: eliminar información redundante, compactar la información guardada, etc.

Durante el análisis de los distintos algoritmos nos encontramos con diversas características, que son importantes para comprender el por qué de nuestro interés en usar la información de búsquedas anteriores junto con la idea de un índice dinámico para optimizar la selección de pivotes dentro del índice, y así mejorar las futuras búsquedas sin preocuparnos por optimizar la memoria ocupada.
Esto es lo que desarrollaremos a continuación en las próximas secciones, donde esta ampliada nuestra propuesta.

Capítulo 3: Propuesta y Trabajo Realizado

3.1. Introducción

Hemos visto previamente los algoritmos y estructuras para implementar los índices que hacen posible la búsqueda de forma eficiente en estas bases de datos particulares. Algo común que hemos detectado en todos los algoritmos que mencionamos es que tanto los pivotes como los centros de los clusters se seleccionan de forma aleatoria. Pero es sabido ya que el conjunto de referencia seleccionado tiene un impacto directo en los resultados de búsqueda.
Estos elementos de referencia destacados son de suma importancia a la hora de mejorar el desempeño de los algoritmos. Nosotros trabajaremos basándonos en la hipótesis de que si los pivotes del índice están bien distribuidos dentro del espacio métrico, la partición del espacio será mejor desde el comienzo de las operaciones, y como consecuencia directa los resultados de búsqueda serán más eficientes. Es por ello que desarrollaremos una idea para concretar una buena elección de estos pivotes e ir cambiándolos para adaptarlos al uso particular de la base de datos, ya que la construcción inicial del índice será realizada mediante el *Sparse Spatial Selection* (SSS), y la 'actualización' se realizará aplicando dos nuevas políticas de selección aquí propuestas.

Se presenta una nueva estructura de indexación y búsqueda por similitud basada en el SSS. Este método basado en la selección dinámica de pivotes, tiene como característica que utiliza al SSS para la selección inicial de pivotes bien distribuidos en el espacio métrico, lo que da lugar a una mejora en el rendimiento de la búsqueda con respecto a una selección inicial aleatoria de los pivotes. Además, tiene otras interesantes características:

- *dinámico*, ya que permite que la base de datos esté inicialmente vacía y crezca en el tiempo.
- *adaptativo*, ya que no es necesario establecer de antemano el número de pivotes a utilizar dado que el propio algoritmo los selecciona según son necesarios para adaptarse a la complejidad del espacio.

Nuestra hipótesis de trabajo es que, en la construcción del índice, se aplica SSS para seleccionar los pivotes iniciales, con lo cual la partición del espacio es más eficiente y con mejores resultados en las búsquedas. Luego, con el pasar del tiempo y de las búsquedas, se aplican unas nuevas políticas de selección de pivotes para eliminar del índice los pivotes que menos discriminen, y otra para la selección de elementos candidatos a pivotes para formar parte del índice, y así lograr adaptar dinámicamente el índice a las búsquedas realizadas durante un determinado tiempo, completando así nuestra propuesta.

En las siguientes secciones estaremos hablando siempre de una instancia de una base de datos métrica *DB* de *n* elementos y con un índice de *k* pivotes $\{p_1, p_2, \cdots p_k\}$ que será dinámico.

Propuesta: *selección dinámica de pivotes que se adaptan a las búsquedas*

Vamos a presentar una nueva técnica para la selección de pivotes, que combina las buenas propiedades de trabajos previos con algunas ideas fomentadas recientemente para la selección dinámica de pivotes.

Sabemos que Sparse Spatial Selection (SSS) dinámicamente selecciona un conjunto de pivotes bien distribuidos en el espacio y adapta el índice a la complejidad de la colección. Es decir, SSS lo adapta al índice según la dimensionalidad intrínseca del espacio métrico. Cuando un nuevo objeto es insertado en la base de datos, SSS lo selecciona como un nuevo pivote si está bastante lejos de los pivotes ya seleccionados, según un parámetro obtenido de forma experimental.

La propuesta de selección dinámica de pivotes que se adaptan a las búsquedas presentada aquí, está basada en el SSS que antes comenté junto con una nueva política de selección para el pivote entrante y otra política de selección para el pivote saliente del índice.

La idea, en resumen, es construir un índice utilizando el SSS tal cual como está explicado en [Pedreira, 2007], logrando obtener un índice que tenga sus pivotes adaptados a la dimensionalidad del espacio

métrico. Luego de que está "estabilizado" tanto el índice como la cantidad de elementos de la base de datos, comienzan las búsquedas.

Guardamos los resultados de búsqueda para en un futuro evaluar los distintos conjuntos de resultados teniendo en cuenta qué pivote discrimino más o menos elementos, y así decidir que pivote "es malo" (o discrimina menos elementos, en comparación con los demás pivotes que forman parte del índice).

Por otro lado también analizamos las listas de elementos que son candidatos a ser resultado de la búsqueda, es decir, vamos a analizar los elementos que no son discriminados por los pivotes del índice y que tiene que ser comparados contra el objeto consulta, formando parte de los resultados de búsqueda,
Esto lo realizamos para poder decidir qué elementos son más difíciles de discriminar con este conjunto actual de pivotes, y de esta forma considerarlos como potenciales candidatos a la hora de actualizar los pivotes que constituyen el índice.

¿Cómo vamos a realizar esta propuesta? Bueno, ya construido el índice con SSS, comienzan las búsquedas y almacenamos la estadística de las discriminaciones realizadas por cada pivote, esto es calculado al momento de obtener los resultados de una búsqueda.
Luego de "*B*" búsquedas, donde este *B* es un parámetro que evaluamos de forma experimental, obtengo los porcentajes de discriminación por parte de los pivotes actuales en el índice. Luego detectamos algún pivote muy poco relevante, es decir un pivote que discrimine poco, en relación a los demás pivotes del índice sujeto a las B búsquedas hechas a la base de datos.
Con este accionar recién enunciado, definimos cual es el pivote candidato a ser reemplazado dentro del índice.

Al mismo tiempo, tenemos que seleccionar un nuevo pivote que entre al índice, para esto proponemos llevar una estadística de las apariciones en los resultados de búsqueda de los elementos. Es decir, se lleva una estadística sobre los elementos $x \in DB$ que quedan más veces en la lista de candidatos obtenida como resultado de las consultas. Luego, el elemento que más veces fue candidato va a ser el que entre como pivote al índice, en reemplazo de la victima seleccionada con la política que recién expliqué, según los porcentajes de discriminación.

De esta forma, gracias a la construcción inicial del índice con SSS, tenemos solucionado el problema de la dimensionalidad, y luego gracias a estas dos nuevas políticas de selección dinámicas, vamos adaptando el índice a las búsquedas que se realizan a lo largo del tiempo.

3.2. Construcción inicial del índice

El algoritmo de construcción para la propuesta que aquí presentamos, puede ser dividido en 2 grandes pasos:

a) el proceso de selección de pivotes
b) el proceso de generación del índice y cálculo de las distancias, en donde se calculan las distancias entre los pivotes y todos los objetos de la base de datos.

Sean *(X, d)* un espacio métrico, $U \subset X$ la base de datos y $M = \max \{d(x, y) \mid x, y \in X\}$ la distancia máxima entre cualquier par de objetos. El conjunto de pivotes se constituye inicialmente con sólo el primer elemento de la colección, luego para todos los elementos $x_i \in U$, x_i es elegido como nuevo pivote si su distancia a cada pivote en el conjunto actual es igual o mayor que $M \times \alpha$, siendo α una constante cuyos valores óptimos están cerca de 0,35 y 0,4 como se explica en [Pedreira, 2007]. Es decir, un objeto de la colección será pivote inicial del índice si está ubicado en nuestro espacio métrico a más de una fracción de la distancia máxima de todos los pivotes. Entonces, también tenemos que calcular el valor de M en esta etapa de construcción.
Por ejemplo, si $\alpha = 0{,}5$, un objeto se selecciona como pivote si está a más de la mitad de la distancia máxima de todos los pivotes.

Los pivotes seleccionados para constituir el índice inicial de esta forma, estarán algo distantes entre sí (a mas de $M \times \alpha$), algo que muchos autores señalan como una característica deseable del conjunto de pivotes.
Nuestra propuesta genera un número de pivotes para el índice inicial que depende de la dimensionalidad del espacio métrico, entonces en el momento de la construcción deberá crecer rápidamente la cantidad de

pivotes en el índice, y este valor se estabilizará al crecer mucho la base de datos. Al momento de la experimentación, evaluaremos el comportamiento del crecimiento de este valor.

3.3. Crecimiento inicial de la colección

Como este método es dinámico, entonces partimos de que la colección de elementos está inicialmente vacía. El primer objeto insertado en la base de datos, u_1 es nuestro primer pivote llamado p_1. Se inserta el segundo (o nuevo) objeto en la base de datos, calculo y almaceno su distancia a todos los pivotes que ya tengo en el índice. Si su distancia a todos es mayor o igual que $M \times \alpha$, lo agrego al conjunto de pivotes.
Es recién en este momento cuando calculo y almaceno la distancia del nuevo pivote a todos los objetos de la base de datos. De esta forma, el conjunto de pivotes no tiene que ser seleccionado de forma aleatoria sobre la colección inicial de objetos, sino que se va constituyendo el índice a medida que crece la base de datos, hasta un determinado momento o "umbral máximo" que alcanzaremos, según la capacidad física de la base de datos.
También tengo que ir actualizando el valor de la variable M, que mantiene la distancia máxima entre cualquier par de objetos, $M = \max\{\, d(x,y) \mid x, y \in X \,\}$.
Y por ultimo generar nuevamente el índice para actualizar distancias dentro de la matriz.
Esta operación de crecimiento inicial lleva un tiempo de computación $\Theta(k \times n)$ y espacio $\Theta(k \times n)$, siendo *k* la cantidad de pivotes en el índice.

3.4. Constantes: M y α

Al ingresar nuevos elementos en la base de datos, en la fase de construcción, y decidir si tengo o no que agregar un nuevo pivote al índice, tengo que ir actualizando el valor de la variable M, que mantiene la distancia máxima entre cualquier par de objetos, $M = \max\{\, d(x,y) \mid x, y \in X \,\}$. Por el momento, este cómputo se está realizando comparando la distancia de todos los elementos con todos los elementos, sin utilizar técnicas avanzadas de programación ni optimización, como lo es la 'programación dinámica', o '*divide and conquer (D&C)*'. Entonces el orden de tiempo de computación es $\Theta(n^2)$ y de espacio $\Theta(n^2)$.

Por otro lado, tenemos que destacar que, si bien es cierto que en esta propuesta no es necesario prefijar de antemano el número de pivotes a utilizar dentro del índice como en la mayoría de los algoritmos que estudiamos en el estado del arte, si tenemos que prefijar el valor de α y que a su vez dicho valor nos condiciona el número de pivotes.
De forma experimental, y basándonos en la información brindada en [Pedreira, 2007], llegamos a la conclusión de que el valor de α debe estar entre 0.35 y 0.40, según sea la dimensionalidad de la colección.
Más adelante en este trabajo, analizaremos el número de evaluaciones de la función de distancia realizadas en función del parámetro α para espacios métricos sintéticos de dimensión 8, 10, 12 y 14. Y así lograr observar que el mejor resultado se obtiene con valores de α situados entre 0.35 y 0.40, aunque la eficiencia del índice propuesto es prácticamente la misma para todos los valores incluidos en ese intervalo.

3.5. Búsquedas

Una vez que la base de datos creció en un tamaño considerable e interesante para la experimentación, y tenemos el índice construido, comenzamos a realizar las búsquedas dentro del espacio métrico.
Sea (*q, r*) la consulta. Primero tengo que calcular la distancia del objeto consulta *q* a todos los pivotes del índice. Una vez completado esto, aplicando la desigualdad triangular $d(x,y) \le d(x,z) + d(z,y)$

descartamos todos los elementos $x_i \in U \mid d(x_i, p_j) - d(d, p_j) \mid > r$ siendo p_j un pivote, ya que por la desigualdad triangular, si se da esta condición, su distancia a *q* será $d(x_i, q) > r$.
Los objetos de nuestra base de datos que no se descarten utilizando esta propiedad formaran la lista de candidatos, $\{u_1, u_2, ..., u_n\} \subset U$ y deben compararse directamente con la consulta de forma exhaustiva.
También, tomaremos esa lista de candidatos $\{u_1, u_2, ..., u_n\} \subset U$ y aumentaremos en una unidad la estadística que llevamos sobre los elementos de la base de datos que forman parte de los resultados de búsqueda.
Aquí la operación de búsqueda lleva un tiempo de computación $\Theta(k \times n)$ y espacio $\Theta(n)$, siendo *k* la cantidad de pivotes en el índice.

3.6. Actualización del índice – Propuesta de nuevas políticas

Hasta el momento ya hemos presentado una estructura para índice, similar a anteriores expuestas en el estado del arte.
Veamos ahora como obtener una nueva versión de dicha estructura, utilizando las nuevas políticas de intercambio que aquí proponemos.

¿Cuándo es "malo" un pivote?

Una vez constituido el índice con el número de pivotes estabilizado y acorde a la dimensionalidad intrínseca, se realiza una consulta de un elemento *q*, y es aquí donde se calcula en tiempo de consulta $d(q, p_i)\ i = 1...k$ es decir, la distancia entre este elemento *q* consulta y todos los pivotes del índice.
Luego si $\max_{1 \le j \le k} \mid d(q, p_j) - d(e, p_j) \mid > r$ entonces el elemento *e* se encuentra fuera del rango de consulta. Tenemos como conclusión que el pivote p_j discrimina de forma exitosa al elemento $e \in DB$.
Vemos entonces que en una consulta se pueden eliminar a lo sumo *n* elementos (es decir, todos los elementos de la base de datos) lo cual habría repartido esa *n* discriminaciones entre los *k* pivotes, también en una consulta podemos saber qué pivote realizó la discriminación de un elemento $e \in DB$.

Siguiendo con este mismo concepto de prestar atención a los elementos discriminados por los distintos pivotes, y con un conjunto interesante de consultas (*B* búsquedas, donde *B* tiene que ser un valor grande y representativo) realizadas a DB definimos "el porcentaje de discriminación del pivote i-ésimo p_i a $[\%Disc(p_i)]$:

$$[\%Disc(p_i)] = Disc(p_i)/(B \times n)$$

donde:
- $Disc(p_i)$: *es la cantidad de elementos que discrimino el pivote* p_i.
- $B \times n$: *representa el total de discriminaciones posibles.*

Luego, diremos que un pivote p_i es "malo" cuando

$$[\%Disc(p_i)] < 1/k$$

donde *1/k* es un umbral que se obtiene de forma experimental, trabajando sobre un espacio métrico sintético o de experimentación, que nos permita controlar su dimensión y la cantidad de elementos del espacio métrico junto con la cantidad de pivotes en el índice.

Si el pivote i-ésimo p_i tiene su porcentaje de discriminación por debajo de este umbral, decimos que el pivote p_i está siendo *muy poco relevante a la hora de discriminar*, por lo menos sujeto a las *B*

búsquedas hechas a la *DB*. Entonces, es aquí cuando lo seleccionamos como una víctima dentro del índice, para en un futuro reemplazarlo.
En el caso de que tengamos más de un elemento pivote que discrimine poco, o que "sea malo", siempre será el último el candidato a ser reemplazado en un futuro.

¿Qué elemento se vuelve pivote? El mejor candidato.

Ya tenemos una política para la selección del pivote victima dentro de nuestro índice, que es aquel que "menos discrimina". Ahora es el momento de seleccionar su reemplazante, luego de realizadas *B* búsquedas en la base de datos.

Existen varias políticas de selección para el nuevo pivote. Como primer política, y la más simple e intuitiva, es la de seleccionar, de modo aleatorio, un elemento que no esté dentro del conjunto de pivotes.
La segunda es la política que propone el SSS, utilizando la misma idea que al momento de crear el índice. Es decir, seleccionamos el nuevo pivote calculando la distancia del elemento contra el conjunto actual de pivotes y que entre ellos estén a una cierta fracción de distancia.

Estas dos políticas no son tenidas en cuenta dentro de nuestra propuesta, ya que seleccionar de modo aleatorio un elemento es 'desperdiciar información' y dejar que el azar ayude a resolver nuestro problema. Utilizando la política que propone SSS sí estamos 'utilizando información', que es la distancia entre los pivotes, que nos asegura que los pivotes estén bien distribuidos en el espacio. Pero este no es nuestro principal objetivo del índice, no queremos que sea 'capaz de descartar más objetos en la búsqueda' porque los pivotes cubren todo el espacio. Queremos que nos dé la mayor cantidad de resultados cercanos a la consulta y en un buen tiempo de respuesta, utilizando información de búsquedas anteriores, y no cobertura del espacio.

El mejor candidato

La tercera política es la que proponemos nosotros aquí, que utiliza un criterio muy similar a la elección del pivote víctima, y la llamaremos "*El mejor candidato*".
La idea es la de contabilizar una estadística de los elementos $e \in DB$ que quedan más veces en la lista de candidatos obtenida como resultado de las consultas.
Esta estadística se almacena en una estructura de datos simple, como lo es un vector de *n* componentes, el cual cuenta las veces que el i-ésimo elemento forma parte de la lista de candidatos. Luego, el elemento que más veces fue candidato va a ser el que la política de selección tome como pivote entrante al índice.

El fundamento de esta política de selección está basado en la idea de que un elemento que formó parte muchas veces de la lista de candidatos es difícil de discriminar por los pivotes actuales que están dentro de nuestro índice. Esto implica que si lo seleccionamos a este elemento como pivote de nuestro índice va a mejorar los porcentajes de discriminación en torno a esa región que lo circunda, en futuras búsquedas.
Este no es el único beneficio de esta política de selección. Como veremos a continuación al momento de la experimentación, esta técnica también adapta los pivotes a la región donde se realizan la mayoría de las búsquedas, lo que transforma al índice en una estructura dinámica y que se adapta de forma automática para seguir cumpliendo su principal objetivo: reducir la cantidad de evaluaciones de la función distancia al momento de una búsqueda.

3.7. Algoritmos

A continuación, enumeraremos los algoritmos que fueron utilizados para realizar una implementación de esta propuesta. Algunos ya fueron mostrados anteriormente, otros son desarrollados aquí.

Construcción del índice

Como este método es dinámico, comenzamos la construcción del índice con la colección de pivotes vacía y teniendo ya un valor obtenido experimentalmente para la constante α y luego debemos calcular el valor de la variable M, que mantiene la distancia máxima entre cualquier par de objetos $M = \max\{ d(x,y) \mid x, y \in X \}$.

El primer objeto insertado en la base de datos, u_1 es el primero en ser seleccionado como pivote p_1. Luego, con el próximo elemento que se inserta en la base de datos, se calcula y almacena su distancia a todos los pivotes que ya tenemos dentro del índice. Si su distancia a todos ellos es mayor o igual que $M \times \alpha$, se añade al conjunto de pivotes. Aquí se calcula y almacena la distancia del nuevo pivote a todos los objetos de la base de datos, y así el conjunto de pivotes comienza vacío y no tiene que ser prefijado de antemano o de forma aleatoria, sobre una colección inicial de objetos, sino que crece al mismo tiempo que crece la base de datos. Esta implementación permite que la estructura sea totalmente dinámica, y que el conjunto de pivotes se adapte a los objetos que se van insertando y su complejidad.
Además, asegura que aunque la colección crezca, los pivotes están bien distribuidos en el espacio métrico. De esta forma, el método sigue siendo eficiente a pesar de que la colección varia (crece o se eliminan elementos) hasta que la colección se estabiliza.

El siguiente pseudocódigo resume este proceso de selección de pivotes iniciales y posteriores inserciones, ya que se realizan de forma similar:

Input: (X,d), U ,M ,α
Output: Pivotes

```
0  Pivots ← x_1
1  FOR ALL x_i ∈ U DO
2        IF( ∀p ∈ Pivotes , d( x_i , p) ≥ M × M × α )
3           THEN Pivotes ← Pivotes ∪ { x_i }
4                 generarIndice()
5           END THEN
6        END IF
7         RecalcularValorDeM()
8         ActualizarValorDeM()
9   END FOR ALL
10  RETURN: Pivotes
```

Búsquedas

Una vez generado el índice, según se explicó anteriormente, ya con un buen número de elementos en la base de datos, vamos a realizar las distintas búsquedas y consultas a la base de datos, haciendo así uso de la estructura índice que aquí proponemos.
Sin entrar en el detalle de cómo se realiza el proceso de búsqueda, podemos resumirlo diciendo que, dado un elemento consulta Q, quiero obtener una lista de elementos candidatos desde la base de datos y finalizar la búsqueda evaluando Q contra cada uno de esos elementos que forman la lista de candidatos, comparando de forma exhaustiva.
El siguiente pseudocódigo resume el proceso de búsqueda, el cual es explicado a continuación más en profundidad:

Input: QueryItem Q y R, como radio de búsqueda
Output: lista de candidatos, {u1, u2,…Un}
Obs: Tenemos ya el índice formado y los elementos en la base de datos.

```
1     FOR ALL ( ∀p_i ∈ Pivotes ) DO
2      queryDistance[i]=calcDistancia(query,indice.getPivot( p_i ));
```

```
3       Stats.nuevaEvaluacionFuncionDist ();
4      END FOR
5      FOR (x_i ∈ U )DO
6         maxdist = 0.0,posicionMaximaDist = 0;
7         FOR ALL ( ∀p_J ∈ Pivotes ) DO
8                 distanciaTemp = indice.obtenerDistancia( x_i , p_i );
9                         tempValue = distanciaTemp - queryDistance[j];
10           distanciaActual = Math.sqrt(Math.pow(tempValue, 2));
11           IF (distanciaActual > maxdist) THEN
12              maxdist = distanciaActual;
13              posicionMaximaDist = j;
14           END IF
15           IF (maxdist > radio) THEN
16            Stats.nuevaDiscriminacion(posicionMaximaDist);
17           END IF
18           IF (maxdist <= radio) THEN
19            Stats.nuevaEvaluacionFuncionDist();
20            listaDeCandidatos.add(getItem( x_i ));
21           END IF
22      END FOR
23      RETURN: listaDeCandidatos;
```

Como se muestra en el algoritmo, comenzamos la búsqueda con el índice ya generado de forma completa y un elemento consulta 'queryItem' Q y R, un radio de búsqueda.
En el primer bucle, de la línea 1 a la 4, calculamos la distancia de queryItem a los pivotes actuales del índice. Luego, se realiza la búsqueda analizando cada elemento contra los pivotes, aplicando la desigualdad triangular $d(x,y) \leq d(x,z) + d(z,y)$ para descartar los elementos utilizando las distancias a los pivotes y acumulándolas para ver si encontramos una distancia mayor a la calculada en la primer iteración, entre el elemento consulta y el pivote (línea 11)
De la línea 13 a la 15, almacenamos cual fue el pivote que discrimina, y luego verificamos que esta distancia obtenida supere el radio de búsqueda, lo que significa que este pivote lo discrimina al elemento. Y entonces, como ese pivote discrimino un elemento más, reflejamos eso en las estadísticas.

Si no lo discrimina, esto lo verificamos en la línea 18, significa que este elemento queda en la lista de candidatos, y se va a realizar una evaluación más de la función distancia.
Entonces, agregamos este elemento a la colección de candidatos que se retornan, aumento en las estadísticas las funciones distancias calculadas hasta el momento y también registro en mis estadísticas quienes formaron parte de la colección de resultados de búsqueda.

Una vez recorridos todos los elementos de la base de datos, en la línea 23 retornamos la lista de candidatos y finaliza la búsqueda.
Solo resta por comparar de forma exhaustiva todos esos elementos contra el elemento de consulta, tarea que se realiza de la misma forma que en otras propuestas, independientemente del índice de trabajo.

Políticas de Selección

Veamos cómo se implementa el algoritmo con la *política de selección de pivote saliente.* Aquí el vector *d* lleva la cuenta de cuantos elementos discrimino el pivote i-ésimo.
Esta estadística debe ser computada cuando se ejecuta cada consulta. Por lo tanto, lleva un tiempo de computación $\Theta(1)$ y espacio $\Theta(n)$.
Veamos el pseudocódigo para implementar la acumulación de estadísticas para luego decidir quien es el pivote que menos elementos discrimina:

Input: DB, discrimina[], pivotes
Output: d con los porcentajes de discriminación.

```
0 initAllStats();
1 FOREACH (Query e on Database)) DO
2   FOR (j = 0 … Database.CANT_INDEX_PIVOT )
3      IF ( x∈ Pivotes discrimina a e ) THEN
4         discrimina[x] = discrimina[x] + 1;
5       END IF
6   END FOR
7 END FOREACH
```

Luego de *B* consultas se determina cual es el pivote con menor porcentaje de discriminación y si este es menor que un umbral constante de tolerancia con valor TOLERANCIA, se lo reemplaza. Entonces,

$$TOLERANCIA = 1 / (1.1 \times k)$$

es un 10% de tolerancia para estabilizar el algoritmo, este parámetro ha sido evaluado de forma experimental. Recordando que K es el número de pivotes actual en nuestro índice, esta constante de tolerancia se utiliza en la siguiente situación:

```
 0  IF (min_{1≤j≤k} discrimina [j] < 1 / (1.1 × k)) THEN {
 1     PivoteSaliente ← GetPivot();
 2          IntercambiarPivotes();
 3     GenerarIndice();
 4  }
```

En la línea 0 se evalúa si el porcentaje de discriminación *j*-ésimo es menor que el umbral de tolerancia. De ser así, se define a este pivote como el que menos discrimina, dejándolo disponible para el intercambio de pivotes dentro del índice, como se indica en la línea 2. Al cambiar un pivote hay que recalcular toda una columna de la matriz distancia del índice (método *GenerarIndice*()) entre el pivote entrante que retorna el método *GetPivot*() y todos los elementos de la base de datos. La complejidad de cambiar un pivote es $n \times \Theta(\text{función distancia})$.
En el caso de que el porcentaje de discriminación no sea menor que el umbral de tolerancia, no realizamos ningún accionar, y continuamos con la inserción de nuevos elementos.

Veamos cómo se implementa la *política de selección de pivote entrante*, el algoritmo en la función "*GetPivot()*", utilizando las estadísticas que cuentan las veces que el i-ésimo elemento forma parte de la lista de candidatos.
Recorremos todas esas estadísticas, y luego, el elemento que más veces fue candidato va a ser el que la política de selección tome como pivote entrante al índice.

Resumimos esta implementación en el siguiente pseudocódigo del Algoritmo con política de selección entrante:

Input: DB, Stats
Output: Stats, un arreglo con las veces que aparecieron los elementos de la base de datos en la lista de candidatos.

```
0 Candidato = NULL; maxCurrentStats = 0;
1 FOREACH ( e on Database)) DO
2    IF (Stats( e ) > maxCurrentStats)
4         THEN Candidato = e ;
5           maxCurrentStats = Stats( e );
6   END IF
7 END FOREACH
8 RETURN Candidato
```

Al comienzo del algoritmo, en la línea 0, inicializamos los valores a cero y NULL para trabajar. En la línea 1 recorremos todos los elementos dentro de la base de datos y a su vez miramos sus estadísticas, así en la línea 2 comparamos con el máximo actual, que representa al elemento que más veces fue candidato y que va a ser el que la política de selección tome como pivote entrante al índice. Si la estadística de este elemento e supera al actual, en la línea 5, lo seleccionamos al elemento como el futuro pivote entrante al índice, y continuamos con la iteración.
Luego, este algoritmo lleva un tiempo de computación $\Theta(1)$ y espacio $\Theta(n)$.

Con esto terminamos de presentar todos los algoritmos relacionados a nuestra propuesta junto con las políticas de selección de pivotes entrantes y salientes del índice.
En la próxima sección se presenta la experimentación y el análisis de los resultados, para la evaluación de nuestra propuesta.

Capítulo 4: Experimentación

Aquí se presentan los resultados de las pruebas preliminares que comparan el rendimiento de nuestra propuesta de mejora al *SSS*.
A lo largo de esta sección se utilizará la experimentación para validar la propuesta ya comentada.
Se mostrará el desempeño de la técnica en bases de datos reales y sintéticos.
En las pruebas se han utilizado las siguientes dos colecciones de datos como *datasets*:

- Espacios vectoriales sintéticos: Colección de 1.000.000 vectores de dimensión 2, 4, 6, 8, 10 y 14, creados sintéticamente, con distribución uniforme y perteneciente al intervalo [-1,1].
- Colección de 14.051 imágenes descargadas de Internet y luego mapeadas a su *Feature Vector*, utilizando la API de Java JAI 1.2.[JAI 1]

Experimento 1: probando las implementaciones.

Sobre los datasets

La ventaja de probar el algoritmo con este tipo de datos (espacios vectoriales sintéticos) es que se puede estudiar su comportamiento en espacios de dimensionalidad conocida.
El desempeño de los algoritmos para las búsquedas por proximidad en espacios métricos, decae a medida que la dimensión del espacio crece. Por lo tanto, es interesante experimentar en espacios donde sea posible variar la dimensión.
Una manera de controlar la dimensión del espacio es generar un conjunto sintético uniformemente distribuido en el cubo unitario, y usar este conjunto como un espacio métrico abstracto.
La función de distancia utilizada con estos conjuntos de datos fue la distancia Euclidiana.

Otro tipo de espacio métrico utilizado es un espacio métrico formado por una colección de 14.051 imágenes. Dichas imágenes de la colección se extrajeron de Internet *(the Color FERET image database)* y se transformó cada una en un vector de características de dimensión 25, utilizando la distancia euclidea para medir la similitud entre ellas [JAI 5].

Estos dos espacios métricos están explicados más en detalle en la tercer parte de este documento, al finalizar el análisis y las conclusiones de nuestra propuesta.

Primeras experimentaciones

Como primeras experimentaciones, y para realizar unas buenas pruebas de concepto de nuestra propuesta sobre un espacio métrico sintético de dimensión 2, se ha realizado la implementación de los siguientes algoritmos:

a) *Aleatorios*
b) *SSS*
c) *SSS + Propuesta*

Antes de continuar, definamos los valores utilizados para esta experimentación inicial, implementada en el lenguaje Java (*versión jdk 1.6*):

1) Espacio Métrico: Espacio Vectorial sintético, números aleatorios entre -1 y 1.
2) Dimensión: 2 (dos), ideal para 'visualizar' el comportamiento.
3) Cantidad de elementos en el espacio métrico: 10.000 (diez mil)
4) Cantidad de elementos consulta: 1.000 (mil), radio de búsqueda 2 (dos).
5) Cantidad de 'épocas' de ejecución: 20 (veinte) épocas.

Una *época* es la ejecución de todas las consultas contra la base de datos. Luego de cada una de ellas se realiza un intercambio de los pivotes que forman parte del índice, según la política de cada implementación, a excepción de *SSS y Aleatorios*, donde se genera todo el índice nuevamente.

Ahora ampliamos brevemente que significan cada uno de esos tres algoritmos utilizados en la primera etapa de experimentación, donde gráficamente, se quiere mostrar como, en nuestra propuesta, los pivotes se van adaptando a las búsquedas focalizadas en una zona puntual, y compararlo con otros algoritmos:

a) *Aleatorios (Random)*: En esta implementación tomamos un número fijo de pivotes que formaran parte del índice, 3 (tres) elegidos al azar en el momento de construcción del índice, ya con la base de datos poblada de los elementos. Al trabajar con el índice para intercambiar pivotes se utilizó *Política Random* para la selección Inicial de Pivotes, selección Posterior de Pivotes, selección del Pivote Entrante al índice y selección del Pivote Saliente del índice.

b) *SSS*: Es la implementación del *Sparse Spatial Selection* (SSS), explicado en [Pedreira, 2007]

c) *SSS + Propuesta*: Implementación de nuestra propuesta de trabajo explicada anteriormente, donde se utilizan las dos nuevas políticas de selección que adaptan el índice.

Sobre las políticas

Se utilizó 3 técnicas para elegir el elemento entrante al índice que ya esta construido, según presentamos antes:

1. Aleatoria.
2. Política del SSS
3. Más candidato.

Aleatoria es la primera y más simple, ya que tenemos que elegir un elemento del conjunto: $\{e \in DB \mid e \notin Pivotes\}$, es decir, un elemento que no sea ya pivote de nuestro índice.
La segunda es la política que propone el SSS, utilizando la misma idea que al momento de crear el índice, evaluando la distancia del elemento contra los pivotes y una constante.
La tercera, utiliza un criterio muy similar a la elección del pivote víctima y que presentamos en la sección anterior.

Experimentación

Se comienza el trabajo de la experimentación, implementando el primero de los algoritmos, es decir se trabaja sobre *Random* que utiliza las políticas aleatorias, considerando para la experimentación:

a) Construcción del índice: Se define un número fijo de pivotes de modo aleatorio, tres en este caso, y se construye el índice, calculando la distancia de los pivotes a todos los elementos de la base de datos, pero esto se realiza una vez que ya están todos los elementos dentro de la *DB*, y no se realizan inserciones una vez construido el índice.

b) Búsquedas: se realizan las búsquedas de todos elementos que forman parte del conjunto de entrenamiento. A esto se lo llama una época. Dicho conjunto de entrenamiento tiene un total de 1.000 (mil) elementos, y las búsquedas tendrán un radio de búsqueda igual a 2 (dos).

c) Se ejecutan 20 épocas, al finalizar cada época, se cambian los pivotes de los índices, aplicando las siguientes políticas:

1. *Pivote saliente*: Se toma un pivote del índice, de forma aleatoria, sin evaluar ningún valor.
2. *Pivote entrante*: Se toma un elemento de la base de datos, cuya única condición a cumplir es que no sea ya un pivote dentro del índice.

Finalizadas las 20 épocas, gracias a que estamos experimentando en un espacio métrico de dimensión 2, se puede graficar la ubicación de los elementos del espacio métrico, el conjunto de entrenamiento y los 3 pivotes que forman parte del índice. Obteniendo los siguientes resultados:

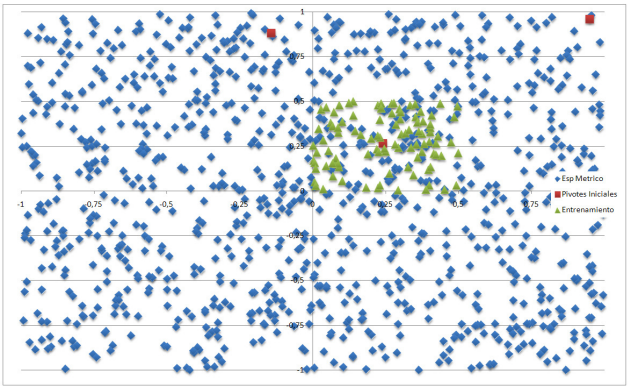


Figura 16: Etapa 1

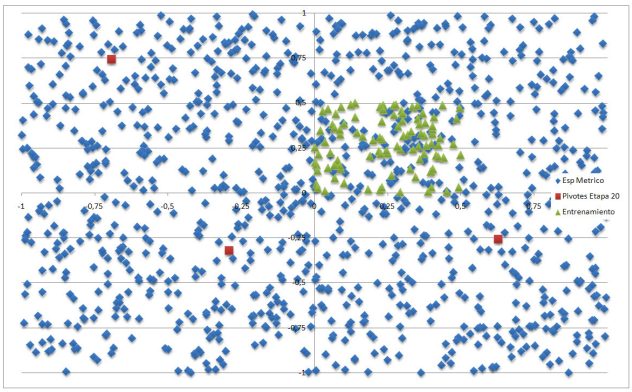


Figura 17: Etapa 20

En el Apéndice A.1 se encuentran los resultados de las 20 etapas, donde se reflejan las distintas posiciones de los pivotes que forman parte del índice.

Como podemos apreciar en las imágenes, ir tomando de forma aleatoria los pivotes iniciales que conforman el índice, junto con los pivotes entrantes y salientes al momento de realizar las actualizaciones del índice no es una gran política, ya que encontramos que los elementos que conforman el índices no estarán algo distantes entre sí, algo que muchos autores señalan como una característica deseable del conjunto de pivotes. Y muchas veces, dichos pivotes quedan muy cerca entre sí o concentrados en zonas pocas pobladas de la base de datos.

Continuamos nuestro proceso de implementación de los tres algoritmos que mencionamos anteriormente, y es el momento de implementar *Sparse Spatial Selection* (SSS), utilizando la información brindada en [Pedreira, 2007] y los mismos parámetros antes enumerados.
En este caso, en cada época reiniciamos todo el índice de la base de datos, porque SSS no realiza intercambio de pivotes, lo realmente interesante de esta propuesta es que la cantidad de pivotes está en función de la dimensión del espacio, y no de la cantidad de elementos insertados.
Entonces, al ejecutar las 20 épocas, vamos a analizar la cantidad de pivotes y su disposición en el espacio, utilizando nuevamente el conjunto de entrenamiento focalizado en una zona determinada, y un radio de búsqueda como antes nombramos.

Finalizada la experimentación, utilizando los mismos valores del apartado anterior, se grafica la ubicación de los elementos del espacio métrico, el conjunto de entrenamiento y los pivotes que forman parte del índice, prestando especial atención a la cantidad de pivotes. Se obtienen los siguientes resultados:

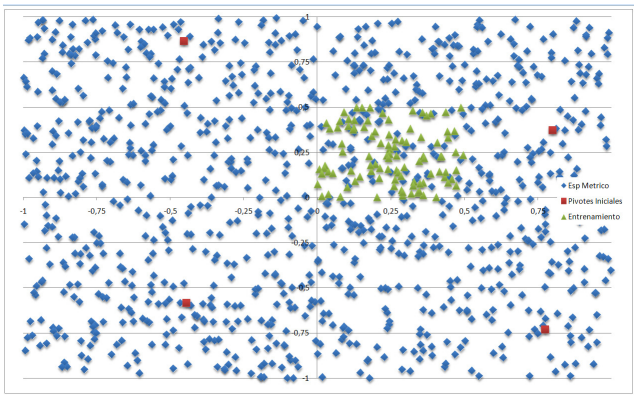


Figura 18: Etapa 1 SSS

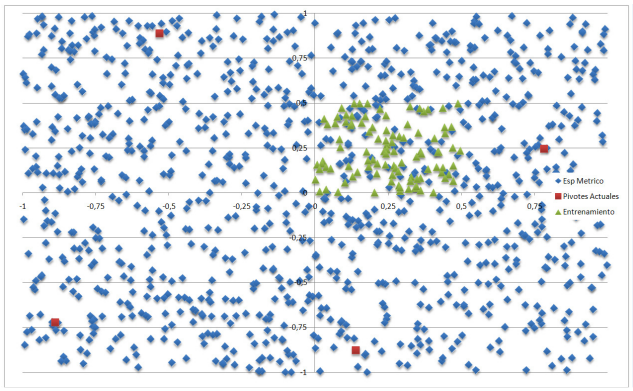


Figura 19 : Etapa 20 SSS

En el Apéndice A.2 se encuentran los resultados de las 20 etapas, donde se reflejan la cantidad de pivotes en el índice y las distintas posiciones de estos en el espacio métrico.

Como podemos apreciar en las imágenes, aquí la cantidad de pivotes durante todas las épocas se mantiene constante en 4 elementos, ya que la dimensión del espacio métrico no varía (mas adelante, analizamos este comportamiento). Lo que si observamos es una leve diferencia entre las ubicaciones de los pivotes, pero siempre respetando una cierta distancia, y encontrándose algo distantes entre sí, una característica deseable del conjunto de pivotes, como ya hemos presentado anteriormente. Una observación importante, es que los pivotes seleccionados cubren una buena cantidad de elementos del espacio métrico.

Por último, esta es la experimentación y testeo inicial de nuestra propuesta, utilizando las siguientes políticas ya mencionadas anteriormente, y las mismas condiciones de trabajo que los dos algoritmos anteriores:

1) Construcción del índice: Idéntica a la propuesta por SSS, sobre una base de datos inicialmente vacía, vamos insertando elementos y decidimos si es un nuevo pivote del índice o no. La cantidad de pivotes está en función de la dimensión del Espacio Métrico, y no en función de la cantidad de elementos insertados. Para la constante α tomamos el valor 0.5, y su justificación será dada más adelante en este trabajo.

2) Pivote Saliente: Analizando las estadísticas registradas durante el uso del índice, seleccionado el pivote que menos discrimina, es decir, se elimina del índice a aquel pivote que menos elementos discrimine para todas las búsquedas de esa época.

3) Elemento candidato a ser Pivote: Nuevamente se analizan las estadísticas, pero en este caso, prestando especial atención a los elemento de la base de datos, y seleccionado a aquel que más veces apareció en los resultados de búsqueda. Este elemento será nuestro candidato a ser pivote entrante del índice.

4) Al finalizar cada época, una vez que se intercambiaron los pivotes, se genera nuevamente el índice, para luego volver a ejecutar las búsquedas de entrenamiento, utilizando el conjunto creado para dicho fin.

Cuando estamos aplicando estos puntos, por el momento y a modo de experimentación, no vamos a permitir que se realicen inserciones, modificaciones o eliminación de los elementos pertenecientes a la base de datos, con el único objetivo de poder controlar y comprar las operaciones.

Finalizada la experimentación, se grafica de forma similar a las dos implementaciones anteriores, concentrándonos en la ubicación de los elementos del espacio métrico, el conjunto de entrenamiento y los pivotes que forman parte del índice. Obteniendo los siguientes resultados:

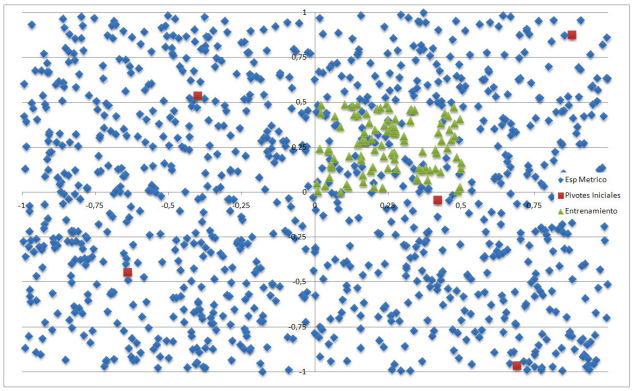


Figura 20: Época 1 Propuesta

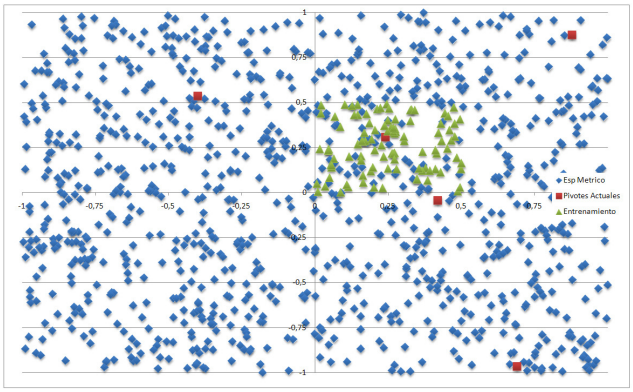


Figura 21: Época 2 Propuesta

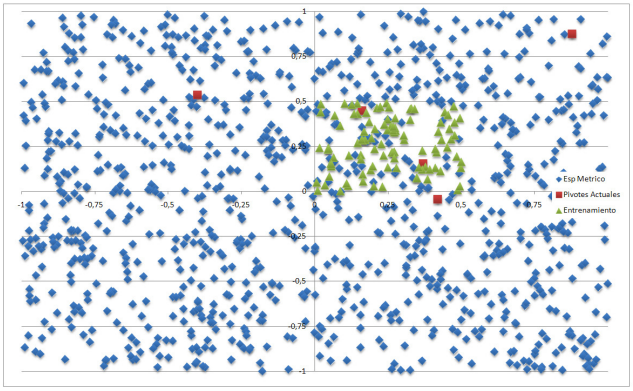


Figura 22: Época 5 Propuesta

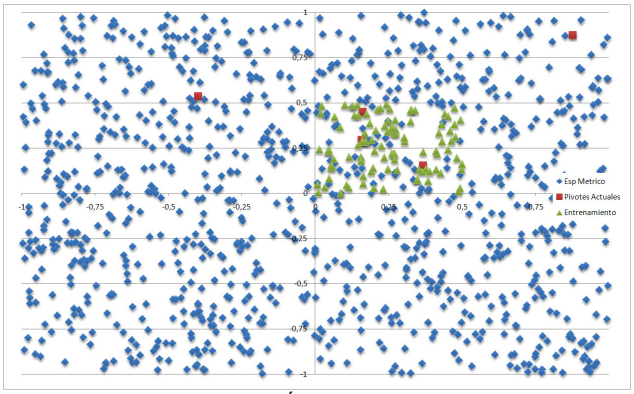


Figura 23: Época 6 Propuesta

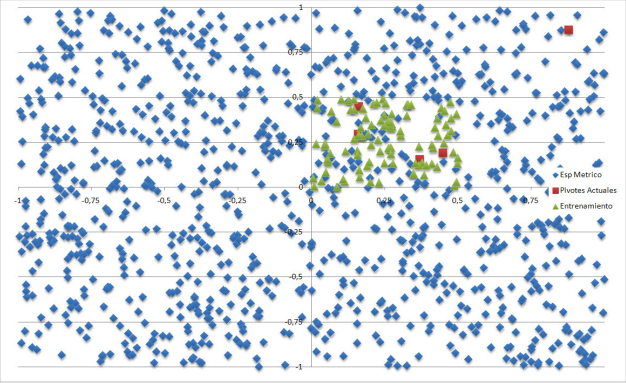


Figura 24: Época 13 Propuesta

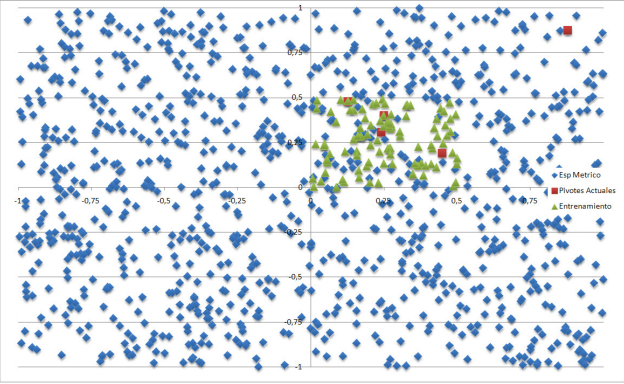


Figura 25: Época 20 Propuesta

En el Apéndice A.3 se encuentran los resultados de las 20 etapas, donde se reflejan las distintas variaciones de los pivotes en el índice y las distintas posiciones de estos en el espacio métrico, mostrando cómo se adaptan a las búsquedas.

Como podemos apreciar en las imágenes, aquí la cantidad de pivotes durante todas las épocas se mantiene constante en 5 elementos. Pero lo principal que observamos es la gran diferencia entre las ubicaciones de los pivotes de las primeras 2 épocas con respecto a las épocas posteriores a las número 6, donde los pivotes del índice se van *adaptando a la zona de búsqueda*, dando una prioridad a los elementos de dicha zona para lograr descartar más elementos.
En las dos primeras épocas podemos destacar cómo se *adaptan rápidamente* los pivotes a la zona de búsqueda, teniendo finalizada la época 2, en cercanías de la zona de búsquedas, ya tenemos 2 pivotes del índice ahí. Luego, en las siguientes iteraciones, van oscilando los pivotes dentro de esa zona hasta la época 5, en donde un nuevo pivote se aproxima a dicha zona de búsquedas. Aquí nuevamente, los pivotes van intercambiándose de posiciones con el correr de las búsquedas, pero siempre en las cercanías de esa zona, hasta finalizada la época 13, momento en el cual un nuevo pivote se suma a ella, y es aquí donde los pivotes sigue intercambiándose por posiciones similares, pero siempre en zonas próximas a las de las búsquedas efectuadas.
Con esta experimentación queremos demostrar cómo funciona la propuesta de este trabajo, en contraste con la más simple de todas las políticas de selección de pivotes y con el método SSS, que es el más novedoso y con un muy buen desempeño, según lo visto en el estado del arte.

Más Experimentación y Análisis de Resultados

Ya tenemos hasta este momento una implementación estable de nuestra propuesta de trabajo, y de dos algoritmos más que utilizaremos de referencia para comparar.

Ahora, en esta sección mostraremos el desempeño de nuestra técnica en una base de datos real y en una de elementos sintéticos. La base de datos real es un conjunto de 14.051 imágenes descargadas desde Internet [JAI 5] y la base de datos sintética fueron vectores en el cubo unitario.

En particular, nos interesa mostrar la cantidad de evaluaciones de la función distancia que son necesarias para responder una consulta usando nuestra técnica basada en las políticas presentadas anteriormente.
También es importante mostrar el costo inicial de construcción, cómo se va adaptando el índice a medida que se van procesando las consultas y la cantidad de discriminaciones que realizan los pivotes que conforman el índice.
Otro experimento interesante en el estudio de la técnica aquí propuesta es cómo afecta la dimensión de los datos al desempeño del índice. Para este nuevo experimento se usaron conjuntos sintéticos distribuidos en el cubo unitario. El rango de dimensiones fue $8 \leq dim \leq 14$. El tamaño de la base de datos fue de 100.000 objetos y las consultas recuperaron el 0,02% de la base de datos en consultas por rango.

Experimento 2: Performance y Análisis de Resultados

Costo de construcción del Índice

Aquí se experimenta con el espacio métrico sintético, buscando evaluar el costo de construcción del índice, teniendo en cuenta qué, a diferencia de otros métodos propuestos, el aquí presentado es dinámico, entonces este costo de construcción obtenido no es el definitivo.
Entonces, experimentaremos con los siguientes valores:

* 100.000 de elementos que se insertaran como máximo en la base de datos.
* k = {8, 10, 12, 14} son todas las dimensiones con las que experimentaremos.
* α = 0,5 valor que fijamos a mano dentro de la propuesta, y que se justifica más adelante en este trabajo.

Se ejecuta el experimento tanto con la implementación de nuestra propuesta, como con SSS para realizar un análisis comparativo, y se obtienen los resultados de la Tabla 1 y la grafica de la figura 26:

Tabla 1: Costo de construcción del índice en espacios vectoriales de dimensión 8, 10, 12 y 14**.**

	Costo de construcción			
Método	**8**	**10**	**12**	**14**
SSS	19898	27837	34426	119320
Propuesta	14744	27607	33739	33660

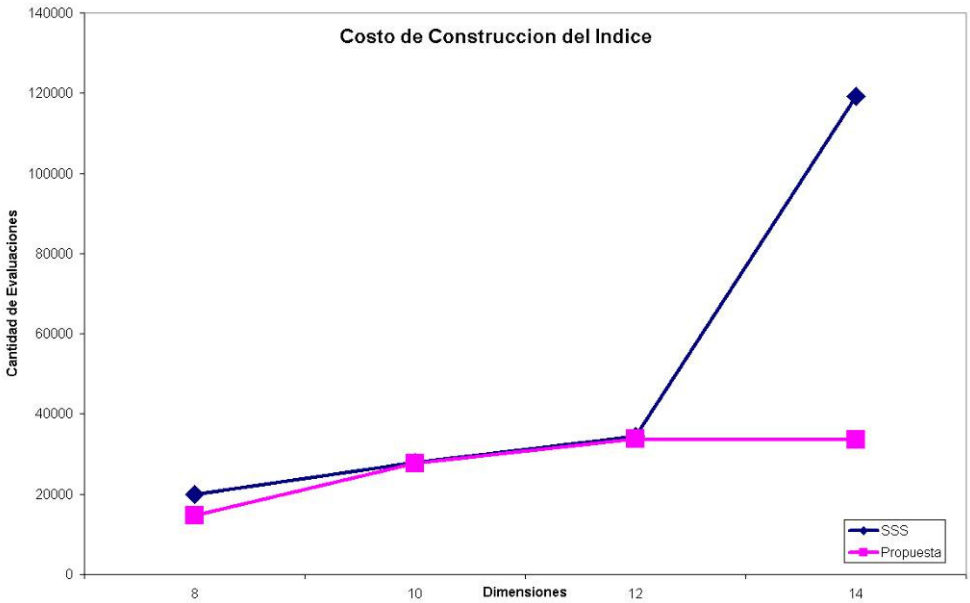


Figura 26: Costo de construcción según la dimensión.

Según se observa en la grafica, en todo momento se mantiene el costo de construcción del índice de la propuesta de este trabajo por debajo de la implementación de SSS. Este resultado es interesante, ya que es

muy importantes la cantidad de evaluaciones de la función distancia al momento de la construcción del índice, y obtener una menor cantidad es obtener un valor de costo menor, lo que genera una interesante mejora al momento de la construcción inicial.

Cantidad de Pivotes en el Índice

A continuación, se quiere demostrar que la propuesta aquí presentada genera de forma dinámica un número de pivotes, que está en función de la dimensionalidad del espacio, y no del número de objetos que conforman la base de datos.
Se experimento con los siguientes valores:

* 100.000 de elementos que se insertaran como máximo en la base de datos.
* K = {8, 10, 12, 14} son todas las dimensiones con las que experimentaremos.
* $\alpha = 0{,}5$ valor fijado intencionalmente dentro de la propuesta.

En la Tabla 2 y en la Figura 27 se muestran la cantidad de pivotes en función de la dimensión y del número de elementos.

Tabla 2: Número de pivotes seleccionados en espacios vectoriales de dimensión 8, 10, 12 y 14**.**

	n, tamaño de la colección (x 10^3)									
dim	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
8	11	12	12	12	13	13	14	14	15	16
10	13	18	20	20	21	21	21	21	21	22
12	25	28	28	31	32	34	34	34	34	34
14	38	47	53	60	60	61	63	66	69	69

Veamos esta información de forma grafica para analizar el comportamiento del índice, a medida que se le insertan elementos, según las distintas dimensiones:

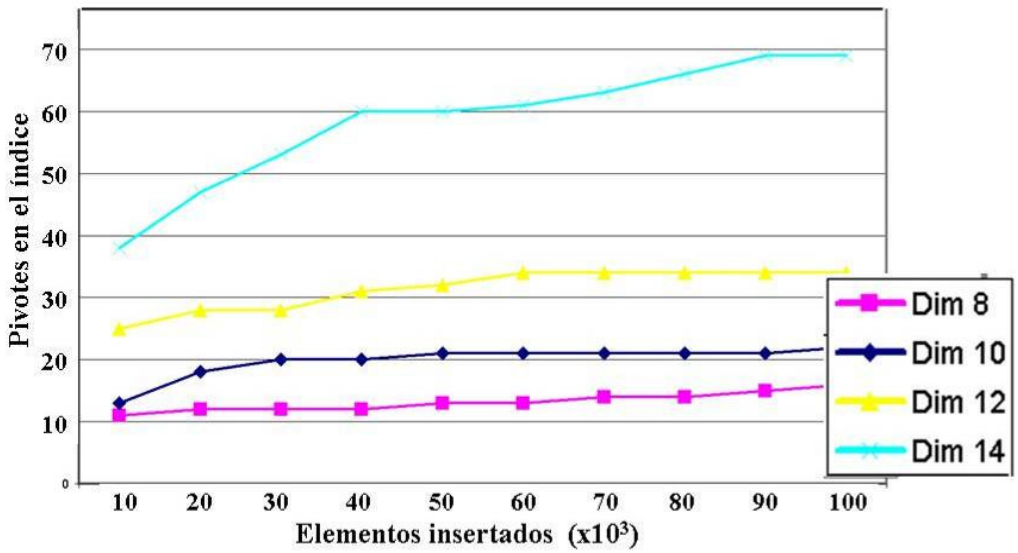


Figura 27: Cantidad de pivotes generados en forma dinámica según la dimensión.

Como primera observación, encontramos que el número de pivotes crece muy rápidamente con las inserciones de los primeros objetos de la colección, luego continua creciendo pero ya en un grado más lento hasta llegar a estabilizarse. Es decir, cuando tenemos pocos elementos insertados, la cantidad de pivotes en el índice sí depende del número de elementos en la base de datos. Ahora, una vez que tengo un número considerable de elementos dentro, al realizar nuevas inserciones no se seleccionarán más pivotes, porque con el conjunto de pivotes actuales ya tenemos cubierto todo el espacio métrico.

Luego claramente se destaca que el número de pivotes en el índice aumenta a medida que aumenta la dimensión del espacio métrico, prestemos especial atención en la diferencia entre la dimensión 12 y la 14. Gráficamente, lo que refleja es que el número de pivotes que forman el índice depende de la complejidad de la colección de objetos.

Al igual que ocurre en la experimentación de SSS, aquí tenemos que el número de pivotes en el índice efectivamente está en función de la dimensión del espacio métrico y no de la cantidad de elementos insertados dentro de la base de datos.

Y así se demuestra, gracias a la experimentación, que el número de pivotes está en función de la complejidad del espacio métrico y no del tamaño.

Análisis de eficiencia en búsqueda

Para esta evaluación, se experimenta con 10.000 objetos dentro de la base de datos, 1.000 consultas y las dimensiones 8, 10, 12 y 14. Se ejecutan 20 épocas, y luego se promedia la información de todas las épocas.
Por cada una de las dimensiones se registran:

#DE = Cantidad funciones distancia evaluadas.
#P = Cantidad de pivotes utilizados en el índice.
#DR = Cantidad de discriminaciones realizadas.

evaluando el método propuesto contra SSS.

En particular, nos interesa mostrar la cantidad de pivotes utilizados en el índice, las comparaciones de distancia necesarias (internas y externas), para responder una consulta usando nuestra técnica propuesta y analizar la cantidad de discriminaciones realizadas en dichas consultas.

Ejecutamos esta experimentación recién presentada con el método de nuestra propuesta, y a continuación lo realizamos con 'SSS', y tabulamos la información obteniendo la Tabla 3:

Tabla 3: Eficiencia en espacios métricos sintéticos con distintos métodos.

	***dim* = 8**			***dim* = 10**			***dim* = 12**			***dim* = 14**		
Método	**#P**	**#DE**	**#DR**	**#P**	**#DE**	**#DR**	**#P**	**#DE**	**#DR**	**#P**	**#DE**	**#DR**
SSS	17	17994	6141634	24	26391	6393097	34	35721	6623490	60	62976	6915951
Propuesta	15	15737	6394202	23	24380	6657667	33	34686	6717067	44	45683	7025895

La información de la Tabla 3 muestra que tenemos el número de pivotes utilizados en el índice de nuestra propuesta siempre es menor que en la implementación de SSS, destacándose una marcada diferencia en la mayor de las dimensiones, con 16 pivotes menos. En las restantes dimensiones es muy poca la diferencia, siendo de a lo sumo 2 pivotes menos en el índice de nuestra propuesta.

Graficando esta información de la columna #P en la tabla 3 obtengo el siguiente resultado:

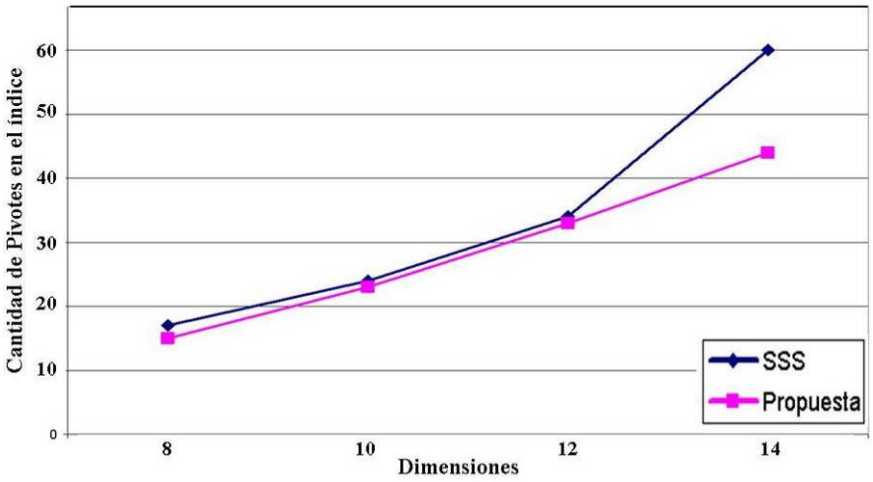


Figura 28: Número de pivotes generados, en función de la dimensión 8, 10, 12 y 14.

Como podemos observar en el grafico, tenemos en nuestra propuesta un número menor de pivotes en el índice, que es un resultado muy importante, ya que la estrategia de selección de pivotes SSS presenta una eficiencia similar a otras más complejas y el número de pivotes que selecciona está cercano al número optimo de pivotes para otras estrategias de selección de pivotes.

Si continuamos analizando los resultados tabulados de este experimento, podemos analizar la cantidad funciones distancias evaluadas según la dimensionalidad. En este caso, nuevamente, vamos a comparar los resultados obtenidos para la implementación de nuestra propuesta contra los obtenidos para SSS.

Graficando dicha información de la columna #DE en la Tabla 3, obtengo el siguiente resultado:

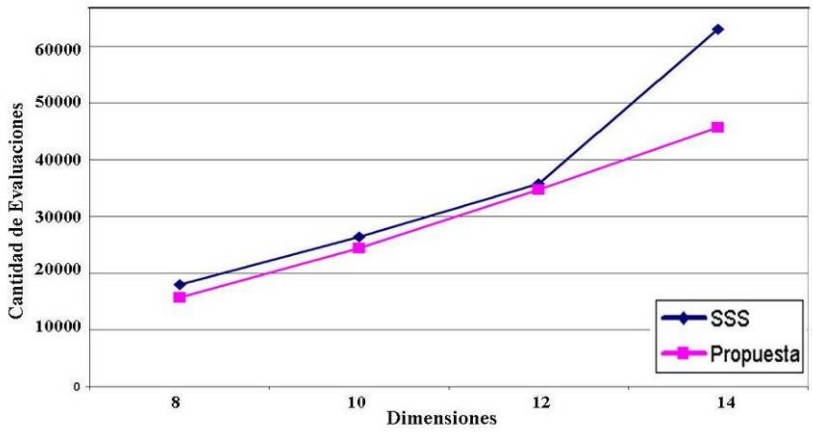


Figura 29: Funciones distancia evaluadas al realizar búsquedas, según dimensiones 8, 10, 12 y 14.

Mirando este grafico claramente vemos que la cantidad de evaluaciones de la función distancia, al momento de realizar las búsquedas, siempre se mantiene por debajo del número de evaluaciones con el método SSS, y en general tiene un crecimiento lineal uniforme al aumentar las dimensiones. Excepto para *dim*=14, donde SSS muestra un leve crecimiento con la cantidad de evaluaciones, con una diferencia de aproximadamente 17.000 evaluaciones, en las demás dimensiones la diferencia nunca supera las 3.000.

Según resultados expuestos en [Pedreira, 2007], el número de evaluaciones de la función distancia en SSS está siempre en torno al mejor resultado obtenido con las técnicas de selección de pivotes y estrategias propuestas en [Bustos, 2001].
Entonces, podemos concluir que la propuesta aquí presentada, al momento de las búsquedas, realiza una cantidad de evaluaciones de la función distancia similar a los mejores resultados obtenidos en trabajos anteriores, utilizando incluso un menor número de pivotes en el índice.

Volviendo a la Tabla 3 presentada anteriormente, observamos la cantidad de discriminaciones realizadas por los pivotes del índice al momento de realizar las búsquedas.

Nuevamente, vamos a comparar los resultados de nuestra propuesta, con los resultados del método SSS, y graficamos dicha información para obtener resultado de la figura 30:

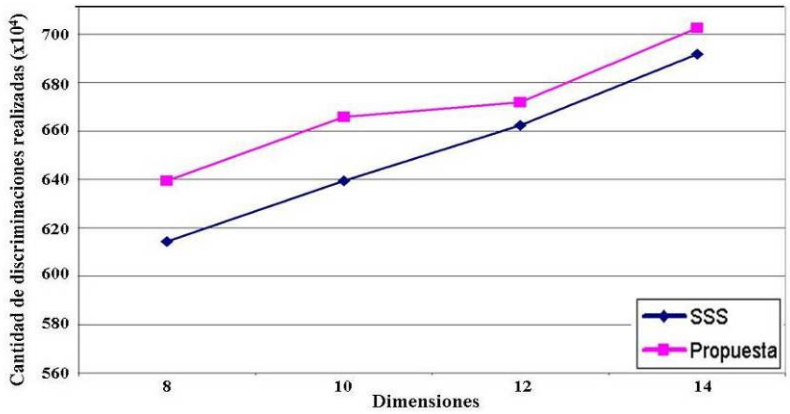


Figura 30: Discriminaciones realizadas por los pivotes, según dimensiones 8, 10, 12 y 14.

Aquí podemos observar como utilizando nuestra propuesta, al momento de las búsquedas, obtenemos un número mayor de discriminaciones por parte de los pivotes que integran el índice, en todas las dimensiones donde se experimentó.
Es en las dos primeras dimensiones donde se refleja una mayor diferencia entre las cantidades de discriminaciones realizadas, mostrando un gran rendimiento los pivotes seleccionados con nuestra propuesta para descartar elementos en las búsquedas, en contraste con los pivotes utilizados en SSS.

Esto es gracias a que con el pasar de las épocas, nuestra propuesta realiza una adaptación de los pivotes, y estos realizan mejores discriminaciones, para con esto reducir la cantidad de evaluaciones de la función distancia y bajar la complejidad al momento de las consultas.

Con este resultado demostramos que es buena la política de selección de pivotes entrantes al índice aquí presentada, y la política de selección del pivote saliente del índice, ya que el comportamiento obtenido de los pivotes en el índice a lo largo del tiempo es mucho mejor que el de resultados anteriores. Además, le damos un carácter dinámico al índice, que claramente tiene sus buenos resultados al momento de realizar las discriminaciones para minimizar la cantidad de evaluaciones de la función distancia.

Experimento 3: sobre el parámetro α

El valor de α condiciona el número de pivotes, y se recomiendan valores entre 0.35 y 0.40 [Paredes, 2002]. Aquí se decide experimentar con valores de α desde 0.32 a 0.54 (avanzando de 0.02) y evaluar la cantidad de pivotes en el índice, el costo de construcción del índice, la cantidad de funciones distancias calculadas y las discriminaciones realizadas por el conjunto de pivotes en el índice. Realizamos este experimento ya que un aumento de α supone una reducción del número de pivotes y esto se nota más en espacios de dimensión mayor.
Se experimenta con las dimensiones 2, 8, 10, 12 y 14, se utilizan en DB 10.000 elementos que conforman el espacio métrico y 1000 elementos de entrenamiento, y se ejecutan 20 épocas. Los resultados obtenidos están tabulados en la Tabla 4.1 - 4.2, y a continuación se explican mejor los experimentos y las graficas asociadas.

Tabla 4.1: Análisis del parámetro α en espacios métricos sintéticos con valores para α desde 0.32 a 0.54.

	Dim 2				Dim 8				Dim 10			
	#P	#DR	#DE	#CI	#P	#DR	#DE	#CI	#P	#DR	#DE	#CI
0,32	35	3963844	7885	17701	441	6607258	144726	194528	953	6928440	291004	380083
0,34	17	3819850	4876	17526	203	6542018	107810	136194	402	6829934	230405	308090
0,36	20	3857351	4891	17825	265	6489169	85066	101320	563	6857041	191865	237293
0,38	7	3829726	4835	16707	86	6368616	62889	57775	141	6787097	131657	133148
0,40	29	3895538	4805	16092	322	6438158	56111	62221	700	6752306	96595	91898
0,42	25	3798645	3833	16650	286	6382819	43800	55995	616	6686642	68068	61367
0,44	11	3875181	4703	14050	102	6153472	35737	34731	193	6614613	62801	56012
0,46	13	3837849	4527	10537	108	6073633	25567	31333	208	6377237	35869	37371
0,48	28	3818781	4701	14028	288	6040610	22165	23295	623	6299942	25203	24056
0,50	22	3158759	2603	12057	263	6088157	18132	22649	570	6294392	26264	25274
0,52	14	3834358	3661	13227	111	5798994	12834	16678	206	6176715	16960	19193
0,54	3	3846199	3507	10141	39	5544585	7624	12481	16	6658126	16887	17732

Tabla 4.2: continuación del análisis de α en espacios métricos sintéticos con valores para α desde 0.32 a 0.54.

	Dim 12				Dim 14			
	#P	#DR	#DE	#CI	#P	#DR	#DE	#CI
0,32	3831	7205660	742113	1262253	5790	7451496	1475606	2952121
0,34	1453	7157664	466113	682250	4564	7375046	905760	1515202
0,36	2021	7087385	336165	423296	4860	7301443	635154	963078
0,38	406	7025236	245548	290951	4046	7231941	413451	469010
0,40	2522	6980621	148421	148422	5129	7249466	269638	292755
0,42	2235	6884732	128139	122789	4977	7183688	213771	235428
0,44	537	6813767	72241	64810	4096	6999077	152973	159451
0,46	617	6822758	69053	61060	4143	7049361	102450	68992
0,48	2285	6683382	49888	37768	5006	6924489	71451	69026
0,50	2051	6515198	32712	34243	4873	6792644	51845	56905
0,52	641	6570080	25031	20614	4151	6707140	43678	33553
0,54	22	6277046	21712	14246	3833	6911107	22852	17044

a) Número de pivotes generados

Esta experimentación nos sirve para demostrar por qué utilizamos el valor de la constante α en 0.5. Según observamos en la grafica de la Figura 31 para todas las dimensiones, la cantidad de pivotes varía según el entorno al valor de α, registrándose algunos máximos y mínimos locales, y grandes amplitudes en las dimensiones mayores. Luego del valor 0.50 se observa decrecer la cantidad de pivotes, según se esperaba.

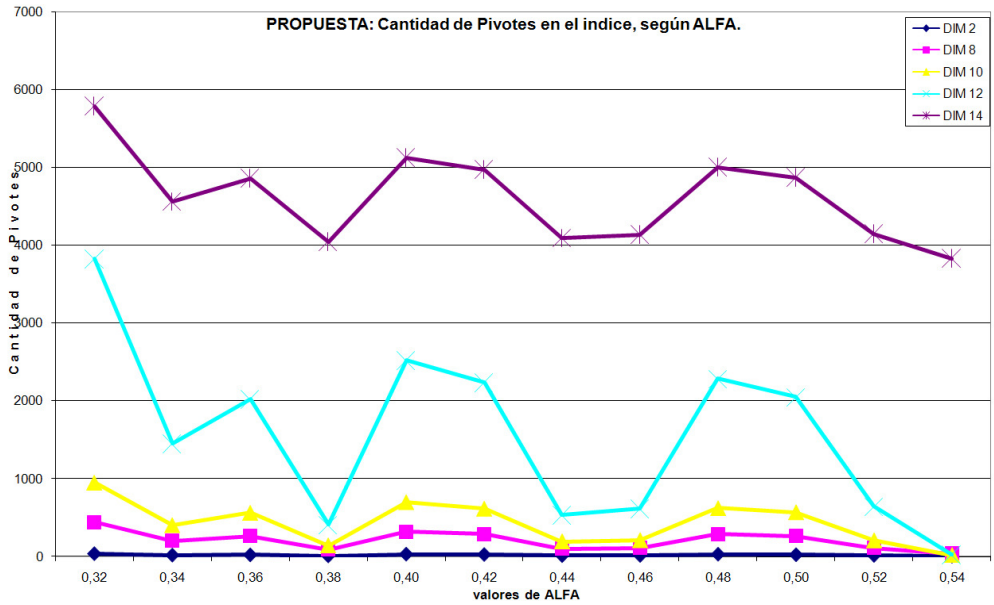


Figura 31: Cantidad de pivotes seleccionados según el parámetro α.

b) Cantidad de evaluaciones de la función distancia

Nos interesa evaluar el impacto de la constante α al momento de las búsquedas en nuestra propuesta y la cantidad de veces que se evalúa la función distancia, es por eso que realizamos el experimento que arroja sus resultados en el grafico de la Figura 32 y en la Tabla 4, antes mostrada.

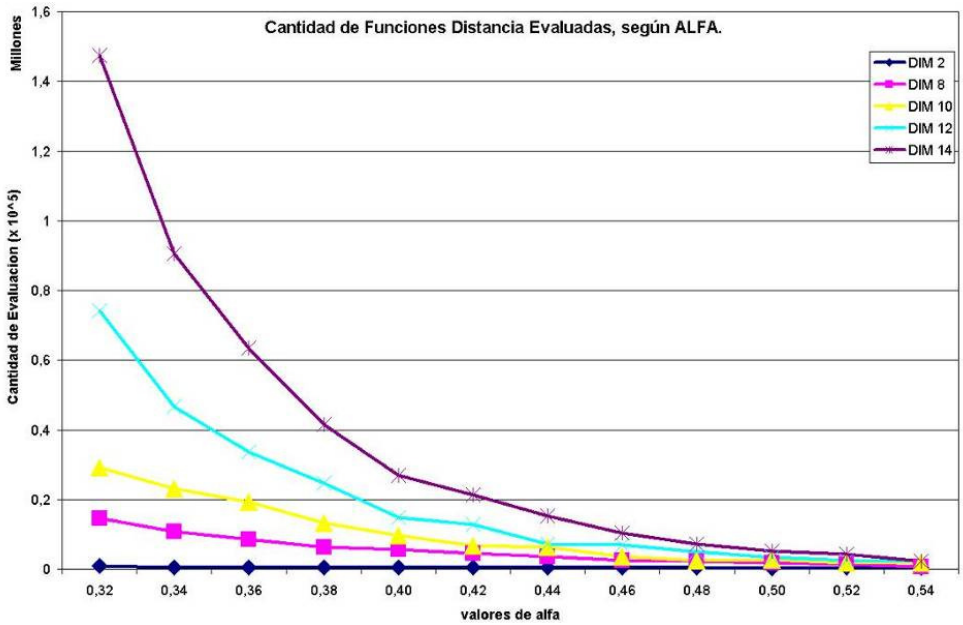


Figura 32: Evaluaciones de la función distancia según el parámetro α**.**

La Figura 32 muestra la cantidad de evaluaciones de distancia realizadas, variando α. En todas las dimensiones se observa un comportamiento uniforme: la cantidad de evaluaciones desciende a medida que aumenta α, ya que al aumentar la distancia entre los pivotes disminuyen las evaluaciones necesarias de la función distancia. Este valor tiene un comportamiento uniforme debido a que se promedian las 20 épocas. En las primeras épocas se encuentran la mayor cantidad de evaluaciones y con el pasar de las épocas los pivotes se van adaptando a las búsquedas. Además se logran más discriminaciones a medida que aumenta α, porque con el pasar de las épocas, en nuestra propuesta, se adaptan los pivotes mejorando las búsquedas, discriminando más elementos y disminuyendo las evaluaciones de la distancia de forma acelerada, como se esperaba.

c) Cantidad de discriminaciones realizadas

En este trabajo se continúa evaluando el impacto de la constante α al momento de las búsquedas, en este caso, sobre la cantidad de discriminaciones, ya que a mayor valor de α, es menor la cantidad de pivotes en el índice, y resulta interesante analizar la cantidad de discriminaciones.
Se realiza el experimento bajo las mismas condiciones ya enumeradas, y se obtiene los resultados de la Tabla 4, resultados que son graficados en la Figura 33.

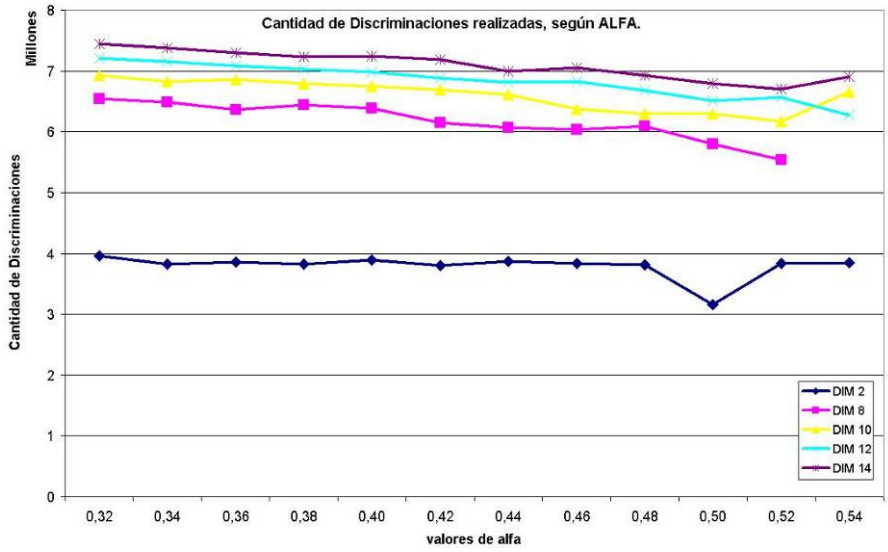


Figura 33: Cantidad de discriminaciones realizadas según el parámetro α.

Como se observa en la Figura 33, la cantidad de discriminaciones realizadas por los pivotes en el índice decrece muy suavemente a medida que aumenta el valor del parámetro α, este comportamiento de nuestra propuesta es idéntico para todas las dimensiones trabajadas, excepto para la menor de las dimensiones, *dim=2*, en donde se encuentra un fuerte mínimo en torno al 0,50. También, en todas las dimensiones, pasado el valor 0.48 se observa una leve disminución.

d) Costo de construcción del índice

Durante la experimentación nos interesa evaluar el impacto de la constante α al momento de la construcción del índice en nuestra propuesta, pero siempre considerando que es una estructura dinámica la que aquí se presenta y este costo no es un valor definitivo.
Se analiza la cantidad de veces que se evalúa la función distancia al momento de la construcción del índice, que arroja sus resultados en el grafico de la Figura 34 y en la Tabla 4, antes mostrada.

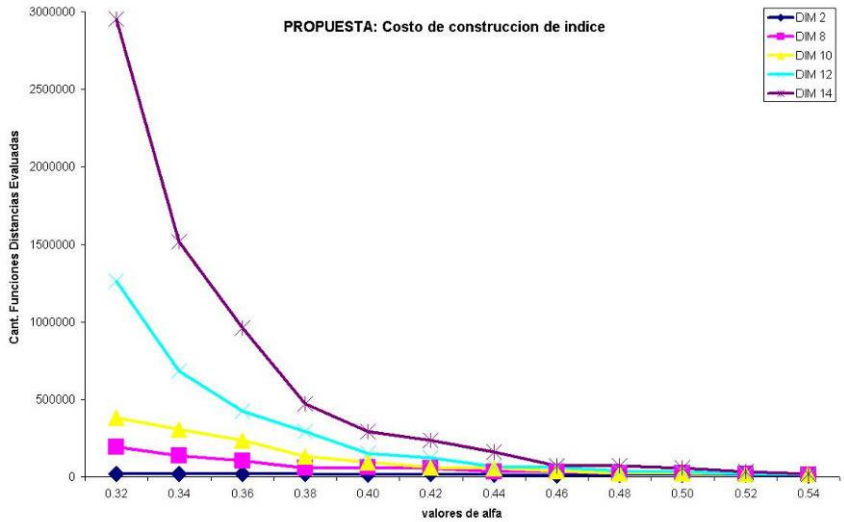


Figura 34: Costo de construcción según el parámetro α.

Claramente podemos destacar un comportamiento uniforme en todas las dimensiones según el parámetro α, en donde al aumentar ese parámetro desciende velozmente la cantidad de funciones distancias evaluadas al momento de construir el índice, según se esperaba gracias a la importancia del parámetro α, que al aumentar dicho valor, disminuye la cantidad de pivotes dentro del índice, y son muchas menos las evaluaciones de la función distancia necesarias para construir el índice.

Experimento 4: Espacio Métrico de Imágenes

Se repite uno de los experimentos anteriores, pero esta vez con el espacio métrico de las imágenes, y utilizando una función distancia especifica del dominio, que se explica en detalle en uno de los apéndices.

La idea es evaluar la eficiencia del índice al momento de las búsquedas, analizando el costo de construcción, la cantidad de discriminaciones, la cantidad de evaluaciones de la función distancia y la cantidad de pivotes en el índice, a través de los experimentos similares a los anteriores.

A modo de ejemplo, veamos cómo se resuelve una consulta en el espacio métrico de imágenes que se explica de forma completa en el Apéndice B.
Dada una imagen como objeto consulta, y un radio de búsqueda *r*, nuestra base de datos nos retorna los elementos más cercanos a dicha imagen con ese radio *r*. Entonces en la Figura 34 tenemos del lado izquierdo a la imagen consulta, y del lado derecho se obtiene un ranking con las imágenes que son resultado de búsqueda, junto con su nombre y la distancia a la consulta. En el apéndice C se muestran los resultados de esa búsqueda, con un radio de búsqueda menor o igual a 10.

Como se puede observar en la Figura 35, es una muy buena performance la que tiene la implementación, ya que en las 4 primeras imágenes del lado derecho obtenemos coincidencia casi perfecta y con una distancia entre las imágenes que es a lo sumo 4, lo que nos permite saber que estamos utilizando de forma optima la función distancia entre los elementos que definen el espacio métrico.
Luego, vemos que las siguientes imágenes resultantes son similares, pero ya encontramos diferencias importantes, una de ellas es que, es otra persona la que se encuentra en las imágenes. Pero tenemos pequeños detalles que siguen siendo importantes a la hora de realizar las comparaciones. Entre esos detalles, podemos nombrar la cantidad de pelo frente al rostro, el porcentaje de rostro en la imagen, la orientación de la cabeza de la persona e incluso que poseen camisa de color claro.

A continuación, para realizar la experimentación y realizar un análisis, se limito el espacio métrico a todas las imágenes en blanco y negro, para así obtener resultados más específicos.

Figura 35: Ranking del resultado de búsqueda

Para esta evaluación, se experimenta con 4.000 objetos dentro de la base de datos que forman el espacio métrico de imágenes reducido sólo a imágenes en blanco y negro, y se utilizan 400 objetos tomados de forma aleatoria como objetos consultas. Este espacio métrico tiene dimensión 25, y en el apéndice está detallada su forma de trabajo.
Se ejecutan 10 épocas, y luego se promedia la información de todas las épocas.
Nuevamente se registran:

#CI = Costo de construcción del índice.
#DE = Cantidad funciones distancia evaluadas.
#P = Cantidad de pivotes utilizados en el índice.
#DR = Cantidad de discriminaciones realizadas.

Aquí se decide experimentar con valores de α desde 0.32 a 0.54 (avanzando de 0.02) y evaluar el costo de construcción del índice, la cantidad de pivotes en el índice, la cantidad de funciones distancias calculadas y las discriminaciones realizadas por el conjunto de pivotes en el índice. Los resultados obtenidos están tabulados en la Tabla 5, y a continuación se explican mejor los experimentos y las graficas asociadas.

Tabla 5: Análisis del parámetro α en espacio métrico de imágenes con valores para α desde 0.32 a 0.54.

	#P	#DR	#DE	#CI
0,32	104	85372	37748	477491
0,34	73	67032	55158	335065
0,36	57	65010	56700	258794
0,38	42	46418	75011	191035
0,40	29	51561	68891	133278
0,42	27	46511	73857	121529
0,44	20	49117	71356	89279
0,46	15	51371	68854	68271
0,48	13	51716	67918	59560
0,50	10	38975	81546	47351
0,52	9	37017	79939	42662
0,54	7	30549	89456	34119

a) Costo de construcción del índice

Por lo ya visto en la experimentación anterior, el valor del parámetro α, condiciona la cantidad de pivotes en el índice y esto impacta en la cantidad de evaluaciones de la función distancia al momento de construir el índice. Se grafica la información de la columna #CI de la Tabla 5 y obteniendo la gráfica de la Figura 36.

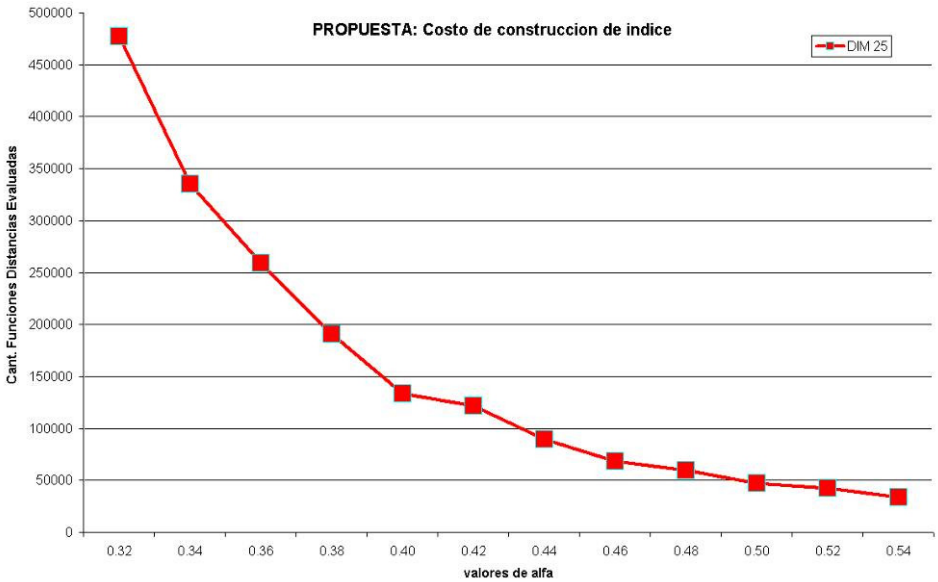


Figura 36: Costo de construcción de índice.

Observando la gráfica se verifica nuevamente que al aumentar el parámetro α desciende velozmente la cantidad de funciones distancias evaluadas al momento de construir el índice, según se esperaba gracias a la importancia del parámetro α, que al aumentar su valor, disminuye la cantidad de pivotes dentro del índice, y son muchas menos las evaluaciones de la función distancia necesarias para construir el índice.

b) Cantidad de pivotes en el índice

De forma similar a la experimentación en espacios métricos sintéticos, analizar la cantidad de pivotes en el índice nos sirve para demostrar porqué utilizamos el valor de la constante α en 0.5. Graficamos los resultados obtenidos de la Tabla 5 en la Figura 37.

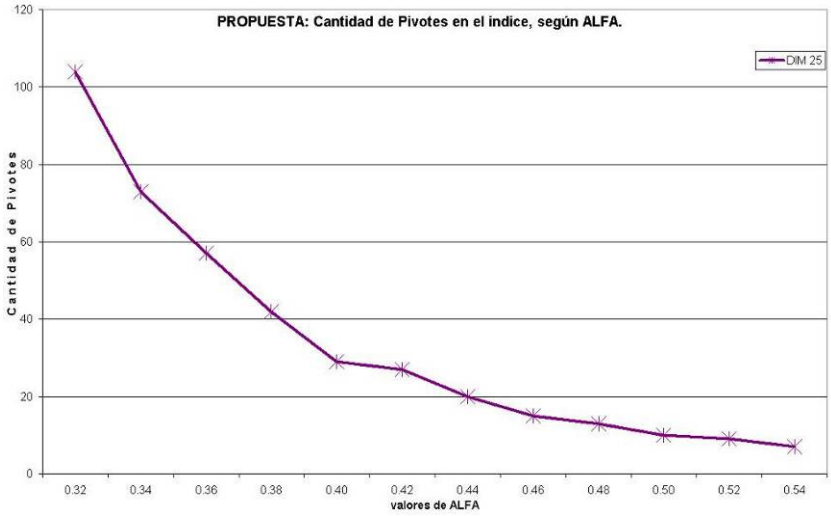


Figura 37: Cantidad de Pivotes en el Índice

Según observamos en la grafica de la Figura 37 para la dimensión 25, la cantidad de pivotes varía de forma decreciente, según se esperaba. Luego del valor 0.50 se observa decrecer velozmente la cantidad de pivotes, validando los resultados obtenidos hasta el momento.

c) Discriminaciones realizadas

En este trabajo se continua evaluando el impacto de la constante al momento de las búsquedas, en este caso, sobre la cantidad de discriminaciones en este nuevo espacio métrico de dimensión 25, ya que a mayor valor de α, es menor la cantidad de pivotes en el índice, y resulta interesante analizar la cantidad de discriminaciones.
Se realiza el experimento bajo las mismas condiciones ya enumeradas, y se obtiene los resultados de la Tabla 5, resultados que son graficados en la Figura 38.

Como se observa en la Figura 38, la cantidad de discriminaciones realizadas por los pivotes en el índice decrece suavemente de forma escalonada a medida que aumenta el valor del parámetro α, este comportamiento valida la propuesta ya que es idéntico a los resultados obtenidos para todas las dimensiones trabajadas de los espacios métricos sintéticos. También aquí, como en todas las dimensiones sintéticas antes presentadas, pasado el valor 0.48 se observa una fuerte disminución.

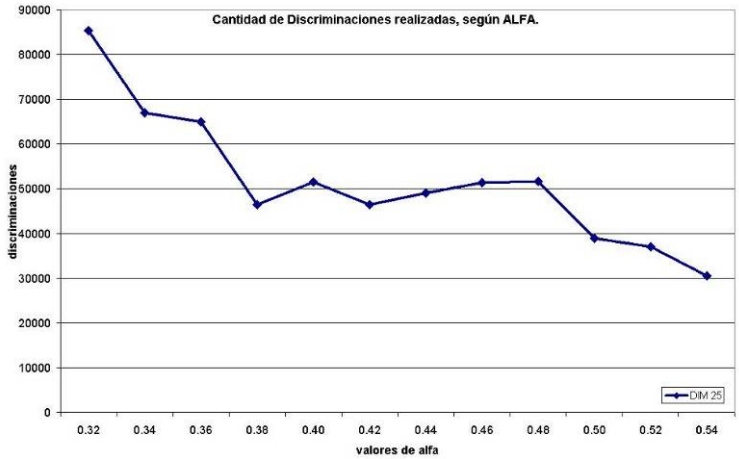


Figura 38: Discriminaciones realizadas por el índice

d) Funciones distancia calculadas

Aquí se analiza la cantidad de veces que se evalúa la función distancia, es por eso que graficamos los resultados de la Tabla 5 en la Figura 39.

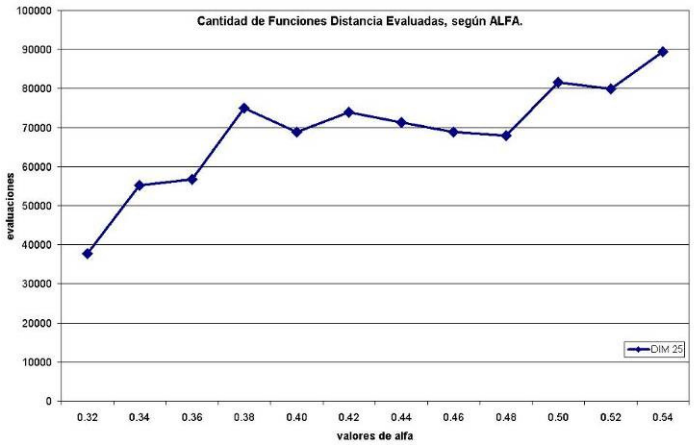


Figura 39: Evaluaciones de la función distancia

La Figura 39 muestra la cantidad de evaluaciones de distancia realizadas, variando el parámetro α. A diferencia que en el espacio métrico sintético, se observa un crecimiento de la cantidad de evaluaciones a medida que aumenta α, ya que al aumentar la distancia entre los pivotes, en teoría, debería disminuir las evaluaciones necesarias de la función distancia.
Este valor tiene un crecimiento escalonado debido a que se promedian las 10 épocas de entrenamiento y a su vez el conjunto de entrenamiento no es *tan representativo* como en el espacio métrico sintético, ya que son muy variadas las imágenes y durante la experimentación muchas consultas han obtenido muchos elementos formando el conjunto de resultados, mientras que otros conjuntos fueron vacíos.

Conclusiones

Luego de estudiar e interiorizarnos sobre bases de datos métricas podemos apreciar la gran importancia que estas tienen a la hora de mejorar la búsqueda de objetos complejos (como datos multimedia, textos, cadenas de ADN, etc...) cada vez más cotidianos en los sistemas de información.

Estas búsquedas por similitud, no por exactitud, de alto costo, fue lo que motivó todo el desarrollo de técnicas para generación de índices que hemos visto en esta oportunidad. Según estudiamos en este trabajo, en muchas aplicaciones y según el dominio de aplicación, la evaluación de la función distancia, es una operación costosa y en muchos de los casos, se utiliza como medida de complejidad a la cantidad de evaluaciones necesarias para resolver la búsqueda.

También se estudió la *maldición de la dimensionalidad,* que describe el fenómeno por el cual la performance de los algoritmos mostrados en el estado del arte disminuye exponencialmente con la dimensión.

Todo el esfuerzo de investigación se ha concentrado en mejorar el desempeño de los índices desde un enfoque teórico. No encontramos ni implementaciones de motores de bases de datos, ni lenguajes extendidos de consulta necesarios para la búsqueda por similitud en espacios métricos. Lo que si encontramos fueron mejoras y combinaciones de distintos métodos de generación de índices, utilizando teorías matemáticas como grafos, recorrido de árboles, optimizaciones por medio de experimentación y demás.

En este trabajo se presenta una nueva estructura de indexación y búsqueda por similitud basada en la selección dinámica de pivotes. Una de sus características más importante es que utiliza al SSS para la selección inicial de pivotes, porque es una estrategia adaptativa que selecciona los pivotes bien distribuidos en el espacio para lograr una eficiencia mayor. La propuesta consiste en agregar dos nuevas políticas de selección de pivotes para que el índice se adapte a las búsquedas cuando el mismo se adaptó al espacio métrico. La estructura propuesta automáticamente se adapta a la región donde se realizan la mayoría de las búsquedas para reducir la cantidad de evaluaciones de distancia, con la política de *'el mejor candidato'* para la selección de pivote entrante, y de '*el que menos discrimina'* para la de pivote saliente del índice.

Durante este trabajo se utilizó la experimentación para validar la propuesta aquí presentada, trabajando en un espacio métrico sintético que nos permita controlar la dimensionalidad, así como también se trabajó con un espacio métrico real de imágenes.

Se comenzó comprobando que nuestra propuesta genera de forma dinámica un número de pivotes que está en función de la dimensionalidad del espacio, y no en función del número de objetos insertados en la base de datos métrica. Esto fue validado con éxito tanto en los espacios métricos sintéticos de varias dimensiones como así también lo fue en el espacio métrico de imágenes.

Luego se continuó la experimentación en los mismos dos espacios métricos, esta vez para evaluar la eficiencia del índice en su construcción y al momento de realizar las búsquedas, variando el parámetro α y la *dimensión* del espacio, para analizar el impacto del valor de estos parámetros, y compararlos con los resultados de SSS.

Del análisis realizado se lograron los resultados esperados, ya que en el espacio métrico sintético la comparación de la propuesta aquí presentada contra la implementación de SSS fue óptima. En todos los casos de las dimensiones, al momento de la construcción del índice y al analizar la cantidad de pivotes del índice se lograron mejores resultados que los obtenidos por SSS, validando así la efectividad de las dos nuevas políticas de selección de pivotes para adaptar el índice.

Ya con la validación de que la propuesta aquí presentada mejora los resultados del SSS al momento de la construcción del índice y en la cantidad del pivotes del índice, se estudiaron los resultados al momento de las búsquedas, prestando especial atención en la cantidad de evaluaciones de la función distancia y de la cantidad de discriminaciones realizadas por los pivotes que forman el índice.

Como era de esperarse, la cantidad de funciones distancias calculadas al momento de las búsquedas, en la propuesta, fue mucho menor que la cantidad de evaluaciones de la función distancia que realiza SSS, esto demuestra que el índice efectivamente se adapta a las regiones en donde se realizan las búsquedas optimizando su función: reducir la cantidad de evaluaciones de la función distancia en el espacio métrico al momento de las búsquedas.

También de esto último podemos concluir que, en la propuesta de este trabajo, el índice realiza un número mayor de discriminaciones al momento de las búsquedas, ya que está mejor adaptado utilizando información histórica de búsquedas anteriores con las dos políticas de "*el mejor candidato*" y de *"el que menos discrimina"*, en contraste con una selección aleatoria de los pivotes que conforman el índice o de una selección inicial estática de pivotes bien distribuidos que cubran el espacio métrico en gran parte, pero que no se intercambien con el correr del tiempo y las búsquedas, como en SSS.

Al momento de analizar la performance de la propuesta, tenemos que hablar del orden de complejidad en el prototipo implementado. El algoritmo de construcción que aquí presentamos, puede ser dividido en 2 grandes pasos: el proceso de selección de pivotes y el proceso de generación del índice y cálculo de las distancias. Esta operación de crecimiento inicial lleva un tiempo de computación $\Theta(k \times n)$ y espacio $\Theta(k \times n)$, siendo *k* la cantidad de pivotes en el índice. Y por otro lado, calcular el valor de la constante **M** lleva un tiempo de computación de $\Theta(n^2)$. Luego de la construcción, la operación de búsqueda, que se realiza de forma tradicional, lleva un tiempo de computación $\Theta(k \times n)$ y espacio $\Theta(n)$ para el conjunto de resultados.
La actualización del índice utilizando la propuesta, lleva un tiempo de computación $\Theta(1)$ y de espacio $\Theta(n)$, tanto para la política de selección de pivote saliente como para la política de selección de pivote entrante. Mientras que la complejidad de cambiar un pivote es $n \times \Theta(\textit{funcion dist})$.

En todos los experimentos desarrollados en los dos espacios métricos que presentamos en la sección anterior, la propuesta de nuestra política de selección dinámica de pivotes que se adaptan a las búsquedas en espacios métricos siempre mostró un comportamiento mejor que una selección de pivotes aleatoria e incluso mejor que *Sparse Spatial Selection* (SSS). Por supuesto que este es el resultado que esperábamos, validando así el algoritmo que periódicamente intenta adaptar los pivotes del índice al uso particular de la base de datos y no relegarlo a un estado inicial estático, como el que se presenta aquí, por medio de la experimentación.
De esta forma, con la estructura de indexación y búsqueda por similitud presentada aquí fue posible mejorar la cantidad de discriminaciones hechas por los pivotes a modo de lograr el objetivo primordial de los índices: bajar la cantidad de evaluaciones de la función distancia.

Para implementar un motor real de objetos de un espacio métrico, es decir, crear un ambiente de bases de datos real, se requiere poder resolver consultas por similitud de forma óptima y eficiente, ya que en aplicaciones reales es factible el hecho de trabajar con grandes cantidades de datos, ya sea por la cantidad de objetos o por el tamaño de cada objeto.
De esto podemos concluir rápidamente que debemos utilizar estructuras adecuadas y eficientes para trabajar en memoria secundaria, y también debemos optimizar la cantidad de E/S.
Existen índices, como el aquí propuesto, que resuelven estos tipos de problemas; pero aún están muy inmaduros para ser implementados en aplicaciones de la vida real por una importante razón, que es la necesidad de trabajar en memoria principal, una característica ya manejada y bien implementada en los índices para bases de datos tradicionales.
Como remarcamos en este trabajo, las líneas de investigación intentan madurar a los índices de los espacios métricos para estas nuevas bases de datos, y así lograr alcanzar un nivel de madurez similar al alcanzado por las bases de datos relacionales.

Trabajo a futuro

Algunas cuestiones por abordar son la evaluación del rendimiento de la propuesta en otros espacios métricos reales, como colecciones de textos en español y/o inglés. Además, a partir de los resultados de la experimentación se puede proponer la implementación de algoritmos que trabajen con índices en memoria secundaria.

También resulta interesante trabajar de forma paralela con el índice y las operaciones de estadísticas, manteniendo por un lado, la creación y actualización del índice, y por el otro lado el manejo de las consultas

Resultaría interesante analizar la evolución de la cantidad de funciones distancia por épocas y variando el radio de búsqueda, ya que en espacios métricos de alta dimensionalidad la cantidad de funciones distancia fue en aumento de forma escalonada.

Otra evaluación interesante a la propuesta aquí presentada es analizar el comportamiento y la eficiencia del índice al momento de las búsquedas cuando se modifica, en un número muy pequeño, la dimensión del espacio métrico, ya sea en una o dos unidades, intentando no recalcular todo el índice, sino seleccionando nuevos elementos pivotes que cubran las nuevas dimensiones.

Mejoras a las políticas de selección

- Utilizar un datawarehouse para entrenar previamente el índice con datos históricos de búsquedas, así el índice inicial podría estar ya adaptado a las clases de consultas habituales.
- Mejora de implementación de la propuesta con la utilización de estructuras de datos más eficientes para llevar las estadísticas o con una implementación en paralelo.
- Otras propuestas para elegir el pivote entrante, tal vez utilizando el concepto propuesto por el algoritmo SSS.

Se sabe de la existencia de los espacios métricos anidados, que son algunos espacios métricos en donde los objetos de la colección pueden agruparse en clusters o subespacios, y distintas dimensiones explican las diferencias entre cada par de objetos en cada uno de los subespacios anidados dentro de un espacio métrico más general.
Una idea interesante para trabajar es la de tratar de identificar estos subespacios y aplicar la propuesta aquí presentada en cada uno de ellos, aplicando la propuesta con distintos valores del parámetro α para, en una primera etapa, identificar los subespacios mediante valores altos de α y obteniendo una cantidad pequeña de pivotes, y en una segunda etapa aplicar nuestra propuesta en cada uno de los subespacios.

Como trabajo futuro de esta investigación resulta interesante poder evaluar el comportamiento de la propuesta experimentando sobre distintos espacios métricos. Y con esto realizar comparaciones contra otras estructuras índices, tal vez de clustering.
También queremos respondernos si esta estructura sería eficiente en memoria secundaria, si es posible mantener los pivotes separados de los elementos para obtener una mejor performance en la memoria secundaria, o incluso investigar si existen otras maneras de combinar estas técnicas de adaptación de los pivotes para búsquedas en espacios métricos utilizando información histórica de las búsquedas realizadas.

Bibliografía y Referencias

[Baeza-Yates, 1994] R. A. Baeza-Yates, W. Cunto, U. Manber, and S. Wu.: Proximity matching using fixed-queries trees. In M. Crochemore and D. Gusfield, editors, Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching, volume LNCS(807), pages 198-212. (1994).

[Bartolini, 2002] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. String matching with metric trees using an approximate distance. In SPIRE 2002: Proceedings of the 9th International Symposium on String Processing and Information Retrieval, pages 271–283, London, UK, 2002. Springer-Verlag.

[Baeza-Yates, 1997] Ricardo Baeza-Yates: Searching: an algorithmic tour. Encyclopedia of Computer Science and Technology, 37:331-359 (1997).

[Brin, 1995] Sergey Brin: Near neighbor search in large metric spaces. In 21st Conference on Very Large Databases (VLDB)(1995).

[Bozkaya, 1997] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In ACM SIGMOD International Conference on Management of Datas, pages 357–368. Sigmod Record 26(2), 1997.

[Bustos , 2001] B.Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity search in metric spaces. In SCCC 2001, Proceedings of theXXI Conference of the Chilean Computer Science Society, pages 33-44. IEEE Computer Science Press .(2001).

[Bustos , 2006] B.Bustos, G. Navarro and D. Keim . Index Structures for Similarity Search in Multimedia Databases.Universidad de Chile

[Burkhard, 1973] Walter A. Burkhard and Robert M. Keller: Some approaches to best-match file searching. Communications of the ACM, 16(4):230-236.(1973)

[Chavez , 2000] E. Chávez and G. Navarro: An effective clustering algorithm to index high dimensional metric spaces. 2000.

[Chavez, 2001a] E. Chavez, G. Navarro, R. Baeza-Yates, and J.L. Marroquin: Searching in metric spaces. ACM Computing Surveys, 33(3):273–321, September 2001.

[Chavez, 2001b] E. Chavez, G. Navarro, A. Marroquin: Fixed queries array: a fast and economical data structure for proximity searching. Multimedia Tools and Applications (MTAP), 14(2):113-135.(2001).

[Chavez, 2005a] E. Chavez and G.Navarro: Metric Databases, Idea Group Inc., Pennsylvania, USA, 2005.

[Chavez, 2005b] E. Chavez and G.Navarro: A compact space decomposition for effective metric indexing. Pattern Recognition Letters, 26(9):1363–1376, 2005.

[Ciaccia, 1997] P. Ciaccia, M. Patella, and P. Zezula M-tree: an efficient access method for similarity search in metric spaces. In Proc. 23rd Conf. On Very Large Databases (VLDB'97), pages 426-435. Software available for download at http://www.db.deis.unibo.it/Mtree/ (1997)

[Gil-Costa, 2008]V.Gil-Costa, M. Marin and N. Reyes: An Empirical Evaluation of a Distributed Clustering-Based Index for Metric Space Databases.In International Workshop on Similarity Search and Applications (SISAP 2008), Cancun, Mexico, Aprill 11-12,pp.386-393, IEEE-CD Press.(2008)

[Guttman , 1984] A. Guttman: R-trees: a dynamic index structure for spatial searching. In ACM SIGMOD International Conference on Management of Data, pages 47–57, 1984

[Figueroa, 2007] Karina M Figueroa Mora: Indexación Efectiva de Espacios Métricos usando Permutaciones. Tesis de Doctor en Ciencias mención Computación, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, Departamento de Ciencias de la Computación, Santiago Chile. (2007)

[Kalantari , 1983] Iraj Kalantari and Gerard McDonald. A data structure and an algorithm for the nearest point problem. IEEE Transactions on Software Engineering, 9:631-634, 1983.

[Navarro, 1999]G.Navarro: Searching in metric spaces by spatial approximation. In Proceedings of String Processing and Information Retrieval Symposium & International Workshop on Groupware (SPIRE'99), pages 141-148. IEEE Computer Science Press. (1999)

[Navarro 2002a] G. Navarro: Searching in metric spaces by spatial approximation. The Very Large Databases Journal (VLDBJ), 11(1):28-46.(2002)

[Navarro, 2006] G. Navarro y R. Paredes, Karina Figueroa, E. Chavez: On the least cost for proximity searching in metric spaces, 2006.

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

[Paredes, 2002] Rodrigo Andrés Paredes Moraleda. Uso de t-spanners para búsqueda en espacios métricos. Universidad de Chile, 2002. Tesis para optar al Título de Ingeniero Civil en Computación y Grado de Magíster en Ciencias, Mención Computación.

[Pedreira , 2007] Spatial Selection of Sparse Pivots for similarity search in metris Spaces. Proc. Of the 33nd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'07) – LNCS vol: 4362,pages 434-445. (2007)

[Pedreira , 2007b] O.Pedreira, Fariña, Brisaboa and Nora Reyes: Similarity search using sparse pivots for efficient multimedia information retrieval. The Second IEEE International Workshop on Multimedia Information Processing and Retrieval (2007)

[Yamaguchi] K. Yamaguchi y S.Masuda Ken Tokoro. Improvements of tlaesa nearest neighbour search algorithm and extension to approximation search.

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

[Yianilos, 1993] P. Yianilos. Data structures and algorithms for nearest-neighbor search in general metric spaces.1993.

[Yianilos, 1999] P. Yianilos: Excluded middle vantage point forests for nearest neighbor search. In Proceeding of the 6th DIMACS Implementation Challenge: Near Neighbour searches ALENEX'99.(1999).

[YU, 2001] C. Yu, B. Chin Ooi, K.L. Tan, and H. V. Jagadish. Indexing the distance: An efficient method to KNN processing. 2001

[JAI 1] Santos, R. July 2004 JAI Stuff : https://jaistuff.dev.java.net . Visitado en Abril 2009.

[JAI 2] Sun Microsystems, Java home page, http://java.sun.com . Visitado en Abril 2009

[JAI 3] Sun Microsystems, JAI (Java Advanced Imaging) 2004 home page, http://java.sun.com/products/java-media/jai/index.jsp . Visitado en Abril 2009

[JAI 4] Rodrigues, L.H. Building Imaging Applications with Java Technology, Addison-Wesley,2001.

[JAI 5] *The Color FERET image database*, http://face.nist.gov/colorferet/ . Descargado en Marzo de 2009.

Agradecimientos

A mis viejos, Carlos y Cristina, que desde el año 2002 apostaron a mi educación universitaria, me bancaron siempre cuando nadie creía en mí, y día a día son un ejemplo de lucha y esfuerzo que trato de seguir. Estoy seguro que sin el apoyo y las enseñanzas de vida de ellos dos, hoy yo no hubiese logrado ser lo que soy.

Agradezco a mi Tía Hilda y mis hermanos Diego y Ariel, cada uno aportando su cuota de esperanza y acompañando cada paso que di en la carrera universitaria, ayudando en las derrotas y festejando en cada pequeño logro, logrando así ser una parte fundamental a lo largo de estos años, y haciéndome sentir muy orgulloso de tenerlos como familia.

A Dios, por darme tan bella familia, con tantas enseñanzas, valores y gestos, que acortaban las distancias entre las frías y solitarias noches de Rosario, y mi querida Cañada de Gómez.

Agradezco a ese pibe que en aquel febrero del 2002 se sentó a mi lado, en mi primer día de clases dentro de la universidad, y que hoy aún continúa a mi lado en cada batalla. Sí, a vos Cristian *"Kily"* Gonzalez, porque me brindaste una amistad inolvidable, de esas que te llenan el corazón, al igual que Hugo*"Cuchi"* Gercek, que con su amistad y enseñanzas de vida (y no de futbol) me acompaño durante toda esta etapa.

Dedico también este trabajo a Vanesa, una gran mujer y excelente persona. Que me acompaño en los últimos años de la carrera y me bancó muchísimas cosas que hicieron posible terminar esta tesina y obtener el título universitario.

A Juan Pablo, "*Rifle"*, Lorandi que me brindó la oportunidad de crecer a nivel profesional, realizar mi primer experiencia laboral, mientras cursaba materias y finalizaba los estudios universitarios, bancándome horarios exóticos, viajes, exámenes y hasta el día de hoy, muchísimos errores que al ser corregidos por él, me continúan dejando enseñanzas.

Dedicado también a excelentes personas y mejores compañeros aun como lo son Damián Alvarez, Federico Bergero y Rafael Namias, cada uno en distintas etapas de la carrera acompañando y ayudándome, ya sea con una simple consulta hasta la revisión de redacción en este trabajo final.
Tampoco puedo dejar de nombrar a los muchos compañeros con los que compartí horas de estudio, como Javier Verbas, Hernán Baro-Graf, Diego Hoolman y Maura Venturini, cada uno en su momento brindando su ayuda.

A mis dos directoras, Claudia Deco y Cristina Bender, por la dedicación y paciencia hacia mí a lo largo de este año para realizar este trabajo final de carrera, mostrándome el camino y ayudando cuando el tiempo era escaso. Por el esfuerzo de las dos para llegar a presentar un artículo en el *XV Congreso Argentino de Ciencias de la Computación 2009*.

Agradezco a la gran cantidad de docentes que dedicaron su tiempo y esfuerzo a nosotros, los alumnos, a lo largo de la carrera, como Raúl Kantor, Ana Casali, Guido Macci, Federico Gimpel, Dante Zanarini y Gabriela Arguirofo.

A los salseros, a los compañeros de trabajo, y a los que día a día me acompañan y forman parte de mi vida, a todos ustedes quiero tener presente en este momento y dedicarles este trabajo.

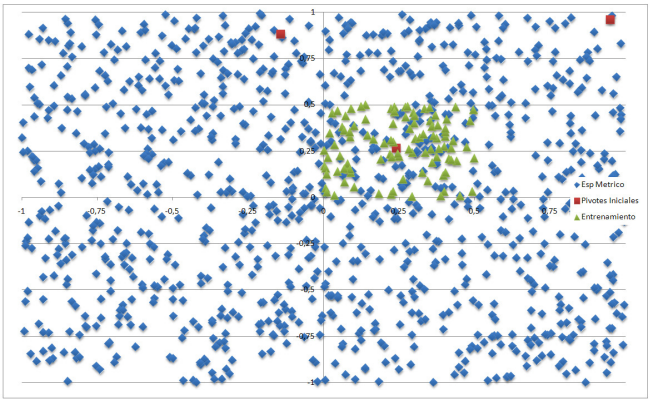
Apéndices

Apéndice A – Screenshots

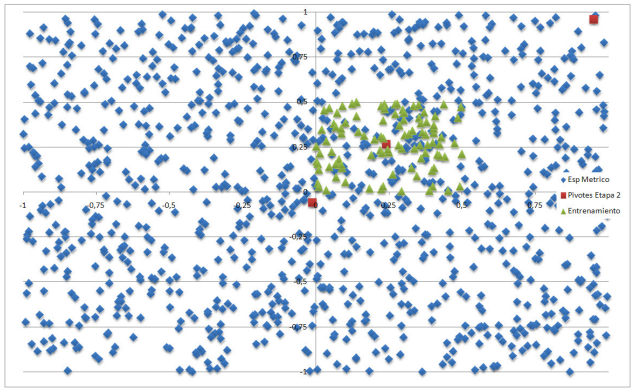
En este apéndice vamos a exponer todos los gráficos que representan las distintas épocas de las experimentaciones de los algoritmos presentados en la sección de implementación de la propuesta. Con estos, analizamos el comportamiento de las distintas implementaciones, según se explican en el apartado de experimentación.

A1 – Random

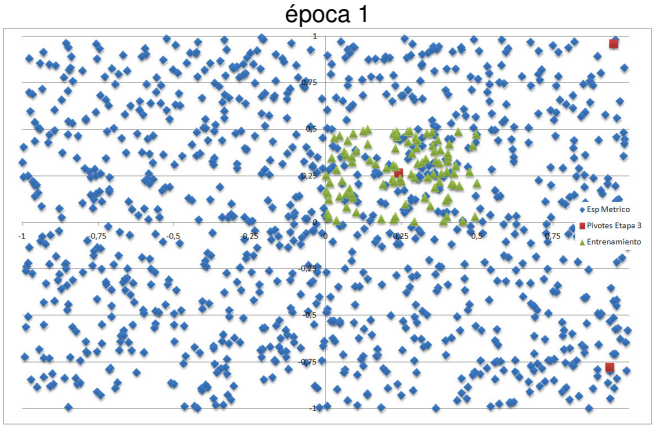
A continuación mostramos los gráficos asociados a la implementación del método '*Random*', donde tenemos: política de selección aleatoria para pivotes iniciales del índice y selección posterior de pivotes de forma aleatoria, tanto pivotes entrante al índice como pivotes salientes del índice.



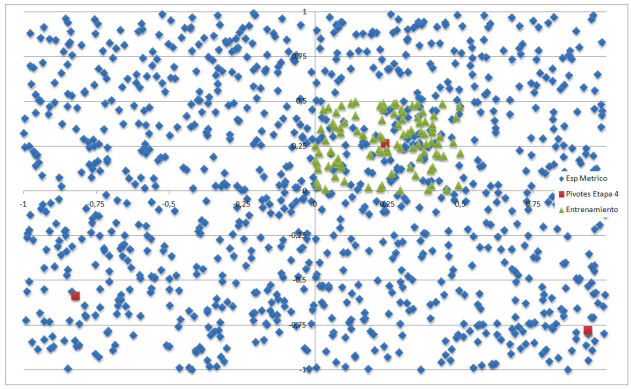
época 1



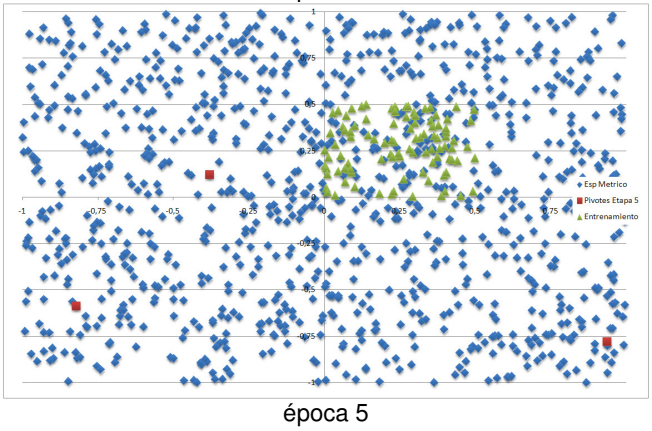
época 2



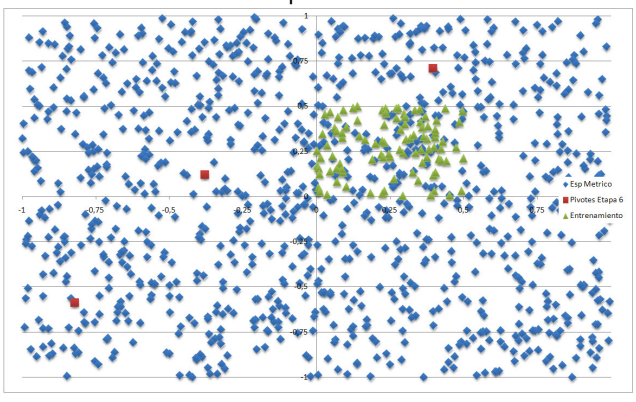
época 3



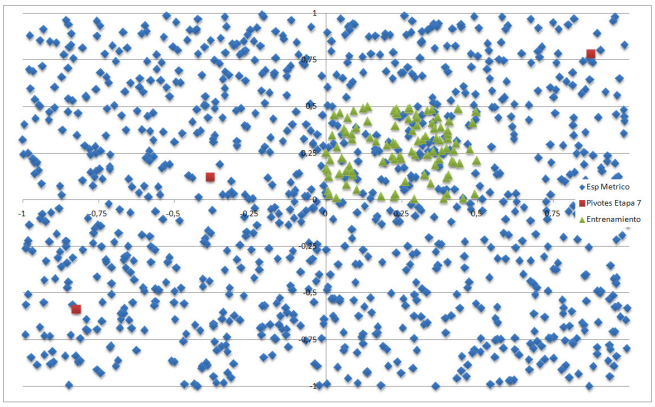
época 4



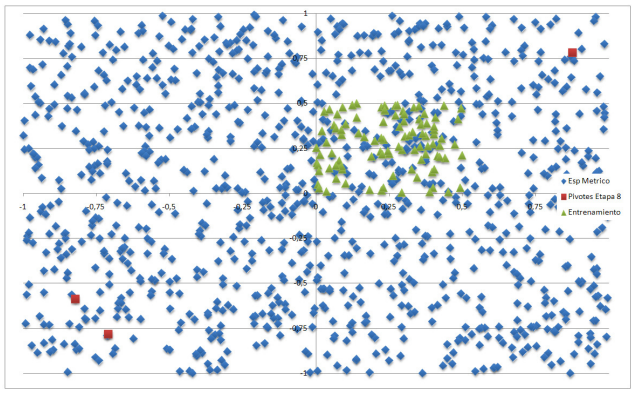
época 5



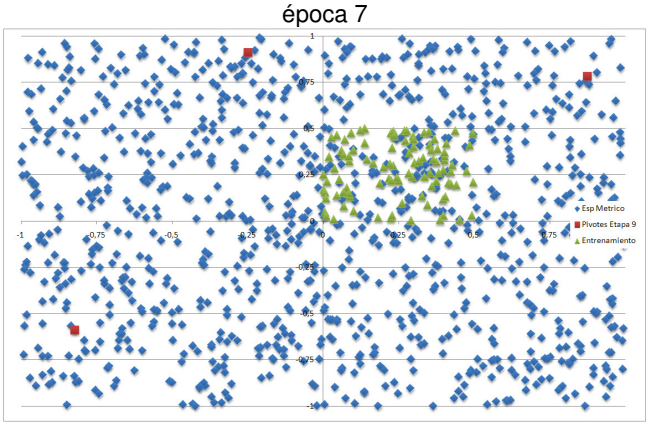
época 6



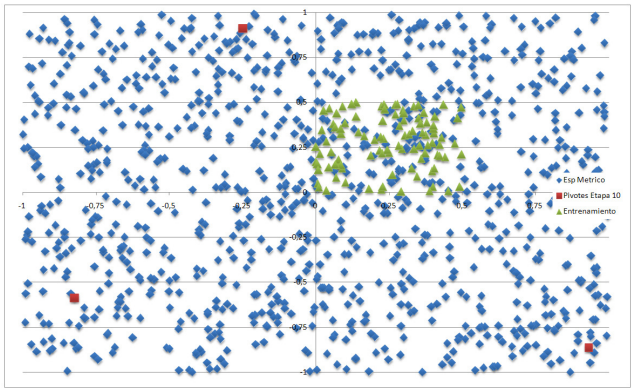
época 7



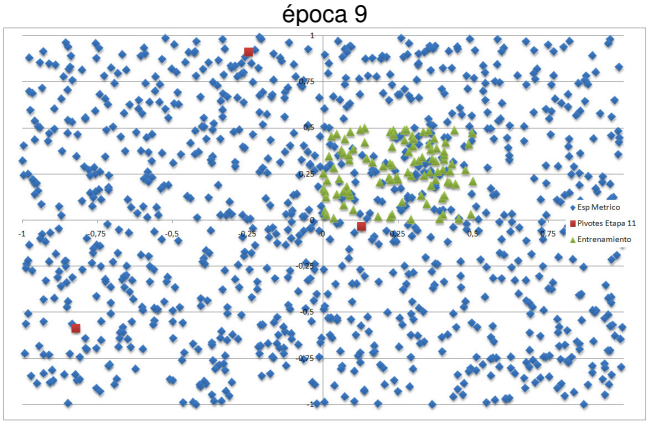
época 8



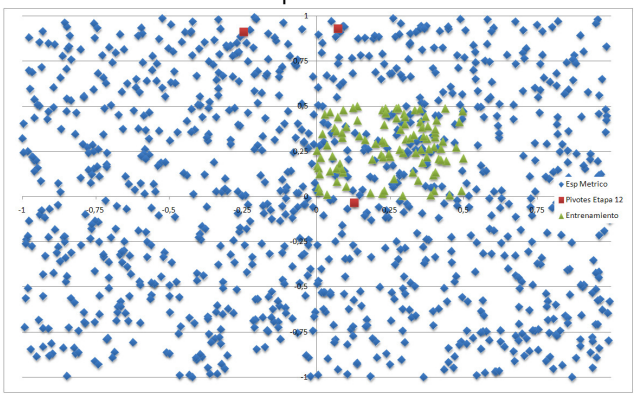
época 9



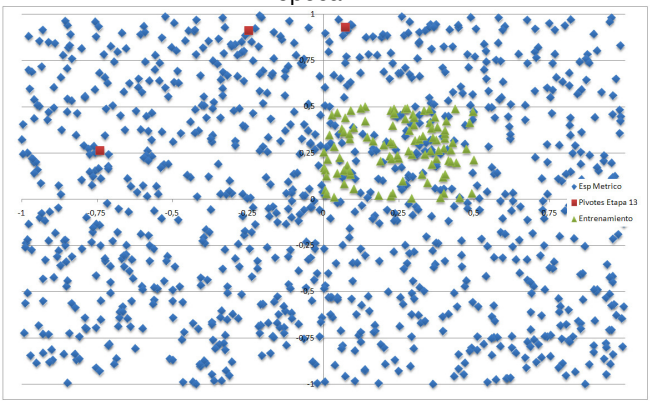
época 10



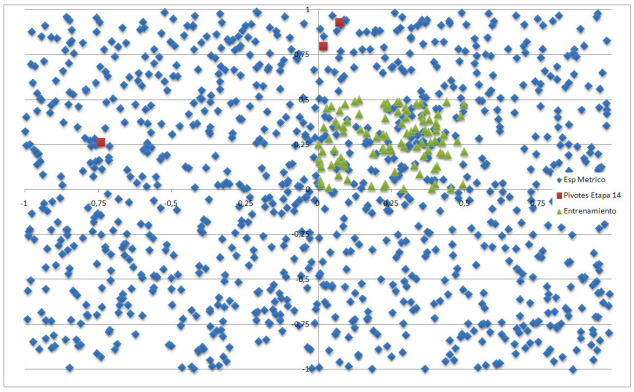
época 11



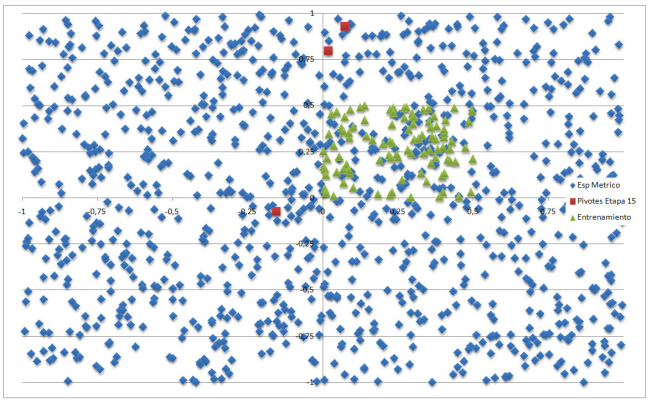
época 12



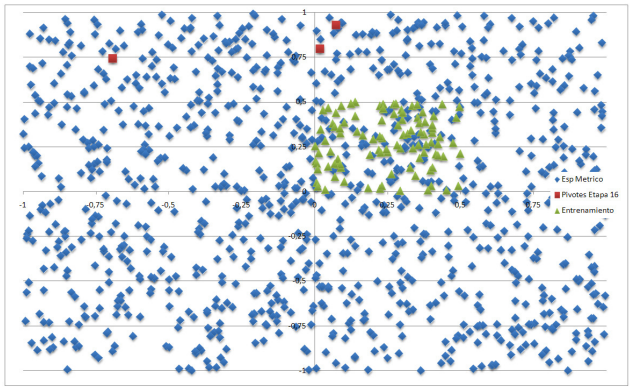
época 13



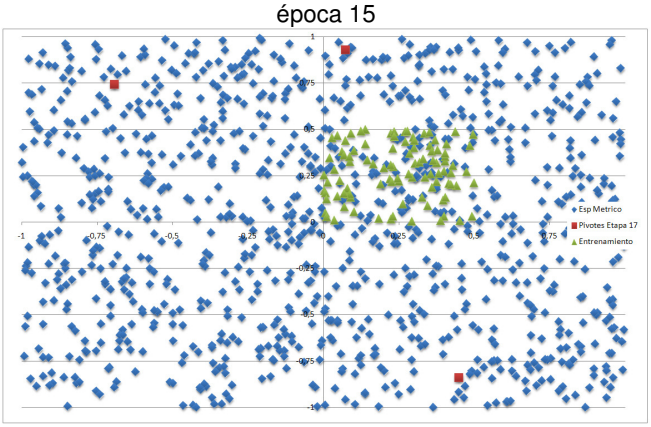
época 14



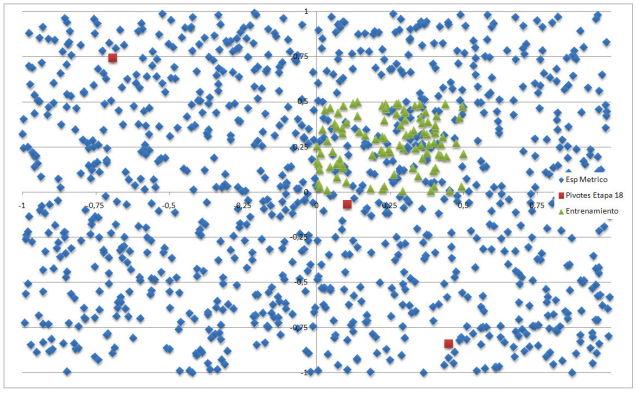
época 15



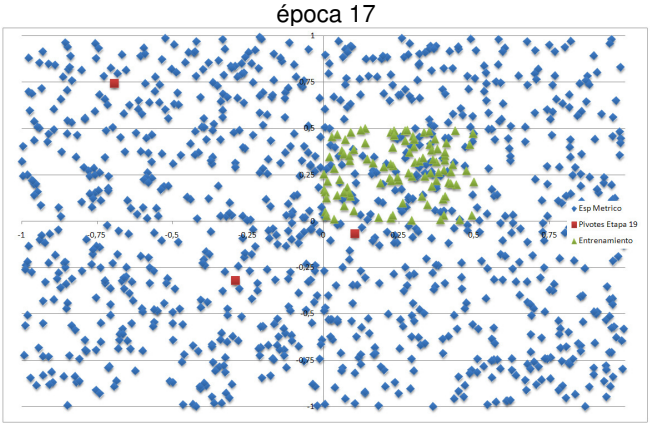
época 16



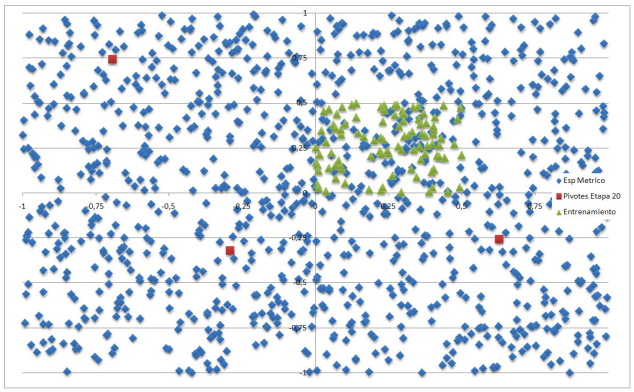
época 17



época 18



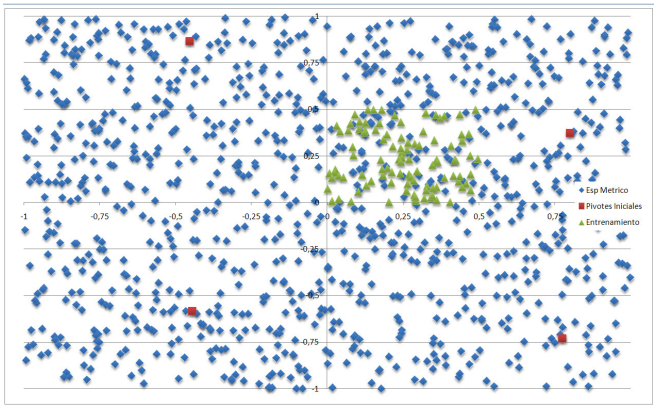
época 19



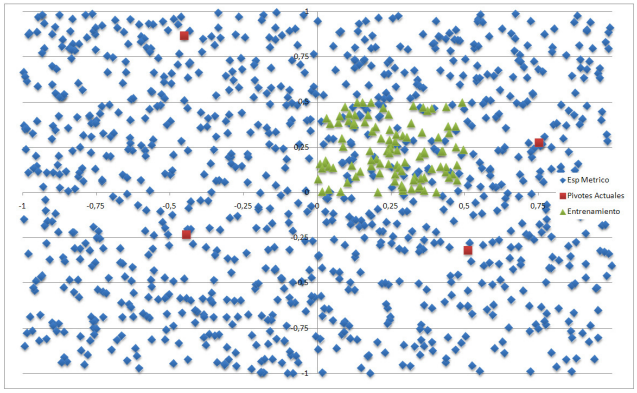
época 20

A2 - Sparse Spatial Selection (SSS)

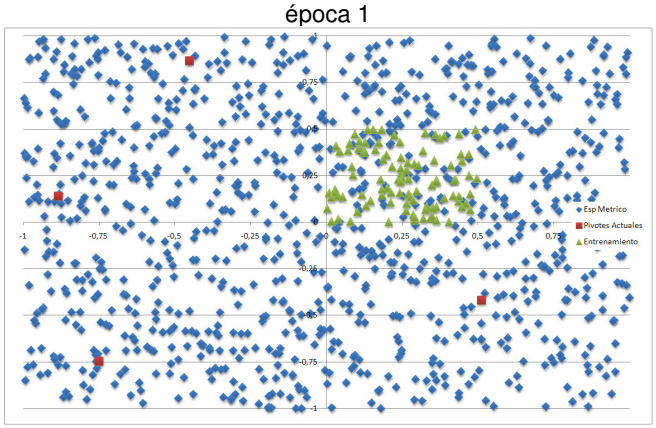
En este apéndice, basado en [Pedreira, 2007] se muestran los gráficos asociados a la implementación del método *SSS*, donde tenemos una política de selección para pivotes iniciales del índice tal que los pivotes estén distribuidos de una forma especial dentro del espacio métrico.



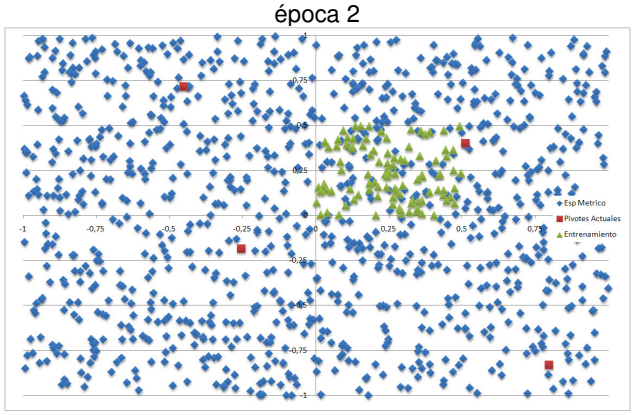
época 1



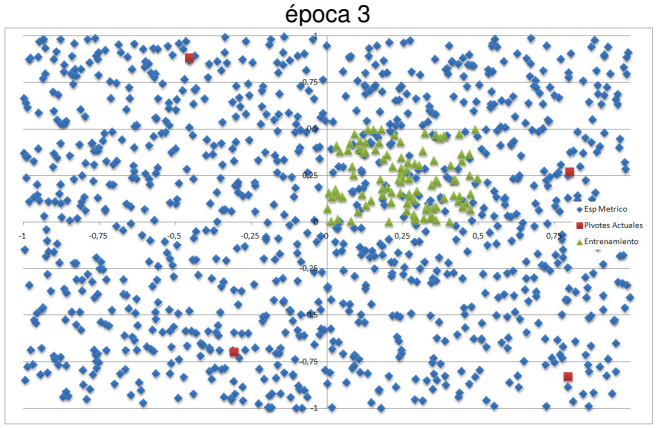
época 2



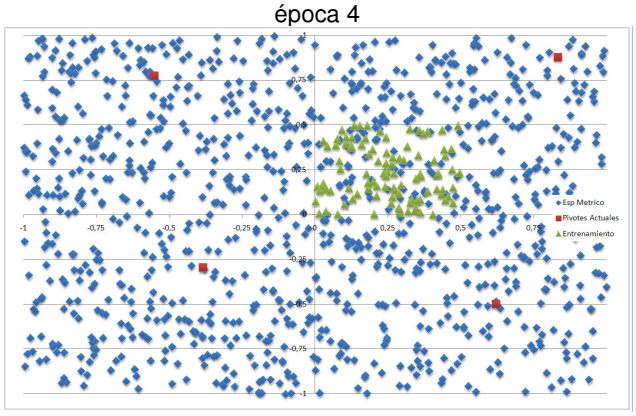
época 3



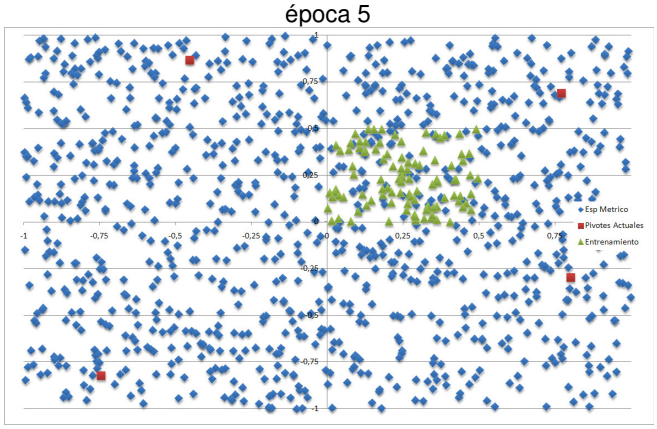
época 4



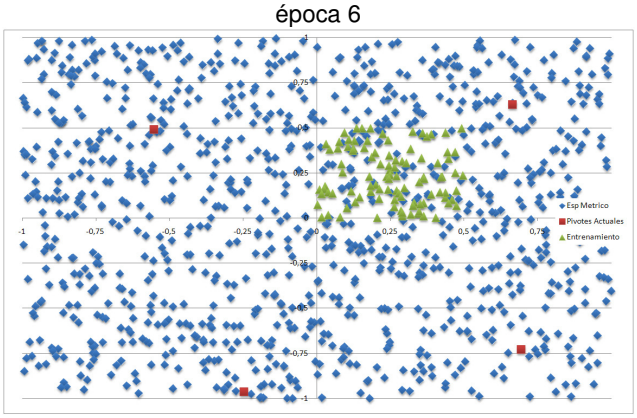
época 5



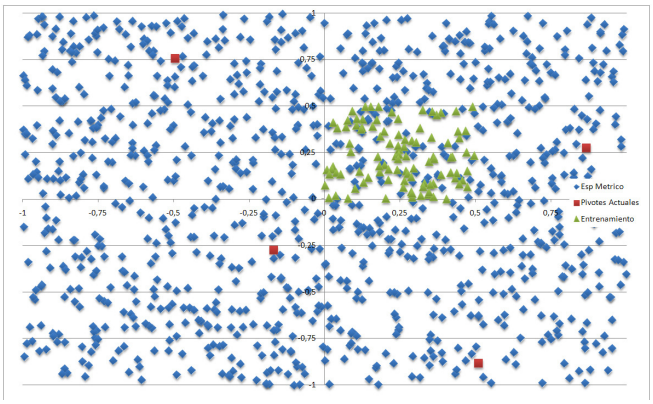
época 6



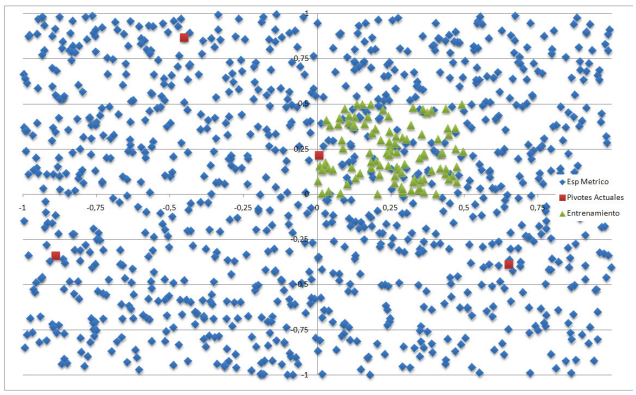
época 7



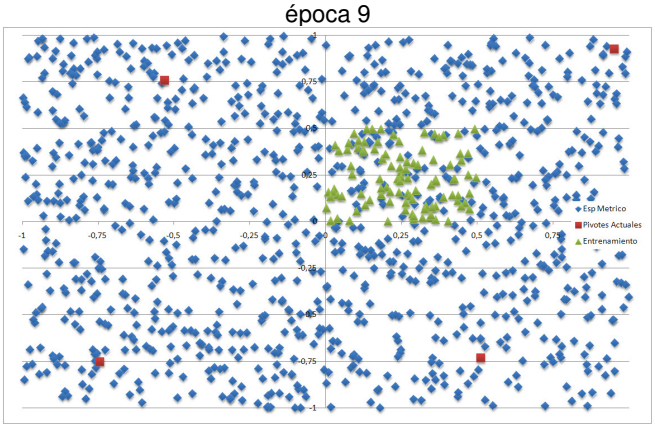
época 8



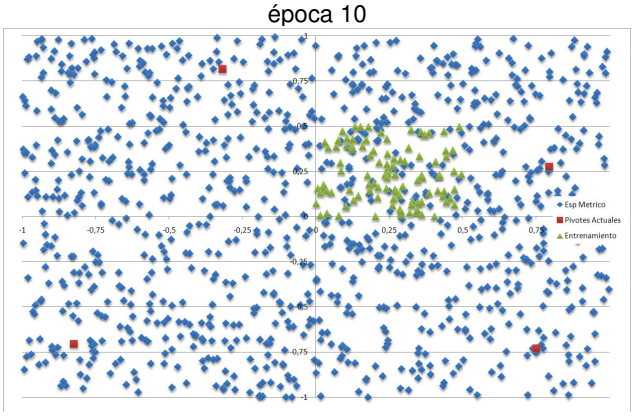
época 9



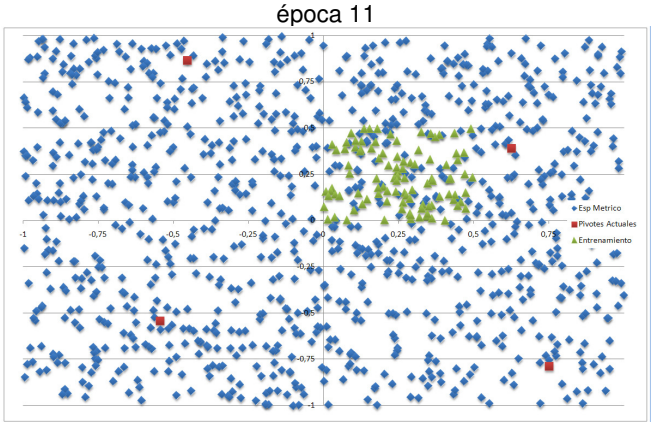
época 10



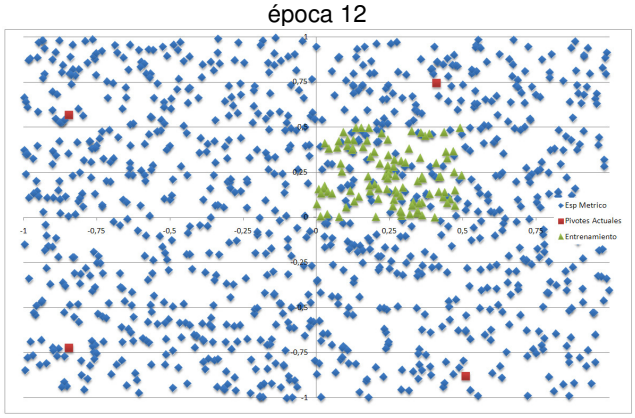
época 11



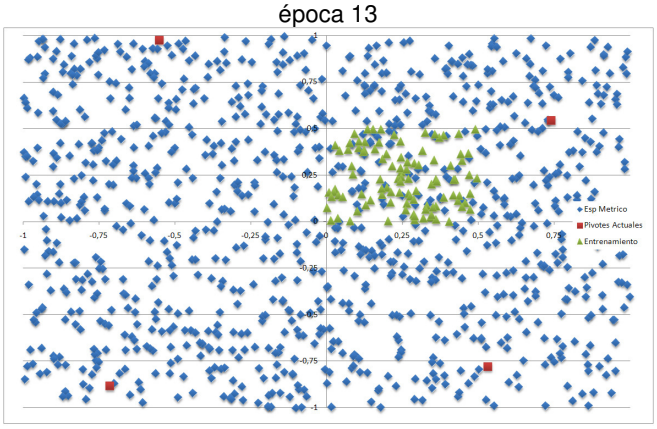
época 12



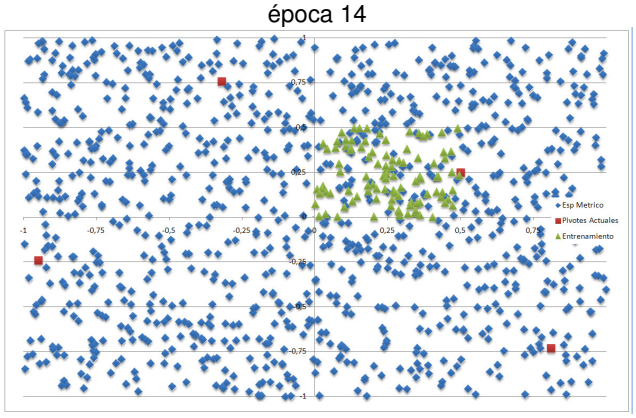
época 13



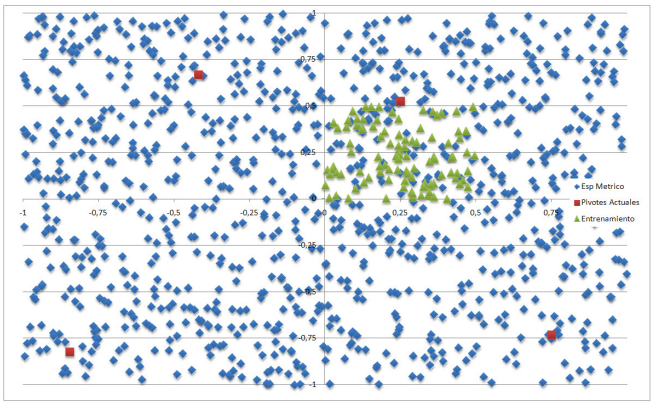
época 14



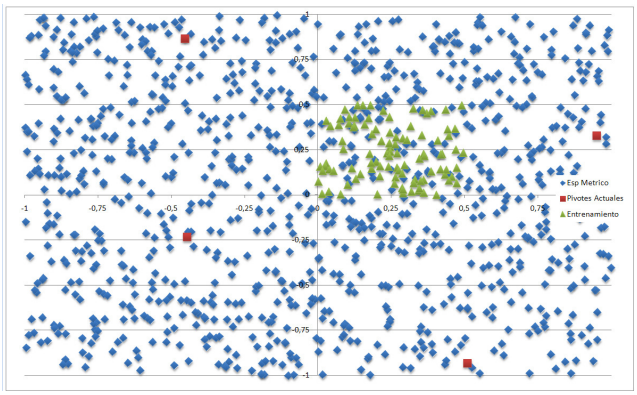
época 15



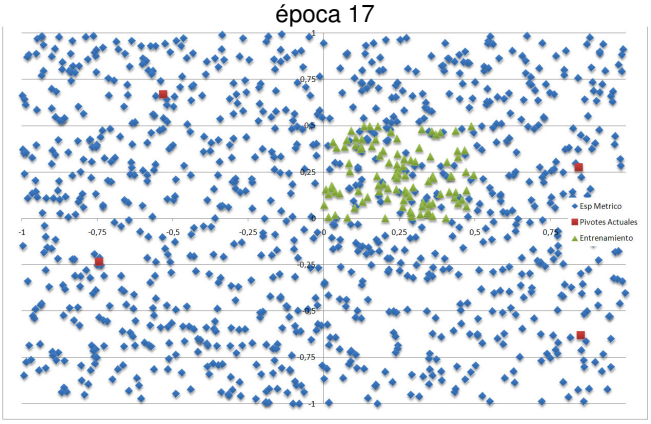
época 16



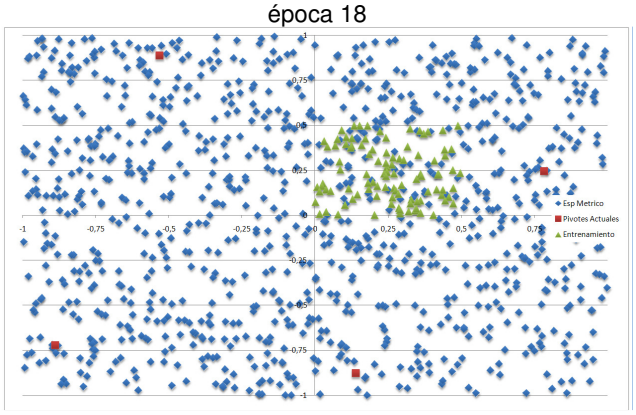
época 17



época 18



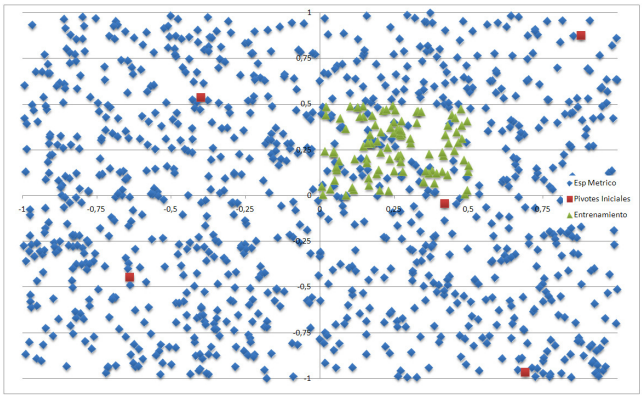
época 19



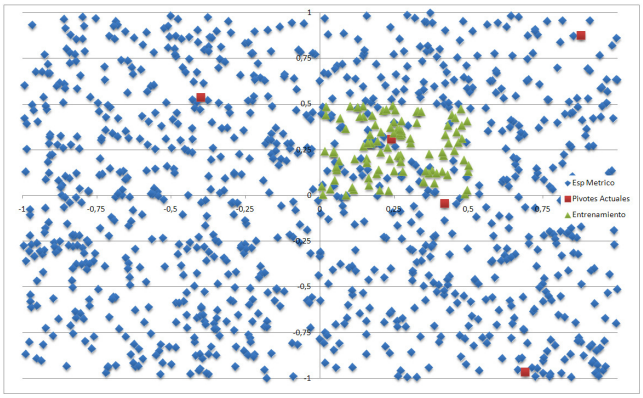
época 20

A3 – Propuesta

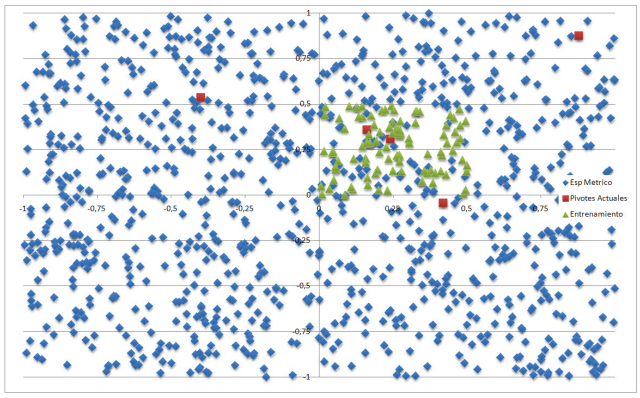
En este apéndice mostramos los gráficos asociados a la implementación de nuestra propuesta, donde comenzamos la construcción del índice aplicando *SSS* y luego tenemos una política de selección para pivotes entrantes y salientes del índice tal como se explico en este trabajo, para que se adapten los pivotes del índice a las búsquedas dentro del espacio métrico.



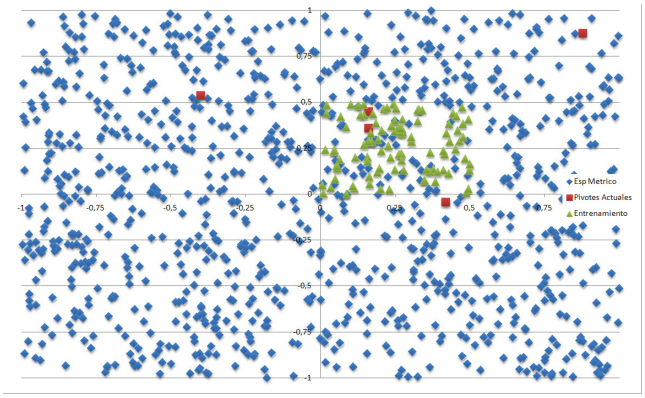
época 1



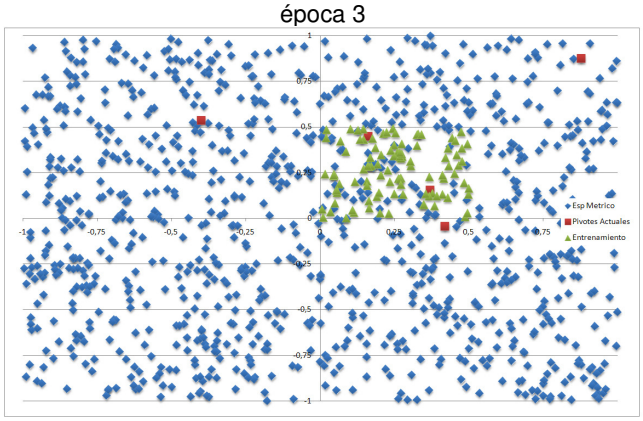
época 2



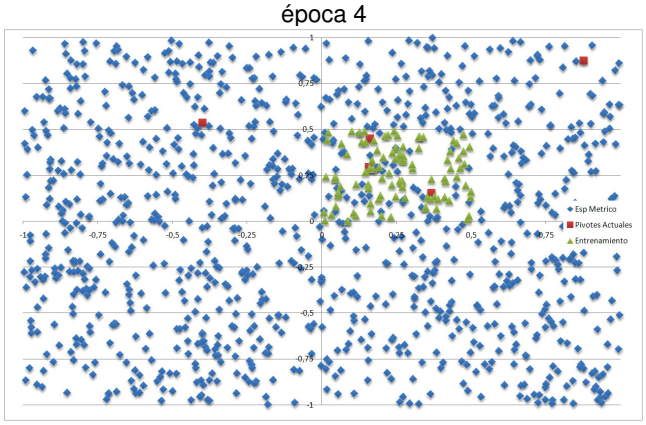
época 3



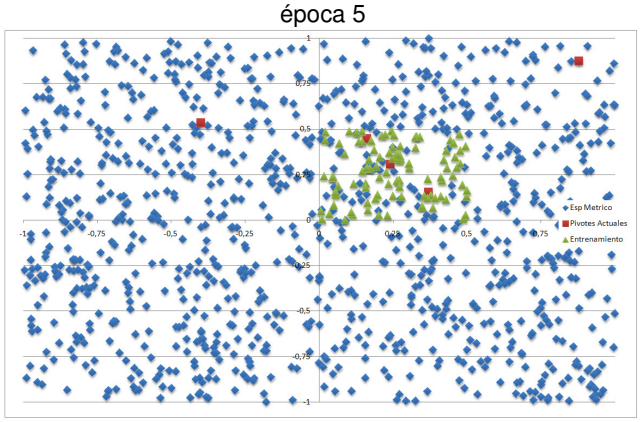
época 4



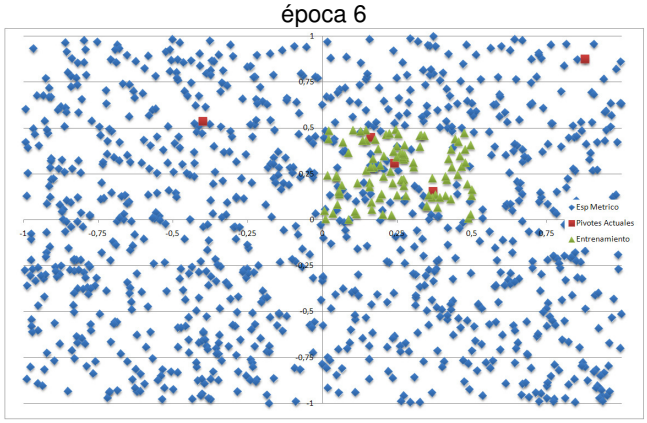
época 5



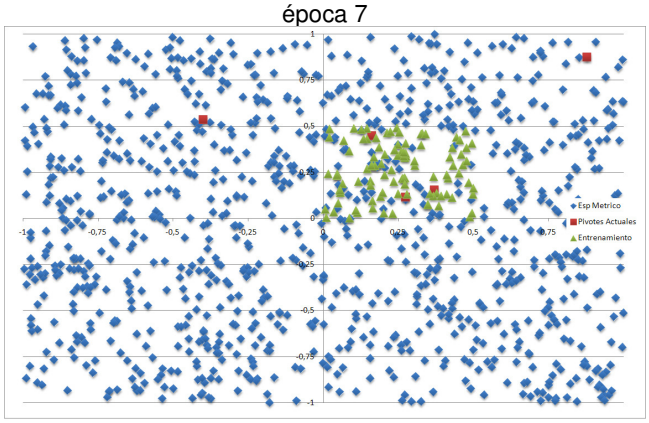
época 6



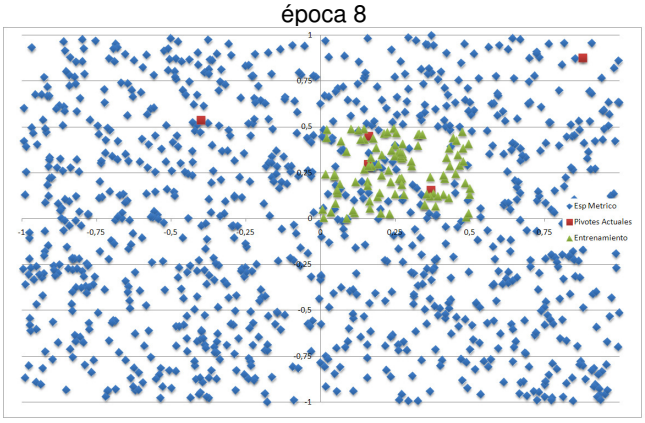
época 7



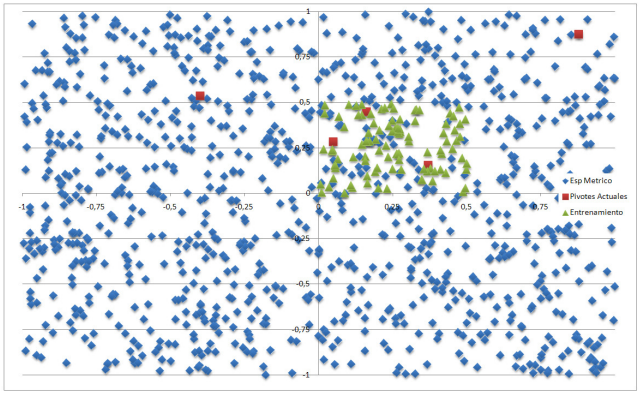
época 8



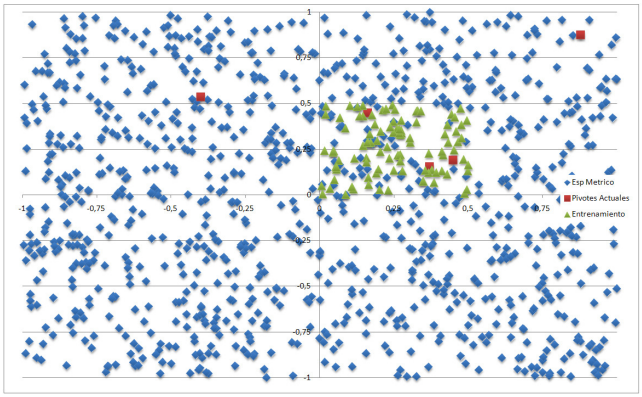
época 9



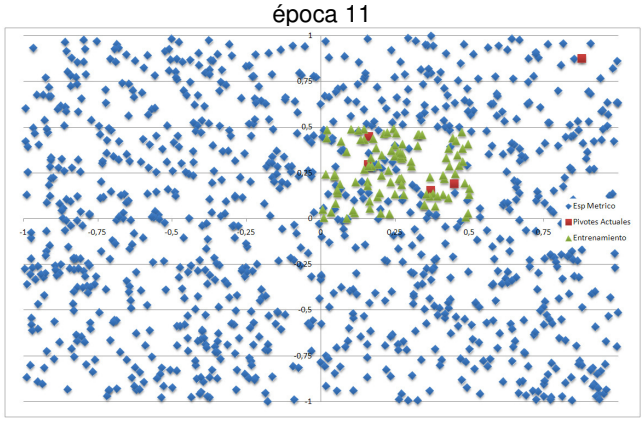
época 10



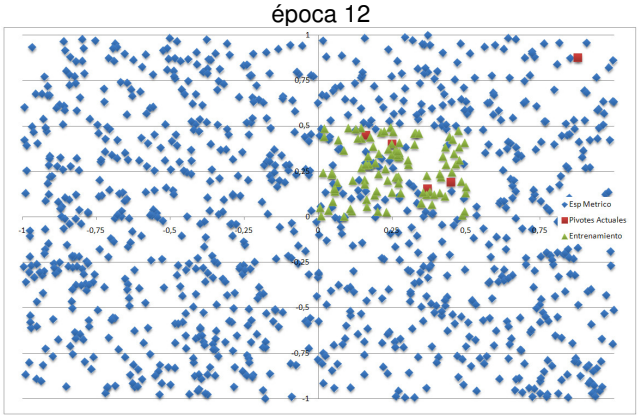
época 11



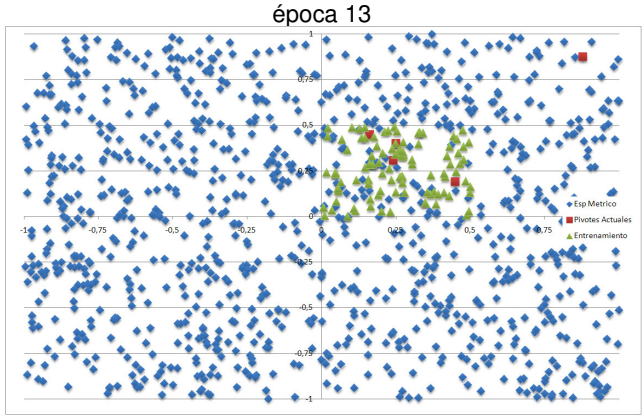
época 12



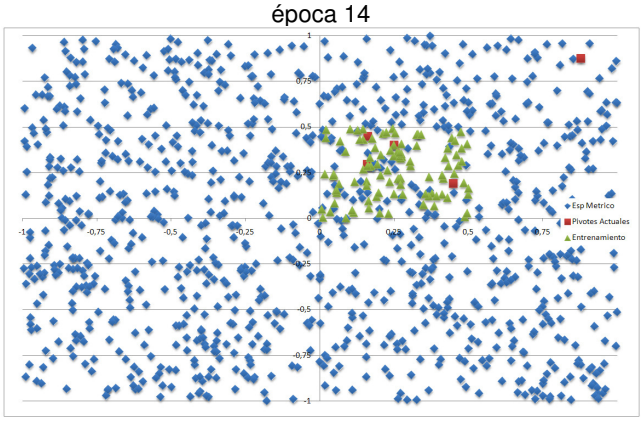
época 13



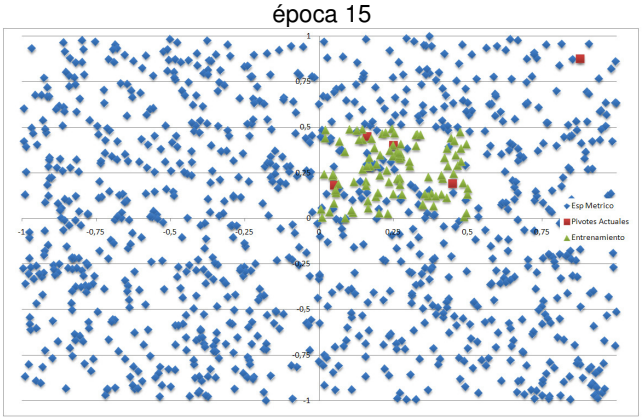
época 14



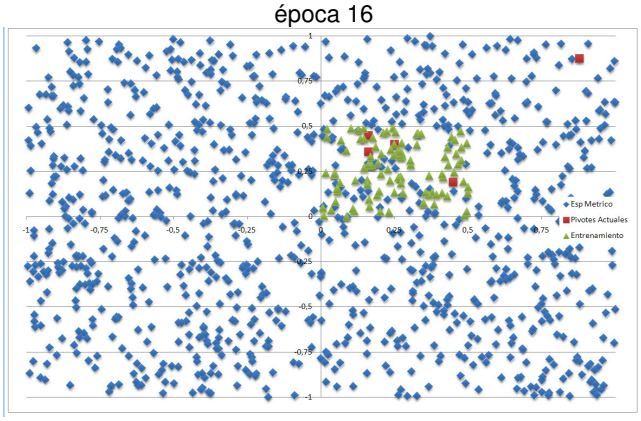
época 15



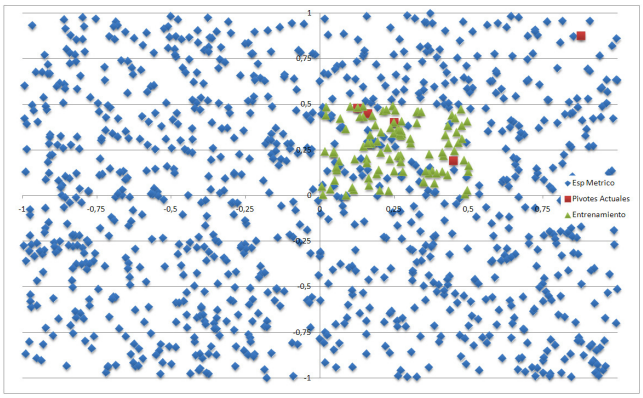
época 16



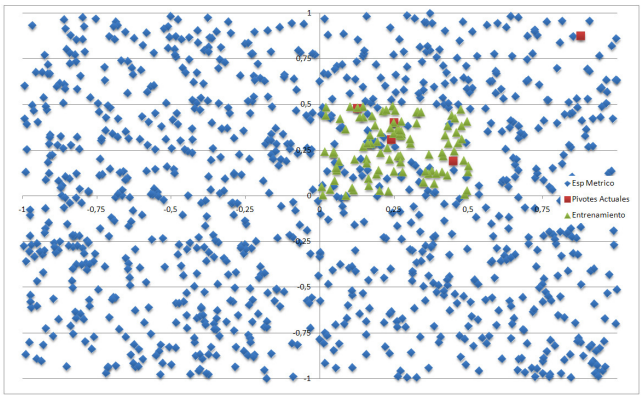
época 17



época 18



época 19



época 20

Si observamos las imágenes que representan las 20 épocas de la propuesta, podemos ver la forma en que los pivotes se adaptan a las búsquedas representadas por los elementos amarillos, y también observamos como de forma muy rápida en las primeras épocas del entrenamiento esto sucede, mientras que ya en las últimas épocas, son solo cambios mínimos que podrían dejarse de lado como una forma de optimizar los cambios.

Apéndice B - Espacios Métricos Utilizados

Aquí se muestra y explica los dataset utilizados al momento de realizar las experimentaciones.
Tenemos los datasets "sintéticos", creados mediante software y como real utilizamos: *The Color FERET image database*, http://face.nist.gov/colorferet/ , descargado en Marzo de 2009.

B1 – Espacio Métrico Sintético

La ventaja de probar algoritmos con este tipo de datos, los espacios vectoriales sintéticos, es que podemos estudiar su comportamiento en espacios de dimensionalidad conocida. Por lo tanto, es interesante experimentar en donde sea posible variar la dimensión.
Una manera de controlar la dimensión del espacio es generar un conjunto sintético uniformemente distribuido en el cubo unitario, y usar este conjunto como un espacio métrico abstracto.

Así fue que definimos nuestro espacio métrico sintético y decidimos variar la dimensionalidad, pero siempre trabajando sobre el cubo unitario y con la misma función distancia que nos mida la similitud entre los objetos.
La función de distancia utilizada con estos conjuntos de datos fue la distancia Euclidiana, y más adelante veremos el código de ejemplo. Como conjunto de entrenamiento, se utilizaron elementos del mismo espacio métrico tomados de forma aleatoria.

Descripción de código y explicaciones

¿Cómo es la estructura de los objetos dentro del espacio métrico sintético? Muy simple, están modelados por la siguiente clase:

```
1.  public class SimpleItem {
2.      static final int BASIC_DIM = 2;
3.      private double coordenadas[];
4.
5.          public SimpleItem(final int dimension) {
6.              coordenadas = new double[dimension]();
7.          }
8.  }
```

En la línea 2 vemos que incluimos la constante *BASIC_DIM=2*, para tener como referencia, en caso de que no se envíe como parámetro la dimensión que tendrán los objetos dentro del espacio métrico.
En la línea 3 esta el arreglo de valores que contendrá las coordenadas del objeto dentro del espacio métrico. Y en la línea 5 tenemos el constructor de los objetos donde se pasa como parámetro la dimensión que utilizaremos.

En el siguiente fragmento de código vamos a crear una colección de "SimpleItem", que no son más que objetos de un espacio métrico sintético en el cual definimos las coordenadas, según una dimensión que es un parámetro al momento de inicializar el espacio métrico.
Las coordenadas de los objetos son números aleatorios tomados del intervalo (-1, 1) y la dimension es asignada al momento de construir el espacio métrico dentro de la variable 'DIMENSION'.

```
1.  ArrayList rval = newArrayList(MAX_ELEMENT_TRAIN);
2.      SimpleItem item;
3.      double coordenadas[];
4.
5.      for (int i = 0; i <MAX_ELEMENT_TRAIN; i++) {
6.          item = new SimpleItem(Database.DIMENSION);
7.          coordenadas = new double[Database.DIMENSION];
8.
9.          for (int j = 0; j < coordenadas.length; j++)
10.             coordenadas[j] = (Math.random()*(-0.5) + 0.5);
11.
12.         item.setCoordenadas(coordenadas);
13.         rval.add(item);
```

```
14.     }
```

Como Espacios Vectoriales sintéticos crearemos una colección de 100.000 vectores de dimensión 2,4,6,8,10 14 y 20, creados de la forma que recién mostramos, tomando así la constante *MAX_ELEMENT_TRAIN* el valor 100.000 y la otra constante *DIMENSION* los valores 2,4,6,8,10 14 y 20.

Como función distancia, utilizamos la distancia euclidiana entre dos puntos en el espacio n-dimensional, que será tomado este número de la constante *DIMENSION*

```
1  public double calcularDistancia(Item item,Item secondItem) {
2          double rval = 0.0, x, y;
3          double[] itemCoord = item.getCoordenadas();
4          double[] pivotCoord = secondItem.getCoordenadas();
5          for (int i = 0; i < Database.DIMENSION; i++) {
6              x = itemCoord[i];
7              y = pivotCoord[i];
8              rval += Math.pow(x - y, 2);
9          }
10           return Math.sqrt(rval);
11       }
```

En la línea 2 vemos que, como parámetros, se toman dos ítems que tienen sus coordenadas de números reales. Luego en la línea 10 vemos que le método retorna la distancia entre los dos vectores n-dimensionales pasados como parámetro, y calculado en el bucle for de la línea 5 a la 9.
Entonces, esta es la explicación básica necesaria para comprender la implementación del espacio métrico sintético. Que es utilizado a lo largo de todas las experimentaciones de la propuesta de este trabajo.

B2 – Espacio Métrico de Imágenes

El lenguaje Java y su interfaz de programación de aplicaciones (API) Java Advanced Imaging (JAI) nos permite crear aplicaciones para la representación, procesamiento y visualización de imágenes.
El lenguaje Java tiene ventajas como su bajo costo de desarrollo, la concesión de licencias y la independencia entre la plataforma (portabilidad).

Aquí las imágenes fueron procesadas utilizando [JAI 1] y también una breve referencia a la API utilizada en la experimentación: *JAI - "Java Advanced Imaging API"* versión 1.1.2 [JAI 4].

La API de JAI tiene como ventajas su flexibilidad y variedad de operadores para el procesamiento de imagen, por ejemplo el re-escalado de imágenes.
En JAI las imágenes pueden ser multidimensionales, es decir, con varios valores asociados a un único píxel, y pueden tener píxel entero o bien con valores de punto flotante, aunque hay restricciones sobre los tipos de imágenes que se pueden almacenar en el disco.
Los píxeles pueden ser empaquetados en diferentes formas o desempaquetados en los datos de la imagen matriz.
Diferentes modelos de color se pueden utilizar. Como uno puede esperar, con el fin de poder representar una variedad de datos de imagen, se debe tratar con una variedad de clases, que ya están en la API y muy bien documentadas.

Sabemos que a menudo existe la necesidad de acceder a los valores de píxel de una imagen con el fin de realizar alguna operación sobre la imagen - algoritmos de clasificación, por ejemplo, y a menudo requieren acceso a todos los píxeles en una imagen para la evaluación y clasificación.

Un método sencillo que puede ser usado para acceder a los píxeles de una imagen es a través del uso de iteradores. Los Iteradores que nos brinda la API JAI permiten el acceso a los píxeles en una imagen en un determinado orden.
Por ejemplo, una instancia de RectIter , que es un iterador dentro de la API JAI, escanea la imagen columna por columna, de la línea superior a la línea inferior, se actualiza automáticamente escaneado píxeles, permitiendo el acceso (de sólo lectura) para todos los píxeles en una banda o de un píxel puntual en la banda actual.

Otro iterador, RandomIter permite el acceso directo a un píxel usando las coordenadas X e Y especificadas por el usuario.

En el caso que necesitemos visualizar los objetos para "*comparar*" los resultados obtenidos, podemos utilizar la siguiente funcionalidad que nos brinda la API JAI, que proporciona un sencillo, pero extensible componente de visualización de imágenes, ejecutado por la clase DisplayJAI.
Este componente hereda de JPanel y puede utilizarse como cualquier otro componente de la API gráfica de Java, y puede ser utilizado como está o ampliado con diferentes propósitos.

Estos elementos son los que utilizaremos para trabajar con los objetos imágenes que me definen el espacio métrico de experimentación.

Descripción de código y explicaciones

A continuación estudiaremos los principales archivos fuentes del prototipo que acompañan este trabajo.

1) Espacio Métrico de Imágenes:
Por el momento, a nivel de pruebas de concepto, lo tengo modelado en los sig archivos:
EspacioMetricoImagenesImpl.java
JPEGImageFileFilter.java

Sobre JPEGImageFileFilter.java
Lo único que realiza es una validación, para asegurarme de que el archivo (u objeto del espacio métrico) es un archivo .jpeg
El tamaño base de las imágenes es: baseSize = 300;

Sobre EspacioMetricoImagenesImpl.java
Esta clase utiliza un muy simple algoritmo de similitud (*Naive Similarity Finder*) para comparar una imagen con todas los demás en el mismo directorio.

En ese mismo archivo, entonces tenemos:
a) Espacio Métrico de Imágenes: es donde se cargan todos los elementos del espacio métrico.
b) Función Distancia: euclidea.
c) Espacio Métrico para entrenamiento: es un subconjunto de elementos, tomados de forma aleatoria del espacio métrico de imágenes.

En el prototipo que acompaña este trabajo, se incluye una aplicación que trabaja como una "*prueba de concepto*" para validar la forma de trabajo del espacio métrico.

El punto de partida para la aplicación es, cuando se abre un pop-up para la carga de un archivo con una imagen que se utilizará como consulta y se inicia la aplicación con el constructor de la clase, que crea la interfaz gráfica de usuario e inicia la tarea de procesamiento de imagen.

Tomamos la imagen de referencia, la re-escalamos a (300,300) píxeles utilizando la "JAI scale" (que es la API de Java para trabajar con imágenes antes presentada) y luego le calculamos "su vector firma" para luego guardarla. Veamos un poco como calculamos su vector firma.

Este método que se muestra a continuación, calcula la firma y devuelve los vectores de la imagen de entrada. Tomamos memoria para el vector a retornar, y para cada uno de los 25 valores de la firma. Luego se promedian los píxeles a su alrededor. Tenemos que recordar que, la coordenada central se encuentra en proporciones de píxel para el momento de promediar los píxeles de su entorno.

```
1.  private Color[][] calcSignature(RenderedImage i) {
2.
3.      Color[][] sig = new Color[5][5];
4.
5.
6.      float[] prop = new float[]
7.              {1f / 10f, 3f / 10f, 5f / 10f, 7f / 10f, 9f / 10f};
8.      for (int x = 0; x < 5; x++)
```

```
9.          for (int y = 0; y < 5; y++)
10.             sig[x][y] = averageAround(i, prop[x], prop[y]);
11.     return sig;
12.  }
```

Y como vemos en este método para calcular el vector firma, tenemos que ver como se promedian los valores de los píxeles en torno a un punto central, esto lo realiza el método *averageAround(),* que promedia los valores de píxel en torno a un punto central y retorna la media como una instancia de Color. Los puntos de coordenadas son proporcionales a la imagen.

```
1.   Color averageAround(RenderedImage i, double px, double py) {
2.      // Get an iterator for the image.
3.      RandomIter iterator = RandomIterFactory.create(i, null);
4.      // Get memory for a pixel and for the accumulator.
5.      double[] pixel = new double[3];
6.      double[] accum = new double[3];
7.      //El tamaño de la zona de muestreo.
8.      int sampleSize = 15;
9.      // pixeles de muestreo.
10.       for (double x = px * baseSize - sampleSize;
                                          x < px * baseSize + sampleSize; x++) {
11.         for (double y = py * baseSize - sampleSize;
                  y < py * baseSize
12.               + sampleSize; y++) {
13.             iterator.getPixel((int) x, (int) y, pixel);
14.             accum[0] += pixel[0];
15.             accum[1] += pixel[1];
16.             accum[2] += pixel[2];
17.          }
18.       }
19.       // Los valores promedio acumulados
20.       accum[0] /= sampleSize * sampleSize * 4;
21.       accum[1] /= sampleSize * sampleSize * 4;
22.       accum[2] /= sampleSize * sampleSize * 4;
23.    return new Color((int)accum[0],(int)accum[1],(int) accum[2]);
24.   }
```

Ahora, necesitamos comparar la imagen "Query" con las demás del directorio, entonces ahora tenemos un componente *X* para almacenar imágenes en una pila, donde *X* es el número de imágenes en el mismo directorio que el original.

Entonces, para cada imagen, calculamos su "vector firma" y su distancia a la firma de referencia.
Es decir, primero re-escalamos cada imagen, y luego le calculo su vector firma. Este es el codigo:

```
1.  RenderedImage[] rothers = new RenderedImage[others.length];
2.  double[] distances = new double[others.length];
3.  for (int o = 0; o < others.length; o++) {
4.       rothers[o] = rescale(ImageIO.read(others[o]));
5.       distances[o] = calcDistance(rothers[o]);
6.  }
```

Veamos ahora como trabaja la función *calcDistance (RenderedImagen img),* que calcula la distancia entre las firmas de una imagen y la referencia uno.
El Vector Firma, para la imagen pasada como parámetro, se calcula en el interior del método.

```
1.   private double calcDistance(RenderedImage other) {
2.       // Calculamos el vector firma para la imagen parámetro
3.       Color[][] sigOther = calcSignature(other);
4. // vamos a calcular la suma de las
5. // distancias entre los valores RGB de los píxeles en la misma posición.
6.
7.       double dist = 0,tempDist;
8.       for (int x = 0; x < 5; x++)
9.           for (int y = 0; y < 5; y++) {
10.              int r1 = signature[x][y].getRed();
```

```
11.             int g1 = signature[x][y].getGreen();
12.             int b1 = signature[x][y].getBlue();
13.             int r2 = sigOther[x][y].getRed();
14.             int g2 = sigOther[x][y].getGreen();
15.             int b2 = sigOther[x][y].getBlue();
16.           tempDist = Math.sqrt((r1 - r2) * (r1 - r2) + (g1 - g2)
17.                   * (g1 - g2) + (b1 - b2) * (b1 - b2));
18.             dist += tempDist;
19.         }
20.       return dist;
21.   }
```

A continuación, ordenamos todos esos resultados, es decir, las distancias entre los elementos de la base de datos y nuestro elemento consulta "*Q*", de la sig manera para poder tener los "*K-Vecinos*" a mi consulta:

```
1.   // Ordenar los vectores "juntos".
2.        for (int p1 = 0; p1 < others.length - 1; p1++)
3.            for (int p2 = p1 + 1; p2 < others.length; p2++) {
4.                if (distances[p1] > distances[p2]) {
5.                    double tempDist = distances[p1];
6.                    distances[p1] = distances[p2];
7.                    distances[p2] = tempDist;
8.                    RenderedImage tempR = rothers[p1];
9.                    rothers[p1] = rothers[p2];
10.                     rothers[p2] = tempR;
11.                     File tempF = others[p1];
12.                     others[p1] = others[p2];
13.                     others[p2] = tempF;
14.               }
15.           }
```

B3 – The Color FERET Database

El programa FERET [JAI 5] se desarrolló desde 1993 a 1997, patrocinado por el Departamento de Defensa y Programa de Desarrollo de Tecnología de Defensa a través de la Agencia de Investigación Avanzada (DARPA) de los Estados Unidos.
Su misión principal era desarrollar la capacidad de reconocimiento facial automático que puede emplearse para ayudar a la seguridad, la inteligencia y la aplicación de la ley personal en el desempeño de sus funciones.

La base de datos de imágenes FERET (el corpus) fue montado por el gobierno para apoyar a un seguimiento y evaluación de las pruebas del reconocimiento de caras utilizando algoritmos de las pruebas y procedimientos. El último cuerpo, se utilizo en este trabajo y esta compuesto por 14.051 imágenes en escala de grises. Todas son imágenes de cabezas humanas tanto de vista frontal, como también con perfiles a la izquierda y a la derecha, o incluso con pequeñas variaciones.

En este trabajo se utilizaron dichas imágenes para conformar el espacio métrico de imágenes, y junto con los algoritmos antes presentados, realizar la validad de la propuesta mediante experimentación.

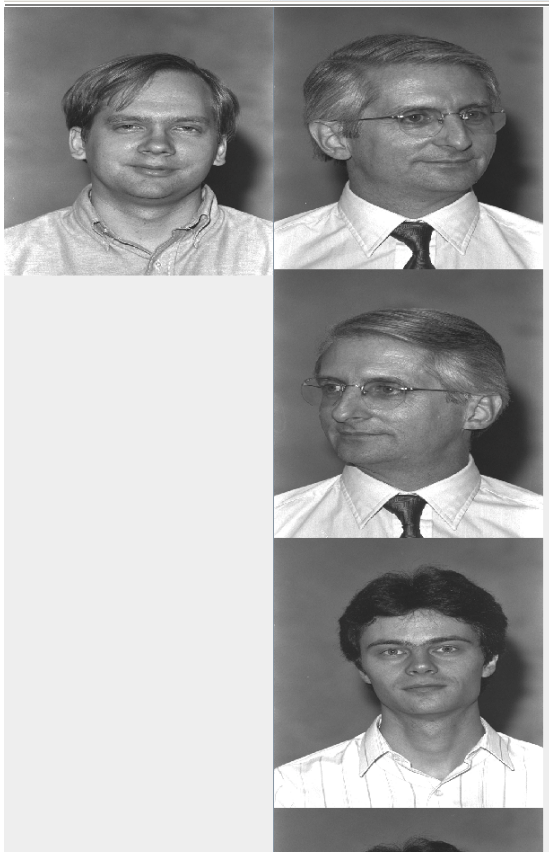
Apéndice C – Búsqueda en Espacios Métricos de Imágenes

Estos son los resultados de la búsqueda en Espacio Métrico de imágenes comentada al momento de la experimentación en este trabajo.
Recordemos que el espacio métrico fue reducido al conjunto de todas las imágenes en blanco y negro.
En la siguiente figura observamos del lado izquierdo a la imagen consulta, y del lado derecho al conjunto de resultados de búsquedas, junto con su nombre y su distancia a la consulta.

imagenGaleria_121.jpg
7.275

imagenGaleria_119.jpg
8.019

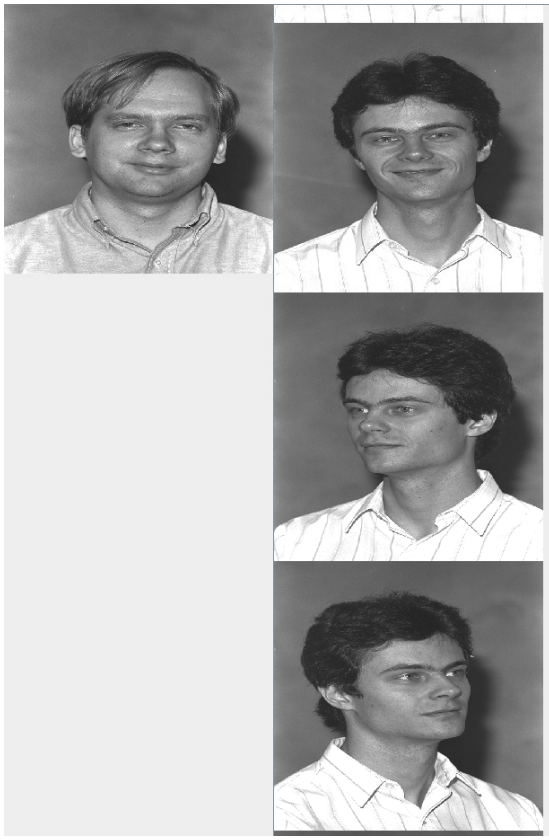
imagenGaleria_3.jpg
8.383



imagenGaleria_5.jpg
8.608

imagenGaleria_4.jpg
8.730

imagenGaleria_6.jpg
8.851



imagenGaleria_18.jpg
9.301

imagenGaleria_17.jpg
10.271

imagenGaleria_16.jpg
10.704