UNIVERSIDAD DEL BÍO-BÍO

FACULTAD DE CIENCIAS EMPRESARIALES

Departamento de Ciencias de la Computación y Tecnologías de la Información.

"Identificación mediante huellas digitales utilizando búsqueda por proximidad sobre el índice métrico M-Tree".

Diego Calderón Sandoval

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO DE EJECUCIÓN EN COMPUTACIÓN E INFORMÁTICA

CHILLÁN
JULIO DE 2008

UNIVERSIDAD DEL BÍO-BÍO

FACULTAD DE CIENCIAS EMPRESARIALES

Departamento de Ciencias de la Computación y Tecnologías de la Información.

"Identificación mediante huellas digitales utilizando búsqueda por proximidad sobre el índice métrico M-Tree".

Diego Calderón Sandoval

PROFESOR GUÍA : DR. GILBERTO GUTIERREZ

PROFESOR INFORMANTE : SR. MIGUEL PINCHEIRA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO DE EJECUCIÓN EN COMPUTACIÓN E INFORMÁTICA

CHILLÁN

JULIO DE 2008

Agradecimientos

Terminar esta tesis ha sido, hasta el momento, para mí el trabajo más importante y de mayor envergadura que haya encarado, pero no es el final de una etapa sino más bien el comienzo de la que considero será la etapa más importante de mi formación profesional.

Estoy muy agradecido especialmente de cada una de las personas que creyeron en mí, me dieron su apoyo, tanto emocional como intelectual, y que estuvieron siempre presentes durante este período de arduo estudio y trabajo, para ayudarme en forma desinteresada, darme su amistad, amor y paciencia.

Antes que nadie quiero agradecer a mis padres Marta Sandoval y Andrés Calderón, mis hermanos Roberto, Cristián y Carlos, y mi amor Cristina Lillo, no sólo por apoyarme en este trabajo, a costa de quitarnos tiempo juntos, sino que también porque sin ellos mi vida carecería de sentido. A Dios por darme la vida, y fortalecer mi espíritu cada día. A mi familia completa que siempre me apoyó y me aconsejó, aún sin entender demasiado mi trabajo. Quiero agradecer especialmente a mi profesor guía el Dr. Gilberto Gutiérrez por creer en mi aún sin conocerme lo suficiente, por haberme brindado todo lo que estuvo a su alcance y que el creyó que yo aprovecharía. A mi profesor informante por sus críticas constructivas y su ayuda que desde un comienzo estuvo dispuesto a brindarme, y a mis compañeros que siempre me dieron su apoyo.

Y por último quiero dar mis más cordiales agradecimientos a: José Saavedra ex-alumno de la Universidad de Chile, Alex Paredes ex-alumno de la Universidad del Bío-Bío, Dr. Marco Patella docente de la Universidad de Bolonia Italia, Clemens Maschner alumno de doctorado de la Universidad de München Alemania, Dra. Nora Reyes docente de la Universidad de San Luis Argentina y Jurgen Kramer docente de la Universidad de Marburg Alemania. Agradezco a cada uno de ellos su apoyo incondicional en cada una de sus áreas de conocimiento, que fueron primordiales para la realización de este proyecto.

Diego Calderón S.

A mis padres y hermanos
por hacer de mí lo que soy, y estar siempre a mi lado

A Cristina Lillo, mi amor
quien ha hecho de mi vida lo más hermoso.

Resumen

Muchas aplicaciones computacionales necesitan buscar información en una base de datos. Tradicionalmente la operación de búsqueda se ha aplicado a *datos estructurados* y las bases de datos tradicionales se construyen alrededor del concepto de búsqueda exacta. En la actualidad han surgido depósitos no estructurados de información y no sólo se consultan nuevos tipos de datos (texto libre, imágenes, audio y video), sino que además ya no es posible estructurar la información de la manera clásica y, aún cuando sea posible hacerlo, nuevas aplicaciones tales como minería de datos (*data mining*) requieren acceder a la base de datos por cualquier campo, no sólo aquellos marcados como "*claves*", como también resulta ser en el caso de identificación mediante huellas digitales.

Un concepto unificador es el de "búsqueda por similitud" o "búsqueda por proximidad", es decir buscar elementos de la base de datos que sean similares a un elemento de consulta dado. La similaridad es modelada usando una función de distancia (o métrica) que satisface ciertas propiedades y el conjunto de objetos es llamado un *espacio métrico*. En general, como la distancia es bastante costosa de calcular, el objetivo es reducir el número de evaluaciones de distancia.

La identificación de personas a través de huellas digitales es una técnica ampliamente utilizada especialmente por las policías de investigaciones de casi todos los países del mundo. Un proceso crítico de la identificación de huellas digitales es el de la búsqueda de un individuo dentro de una base de datos con gran cantidad de información, pues el tiempo de espera es notablemente alto, dependiendo de la cantidad de huellas almacenadas.

En este proyecto se trata de dar solución a esa gran problemática, utilizando un índice métrico denominado M-Tree, el cual permite realizar búsquedas por proximidad o similitud utilizando como cálculo de distancia entre dos huellas digitales el matching basado en minucias.

Para el proceso de matching se describe la aplicación de la Transformada de Hough Generalizada, para obtener unos valores específicos que permiten alinear dos huellas digitales y así poder optar a más apareamientos de minucias. Para el proceso de comparación de huellas, una vez alineadas, se describe un algoritmo basado en coordenadas polares.

Se ha implementado un algoritmo de matching de huellas digitales que arroja resultados aceptables considerando que se ha utilizado un algoritmo de extracción de minucias con muchas deficiencias, pero que promete muy buenos resultados mejorando dicho algoritmo. Además, se ha logrado implementar un interesante algoritmo de búsqueda por proximidad sobre espacios métricos. De esta forma se han obtenido muy buenos resultados en un tiempo mucho menor que los algoritmos de búsquedas actuales, en donde sólo se clasifican las huellas por su tipo, o simplemente realizan una búsqueda secuencial.

Este trabajo constituye un aporte valioso al desarrollo e implementación de algoritmos biométricos en Chile, pues esta área sólo se está incursionando en estos últimos años en nuestro país, y la propuesta de un algoritmo de búsqueda distinto a los actuales (en la misma área) da una base para futuros estudios o propuestas que mejoren lo que ya se ha logrado.

Índice General

Índice de Figuras.

Índice de Tablas.

Capítulo I: Introducción

Capítulo I – Introducción

1.1 Introducción

Con la evolución de las tecnologías asociadas a la información, nuestra sociedad está cada día más conectada electrónicamente. Labores que tradicionalmente eran realizadas por seres humanos, gracias a las mejoras tecnológicas, son realizadas por sistemas automatizados. Dentro de la amplia gama de posibles actividades que pueden automatizarse, aquella relacionada con la capacidad para establecer la identidad de los individuos ha cobrado importancia y como consecuencia directa, la *biometría* se ha transformado en un área emergente **[Mill94]**. La biometría es la ciencia que se dedica a la identificación de individuos a partir de una característica anatómica o un rasgo de su comportamiento. Una característica anatómica tiene la cualidad de ser relativamente estable en el tiempo, tal como una huella dactilar, la silueta de la mano, patrones de la retina o el iris. Un rasgo del comportamiento es menos estable, pues depende de la disposición psicológica de la persona, por ejemplo la firma. No cualquier característica anatómica puede ser utilizada con éxito por un sistema biométrico. Para utilizarlo, se debe cumplir con las siguientes características: *Universalidad, Unicidad, Permanencia y Cuantificación* **[HJ98]**. Un indicador biométrico que satisface estos requisitos es la huella dactilar. Este indicador ha sido utilizado por los seres humanos para identificación personal por más de cien años **[Ele73]**. En la actualidad las huellas dactilares representan una de las tecnologías biométricas más maduras y son consideradas pruebas legítimas de evidencia criminal en cualquier corte del mundo.

Una huella dactilar es la representación de la morfología superficial de la epidermis de un dedo. Posee un conjunto de líneas que, en forma global, aparecen dispuestas en forma paralela. Sin embargo, estas líneas se intersectan y a veces terminan en forma abrupta. Los puntos donde éstas terminan o se bifurcan se conocen técnicamente como *minucias*.

Las huellas dactilares, una vez reconocidas sus minucias son almacenadas en una base de datos junto a los datos personales del individuo, y anteriormente esa información es insertada en un índice métrico denominado M-Tree, el cual posteriormente mejoraría el tiempo de búsqueda de una cierta huella digital.

Para concluir si una huella digital se corresponde o no con alguna huella almacenada en la base de datos, se lleva a cabo un procedimiento de búsqueda en el índice métrico, para el cual es necesario un cálculo de distancia, denominado *matching* o comparación de minucias. El matching de huellas dactilares consiste en encontrar el grado de *similaridad* entre dos vectores de características cuyas componentes representan a las minucias de cada huella. Este cálculo de distancia es utilizado por el índice métrico para retornar todas aquellas huellas, junto con los datos personales del individuo, que estén dentro de un rango (10 o más minucias), sin la necesidad de comparar la huella dactilar de entrada con cada una de las huellas almacenadas, minimizando así el tiempo de respuesta.

Pero ¿para qué diseñar un sistema informático de reconocimiento de huellas dactilares si ya existen en el mercado?. Sería bueno ofrecer las bases de un sistema dactiloscópico, con tecnología nacional, donde los chilenos: la policía científica, las comisarías, el ejército, y las compañías sepan cómo están clasificando las huellas de los ciudadanos o de sus empleados. Además, porque los sistemas que venden empresas extranjeras son cajas negras, que no permiten adaptarse en forma exacta a las necesidades.

En Chile, en Junio de 2007 se ha iniciado el primer proyecto científico de este tipo denominado “Desarrollo Nacional de Algoritmos que faciliten la Utilización de Soluciones de Biometría en Chile”, más conocido como Algoritmos Biométricos **[Lan08]**, con una duración de 2 años. En él participan Red Universitaria Nacional REUNA, como institución líder, Universidad Católica del Norte (UCN), Universidad de Tarapacá (UTA), Universidad de Atacama (UDA), Universidad de La Frontera (UFRO) y la empresa Biokey. En donde cada universidad participante ha asumido una función en el desarrollo del proyecto, el que se ha dividido en las fases de pre-procesamiento y extracción de minucias, control de calidad, algoritmos de matching, y algoritmos de búsqueda y clasificación.

1.2 Glosario

Bifurcación: Es una rama que se forma cuando se separan dos crestas en una imagen de huella digital.

Biometría: Medición de características tanto físicas como de comportamiento, propias de una persona, como pueden ser la huella digital, lectura del iris, reconocimiento de la voz, etc.

Crestas: Son las marcas sobresalientes que se encuentran en la yema de los dedos.

Datos biométricos: Información extraída de una característica propia de una persona, la que es usada para construir una plantilla con la que se compararán posteriormente las siguientes mediciones.

Enrolamiento: Proceso mediante el cual se colecciona las minucias de una huella digital, preparándolas para su posterior almacenamiento. Dichas minucias identifican en forma única a una persona.

Extracción: El proceso para transformar la imagen de huella digital en datos biométricos.

FAR: False-acceptance rate, que se define como la frecuencia relativa con que un impostor es aceptado como un individuo autorizado

Fin de cresta: Un punto en el cual termina una cresta.

FRR: False-rejection rate, definida como la frecuencia relativa con que un individuo autorizado es rechazado como un impostor.

Gradiente: Para una imagen *g(x, y)* el gradiente en el punto con coordenadas *(x, y)* es el vector cuyas componentes cartesianas son las tasas de cambio en las tonalidades de gris en las direcciones definidas por *x* e *y*. **[GW93]**.

Matching: Procedimiento de alineación y comparación de huellas digitales.

Minucia: Pequeños detalles encontrados en la imagen de huella digital, los que pueden ser bifurcaciones o fines de línea.

Query: Conjunto de minucias pertenecientes a una huella digital de entrada.

Surco: Son las marcas encontradas entre dos crestas en una imagen de huella digital.

Template: (Plantilla). Conjunto de minucias pertenecientes a una huella digital que se encuentran almacenados en la base de datos.

1.3 Definición del problema

Los sistemas de identificación de huella digital no son muy frecuentemente utilizados en la actualidad, por su constante estudio investigativo de mejora en la rapidez de respuesta, v/s la necesidad de una buena precisión en la identificación. Esto genera un área de investigación que con varios aportes a nivel mundial, todavía no se llega a consenso de qué algoritmos son los más eficientes y eficaces y cuáles entregan los mejores resultados.

La técnica mas utilizada, en esta área, para segmentar las bases de datos, y así reducir la cantidad de comparaciones, es la de preclasificar las huellas digitales dependiendo de su tipo. Por ejemplo el sistema norteamericano (clasificación de Henry) que define las siguientes clases: arco (*arch*), arco de carpa (*tented arch*), lazo izquierdo (*left loop*), lazo derecho (*right loop*) y anillo de cresta (*whorl*). La cual es una muy buena técnica, pero que aún sigue realizando demasiadas comparaciones para entregar un resultado, pues al tener miles o millones de huellas almacenadas solo las dividen en 5 grupos, y realiza la comparación con cada una de las huellas del grupo seleccionado. Por lo tanto hoy en día existe la necesidad de tener un procedimiento de búsqueda aún más eficiente que obtenga buenos resultados sin la necesidad de hacer demasiadas comparaciones, las cuales son de un costo de procesamiento muy alto, y hacen que los tiempos de respuesta de este tipo de búsquedas sean excesivamente elevados.

1.4 Descripción de la solución

Como ha sido mencionado anteriormente el problema principal es la identificación de una persona a través de su huella digital, en sistemas de grandes bases de datos, en un tiempo mínimo (dependiendo de la cantidad de información almacenada). Para dar solución a este problema es que se propone como proyecto el diseño e implementación de un sistema que permita realizar estas búsquedas de una forma más eficiente a las actuales, en donde no sea necesario realizar demasiadas comparaciones de huellas digitales para poder obtener la identidad de una persona. Y la creación de un sistema de matching de huella digital, implementando un algoritmo de alineación de huellas y otro de comparación.

Dicho sistema cuenta con una técnica de indexación capaz de soportar la ejecución de consultas de similitud a través de un índice métrico denominado M-Tree, inspirado en algunos árboles métricos que reducen los cálculos de distancia y en los métodos de acceso multidimensionales como R-Tree **[Gut84]** y sus variantes, que permiten reducir el costo de acceder a una página en disco. M-Tree es un árbol balanceado, capaz de ocuparse de archivos de datos dinámicos, y como tal no requiere de reorganizaciones periódicas.

El matching de huellas digitales cuenta con dos etapas, una de alineación y otra de comparación **[JR97]**. La primera está basada en la transformada de Hough, en donde a través de esta técnica es posible obtener los valores necesarios para rotar y trasladar las minucias de la huella de entrada (*Query*) con respecto a la huella digital almacenada (*Template*), ya que en distintas capturas se pueden variar las ubicaciones y grado de inclinación local de las minucias. La segunda etapa es la de comparación, en donde los conjuntos de minucias de cada huella una vez alineados, que son representados por coordenadas cartesianas, son convertidas al sistema de coordenadas polares. Luego se procede a calcular la distancia de edición entre las coordenadas del conjunto de minucias Query (en representación polar) y todos los conjuntos de minucias Template que el índice métrico estime convenientes, además de calcular la diferencia entre los ángulos de orientación local de las minucias. Estos valores son ponderados con valores arbitrarios para disminuir la posibilidad de errores por diferencias que puedan existir en el proceso de extracción de minucias. Si la “distancia total” es menor que un valor umbral diremos que la huella Query corresponde o es similar a la huella Template.

Capítulo II: Marco Teórico

Capítulo II - Marco Teórico

El marco teórico que envuelve el material necesario para poder cumplir con los objetivos fijados dentro de este proyecto, se dividen en 5 temas, para facilitar la lectura y comprensión de las materias estudiadas que sirven para el posterior desarrollo del mismo.

Los temas están divididos de la siguiente forma: Huella Digital, Verificación e Identificación de huellas digitales, Espacios métricos, Algoritmos de indexación, y por último Índice métrico "M-Tree".

2.1 Huella Digital

Las huellas digitales son características exclusivas de los primates. En la especie humana se forman a partir de la sexta semana de vida intrauterina y no varían en sus características a lo largo de toda la vida del individuo **[RKCJ96].**

Los dibujos que aparecen visibles en la epidermis, está demostrado científicamente y comprobado por la experiencia, que son **perennes, inmutables** y **diversiformes**:

- Son **perennes**, porque desde que se forman en el sexto mes de la vida intrauterina, permanecen indefectiblemente invariables en número, situación, forma y dirección hasta que la putrefacción del cadáver destruye la piel.

- Son **inmutables**, ya que las crestas papilares no pueden modificarse fisiológicamente. Si hay un traumatismo poco profundo, se regeneran y si es profundo, las crestas no reaparecen con forma distinta a la que tenían, sino que la parte afectada por el traumatismo resulta invadida por un dibujo cicatrizal.

- Son **diversiformes**, pues no se ha hallado todavía dos impresiones idénticas producidas por dedos diferentes.

Son únicas e irrepetibles aún en gemelos idénticos, debido a que su diseño no está determinado estrictamente por el código genético, sino por pequeñas variables en las concentraciones del factor del crecimiento y en las hormonas localizadas dentro de los tejidos. Cabe señalar que en un mismo individuo la huella de cada uno de sus dedos es diferente. En la Figura 2.1 se muestran algunas características de una huella digital.

Figura 2.1: Características de una huella digital

2.1.1 Biometría de huellas digitales.

El termino biometría viene del griego "*bio*" que significa vida y "*metria*" que significa medida o medición, de acuerdo al diccionario de la real academia de la lengua española biometría es el estudio mensurativo o estadístico de los fenómenos o procesos biológicos, sin embargo más recientemente y para el tema que nos concierne el significado de biometría es el conjunto de métodos automatizados que analizan determinadas características humanas para identificar o autentificar personas **[Gal07]**.

La biometría aprovecha que hay ciertas características biológicas o conductuales singulares e inalterables, por lo que pueden ser analizados y medidos para crear una huella biométrica. Estas características son difíciles de perder, transferir u olvidar y son perdurables en el tiempo.

La "biometría informática" es la aplicación de técnicas matemáticas y estadísticas sobre los rasgos físicos o de conducta de un individuo, para "verificar" identidades o para "identificar" individuos. En las tecnologías de la información (TI), la autentificación biométrica se refiere a las tecnologías para medir y analizar las características físicas y del comportamiento humanas con propósito de autentificación. Las huellas dactilares, las retinas, el iris, los patrones faciales, de venas de la mano o la geometría de la palma de la mano, representan ejemplos de características físicas (estáticas), mientras que entre los ejemplos de características del comportamiento se incluye la firma, el paso y el tecleo (dinámicas). La voz se considera una mezcla de características físicas y del comportamiento, pero todos los rasgos biométricos comparten aspectos físicos y del comportamiento **[Gal07]**.

Los rasgos usados por sistemas biométricos deben tener las siguientes características:

- Universalidad: todas las personas deben poseerlos.
- Unicidad: no debe haber dos personas con los rasgos seleccionados iguales.
- Permanencia: deben ser invariantes con el tiempo.
- Cuantificación: admiten medidas cuantitativas.

Las huellas dactilares son el rasgo biométrico que mejor cumple todas estas características. Realmente, son un buen argumento para quienes creen que fuimos creados por un ser superior (parece que Dios nos ha dotado de un "código de barras") **[Rod07]**.

2.1.2 Procedimiento de captura y enrolamiento

En 1960, la sede del FBI en Londres y el Departamento de Policía de París comenzaron estudios sobre los sistemas automáticos de identificación de huellas dactilares. Con la introducción de la tecnología, los archivos fueron clasificados en archivos criminales automatizados, y hoy en día toda esa información está en formato digital **[HHD07]**. A pesar de que se crearon varias extensiones de los Rasgos de Galton **[HJ98]**, la mayoría no se usan en los sistemas automáticos de identificación de huellas dactilares. De hecho, para la representación de huellas dactilares y su emparejamiento, el FBI sólo usa las bifurcaciones y la terminación de crestas, las que se muestran en la Figura 2.2.

Con este conjunto de puntos, el software biométrico de huella digital genera un modelo en dos dimensiones, según se muestra en la Tabla 2.1, siendo éste modelo representado por las ubicaciones de las minucias dentro de un sistema de coordenadas cartesianas y los ángulos que forman estas minucias con el eje *x*, las cuales se almacena en una base de datos, con la debida referenciación de la persona que han sido objeto del estudio.

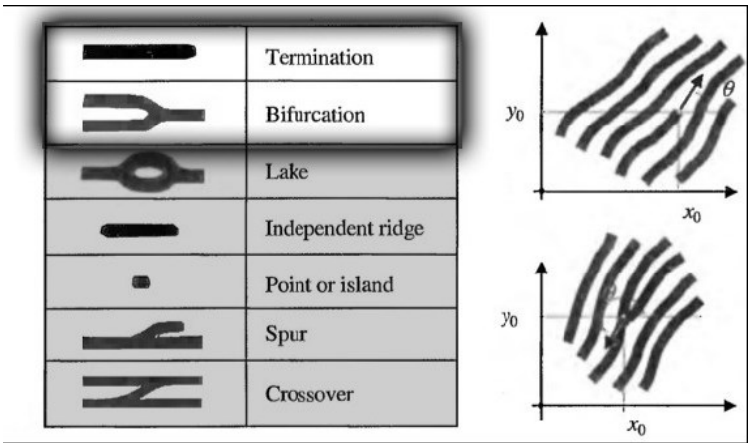


Figura 2.2: Tipos de minucias

Para ello, la ubicación de cada punto característico o minucia se representa mediante una combinación de números *(x, y)* dentro de un plano cartesiano, los cuales sirven como base para crear un conjunto de vectores que se obtienen al unir las minucias entre sí mediante rectas cuyo ángulo y dirección generan el trazo de un prisma de configuración única e irrepetible. Para llevar a cabo el proceso inverso o verificación dactilar, se utilizan estos mismos vectores, no imágenes.

Cada minucia se representa por la 4-tupla *{x, y, t, θ}* en donde como se mencionaba anteriormente *x* e *y* son las coordenadas de la posición de la minucia, *t* es el tipo de minucia (terminación o bifurcación), y *θ* es el ángulo de la minucia formado por la orientación de la minucia con respecto al eje *x* **[IG07]**.

			
El dedo es leído por un lector de huellas.	La imagen es procesada, y luego se identifican las minucias.	Luego se genera una plantilla con estos puntos característicos.	El lector guarda y reconoce un conjunto de números que solo podrán ser reconocidos como una plantilla.

Tabla 2.1: Modelo de 2 dimensiones de una huella digital.

2.1.3 Adquisición y reconstrucción de la huella

Para esta etapa fue necesario utilizar una librería descrita en **[BP05]**, que permite almacenar las minucias de una huella digital posteriormente a un exhaustivo procedimiento de digitalización de la imagen de la huella digital **[Baez05]**. Este procedimiento cuenta con diversas etapas que son explicadas a continuación:

2.1.3.1 Mejoramiento de la imagen

El objetivo de esta etapa es obtener una versión mejorada de la imagen adquirida para favorecer el desempeño en las labores de enrolamiento y verificación, lo que quiere decir que se necesita obtener una estructura más nítida de las crestas y surcos presentes en la imagen de la huella, lo que facilitará el cálculo de la cuenta de crestas y la obtención de información de cada minucia. La clasificación anterior permite definir con mayor precisión el objetivo de la etapa de mejora: realzar la nitidez de la estructura de crestas y surcos de las imágenes de huellas digitales en las regiones recuperables y eliminar de las etapas de procesamiento a las regiones irrecuperables.

Para alcanzar el objetivo antes mencionado se considera la aplicación en serie de cinco etapas:

2.1.3.1.1 Normalización de la imagen

El objetivo de la etapa de normalización de la imagen de huella digital es reducir las variaciones en los niveles de grises de las crestas y los surcos sin reducir la información

útil, facilitando el trabajo en las tareas posteriores. En la Figura 2.3 se muestra un ejemplo de una imagen normalizada.

a) b)

Figura 2.3: a) Huella sin normalizar. b) Huella normalizada

La imagen normalizada $N(x,y)$ de la imagen original $I(x,y)$ se obtiene de imponer a ésta última un valor medio y una varianza a priori.

Sea μ el valor medio y σ la desviación estándar del nivel de gris de la imagen original $I(x,y)$, sea μ_0 el valor medio y σ_0 la desviación estándar deseadas para $N(x,y)$, la ecuación (2-1) entrega lo deseado:

$$N(x,y) = \mu_0 + \frac{\sigma^0}{\sigma} \cdot abs(I(x,y) - \mu) \qquad (2\text{-}1)$$

donde *abs()* denota el valor absoluto.

2.1.3.1.2 Cálculo del campo de orientación

El campo de orientación de una imagen de huella digital representa la direccionalidad de las crestas de la huella, como se muestra en la Figura 2.4. Esta etapa juega un papel muy importante en el análisis de imagen de huella digital.

Existe un método el cual estima la dirección de la estructura de crestas y surcos de una vecindad de píxeles en torno al de coordenadas *(x, y)* directamente a partir de la información de las componentes de la imagen gradiente $G_x(x, y)$ y $G_y(x, y)$ en esa vecindad **[RJ92]**.

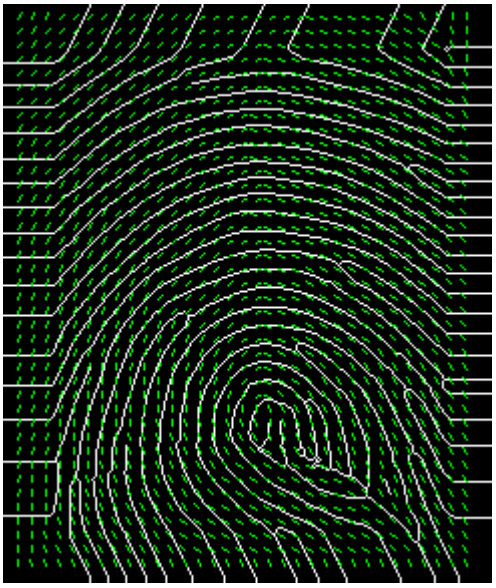


Figura 2.4: Campo de orientación de una huella digital

La solución propuesta considera lo siguiente: se desea encontrar un vector unitario que defina una dirección para una zona de píxeles. También se desea que ese vector sea "el que mejor" represente la dirección de la tangente a la estructura de crestas y surcos de esa zona.

Para encontrar ese vector *v* se buscará otro vector *g*: el asociado a la dirección del gradiente de esa zona. Es claro que el ángulo de *v* (respecto de un eje fijo) es igual al ángulo de $g \pm \frac{\pi}{2}$, ya que por definición ambos vectores son ortogonales.

Sea $W \times V$ la dimensión de la zona considerada para encontrar la dirección del gradiente, y θ el ángulo que define la dirección de *g* en esa vecindad (medido con respecto al eje horizontal en sentido contrario a los punteros del reloj). Entonces, θ queda definido por (2-2).

$$\theta = \frac{1}{2} \cdot \tan^{-1}\left(\frac{\sum_{i=1}^{v}\sum_{j=1}^{w} 2 \cdot G_x(i,j) \cdot G_y(i,j)}{\sum_{i=1}^{v}\sum_{j=1}^{w} \left(G_x^2(i,j) - G_y^2(i,j) \right)} \right) \qquad (2\text{-}2)$$

Para obtener la imagen direccional es necesario particionar la imagen *N(x,y)* en bloques disjuntos de $W \times V$ píxeles. Luego se realiza el cálculo de (2-2) en cada partición y se almacena en D (luego de sumar $\pm\frac{\pi}{2}$ radianes en cada posición).

Como se puede apreciar en (2-2) y por lo mencionado anteriormente, es necesario en primer lugar calcular la *imagen gradiente*. Una buena alternativa para tal efecto es utilizar los operadores de *Sobel*[1].

2.1.3.1.3 Filtro low-pass

La presencia de ruido y de crestas y surcos dañados en la imagen *I(x,y)* se mantiene en la imagen normalizada *N(x,y)*. Lo que implica que la información almacenada en *D(x,y)* no será siempre correcta. Para solucionar este inconveniente se utiliza un filtro (low-pass) para modificar las direcciones locales incorrectas al filtrar la componente *x* y la componente *y* de la imagen direccional. Para llevar a cabo el filtrado low-pass, *D(x,y)* necesita ser convertida a un campo de vectores continuo, para ello se definen las componentes $D_x(x,y)$ y $D_y(x,y)$ del campo de vectores *D(x,y)* como se ve en (2-3):

$$D_x(x,y) = \cos(2 \cdot D(x,y))$$
$$D_y(x,y) = \operatorname{sen}(2 \cdot D(x,y)) \qquad (2\text{-}3)$$

Con el campo de vectores resultantes, el filtrado low-pass puede ser realizado como se muestra en (2-4)

$$\widetilde{D}_x(x,y) = W(D_x(x,y))$$
$$\widetilde{D}_y(x,y) = W(D_y(x,y)) \qquad (2\text{-}4)$$

donde W es el filtro low-pass 2D.

[1] El operador *Sobel* calcula el gradiente de la intensidad de brillo de cada píxel dando la dirección del mayor incremento posible (de negro a blanco) además calcula el monto de cambio en esa dirección, es decir, devuelve un vector.

Luego de aplicar el filtrado low-pass, éste entrega las nuevas componentes, $\widetilde{D}_x(x,y)$ y $\widetilde{D}_y(x,y)$ que deben agregarse a la fórmula (2-2) (del cálculo del ángulo de la dirección del bloque) de la siguiente manera:

$$D(i,j) = \frac{1}{2} \cdot tan^{-1} \cdot \left(\frac{\widetilde{D}_y(x,y)}{\widetilde{D}_x(x,y)} \right) \tag{2-5}$$

Estas componentes se traducirán finalmente en el *campo de orientación suavizado*.

2.1.3.1.4 Estimación de la frecuencia de cresta

La *imagen de frecuencia F(x,y)* representa la frecuencia local de los surcos (y crestas) en una huella digital.

El primer paso en la etapa de estimación de frecuencia es dividir la imagen en bloques de tamaño $W \times W$.

El siguiente paso es proyectar los valores de escala de gris de todos los píxeles ubicados en cada bloque a lo largo de una dirección ortogonal a la orientación local de los surcos. Esta proyección es una figura sinusoidal con los puntos locales mínimos correspondientes a los surcos de la huella. Una explicación gráfica de eso puede verse en la Figura 2.5.

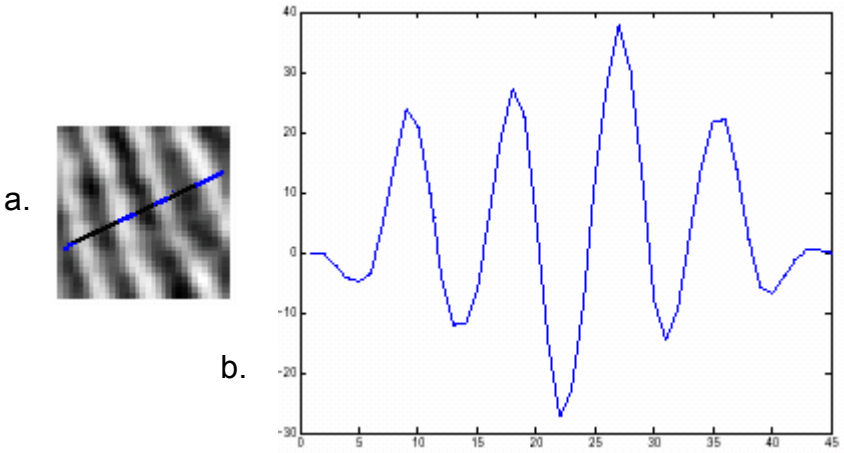


Figura 2.5: a) Bloque de 32x32 con una línea ortogonal a la orientación de los surcos. b) La onda sinusoidal proyectada del bloque.

Por tanto, para cada bloque centrado en *(i, j)*, se debe generar una ventana orientada de tamaño $l \times w$ en la dirección del surco.

La obtención de la proyección viene dada por la fórmula (2-6)

$$X[k] = \frac{1}{w} \cdot \sum_{d=0}^{w-1} G(u,v), \quad k = 0,1,...,l-1 \tag{2-6}$$

donde:

$$u = i + \left(d - \frac{w}{2}\right) \cdot \cos(O(i,j)) + \left(k - \frac{l}{2}\right) \cdot \text{sen}(O(i,j))$$
$$v = j + \left(d - \frac{w}{2}\right) \cdot \text{sen}(O(i,j)) - \left(k - \frac{l}{2}\right) \cdot \cos(O(i,j)) \tag{2-7}$$

El vector X[] (2-6) contendrá los valores de la proyección, computacionalmente hablando.

Si no se encuentran minucias o puntos singulares en el bloque, la proyección forma una onda sinusoidal *discreta*.

Sea *T(i, j)* el número de pixeles promedio entre dos picos consecutivos de la proyección (o dicho de otra forma, el espacio entre crestas), entonces la frecuencia de crestas *F(i, j)* para un bloque centrado en el pixel *(i, j)* se define en 2-8:

$$F(i,j) = \frac{1}{T(i,j)} \tag{2-8}$$

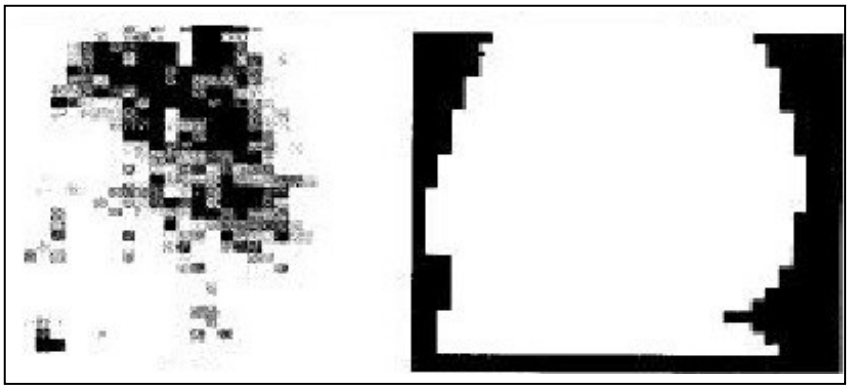
Para una imagen de huella digital escaneada a una resolución fija, el valor de la frecuencia de crestas y surcos local está entre cierto rango. Para una imagen de 500 dpi el rango será [1/3, 1/25]. Utilizando esta información, si el valor estimado de la frecuencia está fuera de este rango, se dirá que no fue posible obtener una *frecuencia válida*.

Cuando las crestas y surcos de la huella están corruptos la proyección no forma una curva sinusoidal bien definida, por lo que la frecuencia puede verse alterada. Si bien esta situación no parece ser un problema, sí lo es si pensamos que una minucia u otro punto singular en la huella también alteran la sinuosidad de esta curva. Para solucionar este

problema y no enmascarar la zona, el bloque necesita ser interpolado a partir de la frecuencia de sus vecinos (que tengan una frecuencia válida).

2.1.3.1.5 Estimación de la región de máscara

Debido a que la imagen contiene "ruido" de fondo, el algoritmo puede generar *minucias* fuera del área ocupada por la huella. Para evitar este problema, se selecciona el área de imagen, definida por todos los bloques de 16×16, en la que existe una alta variación del nivel de grises en la dirección normal de las *crestas* existentes (el campo orientación normal de las *crestas* se había calculado previamente). Por consecuencia, el área de la imagen con ruido, que será excluida en las siguientes etapas, se define por bajas variaciones en todas las direcciones. En la Figura 2.6 se muestran las variaciones de una huella y la región de interés obtenida a partir de esta.



a) b)

Figura 2.6. a) Variaciones de la huella. b) Región de interés

2.1.4 Mejoramiento de la calidad de la imagen

El desempeño de los algoritmos de extracción de minucias y otras técnicas de reconocimiento de huellas digitales está directamente relacionado con la calidad de la imagen de la huella digital de entrada. Jain y otros **[Lop04]** proponen un método efectivo para el mejoramiento de la imagen basándose en *filtros de Gabor* **[Gab46]**.

Los métodos de mejoramiento de huellas basados en los filtros de Gabor han sido ampliamente usados para facilitar múltiples situaciones sobre huellas digitales tales como la comparación de huellas y la clasificación. Los filtros de Gabor son filtros de *paso de banda*

que tienen la propiedad de ser sensibles a la frecuencia y a la orientación, lo que quiere decir que los filtros pueden ser refinados con valores de frecuencias y de orientación específicos.

Basándose en la orientación local y la frecuencia de surcos alrededor de cada píxel, el filtro de Gabor es aplicado a cada píxel de la imagen. El efecto del filtro es que mejora los surcos orientados en la dirección de la orientación local y suaviza (disminuye) todo lo que esté orientado en forma diferente.

La función de Gabor usada es la que se presenta en (2-9):

$$G(x,y;\theta,f)=\exp\left(\frac{-1}{2}\cdot\left(\frac{x_\theta^2}{\sigma_x^2}+\frac{y_\theta^2}{\sigma_y^2}\right)\right)\cdot\cos(2\cdot\pi\cdot f\cdot x_\theta),$$

$$x_\theta = x\cdot\cos(\theta)+y\cdot\operatorname{sen}(\theta),$$

$$y_\theta = -x\cdot\operatorname{sen}(\theta)+y\cdot\cos(\theta), \qquad (2\text{-}9)$$

donde θ es la orientación del filtro de Gabor, f es la frecuencia, σ_x y σ_y son las desviaciones estándar del *envoltorio Gaussiano* **[Tha03]** a lo largo de los ejes x e y, respectivamente. Dado que estos valores están basados en datos empíricos y a medida que mayores son los valores más resistente al ruido es el algoritmo, se utilizarán, para comenzar, valores arbitrarios que pueden variar (pueden producirse surcos falsos como causa de utilizar un valor incorrecto). De hecho en el algoritmo original descrito en **[Hong98]** estos valores fueron fijados en 4 y 4, respectivamente.

Y finalmente, la aplicación del filtro de Gabor para obtener la imagen mejorada *E(x,y)* resulta como en (2-10):

$$E(x,y)=\sum_{u=-\frac{w_x}{2}}^{\frac{w_x}{2}}\sum_{v=-\frac{w_y}{2}}^{\frac{w_y}{2}}G(u,v;O(i,j),F(i,j))\cdot N(i-u,j-v), \qquad (2\text{-}10)$$

donde O es la imagen de orientación, F es la imagen de frecuencia, N es la imagen normalizada, y w_x y w_y son el ancho y alto de la máscara del filtro de Gabor, respectivamente.

2.1.5 Extracción de minucias

2.1.5.1 Binarización

Se refiere a la transformación de la imagen de escala de grises a blanco y negro puros. Ver Figura 2.7.

a) b)

Figura 2.7: a) Huella digital original b) Huella digital binarizada

Muchos algoritmos de extracción de minucias operan en imágenes binarias donde hay sólo dos niveles de interés: los píxeles negros representan las crestas y los blancos representan los valles. La binarización mejora el contraste entre crestas y valles en la imagen de la huella y en consecuencia facilita la extracción de minucias.

La binarización es un filtro muy simple que dado un cierto umbral, determinado a priori, los tonos se transformarán a negro bajo el umbral y blanco sobre éste.

Tiene sentido aplicar un algoritmo tan básico como este puesto que la variación de tonalidades en los surcos sigue siempre el mismo patrón, esto es, la menor tonalidad estará siempre relacionada con un surco y la mayor a una cresta.

2.1.5.2 Adelgazamiento

La última etapa de filtrado de la imagen antes de extraer las minucias es el adelgazamiento.

Esta técnica considera el adelgazamiento (thinning) de las crestas de la imagen normalizada de tal forma que las crestas tengan por ancho un píxel.

Mediante *operadores morfológicos* o relaciones en vecindades es posible adelgazar la estructura binarizada de las crestas. El filtro sucesivamente erosiona los píxeles en negro hasta que tienen un ancho mínimo. Cada iteración comienza examinando el vecindario de cada píxel de la imagen binaria, y basado en un criterio particular de borrado de píxeles, chequea cuando un píxel debe ser borrado o no.

La aplicación del algoritmo de adelgazamiento en imágenes de huellas digitales debe preservar la conectividad en la estructura de crestas. La imagen adelgazada es posteriormente usada en la extracción de minucias. En la Figura 2.8 se puede apreciar el resultado de aplicar el filtro de adelgazamiento a una imagen.

a) b)

Figura 2.8: a) Huella digital original b) Huella digital adelgazada

2.1.5.3 Extracción de características

En esta búsqueda sólo se consideran las minucias de término y las de bifurcación de crestas. Para llevar a cabo la verificación de huellas dactilares se creará un vector de características cuyas componentes tendrán información asociada a la representación estándar de las

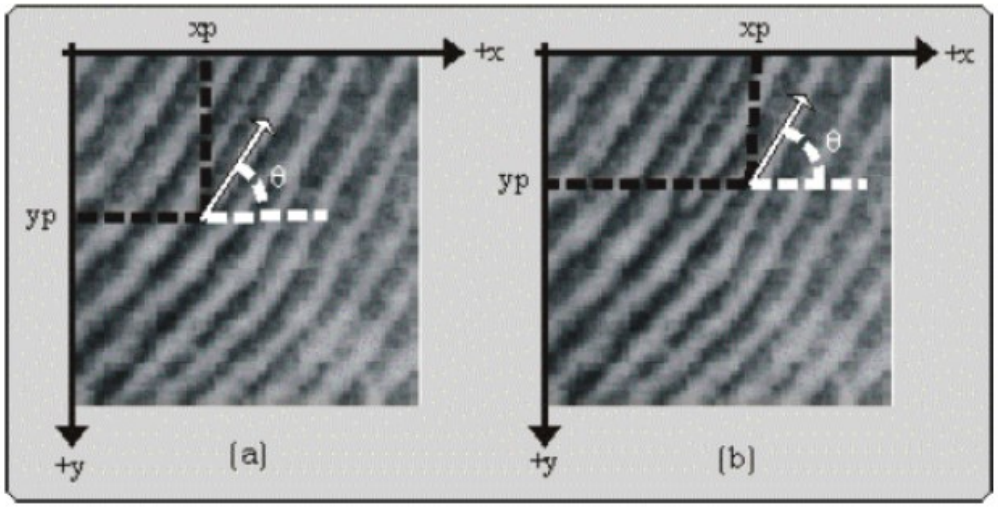


Figura 2.9: Posición y orientación de una minucia de bifurcación (a) y terminación (b).

minucias de una huella. Cada componente del vector de características es una estructura donde se encuentran las coordenadas de posición de la minucia, su orientación en la cresta (ver Figura 2.9) que será la misma que la de la cresta a la que pertenece y, eventualmente, su tipo. La estructura del vector de características resulta adecuada para las labores de confrontación debido a que:

- Las minucias capturan mucha información local, además de definir en conjunto un alto poder de discriminación (son patrones únicos para cada persona).
- Las representaciones basadas en minucias son de almacenamiento eficiente y son adecuadas para algoritmos de confrontación (verificación de autenticidad entre la huella de un sujeto y un conjunto de huellas, refiriéndonos a huella como el vector de características, claro está) automatizados.
- La detección de minucias es relativamente robusta a varias fuentes de degradación de la huella.

Para lograr obtener el conjunto de minucias de una huella digital, es necesario realizar un procesamiento previo sobre la imagen que consiste en obtener el campo de direcciones, el cual es recibido como parámetro por el método `ExtraerConjuntoDeMinucias()`.

Entradas: La imagen limpia y la imagen direccional.
Salida: Un conjunto de minucias
Precondición: La imagen debe estar limpia, en otras palabras, ya se le aplicaron todos los filtros necesarios.
Detalles:
Alto: Alto de la imagen procesada
Ancho: Ancho de la imagen procesada

```
Para n = 1 hasta Alto
   Para m = 1 hasta Ancho
      Si pixel(m, n) es negro
      Verificar los 8 vecinos de (m,n) utilizando la matriz de 3 x 3 y
      acumular los pixeles oscuros que encuentre en el vecindario
         Si resultado del acumulador es 1
            encontramos un fin de línea, se agrega al conjunto de minucias.
         Si resultado del acumulador es 2
            encontramos un borde (no es una minucia)
         Si resultado del acumulador es 3
            encontramos una bifurcación, se agrega al conjunto de minucias.
      Fin si
   Fin para
Fin para
Retornar el conjunto de minucias.
```

Figura 2.10: Pseudo-código algoritmo extraer minucias

Para encontrar las minucias debemos recorrer la imagen en busca de fines de línea o bifurcaciones, utilizando una máscara de 3×3 (se muestra en la Figura 2.11), la que se aplicará a cada píxel negro que encontremos en la matriz. Se analizan los 8 vecinos de píxel *(x, y)* y contamos los píxeles negros que encontremos entre ellos.

Dependiendo del resultado obtenido al analizar esta matriz, podremos saber con gran exactitud el tipo de minucia que hemos encontrado. Si el resultado de la fórmula es 1, estamos hablando de un fin de línea, si el resultado es 2 es un borde (no es una minucia), y si el resultado es 3 podemos decir que encontramos una bifurcación.

Además de conocer el tipo de huella encontrada, podemos obtener las características propias de la minucia como son el ángulo de orientación local y su posición en la imagen.

N(1)	N(2)	N(3)
N(8)	N(0)	N(4)
N(7)	N(6)	N(5)

Figura 2.11: Matriz usada para reconocer minucias

Debido a quiebres en las crestas, se pueden producir minucias que no siempre son correctas. En una etapa de post-procesamiento, es necesario eliminar las minucias falsas. Las minucias que se deben borrar cumplen alguna de las siguientes condiciones:

1. Cresta rota: Se produce cuando dos fin de línea están en la misma dirección y la distancia entre ellas no supera un valor de umbral definido previamente.

2. Espuelas: Un fin de línea se conecta con una bifurcación y la distancia entre ellas está dentro del valor umbral.

3. Puentes: Hay dos bifurcaciones conectadas a diferentes crestas.

Sólo una vez que hemos eliminado las minucias falsas, hemos finalizado la tarea de detección de minucias.

En la Figura 2.12 se muestran las "deformaciones" antes mencionadas.

a b c

Figura 2.12: a) Cresta Rota b) Espuela c) Puente

2.2 Verificación-Identificación

2.2.1 Emparejamiento de huellas digitales

El “emparejamiento" o calce automático de huellas digitales bajo el control de un computador para fines generales es el proceso de comparación de dos huellas digitales diferentes para determinar si las líneas proceden del mismo dedo, y por consiguiente de la misma persona. En el emparejamiento automático de dos huellas digitales se utiliza un número, conocido como la puntuación del “emparejamiento" para indicar el grado en que dos huellas digitales dadas se parecen. Cuanto mayor sea la puntuación, tanto más alta será la probabilidad de que las dos huellas digitales procedan del mismo dedo.

2.2.2 Identificación de huellas digitales

La “identificación" automática de huellas digitales es el proceso de comparación de una huella digital dada, denominada una “huella de búsqueda" o “Query” con una base de datos que contiene un conjunto de huellas digitales denominadas “huellas de archivo" o “Template”, a fin de determinar si la base de datos contiene un Template que proceda del mismo individuo que el de la huella de búsqueda. Por consiguiente, si la base de datos contiene N Template, una identificación es equivalente a N emparejamientos. La comparación de la huella de búsqueda frente a cada una de las N huellas de archivo produce N puntuaciones de los emparejamientos, las cuales son clasificadas por valores decrecientes.

La identificación se traduce típicamente en una lista corta de “candidatos", los cuales son las huellas de archivo que han producido las mejores puntuaciones de los emparejamientos. La precisión de un sistema automático para el emparejamiento y la identificación de las huellas digitales se mide por su capacidad para situar la huella correcta en la lista de candidatos principales.

2.2.3 Verificación de huellas digitales

La “verificación” automática de huellas digitales es el proceso de comparación de una huella digital dada, con una huella digital obtenida directamente desde la base datos por un

identificador otorgado por la persona quien dice ser poseedora de la huella digital de entrada. Con este proceso se "asegura" que la persona sea quien dice ser.

2.2.4 Exactitud en la identificación: medidas de desempeño

La verificación o identificación biométrica por medio de huellas digitales tiene un grado de seguridad tan alto debido a que nadie podría sustraer, copiar o reproducir los elementos usados en ella, ya que son elementos inherentes a su portador, sin embargo puede estar sujeta a errores de:

2.2.4.1 FAR: Falsa aceptación.

Cuando se acepta a alguien que No es; por ejemplo, alguien podría clonar una credencial de identificación, o adueñarse de los números confidenciales de una persona para hacer una transacción en perjuicio de su legítimo dueño y hasta falsificar su firma, o para éste caso que simplemente el sistema de verificación de huella digital falló a la hora de realizar la comparación.

2.2.4.2 FRR: Falso rechazo.

Consiste en no aceptar a alguien que Sí es, pero su identificación no se pudo realizar debido a múltiples motivos, como puede ser: que la imagen de la huella esté muy dañada, o a que tenga una capa de cemento o de pintura, o a que el lector no tenga la calidad suficiente para tomar correctamente la lectura.

2.2.5 Sistemas automáticos

Con el incremento en la potencia de los computadores, se han desarrollado sistemas para automatizar las tediosas tareas de clasificación y emparejamiento. En función de su utilización, se distinguen dos sistemas biométricos basados en huellas dactilares:

2.2.5.1 AFAS: Automatic Fingerprint Authentication System.

La entrada de un AFAS es una identidad y la imagen de una huella dactilar. La respuesta es un "boolean" que indican si la imagen pertenece a la persona de la que se proporciona su

identidad. El sistema compara la imagen de entrada con la que tiene la identidad almacenada en la base de datos.

2.2.5.2 AFIS: Automatic Fingerprint Identification System.

En un AFIS sólo se pasa la imagen como entrada y a la salida se recibe una lista de identidades de personas que pueden tener la huella dactilar dada y una puntuación para cada identidad indicando la similitud entre su huella y la proporcionada como entrada. En este caso, el sistema compara la imagen de entrada con muchos registros de la base de datos.

En la vanguardia de este tipo de sistemas tenemos el Integrated Automated Fingerprint Identification System (IAFIS) **[IAFIS07]**. El IAFIS es un sistema de huella digital e historial criminal mantenido por el FBI de los Estados Unidos. El sistema proporciona capacidades de búsqueda de huella digital automatizadas, almacenamiento de imágenes en formato digital, intercambio de huellas durante 24 horas al día y 365 días al año. La búsqueda se realiza entre una gigantesca base de datos en las que hay almacenada millones de huellas, y los resultados se proporcionan en un plazo de tan solo 2 horas.

2.3 Espacios Métricos

La geometría del espacio tridimensional en el que estamos sumergidos nos resulta muy natural. Conceptos tales como distancia, longitud, ángulo, perpendicularidad son de uso cotidiano. En matemática frecuentemente se pueden agrupar ciertos objetos en espacios abstractos y definir entre ellos relaciones semejantes a las existentes entre los puntos del espacio ordinario. El paralelismo que se establece así entre los espacios abstractos y el espacio euclideano permite visualizar y lograr un entendimiento más profundo de estos objetos. En algunas aplicaciones el planteo más simple que puede usarse es el de espacio métrico.

Un espacio métrico es un conjunto de puntos en los que está definida la noción de distancia entre puntos. Podemos usar la función de distancia o métrica para definir los conceptos

fundamentales del análisis, tales como convergencia, continuidad y compacidad. Un espacio métrico no necesita tener ninguna clase de estructura algebraica definida en él, es decir, puede no tener sentido la suma de elementos del espacio o la multiplicación de un elemento por un número real o complejo. Sin embargo, es muy frecuente el uso de espacios métricos que son a su vez espacios vectoriales, con una métrica derivada de una norma que mide la longitud de un vector.

La noción de semejanza en el caso de las huellas digitales es compleja **[CBYN05]**, por lo tanto su representación como un espacio vectorial, siendo éste a su vez un espacio métrico, se ve beneficiada por la posibilidad de realizar búsquedas de similitud entre ellas.

2.3.1 Métricas

Ahora se introduce la notación básica para el problema de satisfacer consultas por proximidad. El conjunto $\boldsymbol{U}$ denotará el universo de objetos *válidos*. Un subconjunto finito de él, $\boldsymbol{S}$, de tamaño $n = |\boldsymbol{S}|$ es el conjunto de objetos donde se busca. $\boldsymbol{S}$ será llamado la *base de datos* o simplemente un conjunto de *objetos* o *elementos* **[CBYN05]**.

La función

$$d : \boldsymbol{U} \times \boldsymbol{U} \rightarrow \boldsymbol{R}$$

denotará una medida de "distancia" entre objetos (es decir, mientras más pequeña es la distancia, más cercanos o similares son los objetos). Las funciones de distancia tienen las siguientes propiedades[2]:

(p1) $\forall\, x,y \in \boldsymbol{U},\ d(x,y) \geq 0$ (positividad)

(p2) $\forall\, x,y \in \boldsymbol{U},\ d(x,y) = d(y,x)$ (simetría)

(p3) $\forall\, x \in \boldsymbol{U},\ d(x,x) = 0$ (reflexibidad)

y en la mayoría de los casos

[2] Para simplificar la presentación, algunas veces nos referimos a O_j, como un objeto, en vez de como el valor de característica del objeto mismo.

(p4) $\forall x,y \in U, x \neq y \Rightarrow d(x,y) > 0$ (positividad estricta)

Las propiedades enumeradas arriba, de la función de similaridad sólo aseguran su definición consistente y no pueden ser usadas para ahorrarse comparaciones en una consulta por proximidad. Si *d()* es en verdad una métrica, es decir si satisface

(p5) $\forall x,y,z \in \boldsymbol{U}, d(x,y) \leq d(x,z) + d(z,y)$ (desigualdad triangular)

entonces el par (***U, d***) se denomina un *espacio métrico*.

De aquí en adelante se utilizará el término *distancia* entendiendo que se hace referencia a una métrica.

Si *a* y *b* son dos números reales, puede pensarse al número real no negativo |*a* - *b*| como la distancia que separa *a* de *b*. Esta operación de asignar distancias a pares de puntos es precisamente lo que da origen a los espacios métricos.

2.3.2 Consultas por proximidad

Existen dos tipos básicos de consultas de interés en espacios métricos **[CBYN05]**:

Consulta por rango $(q, r)_d$**.** Recuperar todos los elementos que están a distancia *r* de *q*. Esto es $\{x \in S, d(q,x) \leq r\}$.

Consulta de *k***- vecinos más cercanos** *k – NN(q)*. Recuperar los *k* elementos más cercanos a *q* en ***S***. Esto es, recuperar un conjunto $A \subseteq S \; tal \; que \mid A \mid = k$ y $\forall x \in S - A, d(q,x) \leq d(q,y)$. Notar que se satisface cualquier conjunto de *k* elementos que cumpla la condición.

La Figura 2.13 ilustra un ejemplo de ambos tipos de consulta. A la izquierda se muestra una consulta por rango con radio *r* y a la derecha una consulta por los *5*-vecinos más cercanos a *q*. En este último caso, para hacer más evidente el tipo de consulta, también se ha graficado

el rango necesario para encerrar al menos a 5 puntos (aunque éste realmente encierra a más de 5 objetos). Así es posible observar que existirían, para esta consulta (dado q y con $k = 5$), otras posibles respuestas. Las consultas se realizan sobre un conjunto de puntos de $\mathbf{R}^2$, como el espacio métrico, para mayor claridad.

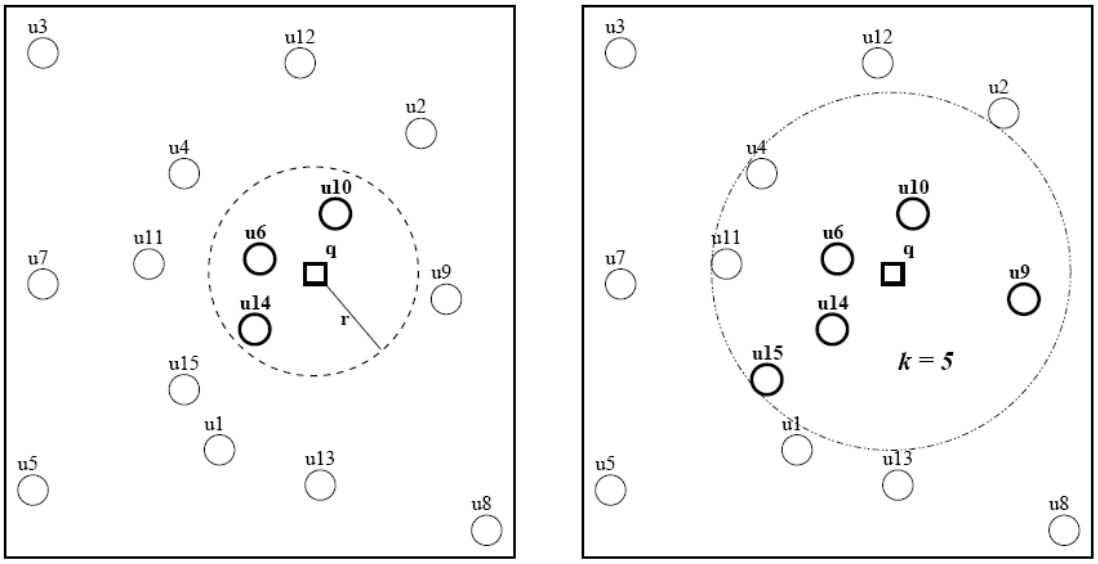


Figura 2.13: Un ejemplo de consulta por rango (izquierda) y por k-vecinos más cercanos (derecha) sobre un conjunto de puntos de $\mathbf{R}^2$.

Una *consulta por rango* será por lo tanto un par $(q, r)_d$ siendo q un elemento de $\boldsymbol{U}$ y r un número real indicando el *radio* (o tolerancia) de la consulta. El conjunto $\{u \in S, d(q,u) \leq r\}$ será llamado la *salida* o *respuesta* de la consulta por rango.

Se usa "*NN*" como una abreviatura de "*Nearest Neighbor*" (vecino más cercano), y se le da el nombre genérico de "*consulta-NN*" a este tipo de consulta y "*búsquedas-NN*" a las técnicas utilizadas para resolverlas. Como se verá más adelante, las consultas-NN se pueden construir sistemáticamente sobre consultas por rango.

El tiempo total para evaluar una consulta se puede dividir en:

- T = # evaluaciones de distancia × complejidad de $d()$ + tiempo extra de CPU + tiempo de E/S.

y desearíamos minimizar *T*. En muchas aplicaciones, sin embargo, evaluar *d()* es tan costoso, como resulta ser en este caso (la comparación de huellas digitales), que todas las demás componentes del costo se pueden ignorar. Éste es el modelo utilizado en el presente trabajo, y por lo tanto la medida de complejidad de los algoritmos será el *número* de evaluaciones de distancias realizadas.

Un *algoritmo de indexación* es un procedimiento para construir de antemano una estructura de datos (llamada *índice*) diseñada para ahorrar cálculos de distancia en donde luego se responden consultas por proximidad. Lo importante es, por lo tanto, diseñar algoritmos de indexación eficientes para reducir las evaluaciones de distancia. Todos los algoritmos de indexación particionan el conjunto ***S*** en subconjuntos. Se construye el índice para permitir determinar un conjunto de subconjuntos candidatos donde se pueden encontrar los elementos postulantes a la consulta. Durante la consulta, se busca en el índice para encontrar los subconjuntos relevantes y luego se inspeccionan exhaustivamente todos estos subconjuntos. Todas estas estructuras se basan sobre la desigualdad triangular para descartar elementos. La Figura 2.14 resume el modelo general utilizado para la indexación y consultas en espacios métricos.

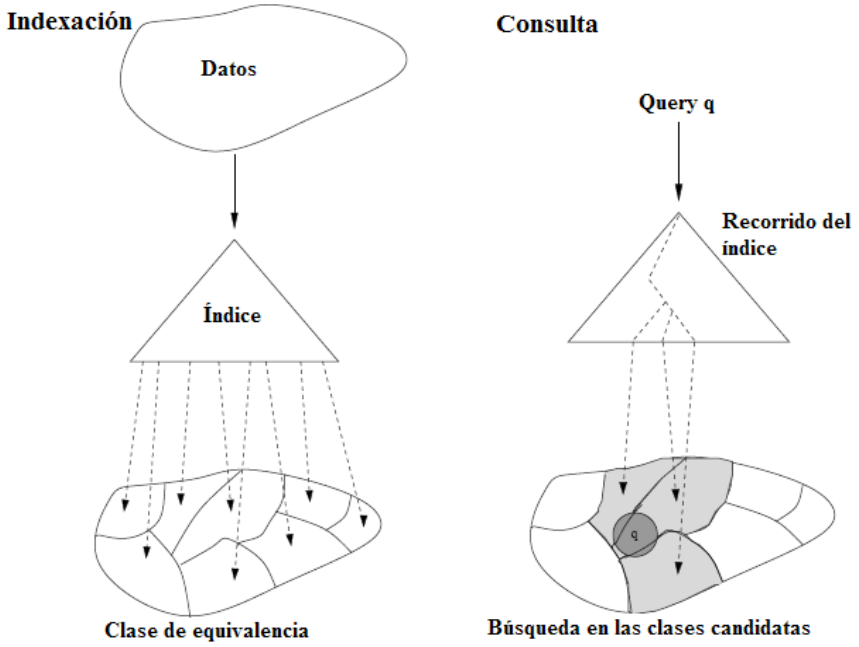


Figura 2.14: Modelo general que se utiliza para la indexación y consulta en espacios métricos.

2.3.3 Ejemplos de espacios métricos

Primero que todo, aquí se muestran las definiciones de algunos espacios métricos, siendo estos los más conocidos:

2.3.3.1 Espacio métrico discreto

Dado un conjunto no vacío cualquiera X definimos la métrica discreta $d()$ en X mediante

$$d(x, y) = \begin{cases} 1 & x \neq y \\ 0 & x = y \end{cases} \tag{2-11}$$

se chequea fácilmente que (X, d) es un espacio métrico.

2.3.3.2 La recta real R

Sea X = **R**, $d(x, y) = |x - y|$ para cada $x, y \in$ R. Los axiomas métricos se cumplen. El conjunto de números complejos C con la función distancia $d(z, w) = |z - w|$ también es un espacio métrico.

2.3.3.3 El espacio euclídeo $\mathbf{R^n}$

Sea X = $\mathbf{R^n}$, el conjunto de todas las n-uplas de números reales. Si $x = (x_1, x_2, ..., x_n)$ *e* $y = (y_1, y_2, ..., y_n)$ son elementos de X, definimos la distancia

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2-12}$$

Ahora se presentan algunos ejemplos de espacios métricos particulares: espacios vectoriales, espacio de documentos representados como vectores y diccionarios (espacios de cadenas de caracteres o strings sobre un alfabeto).

2.3.3.4 Espacios Vectoriales

Si los elementos del espacio métrico *(**U**, d)* son realmente tuplas de números reales, entonces el par se denomina *espacio vectorial*. Un espacio vectorial de dimensión finita D

es un espacio métrico particular donde los objetos se identifican por *D* números reales $(x_1, x_2, ..., x_D)$ y cada x_i es llamada "coordenada" del objeto (en adelante los llamaremos espacios vectoriales o espacios *D*-dimensionales). Existen distintas funciones de distancia que se pueden usar en un espacio métrico, pero las más usadas son las de la familia de L_s o *familia de distancias de Minkowski*, que se define como:

$$L_s((x_1,...,x_D),(y_1,...,y_D)) = \left(\sum_{1 \le i \le D} | x_i - y_i |^S \right)^{1/S} \quad (2\text{-}13)$$

Algunos ejemplos de métricas pertenecientes a esta familia de distancias son la distancia L_1 conocida como *distancia Manhattan*, la L_2 que es más conocida como *distancia Euclídea*, y se corresponde a nuestra noción habitual de distancia, y la distancia L_∞, conocida también como *distancia del máximo*.

La distancia L_2, se define como:

$$L_2((x_1,...,x_D),(y_1,...,y_D)) = \left(\sum_{i=1}^{D} | x_i - y_i |^2 \right) \quad (2\text{-}14)$$

En muchas aplicaciones los espacios métricos son en realidad espacios vectoriales, es decir que los objetos son puntos *D*-dimensionales y la similaridad se puede interpretar geométricamente. Un espacio vectorial permite más libertad al diseñar algoritmos de búsqueda, porque es posible usar la información de coordenadas y geométrica que no está disponible en los espacios métricos generales.

Las estructuras de búsqueda para espacios vectoriales más populares son *Kd-trees* **[Ben75]**, los *R-trees* **[Gut84]**, los *Quad-trees* **[Sam84]** y los *X-trees* **[BKK96]** que son los más recientes. Todas estas técnicas usan ampliamente la información de coordenadas para agrupar y clasificar puntos en el espacio. Desafortunadamente estas técnicas son muy sensibles a la dimensión del espacio. Los algoritmos de búsqueda de puntos más cercano y por rango dependen exponencialmente de la dimensión del espacio **[Cha94]**.

2.3.3.5 Modelo Vectorial para Documentos

En recuperación de la información **[BYRN99]** se define un documento como una "unidad de recuperación", la cual puede ser un párrafo, una sección, un capítulo, una página Web, un artículo o un libro completo. Los modelos clásicos en recuperación de la información consideran que cada documento está descrito por un conjunto representativo de palabras claves llamadas términos, que son palabras cuya semántica ayuda a definir los temas principales del documento.

Uno de estos modelos, el *modelo vectorial*, considera un documento como un vector *t*-dimensional, donde *t* es el número total de términos de la colección. Cada coordenada *i* del vector está asociada a un término del documento, cuyo valor corresponde a un "peso" positivo w_{ij} si es que dicho término (el término *i)* pertenece al documento *j* ó 0 en caso contrario. Si ***D*** es el conjunto de documentos y d_j es el *j*-ésimo documento perteneciente a *D* entonces $d_j = (w_{1j}, w_{2j}, ..., wi_j)$.

En el modelo vectorial se calcula el *grado de similitud* entre un documento *d* y una consulta *q*, la cual se puede ver como un conjunto de términos o como un documento completo, como el grado de similitud entre los vectores $\vec{d}_j$ y $\vec{q}$. Esta correlación puede ser cuantificada, por ejemplo, como el coseno del ángulo formado entre ambos vectores:

$$\text{sim}(d_j, q) = \frac{\vec{d}_j \cdot \vec{q}}{|\vec{d}_j| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} wij \times wiq}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \times \sum_{i=1}^{t} w_{iq}^2}} \qquad (2\text{-}15)$$

donde w_{iq} es el peso del *i*-ésimo término en la consulta *q*.

Los pesos de los términos pueden calcularse de varias formas. Una de las más importantes es utilizando esquemas *tf-idf*, en donde los pesos están dados por:

$$w_{ij} = f_{i,j} \times \log\left(\frac{N}{n_i}\right) \qquad (2\text{-}16)$$

donde ***N*** es el número total de documentos, n_i es el número de documentos en donde el *i*-ésimo término aparece, y $f_{i,j}$ es la *frecuencia normalizada* del *i*-ésimo término, dada por:

$$f_{i,j} = \frac{freq_{i,j}}{máx_{\ell=1..t}(freq_{\ell,j})} \tag{2-17}$$

donde $freq_{i,j}$ es la frecuencia del *i*-ésimo término en el documento d_j, y $máx_{\ell=1..t}(freq_{\ell,j})$ es el máximo valor de frecuencia sobre todos los términos contenidos en d_j.

Si se considera que los documentos son puntos en un espacio métrico, el problema de la búsqueda de documentos similares a una consulta dada se reduce a una búsqueda en proximidad en el espacio métrico. Dado que *sim(d_j, q)* sólo es una medida de similaridad, que en particular no cumple con la desigualdad triangular, se utiliza el ángulo formado entre los vectores $\vec{d}_j$ y $\vec{q}, d(d_j, q) = arc\cos(sim(d_j, q))$, como función de distancia y por lo tanto se la denomina "distancia coseno" **[BYRN99]**. Por lo tanto, ***(D, d)***, es un espacio métrico.

2.3.3.6 Diccionarios

Otro ejemplo posible de espacio métrico corresponde a un conjunto de palabras o diccionario. En este caso los objetos son palabras (o strings) en un determinado lenguaje (por ejemplo, Inglés, Español, Francés, Italiano, etc.).

Para medir la similitud entre palabras se utiliza la función de distancia conocida como *distancia de edición* (o *distancia de Levenshtein*). Se define como la cantidad de inserciones, eliminaciones o cambios de caracteres que hay que realizar para convertir una palabra en la otra. Claramente ésta es una función de distancia *discreta* y tiene muchas aplicaciones en recuperación de texto, procesamiento de señales y biología computacional **[Nav01]**.

Si se consideran las palabras como puntos de un espacio métrico, el problema de buscar palabras similares a una dada se reduce a una búsqueda por proximidad en el espacio métrico. El caso particular de un diccionario es de interés en aplicaciones de correctores ortográficos.

2.3.4 La huella digital sobre el modelo de espacio métrico

El modelo de espacio vectorial puede ser restrictivo para algunas aplicaciones.

Es difícil mapear cada objeto en un punto *d*-dimensional de un espacio vectorial mientras se mantiene la precisión de la representación de semejanza o distancia entre objetos usando alguna métrica en ese espacio.

Esto ocurre cuando la noción de semejanza es compleja y dependiente del dominio, como en el caso de similaridad entre huellas digitales **[CBYN05]**, siendo éste un espacio métrico que cumple con las propiedades definidas en la Sección **2.3.1**.

Para la implementación de un sistema de búsqueda de huellas digitales sobre un espacio métrico, el índice es construido con sólo la función de distancia métrica, sin conocer la representación usada para calcular la distancia. Cualquier caso particular de espacio métrico puede ser usado para el modelo vectorial, ya que el mismo es un caso particular de espacio métrico donde los objetos son representados por *d* coordenadas de valores reales.

2.3.4.1 ¿Porqué un espacio métrico?

Es deseable buscar algoritmos en los cuales la complejidad no dependa estrictamente de la dimensión, sino más bien de propiedades intrínsecas de los datos. Sobre el modelo de espacio métrico existen varias técnicas conocidas para resolver las consultas en un número sub-lineal de cálculos de distancia si es posible preprocesar los datos, como lo realiza el índice métrico M-Tree (ver Sección **2.5**) utilizando, para este proyecto, el algoritmo de matching de huellas digitales como función de distancia (ver Sección **4.1**).

En la Figura 2.15 se muestra la imagen de una huella digital representada como un espacio métrico mediante sus características.

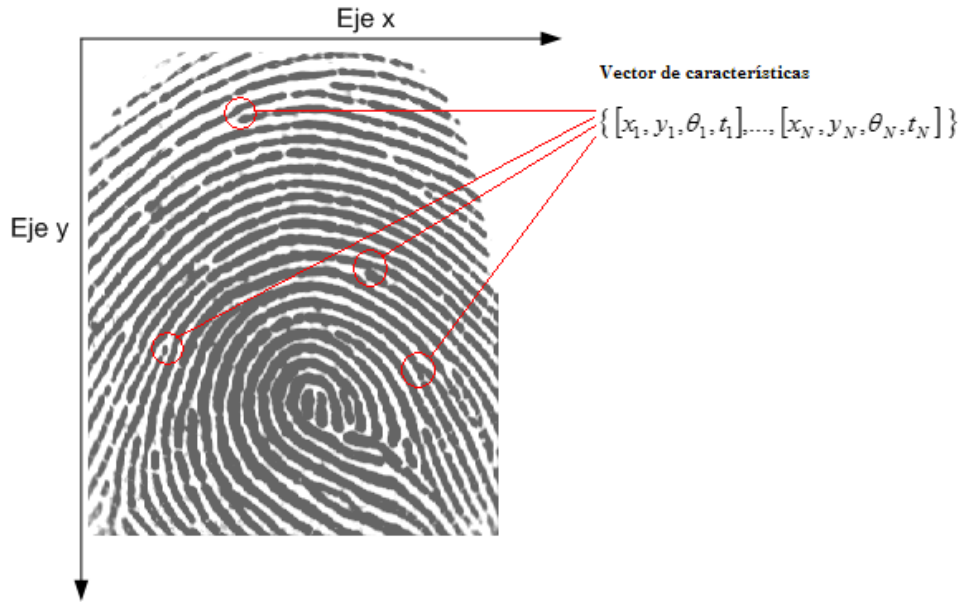


Figura 2.15: Imagen de una huella digital representada como un espacio vectorial.

2.4 Algoritmos de indexación

Los algoritmos de búsqueda en espacios métricos generales se dividen en dos grandes áreas: *algoritmos basados en pivotes* y *algoritmos basados en particiones compactas* (ver **[CNBYM01]** para un análisis detallado).

El objetivo de estos algoritmos es reducir al mínimo el número de evaluaciones de distancia necesarias para responder a búsquedas por rango y a búsquedas de *k*-vecinos más cercanos.

La Figura 2.16 ilustra la clasificación de los índices de acuerdo a sus características más importantes.

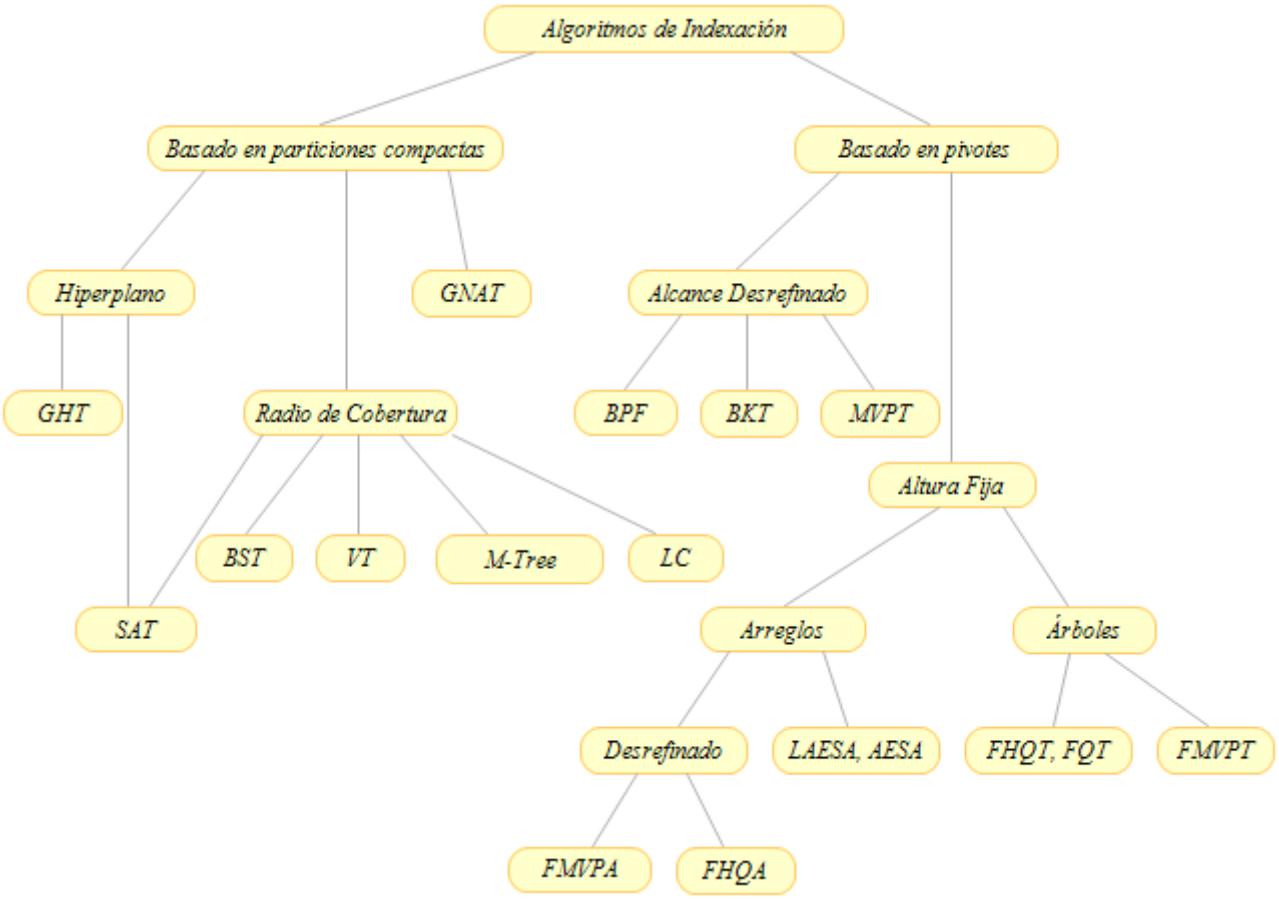


Figura 2.16: Taxonomía de los algoritmos existentes para búsqueda por proximidad en espacios métricos.

En la Tabla 2.2 se muestra el nombre completo de cada una de las estructuras de datos para búsqueda por proximidad en espacios métricos (que aparecen clasificadas en la Figura 2.11), junto a sus respectivas abreviaturas y la referencia del artículo en donde se propuso cada una de ellas.

Analizamos brevemente aquí en qué consisten los algoritmos de cada una de estas grandes áreas, para dar el contexto de este trabajo.

Nombre Abreviado	Estructura	Referencia
BK-tree	Burkhard-Keller tree	[BK73]
FQ-tree	Fixed Queries tree	[BYCMW94]
FHQ-tree	Fixed Height FQ-tree	[BYCMW94, BY97, BYN98]
FQ-arrays	Fixed Queries arrays	[CMBY99, CMN01]
Metric Tree	Metric Tree	[Uhl91b]
VP-tree	Vantage Point tree	[Yia93, Chi94]
MVP-tree	Multi-Vantage Point tree	[BO97]
VP-forest	Excluded Middle Vantage Point Forest	[Yia99]
BS-tree	Bisector tree	[KM83]
GH-tree	Generalized-Hyperplane tree	[Uhl91b]
GNA-tree	Geometric Near-neighbor Access tree	[Bri95]
V-tree	Voronoi tree	[DN87]
M-tree	M-tree	[CPZ97]
LC	List of Clusters	[CN00a]
SA-trees	Spatial Aproximation trees	[Nav99]
AESA	Approximating Eliminating Search Algorithm	[Vid86]
LAESA	Linear AESA	[MOV94]

Tabla 2.2: Nombres de las estructuras de datos para búsquedas por proximidad en espacios métricos, con las referencias a los artículos en donde se propone cada una de ellas.

2.4.1 Algoritmos basados en pivotes

La idea es usar un conjunto de k elementos distinguidos ("pivotes") $p_1, p_2, ..., p_k \in S$ y almacenar, para cada elemento x de la base de datos, su distancia a los k pivotes $(d(x, p_1), ..., d(x, p_k))$. Dada una query q, se calcula su distancia a los k pivotes $(d(q, p_1), ...d(q, p_k))$. Ahora, si para algún pivote p_i se cumple que $| d(q, p_i) - d(x, p_i) | > r$, entonces por la desigualdad triangular conocemos que $d(q, x) > r$ y, por lo tanto, no es necesario evaluar explícitamente $d(x, p)$. Los elementos que no se puedan descartar por medio de esta regla se deben comparar directamente contra la query.

Algoritmos tales como *AESA* **[Vid86]**, *LAESA* **[MOV94]**, *Spaghettis* y sus variantes **[CMBY99, NN97]**, *FQ-trees* y sus variantes **[BYCMW94]** y *FQ-arrays* **[CMN01]**, son casi las implementaciones más directas de esta idea, y difieren básicamente en su estructura extra para reducir el costo de CPU de encontrar los puntos candidatos, pero no en el número de evaluaciones de distancia que realizan.

Existen algunas estructuras de datos como árboles que utilizan esta idea de una manera indirecta: seleccionan un pivote como la raíz del árbol y dividen el espacio de acuerdo a las distancias a la raíz. A cada subárbol le corresponde una porción (el número y amplitud de las porciones difiere de acuerdo a las estrategias). En cada subárbol, se selecciona un nuevo pivote y así sucesivamente. La búsqueda realiza un backtrack sobre el árbol y se utiliza la desigualdad triangular para podar subárboles, es decir, si *a* es la raíz del árbol y *b* la raíz de un hijo tal que $[d(q,a)-r, d(q,a)+r]$ no intersecte a $[x_1, x_2]$. Las estructuras de datos que usan esta idea son los *BK-trees* y sus variantes **[BK73, Sha77]**, los *Metric trees* **[Uhl91b]**, *TLAESA* **[MOC96]**, y los *VP-trees* y variantes **[Yia93, BO97, Yia00]**.

2.4.2 Algoritmos basados en particiones compactas

La segunda tendencia consiste en dividir el espacio en zonas o regiones tan compactas como sea posible. Normalmente se realiza recursivamente, y se almacena un punto representativo ("centro") para cada zona más algunos pocos datos extras que permitan descartar rápidamente la zona durante una búsqueda. Se pueden usar dos criterios para delimitar una zona.

El primero es el área de *Voronoi*, donde seleccionamos un conjunto de centros y colocamos cada punto dentro de la región de su centro más cercano. Las áreas se delimitan con hiperplanos y las zonas son análogas a las regiones de Voronoi en espacios vectoriales. Sea $\{c_1,...,c_m\}$ el conjunto de centros. Durante la búsqueda se evalúa $d(q,c_1),...,d(q,c_m)$, eligiendo el centro c_j más cercano y descartando cada zona cuyo centro c_i satisfaga que $d(q,c_i) > d(q,c_j) + 2r$, porque su área de Voronoi no puede intersectar la "bola" de la query (con centro *q* y radio *r*)[3].

El segundo criterio es el de *radio de cobertura* $rc(c_i)$, el cual es la máxima distancia entre el centro c_i y un elemento de su zona. Si $d(q,c_i) - r > rc(c_i)$, entonces no es necesario considerar la zona *i*.

[3] Se denomina "bola" de la query *q* al conjunto de puntos que se encuentra a distancia a lo más *r* de *q*.

Las técnicas se pueden combinar. Algunas que usan sólo hiperplanos son los *GH-trees* y sus variantes **[Uhl91b, NVZ92]**, y los *Voronoi trees* **[DN87, Nol89]**. Algunas que usan sólo radios de cobertura son los *M-trees* **[CPZ97]** y *Lista de Clusters* **[CN00a]**. Uno que usa ambos criterios es el *GNA-tree* **[Bri95]**.

2.5 Índice Métrico: M-tree.

2.5.1 Introducción

Actualmente existe la necesidad de manejar de manera automatizada (mediante computador) objetos de tipos diferentes (imagen, video, voz, huellas digitales etc.) **[BFPR06]** a los tradicionales (string, integer, etc.). Esta necesidad ha tenido como consecuencia el desarrollo de las bases de datos multidimensionales.

En estos tipos de bases de datos, es esencial la operación que permite recuperar objetos basado en el contenido. Por ejemplo, si tenemos almacenados en una base de datos multimedia objetos que corresponden a imágenes de caras de un conjunto de personas, la idea es dada la imagen de la cara de una persona poder recuperar desde la base de datos aquellas imágenes similares a la imagen de entrada. O como en este estudio, la idea es que a partir de una imagen de huella digital de entrada, recuperar de una base de datos todas aquellas huellas digitales similares a la primera.

Es claro que el procesamiento secuencial de la base de datos para este tipo de consultas, no es práctico, debido a la gran cantidad de tiempo que se necesitaría evaluar esta consulta. Una primera solución al problema de la indexación es utilizar los métodos de acceso espaciales (SAMs) como R-tree **[Gut84]** y sus variantes. Sin embargo la aplicación de tales métodos a objetos de un espacio métrico presenta los siguientes supuestos:

1. Los objetos a indexar, se presentan por medio de valores de características en un espacio vectorial multidimensional.

2. La distancia de similitud de dos objetos debe basarse en alguna función de distancia L_s metric, como la distancia Euclidiana (ver Sección **2.3.3.4**).

Además, desde el punto de vista de la función, los SAMs asumen que la comparación de claves (valores de características) es una operación trivial con relación al costo de acceder a una página del disco, lo cual no es siempre el caso en aplicaciones multimedia. Consecuentemente, en el diseño de estas estructuras no se ha intentado reducir el número de cálculos de distancia.

Un acercamiento más general para el problema de "indexación por similitud" ha ganado alguna popularidad en estos últimos años, liderando el desarrollo de los así llamados árboles métricos (vea a **[CPZ97]**). Los árboles métricos sólo consideran distancias relativas de objetos (en vez de sus posiciones absolutas en un espacio multidimensional) para organizar y dividir en partes el espacio de búsqueda, y justamente requiere que la función usada mida la distancia (distancia de similaridad) entre objetos métricos (vea Sección **2.4.2**), a fin de que la propiedad de desigualdad triangular tenga que ser aplicada y pueda usarse para disminuir el espacio de búsqueda.

Aunque la efectividad de árboles métricos ha sido claramente demostrada, los diseños actuales padecen de ser inherentemente estáticos, lo cual limita su aplicabilidad en ambientes dinámicos de base de datos. Contrariamente de SAMs, los árboles métricos conocidos sólo han tratado de reducir el número de cálculos de distancia requerido para contestar una consulta, no poniendo ninguna atención en los costos de I/O.

El M-Tree es un árbol métrico paginado, el cual ha sido explícitamente diseñado para ser integrado con otros métodos de acceso en sistemas de base de datos. Demostrando tal posibilidad implementando el M-Tree con la librería GIST (Árbol de Búsqueda Generalizado), el cual le permite a los métodos de acceso específicos ser añadido a un sistema extensible de la base de datos.

El M-Tree es un árbol balanceado, capaz de ocuparse de ficheros de datos dinámicos, y como tal no requiere reorganizaciones periódicas. El M-tree puede indexar objetos usando características comparadas por funciones de distancia, no importando si no pertenecen a un espacio vectorial o no utiliza una L_s metric (ver Sección **2.3.3.4**), así considerablemente

extiende los casos para el cual es posible procesar una consulta eficiente. Desde que el diseño de M-Tree está inspirado por ambos principios de árboles métricos y métodos de acceso a la base de datos, la optimización de la función le concierne a la CPU (cálculo de distancia) y los costos de I/O. Después de proveer algún fondo preliminar en la Sección **2.4.2,** la Sección **2.4.3** introduce los principios básicos de M-Tree y los algoritmos. En la Sección **2.4.4**, se discuten las alternativas disponibles para implementar la estrategia usada para manejar overflows (desbordamiento) del nodo. En la Figura 2.17 se muestra un dibujo de un árbol M-Tree creado con puntos de coordenadas dentro de un espacio 2-dimensional, con el fin de demostrar de forma clara y simple como se estructura gráficamente éste índice métrico. Para este ejemplo la función de distancia utilizada es la distancia Euclidiana.

2.5.2 Preliminares

Indexar un espacio métrico tiene la intención de proveer un soporte eficiente para contestar consultas de similitud, o sea consultas cuyo propósito es recuperar objetos de ***S*** que sean "similares" para un objeto dado en la consulta, y donde la distancia de similitud entre objetos es medida por una función específica de distancia métrica *d*.

Formalmente, un espacio métrico es un par, *M = (D, d)*, dónde *D* es un dominio de valores de características - la clave de indexación - y *d()* es una función total (distancia) descrita por las propiedades definidas en la Sección **2.3.1**.

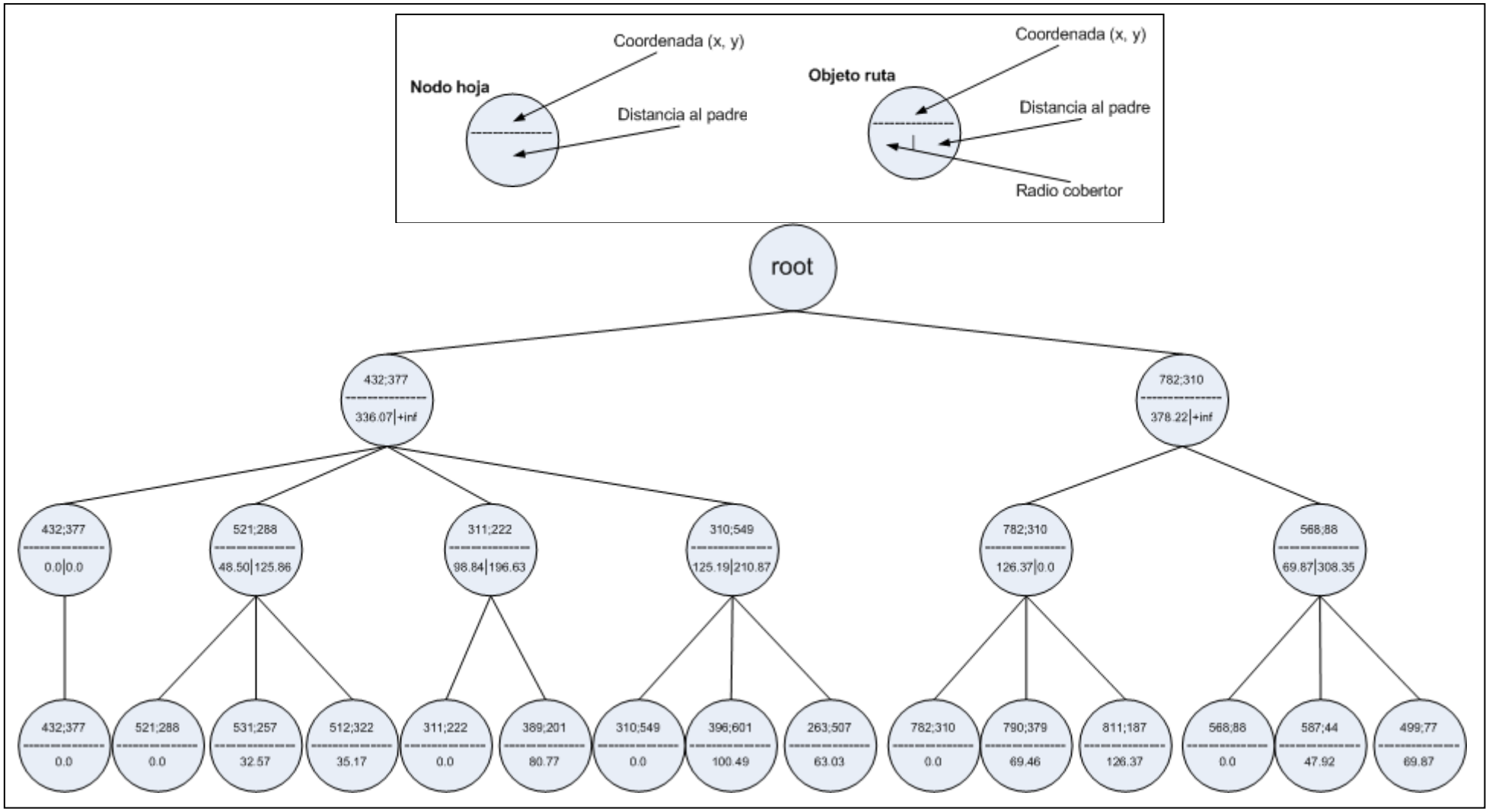


Figura 2.17: Demostración gráfica del M-Tree con puntos de coordenadas en un espacio 2-dimensional.

Ya ha habido algunos intentos para abordar el difícil problema de indexación de espacios métricos. El algoritmo FastMap **[FL95]** transforma una matriz de distancias en un set de puntos de dimensiones bajas, lo cual luego puede ser indexado por algún método SAM. Sin embargo, FastMap asume que el conjunto de datos es estático e introduce errores de aproximación en el proceso de mapeo. El VP-tree **[Yia93, Chi94]** tiene la ventaja de dividir en partes un conjunto de datos según las distancias que los objetos tienen en relación a un punto de referencia.

El valor medio de tales distancias es usado como un separador para dividir en partes los objetos en dos subconjuntos simétricos, para el cual el mismo procedimiento puede ser aplicado recursivamente. El MVP-Tree **[BO97]** extiende esta idea usando múltiples posiciones de ventaja, y aprovecha las distancias pre-calculadas para reducir el número de cálculos en el tiempo de la consulta.

El diseño del GNAT **[Bri95]** aplica un estilo diferente de partición - el llamado hiperplano generalizado. En el caso básico, se escogen dos objetos como referencia y cada uno de los objetos restantes es asignado al objeto referente más cercano. Así es que los subconjuntos obtenidos recursivamente pueden ser elegidos otra vez, cuándo sea necesario.

Ya que todas las organizaciones citadas anteriormente construyen árboles por medio de un proceso recursivo, estos árboles no garantizan permanecer balanceados en caso de inserciones y supresiones, y requieren costosas reorganizaciones para impedir funciones de degradación.

2.5.3 El M-Tree

El diseño de M-Tree combina ventajas de balanceo y dinamismo de SAMs, con las capacidades de los árboles métricos estáticos de indexar objetos usando características y con funciones de distancia que no necesariamente pertenezcan a un espacio vectorial sino que sólo cumplan con las propiedades métricas descritas en la Sección **2.3.1**.

El M-Tree divide en partes el conjunto de objetos con base en sus distancias relativas, medidas por una función de distancia *d*, y almacena estos objetos en nodos de tamaño fijo[4], que corresponden a regiones obligadas del espacio métrico. El M-Tree desconoce completamente la función de distancia *d*, por lo tanto se considera que la implementación de la función es una caja negra para el M-Tree. La descripción del diseño y la aplicabilidad de M-Tree están rigurosamente descritas en **[DR99]**. Esta descripción del M-Tree, principalmente se concentra en asuntos de implementación.

2.5.3.1 La Estructura de Nodos de M-Tree

Los nodos hoja de cualquier M-Tree almacenan todos los objetos indexados (base de datos), representados por sus claves o las características, mientras que los nodos internos almacenan los objetos de ruteo (ver Sección **2.5.4**).

Para cada objeto de ruteo O_r, hay un puntero asociado, denotado por $ptr(T(O_r))$, que apunta la raíz de un subárbol, $T(O_r)$, llamado el árbol cobertor de O_r. Todos los objetos en el árbol cobertor de O_r, están dentro de la distancia $r(O_r)$ de O_r, $r(O_r) > 0$, que es designado el radio cobertor de O_r, y forma una parte de la entrada de O_r en un nodo particular de M-Tree. Finalmente, un objeto de ruteo O_r es asociado con una distancia para $P(O_r)$, su objeto padre, ese es el objeto de ruteo el cual referencia al nodo donde la entrada O_r está almacenada. Obviamente, esta distancia no está definida para entradas en la raíz del M-Tree. La información general para una entrada del objeto de ruteo está resumida en la siguiente tabla:

Or	Características objeto de ruteo
ptr(T(Or))	Puntero a la raíz de T(O_r)
r(Or)	Radio cobertor de O_r
d(Or, P(Or))	Distancia de O_r al nodo padre

Tabla 2.3: Información general de un objeto de ruteo.

Un nodo hoja (objeto O_j) de una base de datos es realmente similar a un objeto ruta, pero para el cual el radio cobertor no es necesario, y el campo del puntero almacena el

[4] Nada impediría usar nodos de tamaño variable, como se hace en la X-Tree [BKK96]. Para la simplicidad, sin embargo, aquí no se considera esta posibilidad.

identificador actual (oid) del objeto, que es usado para proveerle el acceso al objeto entero posiblemente residente en un Archivo[5] separado de datos. En Resumen, la entrada en nodos hoja está estructurada de la siguiente forma:

O_r	Características objeto DB
$oid(O_j)$	Objeto identificador
$d(O_j, P(O_j))$	Distancia de O_j al nodo padre

Tabla 2.4: Información general de un nodo hoja.

2.5.3.2 Procesando Consultas de Similitud

Antes de presentar algoritmos específicos para construir el M-Tree, se muestra cómo la información almacenada en nodos es usada para procesar consultas de similitud. Aunque la función de algoritmos de búsqueda es mayormente influenciada por la construcción actual del M-Tree, la exactitud y la lógica de búsqueda son independientes de tales aspectos.

En todos los algoritmos que se presentan, se tiene por objetivo disminuir, además del número de nodos accedidos, también el número de cálculos de distancia necesarias para procesar las consultas. Esto tiene particular importancia cuando la búsqueda resulta tener mayor uso de CPU que I/O, que podría ser el caso para funciones de distancia que requieren de mucho cálculo. Por este propósito, toda la información sobre distancias almacenadas (pre-computadas) en los nodos de M-Tree, o sea $d(O_i, P(O_i))$ y $r(O_i)$, son usados para aplicar efectivamente la desigualdad triangular.

2.5.3.3 Búsqueda por Rango

La búsqueda por rango *(Q, r(Q))* selecciona todos los objetos de la BD tal que $d(O_j, Q) \le r(Q)$. El algoritmo RS comienza por el nodo raíz y recorre recursivamente todos los caminos que cumplen la desigualdad anterior. El pseudo-código del algoritmo de éste tipo de búsqueda se muestra en la Figura 2.18.

[5] Por supuesto, el M-Tree también puede ser utilizado como una organización de datos primaria, dónde los objetos enteros se guardan en las hojas del árbol.

```
RS(N : node, Q : query − object, r(Q) : search _ radius){
sea O_p el objeto padre del nodo N
Si N no es un nodo hoja entonces {
  ∀O_r en N hacer :
    Si | d(O_p,Q) − d(O_r,O_p) | ≤ r(Q) + r(O_r) entonces {
      Calcular d(O_r,Q);
      Si d(O_r,Q) ≤ r(Q) + r(O_r) entonces
        RS(*ptr(T(O_r)),Q,r(Q));
    }
 }else{ ∀O_j en N hacer :
  Si | d(O_p,Q) − d(O_j,O_p) | ≤ r(Q) entonces {
    Calcular d(O_j,Q);
      Si d(O_j,Q) ≤ r(Q) entonces
        agregar oid(O_j) al resultado;
  }
 } }
```

Figura 2.18: Pseudo-código del algoritmo búsqueda por rango.

Ya que, al acceder al nodo N, la distancia entre Q y O_p, el objeto padre de N, ya ha sido calculado, es posible podar un subárbol sin la necesidad de calcular ninguna otra distancia. La condición solicitada para la poda es como sigue **[CPZ97]**:

Lemma 1: *Si* $d(O_r,Q) > r(Q) + r(O_r)$, *entonces, para cada objeto* O_j *en* $T(O_r)$, *es* $d(O_j,Q) > r(Q)$. *Así, la búsqueda* $T(O_r)$ *puede ser podada en forma segura.*

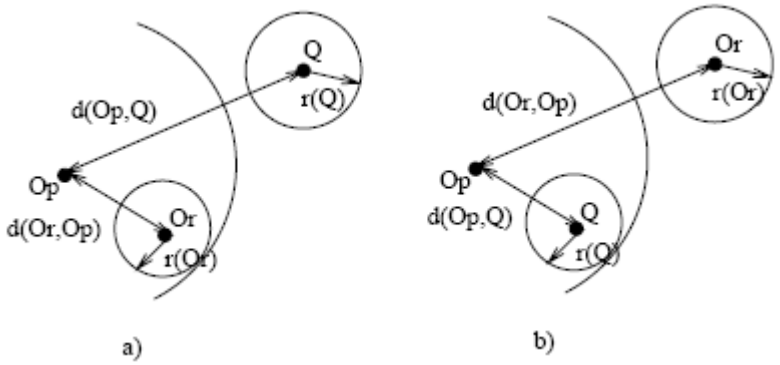


Figura 2.19: Acción aplicada para evitar cálculos de distancia.

De hecho, ya que $d(O_j, Q) \geq d(O_r, Q) - d(O_j, O_r)$ (desigualdad triangular) y $d(O_j, O_r) \leq r(O_r)$ (def. de radio cobertor), es $d(O_j, Q) \geq d(O_r, Q) - r(O_r)$. Tomando en cuenta que, por hipótesis, $d(O_r, Q) - r(O_r) > r(Q)$.

Para aplicar el Lemma 1, la distancia $d(O_r, Q)$ tiene que ser calculada. Esto puede ser evitado aprovechándose del siguiente resultado **[CPZ97]**.

Lemma 2: *Si* $| d(O_p, Q) - d(Or, O_p) | > r(Q) + r(O_r)$, *entonces* $d(O_r, Q) > r(Q) + r(O_r)$.

Ésta es una consecuencia directa de la desigualdad triangular, lo cual le garantiza que $d(O_r, Q) \geq d(O_p, Q) - d(O_r, O_p)$ (la Figura 2.6 a) y $d(O_r, Q) \geq d(O_r, O_p) - d(O_p, Q)$ (la Figura 2.6 b).

2.5.3.4 Construyendo el M-Tree

Los algoritmos para construir el M-Tree especifican cómo insertar los objetos, y cómo se tratan los overflows y underflows de los nodos.

El algoritmo de inserción (Figura 2.20) baja recursivamente el M-Tree para localizar el nodo hoja más adecuado para acomodar un nuevo objeto, O_n, provocando posiblemente una división (split) si el nodo está lleno. La razón básica usada es determinar el nodo hoja "más adecuado" que debe descender, en cada nivel del árbol, a lo largo de un subárbol, $T(O_r)$, para el cual no es necesaria ninguna extensión del radio cobertor, o sea $d(O_r, O_n) \leq r(O_r)$. Si con esta propiedad los múltiples subárboles existen, el primer objeto O_n más cercano a O_r es escogido. Esta heurística intenta obtener adecuados clusters de subárboles, que tiene un efecto beneficioso en la función.

Si no existe ningún objeto de ruteo para el cual $d(O_r, O_n) \leq r(O_r)$, la elección es minimizar el incremento del radio cobertor, $d(O_r, O_n) - r(O_r)$. Esto está estrechamente relacionado

con el criterio heurístico que sugiere minimizar el "volumen" global del radio cobertor del objeto de ruteo en el nodo actual.

```
Insert(N : node, entry(On) : M − tree _ entry){
  N es el set de entradas en el nodo N;
  Si N no es un nodo hoja entonces{
    N_in = entradas tal que d(O_r, O_n ≤ r(O_r));
    Si N_in ≠ 0 entonces
      entry(O_r*) ∈ N_in : d(O_r*, O_n) es mínima
    Sino{ entry(O_r*) ∈ N : d(O_r*, O_n) − r(O_r*) es mínima
        r(O_r*) = d(O_r*, O_n);
    }
    Insert(*ptr(T(O_r*)), entry(O_n));
  }Sino{  /* N es un nodo hoja */
    Si N no está lleno entonces
      almacenar entry(O_n) en N
    Sino Split(N, entry(On));
  }
}
```

Figura 2.20: Pseudo-código del algoritmo de inserción.

La determinación del set N_{in} - objetos de ruteo para el cual no es necesaria ninguna extensión del radio cobertor – se puede ser optimizar ahorrando cálculos de distancia.

Del Lemma 3.2, substituyendo O_n por Q y seteando $r(O_n) \equiv r(Q)=0$, se deriva lo siguiente:

$$Si\ |d(O_p,O_n)-d(O_r,O_p)|>r(O_r), \text{ entonces } d(O_r,O_n)>(O_r).$$

De lo cuál se entiende que $O_r \notin N_{in}$. Note en que ésta optimización no puede ser aplicada en el nodo raíz.

2.5.3.5 Manejo de divisiones (Split)

Como cualquier otro árbol balanceado dinámico, M-Tree crece de abajo hacia arriba. El overflow de un nodo *N* es manejado ubicando un nuevo nodo, *N'*, en el mismo nivel de N, particionando las entradas entre estos dos nodos, y promoviendo al nodo del padre, N_p, dos objetos de ruteo para referenciar a los dos nodos. Cuando la raíz se divide, se crea una nueva raíz y el M-Tree crece un nivel hacia arriba.

```
Split(N : node, E : M − tree _ entry){
  N = entradas del nodo N ∪ {E};
  Si N no es la raíz entonces
    O_p el padre de N, almacenar en el nodo N_p;
  Asignar un nuevo nodo N';
  Promote(N, O_p1, O_p2);
  Partition(N, O_p1, O_p2, N_1, N_2);
  Almacenar N_1's entradas en N y N_2's entradas en N';
  Si N es la raíz actual entonces{
    Asignar un nuevo nodo raíz, N_p;
    Almacenar entry(Op1) y entry(Op2) en N_p;
  }Sino{
    Reemplazar entrada(O_p) con entrada(O_p1) en N_p;
    Si el nodo N_p está lleno entonces
      Split(N_p, entry(O_p));
    Sino Almacenar entry(Op2) en N_p;
  }
}
```

Figura 2.21: Pseudo-código del algoritmo de Split (división).

El método *Promote* escoge, según algún criterio específico, dos objetos de ruteo, O_{p1} y O_{p2}, para ser insertados en el nodo padre, N_p. El método *Partition* divide las entradas del nodo *overflown* (el set N) en dos subconjuntos disjuntos, N_1 y N_2, los cuáles son almacenados en los nodos *N* y *N'* respectivamente. Una implementación específica del método *Promote* y el método de *Partition* define lo que llamamos *política de división*. A diferencia de otros diseños de árboles métricos (estáticos), cada uno confiando en un criterio específico para

organizar objetos, M-Tree ofrece una posibilidad de implementar alternativas de la política de división, para sincronizar la función dependiendo en las necesidades específicas de la aplicación (ver Sección **2.5.4**).

A pesar de la especificación de la política de división, la semántica del radio cobertor se debe conservar. Si el nodo divido es una hoja, entonces el radio cobertor de un objeto promovido, expresado como O_{p1}, es obtenido por (2-42).

$$r(O_{p1}) = \max\{d(O_j, O_{p1}) \mid O_j \in \mathrm{N}_1\} \tag{2-18}$$

Mientras que si el *overflow* ocurre en un nodo interno

$$r(O_{p1}) = \max\{d(O_r, O_{p1}) + r(O_r) \mid O_r \in \mathrm{N}_1\} \tag{2-19}$$

que garantiza que $d(O_j, O_{p1}) \le r(O_{p1})$ puede aplicarse a cualquier objeto en $\mathrm{T}(\mathrm{O_{p1}})$.

2.5.4 Políticas de división.

La política de división "ideal" debería promover los objetos O_{p1} y O_{p2}, y particionar los otros objetos de tal manera de obtener dos regiones que tengan un "volumen" mínimo y un "*overlap*" mínimo. Ambos criterios tienen la intención de mejorar la efectividad de los algoritmos de búsqueda, desde tener pequeñas regiones hasta llegar a árboles adecuadamente agrupados y reducir la cantidad de espacio muerto indexado - espacio donde no se encuentran objetos - y tener pequeños (posiblemente nulo) *overlap* entre regiones reduciendo el número de caminos por ser atravesados para contestar una consulta.

El criterio de volumen mínimo induce a idear políticas de división que traten de minimizar los valores del radio cobertor, mientras que el requisito del mínimo overlap sugiere que, para los valores fijos del radio cobertor, se debería maximizar la distancia entre los objetos de ruteo[6].

[6] Note que, sin un conocimiento detallado de la función de distancia, es imposible cuantificar la cantidad exacta de overlap de dos regiones poco disjuntas en un espacio métrico.

2.5.4.1 Escogiendo los Objetos de ruteo

El método *Promote* determina, dado un set de entradas, *N*, promover dos objetos y almacenarlos en el nodo padre. Los algoritmos especificados anteriormente, pueden ser clasificados adecuadamente aunque no se confirme el rol del objeto padre original.

Definición: Una política de división confirmada escoge uno de los objetos promovidos, expresado como O_{p1}, el objeto O_p, o sea el objeto padre del nodo dividido.

En otros términos, una política de división justamente "extrae" una región, centrado en el segundo objeto de ruteo, O_{p2}, de la región que todavía permanecerá centrada en O_p. En general, esto simplifica la ejecución de división y reduce el número de cálculos de distancia.

- **m_RAD**: este algoritmo es el más complicado en términos de cálculos de distancia. Considera todos los pares posibles de objetos y, después de dividir el set de entradas, promueve el par de objetos para los cuales la suma de radio cobertor, $r(O_{p1} + r(O_{p2}))$, es mínima.

- **mM_RAD**: este es similar a m_RAD, pero minimiza el máximo de los dos radios.

- **M_LB_LIST:** esta política difiere del anterior en que sólo usa las distancias almacenadas pre-calculadas, donde $O_{p1} \equiv O_p$, el algoritmo determina a O_{p2} como el objeto más lejano de O_p, es decir:

$$d(O_{p2}, O_p) = \max_j \{d(O_j, O_p)\} \tag{2-20}$$

- **RANDOM**: esta variante selecciona en una forma aleatoria el objeto(s) referenciado. Aunque no es una muy buena estrategia, es acelerada y su actuación puede ser utilizada como una referencia para otras políticas.

- **SAMPLING**: esta es la política RANDOM, pero iterado sobre una prueba de objetos de tamaño *s > 1*. Para cada uno de los *s(s - 1) /2* pares de objetos en el

ejemplo, las entradas son distribuidas y se establece un potencial radio cobertor. Se selecciona el radio cobertor con el máximo resultado. En caso de que la promoción sea confirmada, sólo las *s* distribuciones diferentes son probadas.

2.5.4.2 Distribución de las Entradas

Dado un set de N entradas y dos objetos ruta O_{p1} y O_{p2}, el problema es cómo dividir eficazmente N en dos subconjuntos, N_1 y N_2. Con este propósito consideramos dos estrategias básicas. La primera es basada en la idea de la *descomposición generalizada del hiperplano* y conduce a divisiones des-balanceadas, mientras que la segunda obtiene una distribución balanceada. Aquellos están inmediatamente descritos como sigue.

- **Hiperplano Generalizado:** Asigna cada objeto $O_j \in N$ al objeto de ruteo más cercano: si $d(O_j, O_{p1}) \leq d(O_j, O_{p2})$ luego asigna O_j a N_l, sino asigna O_j a N_2.

- **Balanced:** Calcula $d(O_j, O_{p1})$ y $d(O_j, O_{p2}) \; \forall O_j \in N$. Repite hasta que N esté vacío:
  - Asigna a N_1 el vecino más cercano de O_{p1} en N y lo elimina de N.
  - Asigna a N_2 el vecino más cercano de O_{p2} en N y lo elimina de N.

Dependiendo de la distribución de datos y sobre la forma cómo los objetos de ruteo son escogidos, una política de división des-balanceada puede conducir a una mejor división de objetos, debido al grado de libertad adicional obtenido. Se debe hacer notar que, al obtener una división balanceada con las dinámicas de SAMs, la expansión de regiones a lo largo de las dimensiones necesarias, en un espacio métrico, se propagan para todas las "dimensiones" incrementando sus radios cobertores.

Aquí se presenta una breve explicación de la estructura de M-Tree, ya que esto es explicado a cabalidad en **[CPZ97]**.

- El árbol métrico contiene todos los objetos de datos en sus nodos hoja. Este crece de abajo hacia arriba. Los nodos superiores contienen “los objetos ruta” que describen

los objetos contenidos en las ramas. Cada objeto ruta tiene un radio cobertor que contiene todos los objetos de su sub-árbol, y las distancias pre-calculadas para cada sub-árbol.

- Cuando el nodo raíz (o cualquier otro nodo) se llena, una operación de inserción genera una división de ese nodo. Dos de los objetos contenidos en ese nodo son escogidos como objetos ruta, los otros objetos son cada uno asignado a uno de ellos, y se crea una nueva raíz. Si el nodo dividido no es la raíz, uno de los dos nodos nuevos es introducido en el nodo padre, lo cual puede conducir a llamadas recursivas de divisiones si hay desbordamiento.

- Cuando se hace una búsqueda por rango, los subárboles son podados si la distancia entre el objeto Query y el objeto ruta es mayor que el radio cobertor del objeto ruta + el radio del Query. Esto y el hecho de que una gran cantidad de distancias ya están pre-calculadas, conduce a una velocidad relativamente alta de operaciones de búsqueda.

- En **[CPZ97]** se evaluaron diferentes estrategias de división. En esta librería de M-Tree en Java, la única regla implementada fue minRAD/MinMax“, la cual es más lenta al crear el árbol que otras estrategias, pero hace que el árbol sea más eficiente para las búsquedas, siendo esto lo fundamental para la aplicación. La regla dice: Esos dos objetos serán promovidos a los objetos ruta cuya suma de radio cobertor sea la mas pequeña. Por consiguiente, todas las combinaciones posibles de objetos ruta y su radio cobertor deben ser computados, siendo esta una operación con complejidad $O(n^{\wedge 3})$.

Capítulo III: Análisis y Diseño

Capítulo III – Análisis y Diseño

3.1 Especificación de requerimientos

3.1.1 Metodología

El proyecto se desarrolla a través del modelo lineal secuencial, comenzando con el análisis de los requisitos del software para comprender la naturaleza del programa a construir, el dominio de la información del software, así como la función requerida y su comportamiento. Se continúa con el diseño en donde se traducen los requisitos en una representación del software para que se pueda evaluar su calidad antes de que comience la codificación, dicho diseño se debe traducir en una forma legible por la máquina y esta tarea se lleva a cabo con la generación de código. Una vez generado el código se comenzará con las pruebas que se centrarán principalmente en los procesos lógicos internos del software y en los procesos externos funcionales.

Se utiliza el paradigma de programación orientada a objetos de modo de definir los programas en términos de clases de objetos. Así también se modela, construye y documenta a través del Lenguaje Unificado de Modelado (UML).

3.1.2 Patrones de Diseño

Los Patrones de Diseño (Design Patterns) son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño es una solución a un problema de diseño no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y re-usable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias).

3.1.2.1 Patrones utilizados

3.1.2.1.1 Patrón "Data Access Object"

El problema que viene a resolver este patrón es el de contar con diversas fuentes de datos (base de datos, archivos, servicios externos, etc.). De tal forma que se encapsula la forma de acceder a la fuente de datos. Este patrón surge históricamente de la necesidad de gestionar una diversidad de fuentes de datos, aunque su uso se extiende al problema de encapsular no sólo la fuente de datos, sino además ocultar la forma de acceder a los datos. Se trata de que el software cliente se centre en los datos que necesita y se olvide de cómo se realiza el acceso a los datos o de cuál es la fuente de almacenamiento.

3.1.2.1.2 Patrón “Síngleton” [Cra99]

El patrón de diseño Síngleton (instancia única) se implementa en todas las clases de la lógica del negocio con el objetivo de restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.

Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

Para asegurar que la clase no pueda ser instanciada nuevamente se reguló el alcance del constructor (con atributos como protegido o privado). Y se restringió el hecho de instanciar un objeto si ya existe.

El patrón Síngleton provee una única instancia global gracias a que:

- La propia clase es responsable de crear la única instancia.
- Permite el acceso global a dicha instancia mediante un método de la clase.

3.1.2.1.3 Patrones GRASP [Cra99]

3.1.2.1.3.1 Creador

Se aplicó este patrón de manera que se tenga la responsabilidad de crear una instancia de otra clase en uno de los siguientes casos:

- B *agrega* los objetos de A
- B *contiene* los objetos de A
- B *registra* las instancias de los objetos A
- B *utiliza* específicamente los objetos A

3.1.2.1.3.2 Bajo acoplamiento

El acoplamiento es una medida de fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo acoplamiento no depende de muchas otras **[Cra99]**.

Al contar con un alto acoplamiento surgen los siguientes problemas:

- Los cambios de las clases afines ocasionan cambios locales.
- Son más difíciles de entender cuando están aisladas.
- Son más difíciles de reutilizar porque se requiere la presencia de otras clases de las que disponen.

De modo de evitar estos problemas se asignó una responsabilidad para mantener bajo acoplamiento en cada una de las clases controladoras, y las clases de la lógica del negocio haciendo que exista poca dependencia para permitir un aumento de la reutilización.

3.1.2.1.3.3 Alta cohesión

La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme **[Cra99]**. Una clase con baja cohesión hace muchas cosas no afines o un trabajo excesivo.

No conviene este tipo de clases pues presentan los siguientes problemas:

- Son difíciles de comprender
- Son difíciles de reutilizar
- Son difíciles de conservar
- Son delicadas: las afectan constantemente los cambios

Se implementó este patrón en cada clase de manera de evitar la existencia de baja cohesión, al realizar éstas sólo lo necesario y lo correspondiente a ellas, que a menudo representan un alto grado de abstracción o han asumido responsabilidades que deberían haber delegado a otros objetos.

3.1.2.1.3.4 Controlador [Cra99]

Se implementó este patrón asignando la responsabilidad del manejo de un mensaje de los eventos del sistema a la clase Controlador y ControladorDAO, que representan la interacción del usuario a través de la interfaz con las clases internas del sistema en donde se encuentra la lógica de las operaciones y el almacenamiento persistente.

3.1.3 Análisis de Requerimientos

Los requerimientos son una descripción de las necesidades o deseos de un producto. La meta primaria de la fase de requerimientos es identificar y documentar lo que en realidad se necesita, en una forma que claramente se lo comunique al cliente y a los miembros del equipo de desarrollo. El reto consiste en definirlos de manera inequívoca, de modo que se detecten los riesgos y no se presenten sorpresas al momento de entregar el producto **[Cra99]**.

Los requerimientos del sistema, pueden ser divididos en dos categorías:

- **Requerimientos funcionales**, los cuales se basan en las principales funcionalidades que el sistema debe realizar.

- **Requerimientos no funcionales** que indican el desempeño necesario para lograr que el sistema sea preciso, robusto y en tiempo real.

Cabe destacar, que a continuación se muestra los requisitos funcionales, no funcionales y planillas combinadas, detallando a qué módulo del sistema pertenece tal análisis.

3.1.3.1 Requerimientos Funcionales

Ref #	Función	Categoría
R1	*Capturar imagen de huella digital*	*Evidente*
R2	*Extraer minucias de la huella digital*	*Oculto*
R3	*Permitir ingresar los datos de una persona*	*Evidente*
R4	*Ver los datos personales de una Persona*	*Evidente*
R5	*Permitir verificar la identidad de una Persona*	*Evidente*
R6	*Permitir identificar la identidad de una Persona*	*Evidente*
R7	*Crear índice a partir de los datos almacenados en una base de datos*	*Oculto*
R8	*Almacenar los datos de la persona incluyendo su huella digital*	*Evidente*
R9	*Crear un índice métrico*	*Oculto*
R10	*Buscar los datos de una persona en el sistema de almacenamiento persistente*	*Oculto*
R11	*Insertar los datos de una persona en el índice métrico*	*Oculto*
R12	*Obtener una lista de personas a través del índice métrico con sus huellas digitales similares a la huella digital de entrada*	*Oculto*

Tabla 3.1: funcionalidades del sistema

3.1.3.2 Requerimientos no funcionales

Atributos	Detalle y restricciones
Facilidad de uso	*Interfaz de usuario visual fácil de comprender*
Tiempo de respuesta	*El proceso de verificación de identidad debe tomar a lo más 1 segundo.* *El proceso de identificación de identidad y el de la creación del índice a partir de los datos de la BD dependen mucho de la cantidad de personas almacenadas, por lo que el tiempo es aproximado a 1 segundo por cada 100 personas.*
Metáfora de interfaz	*Ventanas de imágenes y botones*
Tolerancia a fallas	*No debe existir pérdida de datos en el mecanismo de almacenamiento persistente frente a algún incidente.*
Plataformas del S.O.	*Multiplataforma*
Lenguaje de programación	*Java*

Tabla 3.2: Atributos del sistema.

3.1.3.3 Planilla combinada

Ref #	Función	Categoría	Atributo
R1	*Capturar imagen de huella digital*	*Evidente*	*Facilidad de uso, Tiempo de respuesta, Metáfora de interfaz*
R2	*Extraer minucias de la huella digital*	*Oculto*	*Tiempo de respuesta*
R3	*Permitir ingresar los datos de una persona*	*Evidente*	*Metáfora de interfaz*
R4	*Ver los datos personales de una Persona*	*Evidente*	*Metáfora de interfaz, Tolerancia a fallas*
R5	*Permitir verificar la identidad de una Persona*	*Evidente*	*Facilidad de uso, Tiempo de respuesta, Metáfora de interfaz*
R6	*Permitir identificar la identidad de una Persona*	*Evidente*	*Facilidad de uso, Tiempo de respuesta, Metáfora de interfaz*
R7	*Crear índice a partir de los datos almacenados en una base de datos*	*Oculto*	*Tiempo de respuesta*
R8	*Almacenar los datos de la persona incluyendo su huella digital*	*Evidente*	*Facilidad de uso, Tiempo de respuesta, Metáfora de interfaz*
R9	*Crear un índice métrico*	*Oculto*	*Tiempo de respuesta*
R10	*Buscar los datos de una persona en el sistema de almacenamiento persistente*	*Oculto*	*Tiempo de respuesta*
R11	*Insertar los datos de una persona en el índice métrico*	*Oculto*	*Tiempo de respuesta*
R12	*Obtener una lista de personas a través del índice métrico con sus huellas digitales similares a la huella digital de entrada*	*Evidente*	*Facilidad de uso, Tiempo de respuesta, Metáfora de interfaz*

Tabla 3.3: Tabla combinada entre requisitos funcionales y no funcionales del sistema.

3.1.4 Casos de Uso

El caso de uso es un documento narrativo que describe la secuencia de eventos de un actor (agente externo) que utiliza un sistema para completar un proceso. Los casos de uso son historias o casos de utilización de un sistema; no son exactamente los requerimientos ni las especificaciones funcionales, si no que simplifican e influyen tácitamente los requerimientos en las historias que narran **[Cra99]**.

3.1.4.1 Actores del sistema

Los actores del sistema sirven para identificar las interacciones que tiene el sistema con el exterior, que pueden representar:

- Un rol que un usuario puede jugar con respecto al sistema.
- Una entidad, como otro sistema o base de datos que reside fuera del sistema.

En este proyecto, los actores identificados son: el administrador que inicia cada una de las funcionalidades del sistema; el capturador de imagen, quien provee la imagen de la huella digital de las personas; la persona, quien provee sus datos personales que se almacenan en el sistema (ver Figura 3.1).

Figura 3.1: Actores del sistema: Administrador <iniciador>, Lector de huella digital, y Persona.

3.1.4.2 Descripción de casos de uso

3.1.4.2.1 Caso de uso: Ingresar persona	
Actores:	Administrador, persona, lector de huella digital.
Propósito:	Agregar una nueva persona al sistema de almacenamiento y al índice métrico.
Resumen:	Almacenar e insertar los datos de una persona incluyendo su huella digital al sistema de almacenamiento y al índice métrico, respectivamente.
Tipo:	Primario.
Referencias cruzadas:	R1, R2, R3, R10

Curso normal de eventos	
Acción de los actores	**Respuesta del Sistema**
1. El administrador solicita el enrolamiento de una nueva persona.	
	2. Se extraen las minucias de la huella digital capturada por el lector de huellas.
	3. Se solicitan los datos personales de la persona al administrador.
4. El administrador ingresa los datos personales de la persona.	
	5. Se almacenan e insertan los datos personales de la persona incluyendo la información de la huella digital en el sistema de almacenamiento y en el índice métrico, respectivamente.

Cursos alternos

Punto 5: El rut ya existe en el sistema. Se emite un mensaje de error.

Tabla 3.4: Descripción de casos de uso Ingresar persona.

3.1.4.2.2 Caso de uso: Verificar persona	
Actores:	Administrador, persona, lector de huellas digitales.
Propósito:	Verificar si la identidad de la persona corresponde al rut ingresado.
Resumen:	Se verifica si la huella digital de entrada corresponde con la huella digital almacenada, la cual es obtenida por el rut ingresado.
Tipo:	Primario.
Referencias cruzadas:	R1, R2, R4, R5, R9
Curso normal de eventos	
Acción de los actores	**Respuesta del Sistema**
1. El administrador solicita verificar la identidad de una persona	
	2. Se extraen las minucias de la huella digital capturada por el lector de huellas.
	3. Se solicita al administrador que ingrese el rut de la persona.
4. El administrador ingresa el rut de la persona.	
	5. Se obtiene del sistema de almacenamiento la información de los datos personales de la persona incluyendo su huella digital.
	6. Se compara la huella digital de entrada con la huella digital almacenada.
	7. Se emite un mensaje indicando si la huella digital de la persona corresponde o no a la huella almacenada.
Cursos alternos	
Punto 5: El rut no existe en el sistema. Se emite un mensaje de error.	

Tabla 3.5: Descripción de casos de uso Verificar persona.

3.1.4.2.3 Caso de uso: Identificar persona	
Actores:	Administrador, persona, lector de huellas digitales.
Propósito:	Obtener una lista de personas con huellas digitales similares a una huella digital de entrada.
Resumen:	
Tipo:	Primario.
Referencias cruzadas:	R1, R2, R11
Curso normal de eventos	
Acción de los actores	**Respuesta del Sistema**
1. El administrador solicita verificar la identidad de una persona	
	2. Se extraen las minucias de la huella digital capturada por el lector de huellas.
	3. El sistema compara la huella digital de entrada con las huellas digitales del índice métrico.
	4. Se entrega una lista de personas que tienen sus huellas digitales similares a la huella digital de entrada.
Cursos alternos	
Punto 3: No existen datos en el sistema. Se emite un mensaje de error.	

Tabla 3.6: Descripción de casos de uso Identificar persona.

3.1.4.3 Diagrama de casos de uso

La Figura 3.7 muestra el diagrama de casos de uso del sistema:

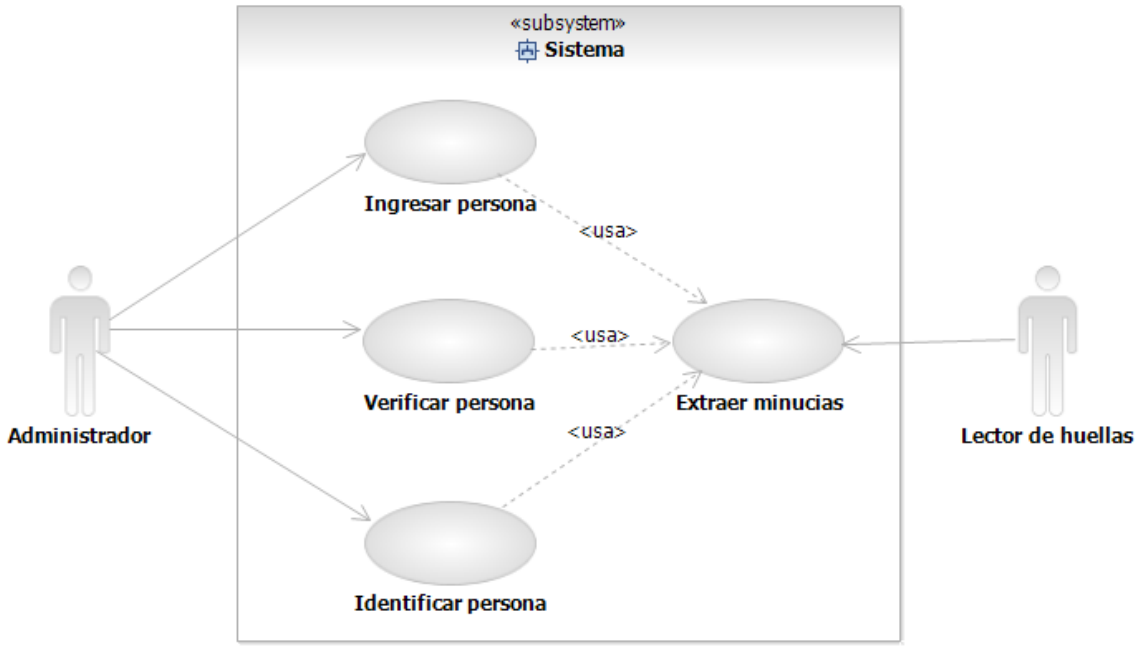


Figura 3.7: Diagrama de casos de uso del sistema

3.2 Análisis del Sistema

En la fase de análisis de un ciclo de desarrollo se investiga sobre el problema, sobre los conceptos relacionados con el subconjunto de casos de uso que se esté tratando. Se intenta llegar a una buena comprensión del problema, sin entrar en cómo va a ser la solución en cuanto a detalles de implementación, sino más a los conceptos que describen lo que se quiere hacer.

3.2.1 Modelo conceptual

En el modelo conceptual se tiene una representación de conceptos del mundo real, no de componentes software. El objetivo de este diagrama es aumentar la comprensión del problema, reflejando conceptos y relaciones básicas del sistema a implementar.

- *Administrador:* Este es uno de los principales actores, es quien inicia y coordina cada acción en el sistema.

- *Persona:* Provee al sistema sus datos personales y su huella digital.
- *Huella digital:* Es el principal componente del sistema, de la huella digital se extraen las características necesarias para identificar a una persona.
- *Característica:* Representa a cada punto característico encontrado dentro de la imagen de la huella digital.

En la Figura 3.8 se puede ver el modelo conceptual del sistema:

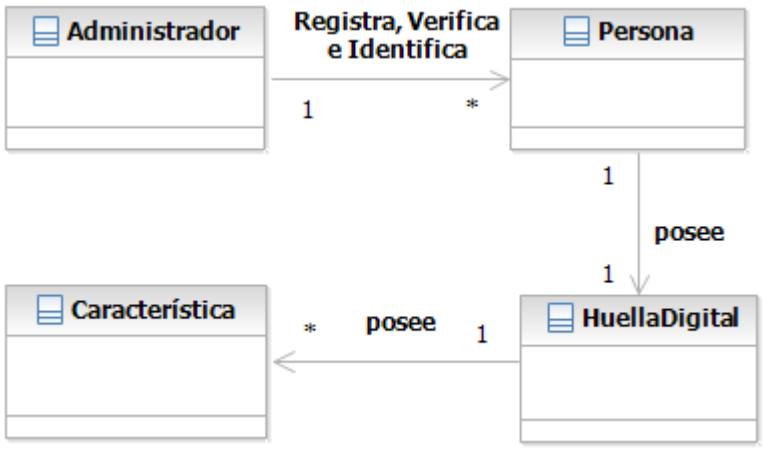


Figura 3.8: Modelo conceptual de sistema.

3.2.2 Diagramas de secuencias del sistema

Los diagramas de secuencia de las Figuras 3.9, 3.10 y 3.11 muestran el comportamiento del sistema, visto éste como una caja negra.

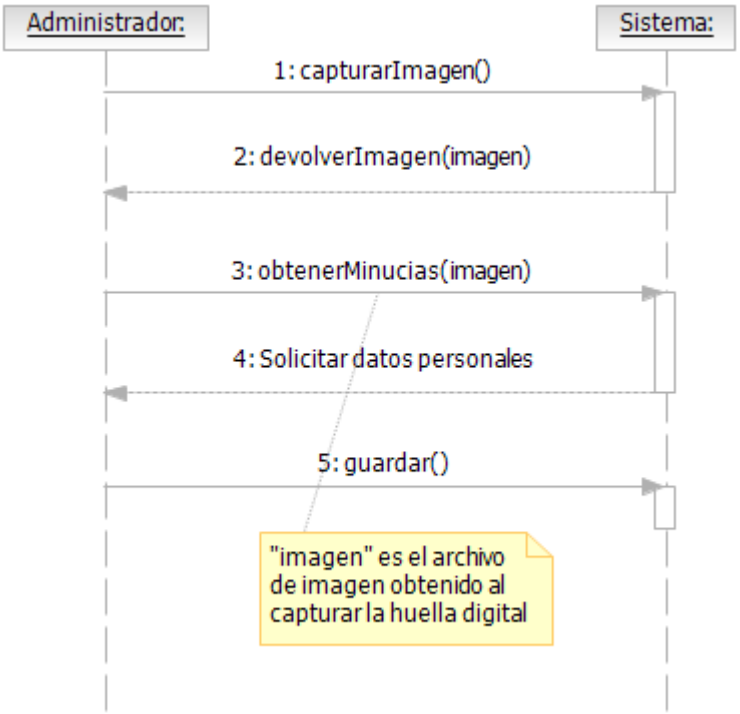


Figura 3.9: Diagrama de secuencia – Ingresar Persona

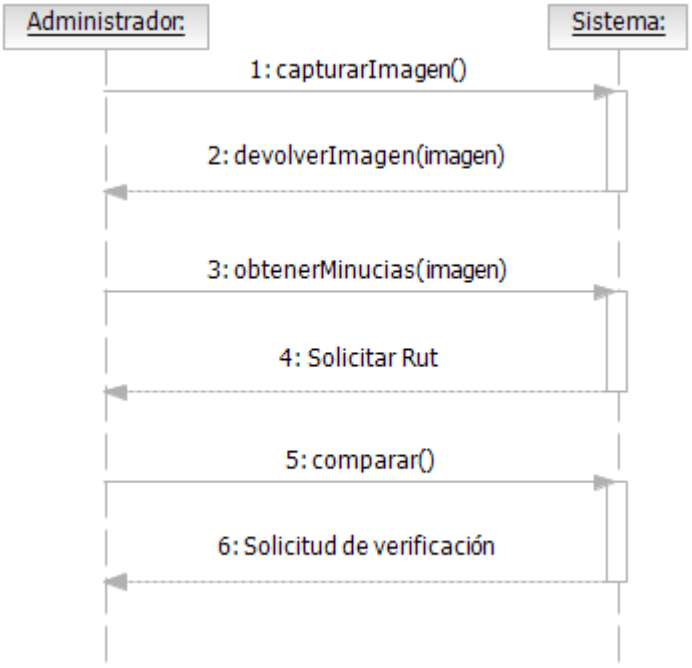


Figura 3.10: Diagrama de secuencia – Verificar Identidad

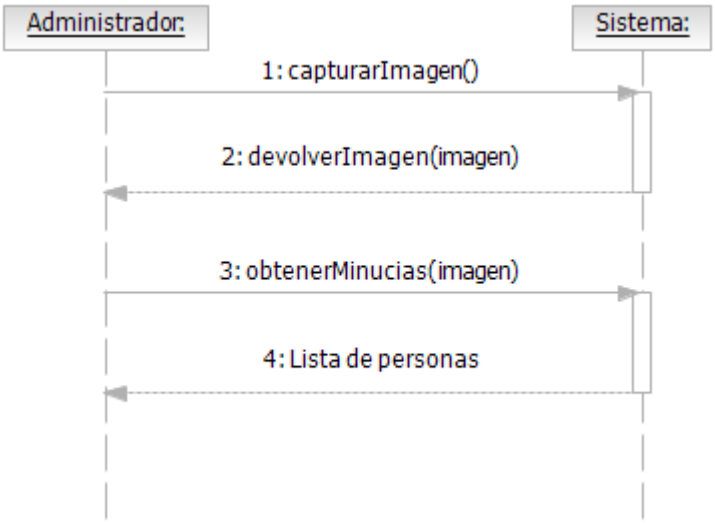


Figura 3.11: Diagrama de secuencia – Identificar Persona

3.3 Diseño del Sistema

En esta etapa se presenta una solución lógica para satisfacer los requerimientos del sistema. Aquí, se muestran los casos de uso reales del sistema, el diagrama de paquetes, el diagrama arquitectónico y los diagramas de clases del sistema, tanto en su versión simplificada como completa. Los diagramas de clases serán divididos por módulos para una mejor comprensión de los mismos.

3.3.1 Casos de uso reales

En las Figura 3.12 se muestra la pantalla que permite realizar el ingreso y enrolamiento de una persona, en donde se deben ingresar sus datos personales y la imagen de su huella digital a través del lector de huellas, en la Figura 3.13 se muestra la pantalla que permite realizar la verificación de identidad de una persona, en donde sólo se de ingresar el rut y la huella digital de la persona, y el sistema indica si la persona es quien dice ser, y por último en la Figura 3.14 se muestra la pantalla que permite realizar la identificación de una persona, en donde sólo es necesario ingresar la huella digital de una persona y el radio cobertor de búsqueda, mostrando posteriormente una lista de persona que tienen huellas digitales similares a la de entrada.

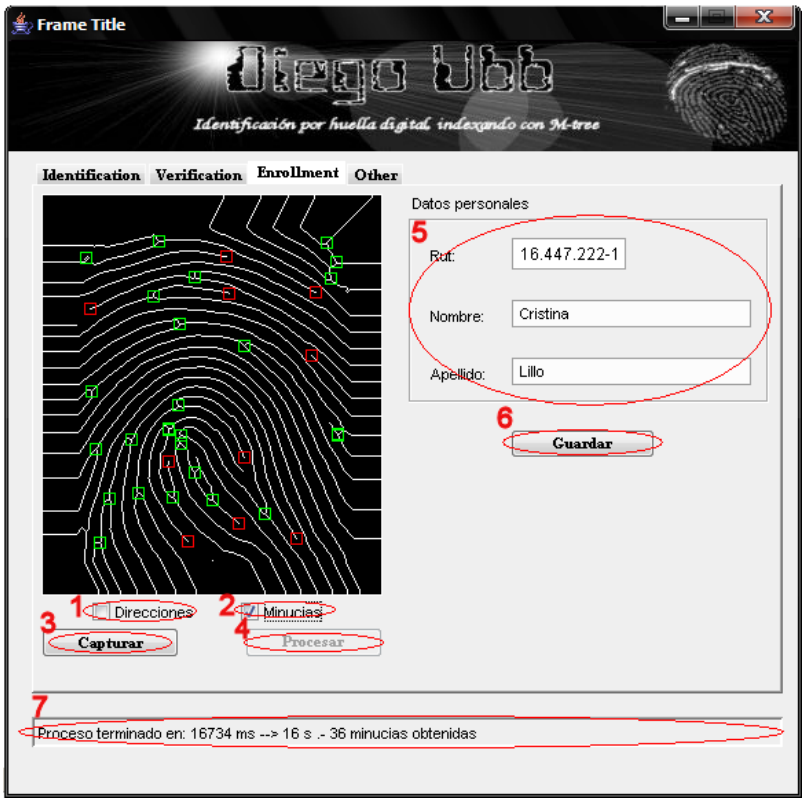


Figura 3.12: Pantalla para el ingreso y enrolamiento de una persona.

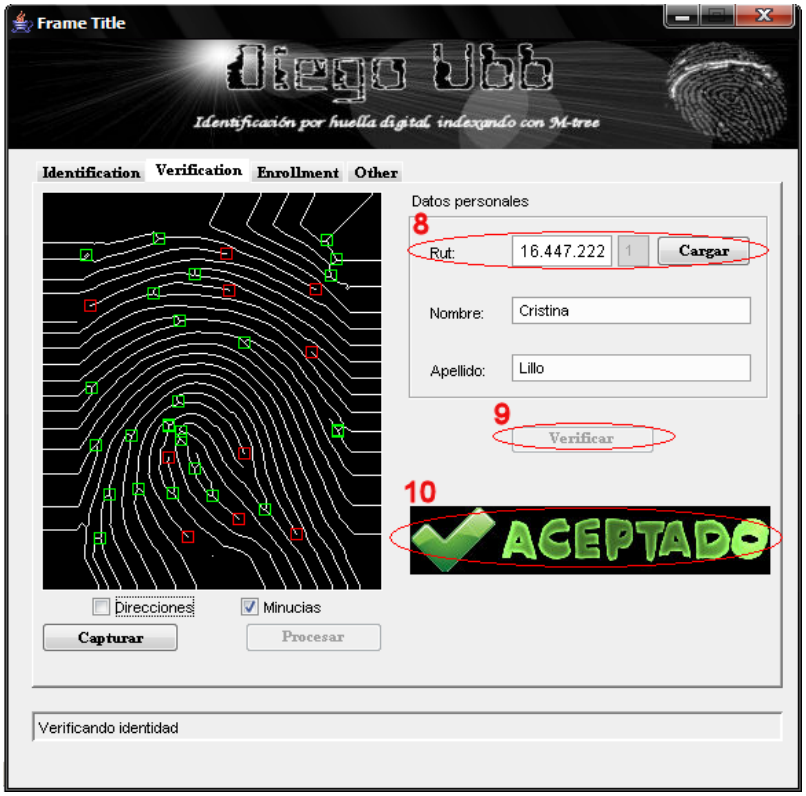


Figura 3.13: Pantalla para la verificación de identidad.

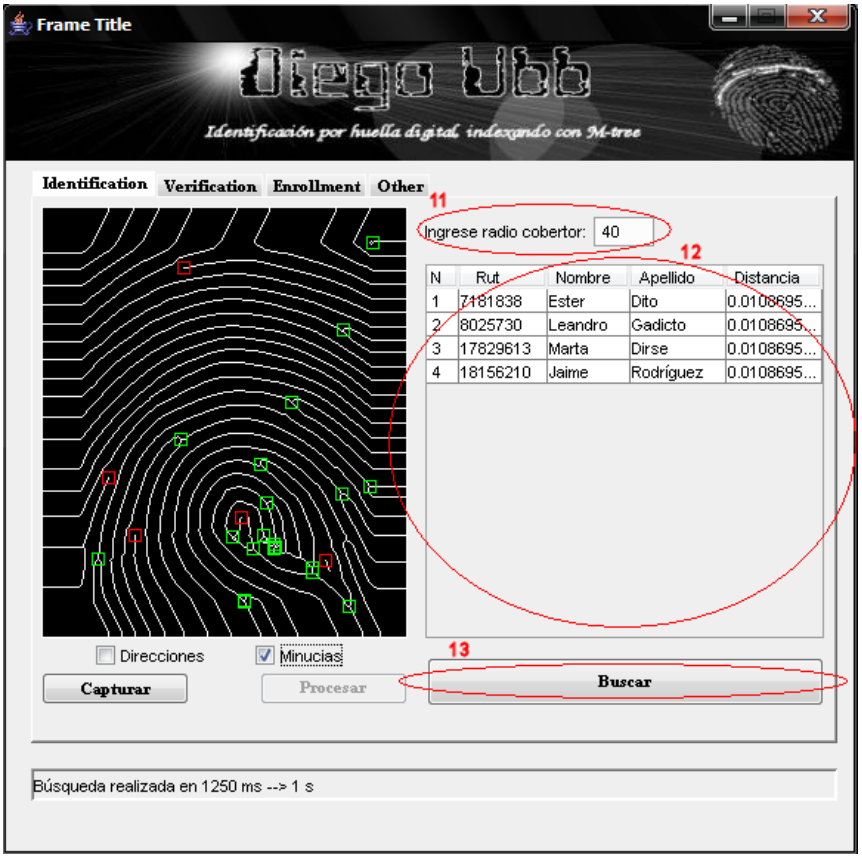


Figura 3.14: Pantalla para la identificación de una persona.

A continuación, en la Tabla 3.7 se muestran las funciones de los casos de uso reales del sistema.

Nro	Componente	Función
1	Checkbox “Direcciones”	Permitir observar las direcciones de las crestas de la huella digital ingresada.
2	Checkbox “Minucias”	Permitir observar las minucias de la huella digital ingresada.
3	Botón “Capturar”	Captura la imagen de la huella digital desde el lector de huellas.
4	Botón “Procesar”	Extrae las minucias de la huella digital.
5	Panel “Datos personales”	Permite ingresar los datos personales de una persona.
6	Botón “Guardar”	Realiza el almacenamiento de los datos personales y las minucias de la huella digital.
7	Label de estado	Muestra el estado de procesamiento del sistema.
8	Botón “Cargar”	Obtiene los datos personales de una persona según el rut ingresado.
9	Botón “Verificar”	Inicia el proceso de verificación de identidad.
10	Label de verificación	Indica si la huella corresponde a la almacenada.
11	Radio cobertor de búsqueda	Permite ingresar el radio para la búsqueda por rango
12	Listado de personas	Muestra todas las personas con huellas similares a la de entrada.
13	Botón “Buscar”	Inicia el proceso de identificación de identidad.

Tabla 3.7: Casos de uso reales del sistema.

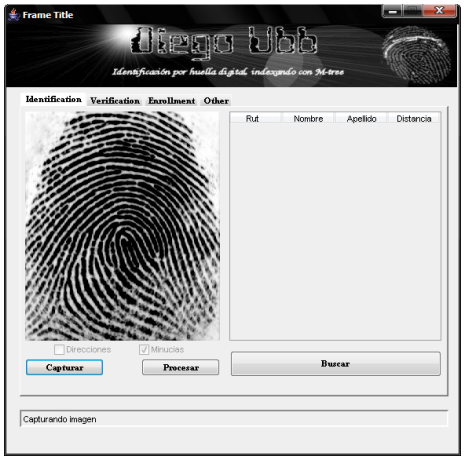
3.3.2 Arquitectura de la solución

La arquitectura utilizada es la clásica de tres capas como se muestra en la Figura 3.15. Dicha arquitectura es común en los sistemas que abarcan una interfaz para el usuario y el almacenamiento persistente de datos.

Esta arquitectura se compone de las siguientes tres capas:

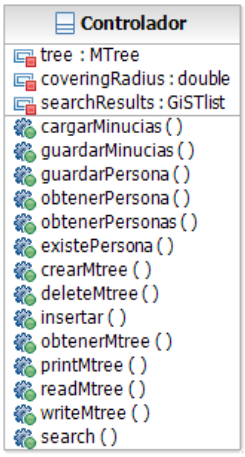
1. Presentación: Incluye interfaz que son las que interactúan con los usuarios.
2. Lógica: Incluye las clases que corresponden a la lógica del problema. Implementa tareas y reglas que rigen el proceso.
3. Almacenamiento: Incluye el mecanismo de almacenamiento persistente, en este caso la base de datos MySQL, y el índice métrico.

Presentación



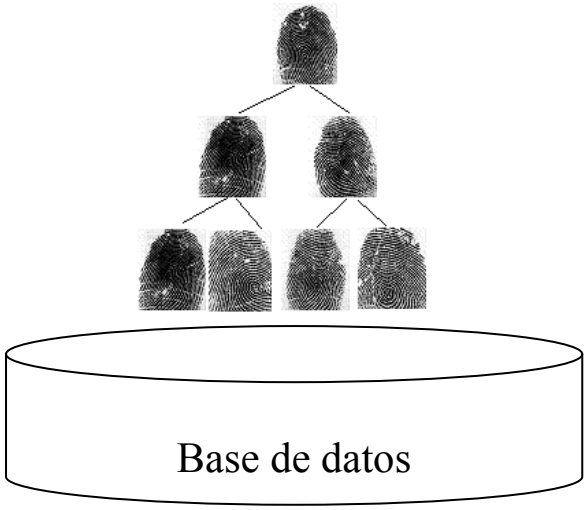
(a)

Lógica



(b)

Almacenamiento



(c)

Figura 3.15: Arquitectura de tres capas.

3.3.3 Diagrama de paquetes

El diagrama de la Figura 3.16 muestra la subdivisión hecha por módulos o paquetes, en donde el paquete central (logic) contiene la clase controlador quien realiza toda la conexión dentro del sistema, siendo esta clase quien controla cada acción, y principalmente es quien realiza la creación del índice métrico utilizando la clase MTree del paquete mtree. Los paquetes logic y persistance son descritos mas abajo, mientras que los paquetes GiST, mtree, ARLib y DLib se describen en la Sección **4.2**.

En este diagrama se muestran las principales conexiones de las clases del sistema, mientras que cada una de las demás conexiones se muestran en las descripciones de los demás paquetes.

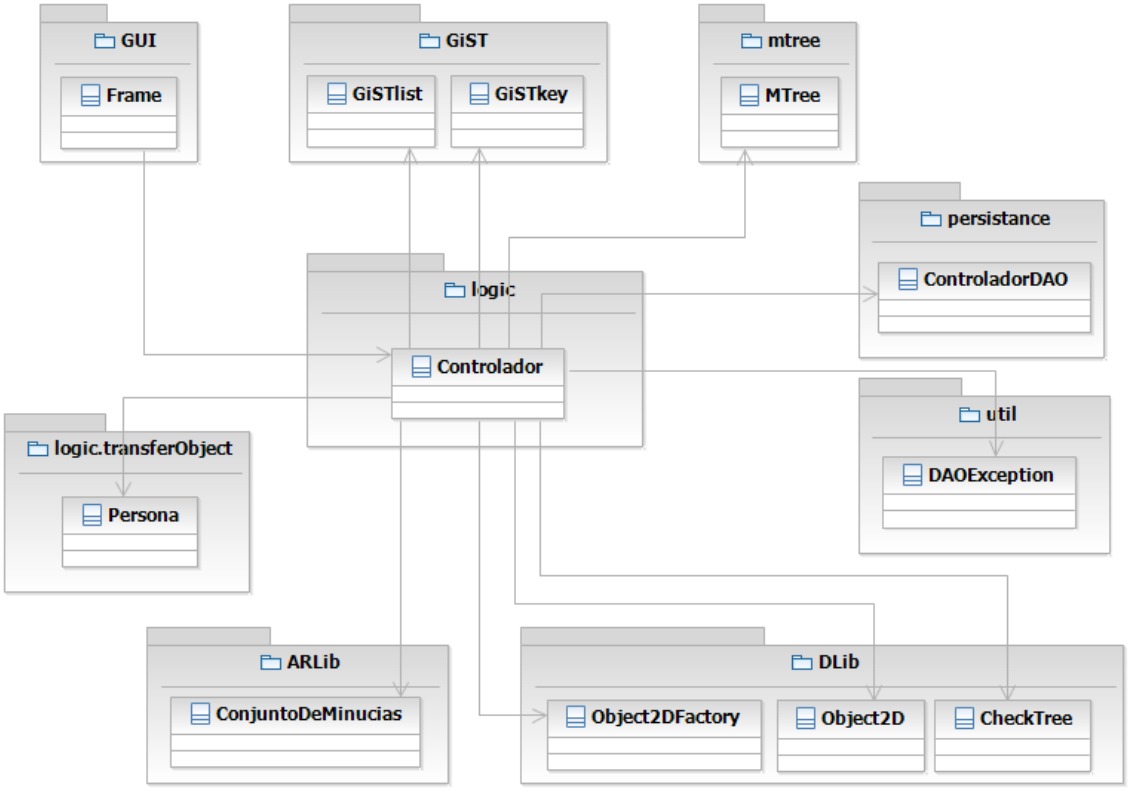


Figura 3.16: Diagrama de paquetes del sistema.

3.3.4 Diagramas de clases por módulos

Los diagramas de clases se dividen en paquetes o módulos y se presentan a continuación. Estos diagramas representan la especificación para las clases software de la aplicación.

3.3.4.1 Diagrama de clases paquete logic

La Figura 3.17 muestra el diagrama de clases del paquete logic.

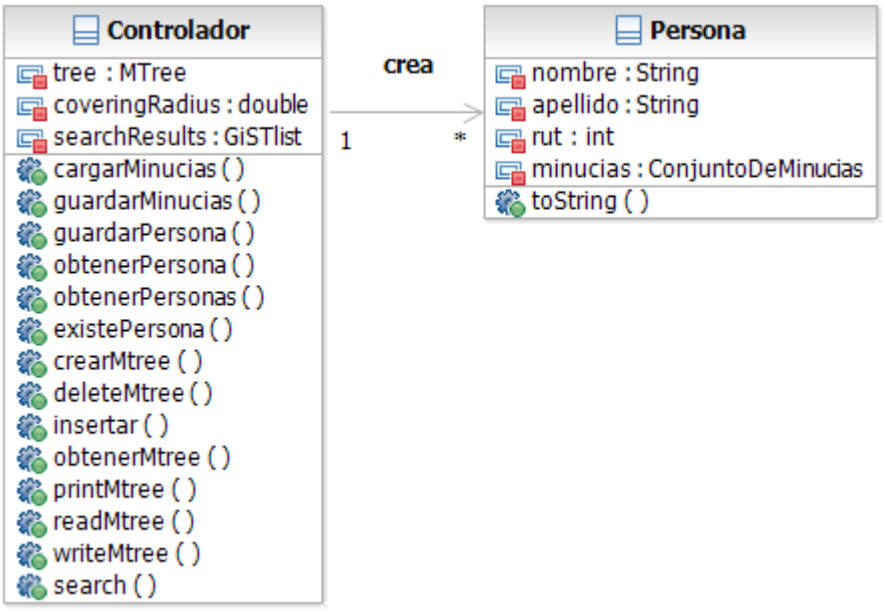


Figura 3.17: Diagrama de clases – Paquete logic.

3.3.4.2 Diagrama de clases paquete persistance

El diagrama de clases del paquete persistance se presenta en la Figura 3.18.

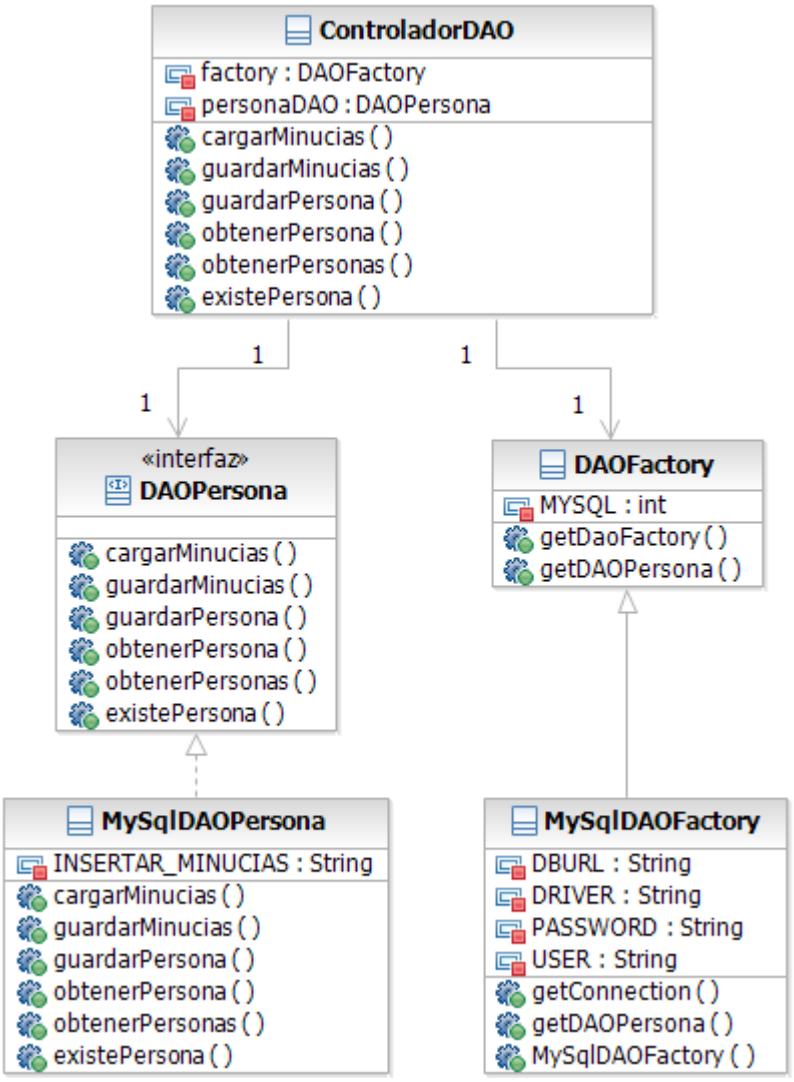


Figura 3.18: Diagrama de clases – Paquete persistance.

3.3.5 Diagrama de Interacción

Un diagrama de interacción explica gráficamente las interacciones existentes entre las instancias (y las clases) del modelo de éstas.

UML define dos tipos de estos diagramas; ambos sirven para expresar interacciones semejantes o idénticas de mensaje:

1. diagramas de colaboración
2. diagramas de secuencia

Los diagramas de colaboración describen las interacciones entre objetos en un formato de grafo o red **[Cra99]**.

3.3.5.1 Diagrama de colaboración ingresarPersona

En la Figura 3.19 se presenta el diagrama de colaboración ingresarPersona.

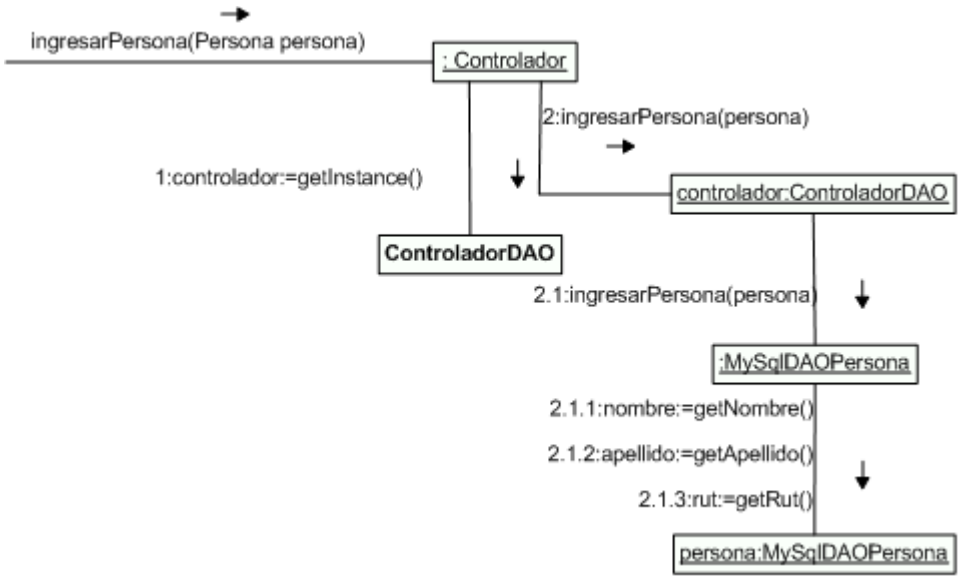


Figura 3.19: Diagrama de colaboración – ingresarPersona

3.3.5.2 Diagrama de colaboración obtenerPersona

En la Figura 3.20 se presenta el diagrama de colaboración obtenerPersona.

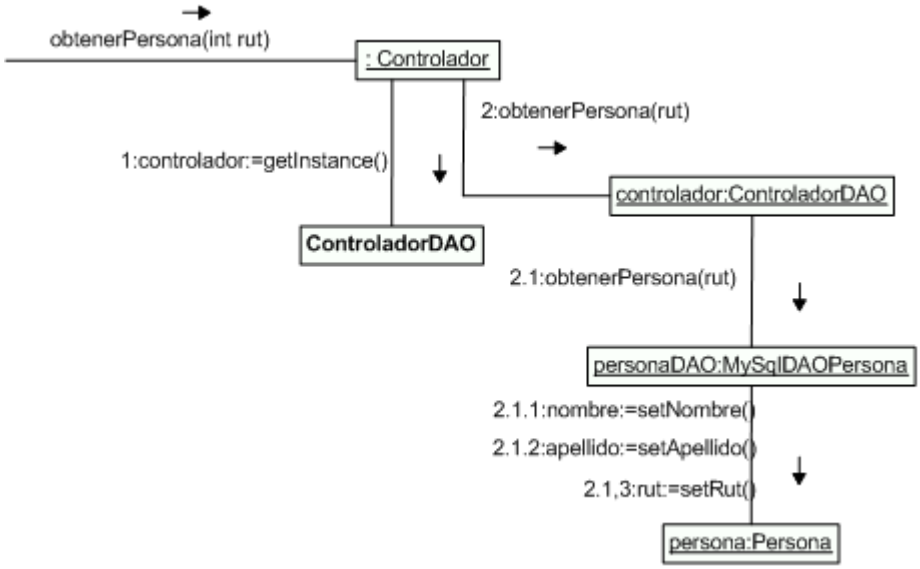


Figura 3.20: Diagrama de colaboración – obtenerPersona

3.3.5.3 Diagrama de colaboración guardarMinucias

En la Figura 3.21 se presenta el diagrama de colaboración guardarMinucias.

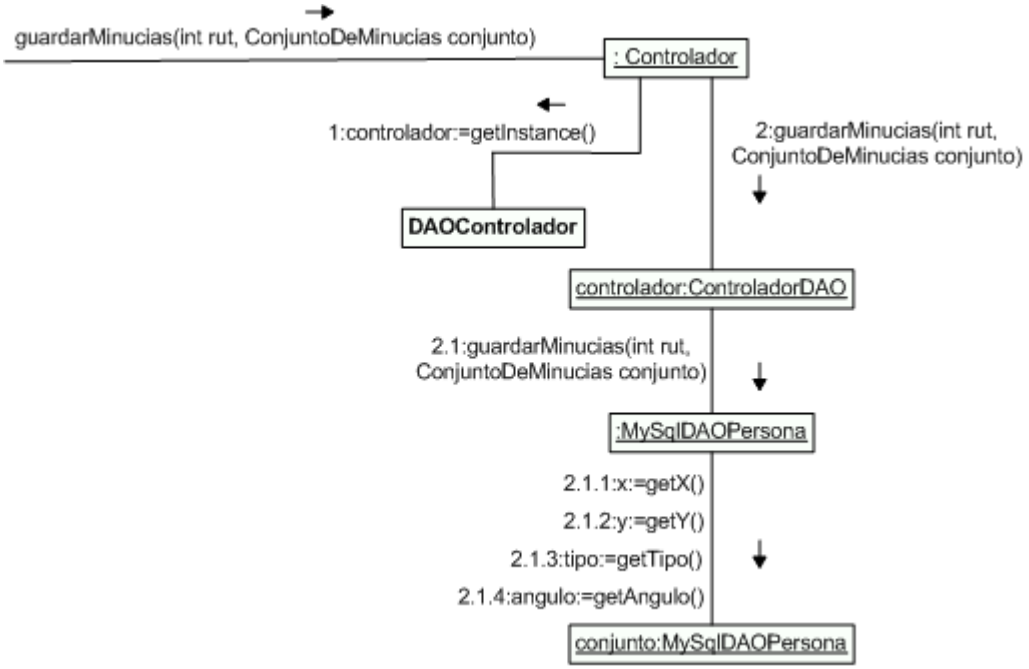


Figura 3.21: Diagrama de colaboración – guardarMinucias

3.3.5.4 Diagrama de colaboración cargarMinucias

En la Figura 3.22 se presenta el diagrama de colaboración cargarMinucias.

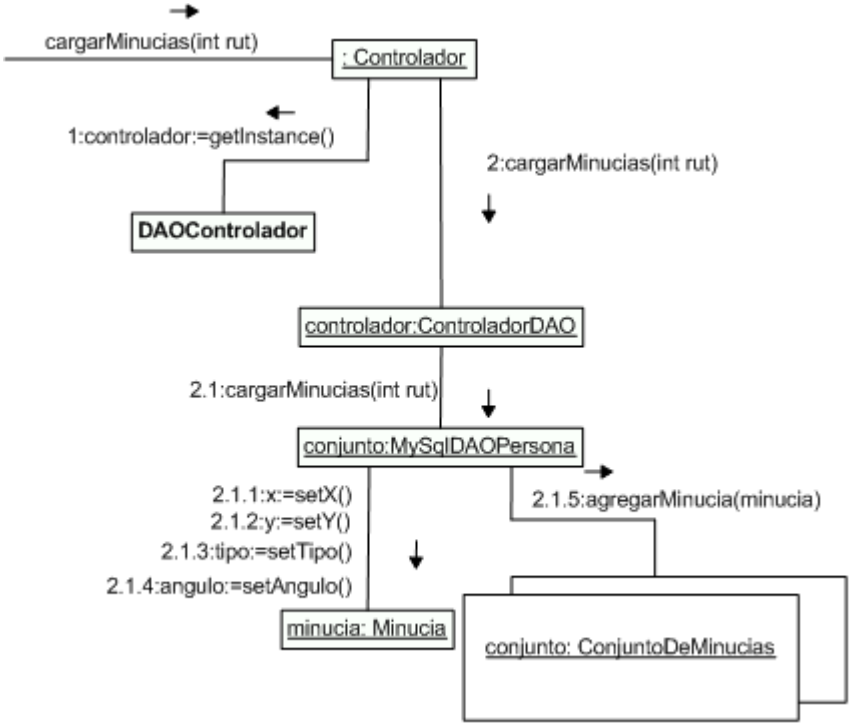


Figura 3.22: Diagrama de colaboración – cargarMinucias

3.3.5.5 Diagrama de colaboración crearMtree

En la Figura 3.23 se presenta el diagrama de colaboración crearMtree.

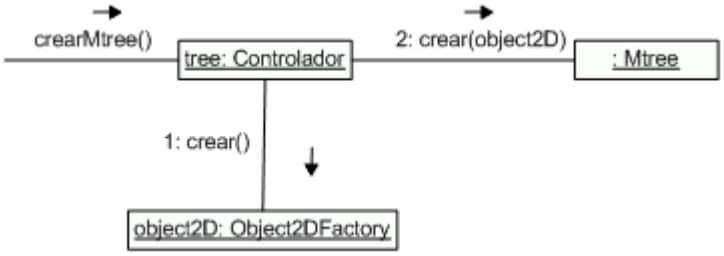


Figura 3.23: Diagrama de colaboración – crearMtree

3.3.5.6 Diagrama de colaboración insertar

En la Figura 3.24 se presenta el diagrama de colaboración insertar.

Figura 3.24: Diagrama de colaboración – insertar

3.3.5.7 Diagrama de colaboración buscar

En la Figura 3.25 se presenta el diagrama de colaboración buscar.

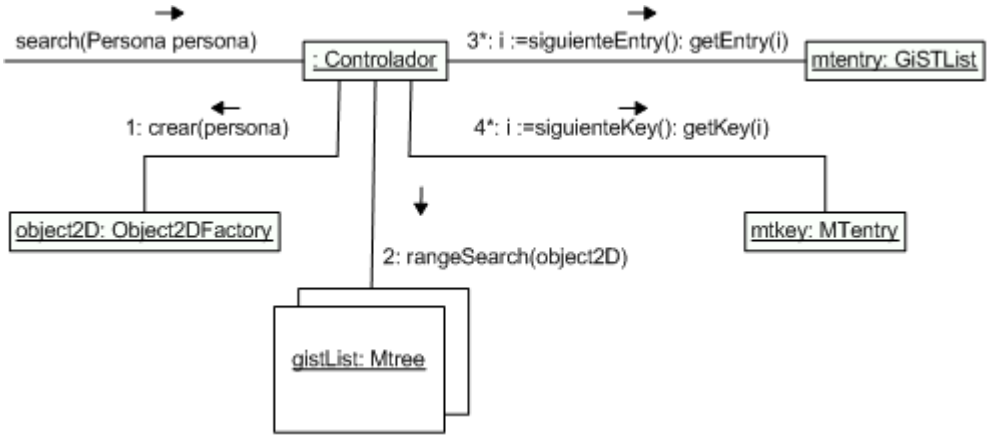


Figura 3.25: Diagrama de colaboración – search

3.3.5.8 Diagrama de colaboración printMtree

En la Figura 3.26 se presenta el diagrama de colaboración imprimir Mtree.

Figura 3.26: Diagrama de colaboración – printMtree

3.3.6 Modelo Entidad Relación

El Modelo Entidad Relación (MER) fue se define como una herramienta de modelamiento de datos que describe las asociaciones que existen entre las diferentes categorías de datos dentro de un sistema de empresa o de información.

Utiliza dos abstracciones principales:

- Entidad
- Atributo

3.3.6.1 Entidad

Es cualquier ente, real o abstracto (tangible o intangible), distinguible en la organización, sobre el cual se desea almacenar datos. Se dibuja como un rectángulo y cada rectángulo representa todas las ocurrencias o presencias de la entidad respectiva. Las entidades deben nominarse en singular y los tipos de suelen clasificarse en:

- Participantes o roles
- Sucesos
- Lugares
- Bienes tangibles

Los datos que describen una entidad toman la forma de atributos.

3.3.6.2 Atributos

Es una característica común a todas las ocurrencias de una entidad concreta y que toma un valor concreto para cada ocurrencia de la entidad que describe. Debe tener más de un valor admisible. El universo de valores puede ser:

- Un intervalo de valores (3,65..16,85)
- Un conjunto limitado de valores ({1, 2, 3, 4}, {soltero, casado, viudo, separado})
- Un valor binario ({Sí, No}, {Activo, Inactivo})
- Un conjunto casi infinito de valores (Nombre Cliente)

Debe existir al menos un atributo de datos que tome un valor único para cada ocurrencia, es decir no pueden existir dos ocurrencias con igual valor para ese atributo.

3.3.6.3 Relación

Es una asociación natural que existe entre una o más entidades, actividades o sucesos que unen a dos o más entidades entre sí. Se representa como rombo que conecta las entidades que relaciona y se debe nominar mediante un verbo o una frase verbal.

Existen dos reglas que definen una relación:

- **Orden**: Define si la relación entre las entidades es obligatoria u opcional, es decir, el número mínimo de ocurrencias de una entidad con respecto a la otra.

- **Cardinalidad**: Define el número máximo de ocurrencias de una entidad para una única presencia de la entidad relacionada.

3.3.6.4 Modelo entidad relación del sistema.

El modelo entidad relación del sistema se presenta en la Figura 3.27.

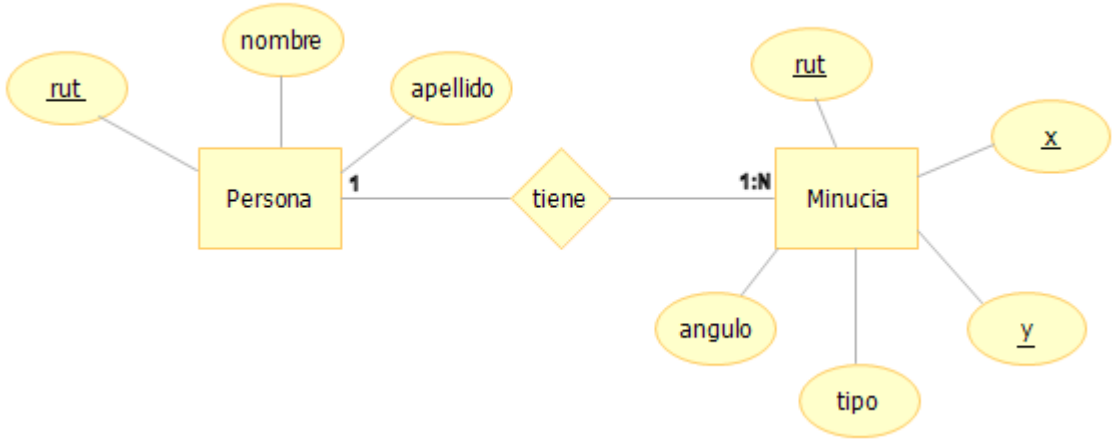


Figura 3.27: Modelo Entidad Relación.

Capítulo IV:

Implementación y Pruebas

Capítulo IV – Implementación y Pruebas

4.1 Matching de minucias

4.1.1 Alineación de minucias basada en la transformada de Hough.

4.1.1.1 THG y la transformación entre los vectores de características Query y Template

El problema del *matching* entre vectores de características asociados a los conjuntos de minucias de dos huellas dactilares tiene, como primera tarea, la determinación de la transformación (rotación, traslación, escalamiento) entre ambas **[Mal03]**.

Esta transformación debe considerar incluso que, para dos vectores de características de una misma huella dactilar pueden existir diferencias entre éstas: la desaparición de algunas minucias, la variación de la posición y orientación local de algunas de éstas debido al ruido que introduce el sensor, y a las deformaciones elásticas que presenta la piel. Esta transformación es, a priori desconocida. De esta manera la THG provee un método para la obtención de esta transformación.

Debido a que en este proyecto las imágenes Query y Template fueron escaneadas por el mismo sensor, no fue necesario considerar en el análisis, escalamiento entre los vectores de características. Sea (q_x, q_y) y β las coordenadas de posición y la orientación local de una minucia perteneciente al vector de características Query. Sean, (t_x, t_y) y α la posición y orientación asociadas a una minucia perteneciente al vector de características Template.

La transformación general entre ambos conjuntos considerará en forma explícita rotaciones y traslaciones, por lo que tendrá la forma (4-1):

$$\begin{pmatrix} q_x \\ q_y \end{pmatrix} = \begin{bmatrix} \cos\theta & -sen\theta \\ sen\theta & \cos\theta \end{bmatrix} \begin{pmatrix} t_x \\ t_y \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \tag{4-1}$$

donde θ es igual a $(\beta - \alpha)$, y $(\Delta x, \Delta y)$ es una traslación arbitraria. El ángulo θ indica la rotación necesaria para que la orientación[7] de la minucia Template coincida con la de la minucia Query. La transformación definida por (4-1) posee tres grados de libertad, esto último indica que el arreglo utilizado para la búsqueda de la transformación será de tres dimensiones: uno para la cuantización del ángulo θ, y dos para el desplazamiento en el eje *x* y en el eje *y*. El cluster del espacio de parámetros que interesa rescatar es aquel que representa la transformación existente entre los vectores Query y Template. Se postula que los parámetros que definen a éste corresponden a los índices asociados al máximo valor del arreglo acumulador utilizado para la transformación. De la discusión anterior resulta evidente que la estrategia para encontrar los parámetros de la transformación (4-1) será aplicar la THG.

Sea *Q* el vector de características Query con dimensión *dimQ* y *T* el vector Template con dimensión *dimT*. Sea además *Q[i]* la *i*-ésima minucia del vector *Q*, y *T[i]* la *i*-ésima minucia del vector *T*. Notemos que el problema (4-1) puede "separarse" en dos etapas: en primer lugar es posible obtener el ángulo de rotación θ entre los vectores de características y luego, a partir de éste, el vector de desplazamiento $(\Delta x, \Delta y)$.

Esto último reduce el costo computacional que involucra el cálculo de una THG 3D al de una THG 1D más una THG 2D **[DR99]**. Además, es importante notar que el rango al que pertenece un ángulo de rotación entre dos huellas y, por ende, entre dos vectores de características está acotado, sea éste $\Delta\theta$. A continuación se presenta un diagrama en bloques del proceso de cálculo de la THG propuesto, para la definición del ángulo de rotación. El caso 2D es análogo.

[7] La orientación de las minucias está basada en coordenadas polares, pues como es sabido que para aplicaciones prácticas la ecuación cartesiana de una línea recta no es de utilidad. Si la recta presenta una pendiente elevada entonces el número de niveles de cuantización será elevado y hará que una representación discreta del espacio de parámetros no sea adecuada. Además, el coeficiente de posición tampoco está acotado.

La Figura 4.1 muestra las principales tareas a realizar por la THG. En primer lugar se restan las orientaciones locales de las minucias Query y Template.

El proceso de cuantificación entrega la componente del arreglo acumulador en donde se recopilará evidencia. El algoritmo continúa acumulando evidencia hasta que todas las comparaciones entre minucias han sido consideradas.

En ese instante se procede a buscar los máximos del arreglo acumulador *A* y a tomar una decisión con respecto a cuál es el mejor parámetro para la transformación.

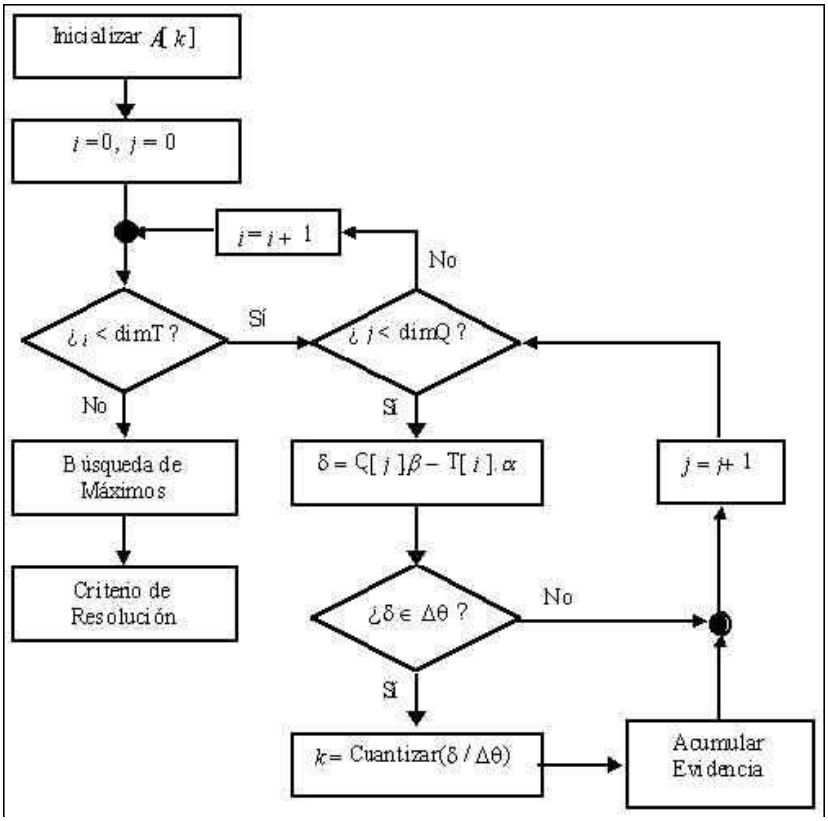


Figura 4.1: Diagrama en bloques de la THG para el ángulo de rotación entre los vectores T y Q.

4.1.1.2 Nuevo Criterio de Resolución

El algoritmo presentado en **[Lor04]** acumula evidencia incrementando al arreglo acumulador en una única posición *k* en cada iteración. El criterio de resolución para determinar la transformación está basado en encontrar la posición donde se encuentra el

máximo del arreglo acumulador. Esta estrategia sólo funciona bien en ausencia de ruido. La nueva propuesta es de incrementar, además de la posición ganadora *k*, a las vecinas inmediatas de ésta. Esto se realiza de la misma forma que en **[Lor04]**, pero incrementando en dos al arreglo acumulador en su posición ganadora y con valor uno a los vecinos inmediatos *k-1* y *k+1*. Para esta estrategia también se buscarán los valores y las posiciones de los seis mayores máximos locales asociados a la transformación que busca la rotación entre los vectores de características.

4.1.1.3 Pseudo-código algoritmo transformada de Hough 1D

A continuación se mostrará el algoritmo de la transformada de Hough generalizada para la obtención del ángulo de rotación, basado en ángulos sexagesimales.

Entradas: Conjunto de minucias Query, y Conjunto de minucias Template.
Salida: Ángulo de rotación.
Detalles:
dimQ: Número de minucias del conjunto Query
dimT: Número de minucias del conjunto Template

```
Inicio
      Inicializar arreglo acumulador A
      Para i=0 hasta dimT, incrementar i en 1
            Transformar los ángulos del conjunto Template a sexagesimales
      Fin para

      Para j=0 hasta dimQ, incrementar j en 1
            Transformar los ángulos del conjunto Query a sexagesimales
      Fin para

      Para i=0 hasta dimT, incrementar i en 1
            Para j=0 hasta dimQ, incrementar j en 1
               1. Se obtiene la diferencia del ángulo de inclinación
                  local de la minucia i del conjunto Query y la minucia j
                  del conjunto Template.

               2. incrementar1D(A, diferencia)
            Fin para
      Fin para

      Buscar el máximo valor del arreglo acumulador, que indicará el
      ángulo de rotación.

      Retornar el ángulo de rotación obtenido.
Fin.
```

Figura 4.2: Pseudo-código algoritmo transformada de Hough 1D

4.1.1.4 Pseudo-código algoritmo transformada de Hough 2D

A continuación se mostrará el algoritmo de la transformada de Hough generalizada para la obtención de los valores que permitirán la traslación de las coordenadas de las minucias del conjunto Query.

Entradas: Conjunto de minucias Query, y Conjunto de minucias Template.
Salida: Valores Δx y Δy para la traslación.
Detalles:
dimQ: Número de minucias del conjunto Query
dimT: Número de minucias del conjunto Template

```
Inicio
      Inicializar arreglo acumulador A
      Para i=0 hasta dimT, incrementar i en 1
            Para j=0 hasta dimQ, incrementar j en 1
               Se obtiene la diferencia de las coordenadas de las
               minucias i y j de los conjuntos Query y Template,
               respectivamente.

               Si las minucias son del mismo tipo entonces
                        Incrementar2D(A, diferenciaX, diferenciaY)
            Fin para
      Fin para

      Buscar los máximos valores del arreglo acumulador, que indicarán
      los valores que permitirán la traslación.

      Retornar los valores obtenidos.
Fin.
```

Figura 4.3: Pseudo-código algoritmo transformada de Hough 2D.

4.1.1.5 Pseudo-código algoritmo incrementar1D basado en el nuevo criterio de resolución

A continuación se mostrará el algoritmo que incrementa el acumulador de evidencia en la posición del arreglo dado por la diferencia de los ángulos. Este algoritmo está basado en el nuevo criterio de resolución propuesto en la Sección **4.1.1.2**.

Entradas: Arreglo A, Entero x

```
Inicio
      Aumentar en dos el arreglo A en la posición x
      Aumentar en uno el arreglo A en la posición x-1
      Aumentar en uno el arreglo A en la posición x+1
Fin
```

Figura 4.4: Pseudo-código algoritmo incrementar1D.

4.1.1.6 Pseudo-código algoritmo incrementar2D basado en el nuevo criterio de resolución

A continuación se mostrará el algoritmo que incrementa el acumulador de evidencia en la posición del arreglo dado por la diferencia de las coordenadas. Este algoritmo está basado en el nuevo criterio de resolución propuesto en la Sección **4.1.1.2**.

Entradas: Arreglo A, Entero x, Entero y

```
Inicio
      Aumentar en dos el arreglo A en la posición x,y

      Aumentar en uno el arreglo A en la posición x-1,y-1
      Aumentar en uno el arreglo A en la posición x,y-1
      Aumentar en uno el arreglo A en la posición x+1,y-1
      Aumentar en uno el arreglo A en la posición x-1,y

      Aumentar en uno el arreglo A en la posición x+1,y
      Aumentar en uno el arreglo A en la posición x-1,y+1
      Aumentar en uno el arreglo A en la posición x,y+1
      Aumentar en uno el arreglo A en la posición x+1,y+1
Fin
```

Figura 4.5: Pseudo-código algoritmo incrementar2D.

4.1.2 Matching de minucias por coordenadas polares.

Si dos minucias están exactamente alineadas entre si, cada par de puntos correspondientes es completamente coincidente. En tal caso, un matching de minucias puede ser logrado simplemente contando el número de pares. Sin embargo, en la práctica, tal situación es difícil de encontrar. Por un lado, el error en determinar y localizar minucias impide que el algoritmo de alineación recupere la transformación relativa exactamente, mientras por otro lado, el esquema de alineación descrito más arriba no modela la deformación no lineal de huellas digitales que es una propiedad inherente de impresiones de la huella digital.

Con la existencia de una deformación no lineal, es imposible recuperar exactamente la posición de cada minucia del conjunto Query con respecto a sus minucias correspondientes en el Template. Por consiguiente, el algoritmo de matching de minucias necesita ser

elástico lo cual quiere decir que debería ser capaz de tolerar, hasta cierto punto, las deformaciones debido a la extracción inexacta de posiciones de minucias y deformaciones no lineales. Usualmente, un matching elástico puede ser logrado colocando un área de tolerancia alrededor de cada minucia del Template, lo cual especifica todas las posiciones posibles de las minucias correspondientes del conjunto Query con relación a las minucias del Template, y restringiendo las minucias correspondientes en la imagen de la Query a estar dentro de esta área **[Mill94]**. Este método no provee una satisfactoria actuación en la práctica, porque las deformaciones locales pueden ser pequeñas mientras las deformaciones globales acumuladas pueden ser muy grandes. Se ha implementado un algoritmo de matching elástico adaptable con la habilidad de compensar los errores de localización de minucias y deformaciones no lineales.

Para lo cual es necesario especificar que:

$$P = \left(\left(x_1^P, y_1^P, \theta_1^P \right)^T, \ldots, \left(x_M^P, y_M^P, \theta_M^P \right)^T \right) \tag{4-2}$$

denota el set de M minucias del Template y

$$Q = \left(\left(x_1^Q, y_1^Q, \theta_1^Q \right)^T, \ldots, \left(x_N^Q, y_N^Q, \theta_N^Q \right)^T \right) \tag{4-3}$$

denota el set de N minucias de la Query que es alineado con el Template anterior, con respecto a una minucia de referencia dada.

4.1.2.1 Convertir a coordenadas polares

Convertir cada punto de minucia al sistema de coordenadas polares con respecto a la minucia de referencia con la cual es realizada la alineación. Para llevar a cabo este proceso se debe aplicar la siguiente ecuación:

$$\begin{pmatrix} r_i \\ e_i \\ \theta_i \end{pmatrix} = \begin{pmatrix} \sqrt{\left(x_i^* - x^r\right)^2 + \left(y_i^* - y^r\right)^2} \\ \tan^{-1}\left(\dfrac{y_i^* - y^r}{x_i^* - x^r}\right) \\ \theta_i^* - \theta^r \end{pmatrix} \tag{4-4}$$

donde $(x_i^*, y_i^*, \theta_i^*)$ son las coordenadas de una minucia, $(x_r, y_r, \theta_r)^T$ es la coordenada de la minucia de referencia, y $(r_i, e_i, \theta_i)^T$ es la representación de las minucias en el sistema de coordenadas polares (r_i representa la distancia radial, e_i representa el ángulo radial, y θ_i representa la orientación de las minucia con respecto a la minucia de referencia).

4.1.2.2 Representación de las minucias

Representar las minucias de la Query y del Template en el sistema de coordenadas polares como string simbólicos concatenando cada minucia en el orden creciente de ángulos radiales:

$$Q_p = \left(\left(r_1^Q, e_1^Q, \theta_1^Q\right)^T, ..., \left(r_N^Q, e_N^Q, \theta_N^Q\right)^T\right) \tag{4-5}$$

$$P_p = \left(\left(r_1^Q, e_1^Q, \theta_1^Q\right)^T, ..., \left(r_M^Q, e_M^Q, \theta_M^Q\right)^T\right) \tag{4-6}$$

donde $(r_*^p, e_*^p, \theta_*^p)$ y $(r_*^Q, e_*^Q, \theta_*^Q)$ representan el radio correspondiente, el ángulo radial, y la orientación de minucias normalizada con respecto a las minucias de referencia, respectivamente.

4.1.2.3 Correspondencia de minucias

Corresponder las cadenas simbólicas resultantes P_p y Q_p con un algoritmo de programación dinámica para encontrar la distancia de edición entre P_p y Q_p que es descrito más abajo.

4.1.2.4 Distancia de edición

Usar la distancia de edición entre P_p y Q_p para establecer la correspondencia de las minucias entre P_p y Q_p. La puntuación del matching, M_{pq}, es entonces calculada según la siguiente fórmula:

$$M_{pq} = \frac{100 N_{pares}}{\max\{M, N\}} \tag{4-7}$$

donde N_{pares} es el número de minucias que caen dentro del límite del área de tolerancia de las minucias Template. Los valores máximos y mínimos de la puntuación del matching son 100 y 1, respectivamente. 100 indica un matching perfecto, mientras que 1 que no hubo ningún matching.

4.1.2.5 Ventajas

El matching de minucias en coordenadas polares tiene varias ventajas. Se ha observado que la deformación no lineal de huellas digitales tiene una propiedad radial. En otras palabras, aquella deformación en una impresión de huella digital usualmente inicia de un cierto punto (región) y se extiende radialmente hacia el exterior de forma no lineal. Por consiguiente, es beneficioso modelar en el espacio polar. Al mismo tiempo, es mucho más fácil formular la rotación, lo cual constituye la parte principal del error de alineación entre una imagen Query y una Template, en el espacio polar que en el espacio cartesiano. El string simbólico generado concatenando puntos en un orden creciente de ángulo radial en coordenadas polares excepcionalmente representa un patrón de puntos. Esto revela que el matching de minucias puede ser logrado con un algoritmo de matching de string.

Generalmente, el matching de string puede ser considerado como la maximización/minimización de una cierta función de costo como la distancia de edición. Intuitivamente, incluir un término elástico en la función de costos de un algoritmo de matching de string puede lograr una cierta cantidad de tolerancia de error.

4.1.2.6 Algoritmo de comparación representado mediante ecuaciones

Dado dos string P_p y Q_p de largo M y N, respectivamente, la distancia de edición, $C(M, N)$, en este algoritmo está recursivamente definido con las siguientes ecuaciones:

$$C(m,n) = \begin{cases} 0 & si\ m = 0,\ o\ n = 0 \\ \min\begin{Bmatrix} C(m-1,n)+\Omega \\ C(m,n-1)+\Omega \\ C(m-1,n-1)+w(m,n) \end{Bmatrix} & 0 < m \le M, \\ & y\ \ 0 < n \le N \end{cases} \tag{4-7}$$

$$w(m,n) = \begin{cases} \alpha \mid r_m^P - r_n^Q \mid + \beta\Delta e + \gamma\Delta\theta & si \mid r_m^P - r_n^Q \mid < \delta, \\ & \Delta e < \in y\ \Delta\theta < \varepsilon \\ \Omega & en\ otro\ caso \end{cases} \tag{4-8}$$

$$\Delta e = \begin{cases} a & si\ (a = (e_m^P - e_n^Q + 360) \bmod 360) < 180 \\ a - 180 & en\ otro\ caso \end{cases} \tag{4-9}$$

$$\Delta\theta = \begin{cases} a & si\ (a = (\theta_m^P - \theta_n^Q + 360) \bmod 360) < 180 \\ a - 180 & en\ otro\ caso \end{cases} \tag{4-10}$$

donde α, β, y γ son valores arbitrarios asociados con cada componente, respectivamente; δ, $\in$, y ε especifican el área de tolerancia; y Ω es un valor elevado pre-especificado para una asignar en caso de incompatibilidad. Tal distancia de edición, hasta cierto punto, captura la propiedad elástica del matching de string. Representa un costo de cambiar un polígono por el otro.

4.1.2.7 Pseudo-código del algoritmo de comparación

Entradas: Conjunto de minucias *Query*, y Conjunto de minucias *Template*.
Salida: Puntuación.
Detalles:
dimQ: Número de minucias del conjunto *Query*
dimT: Número de minucias del conjunto *Template*

```
Inicio
      Inicializar matriz de distancias M
   1. Rotar las minucias del conjunto Query utilizando el ángulo de
      rotación obtenido de la transformada de Hough 1D
   2. Trasladar las minucias del conjunto Query utilizando unos valores
      de traslación obtenidos de la transformada de Hough 2D
   3. Transformar los conjuntos de minucias Query y Template al sistema
      de coordenadas polares


   Para i=0 hasta dimT, incrementar i en 1
      Para j=0 hasta dimQ, incrementar j en 1
            Calcular diferencias del radio, theta, y alfa de la minucia
            polar i con respecto al radio, theta y alfa de la minucia
            polar j, respectivamente
            Si las diferencias están dentro de un cierto umbral entonces
                  M[i][j]=suma de las diferencias obtenidas
            Sino M[i][j]=un valor elevado especificado como Penalti
      Fin para
   Fin para

   Obtener la cantidad de distancias que se encuentran dentro del área de
   tolerancia establecido.

   Calcular la puntuación del matching
   Retornar la puntuación
Fin
```

Figura 4.6: Pseudo-código algoritmo de comparación.

4.2 Implementación del M-Tree

El M-Tree (árbol métrico) puede usarse para hacer operaciones de búsqueda en conjuntos de datos de espacios métricos de forma muy rápida. Pues, como se ha mencionado anteriormente, es un índice creado con las ventajas que poseen los métodos de acceso multidimensionales (SAMs) y algunos otros árboles métricos.

En este proyecto se utiliza una librería de M-Tree en Java creada por Clemens Maschner de la Universidad de München Alemania **[Mas02]**, basada en la librería de M-Tree en C++ diseñada por Patella, Ciaccia, y Zezula de la Universidad de Bologna Italia **[Pat98]**, y la librería GiST, también en Java, creada en la Universidad de Sudáfrica **[Gist07]**.

El diagrama de clases del paquete Mtree se presenta en la Figura 4.7.

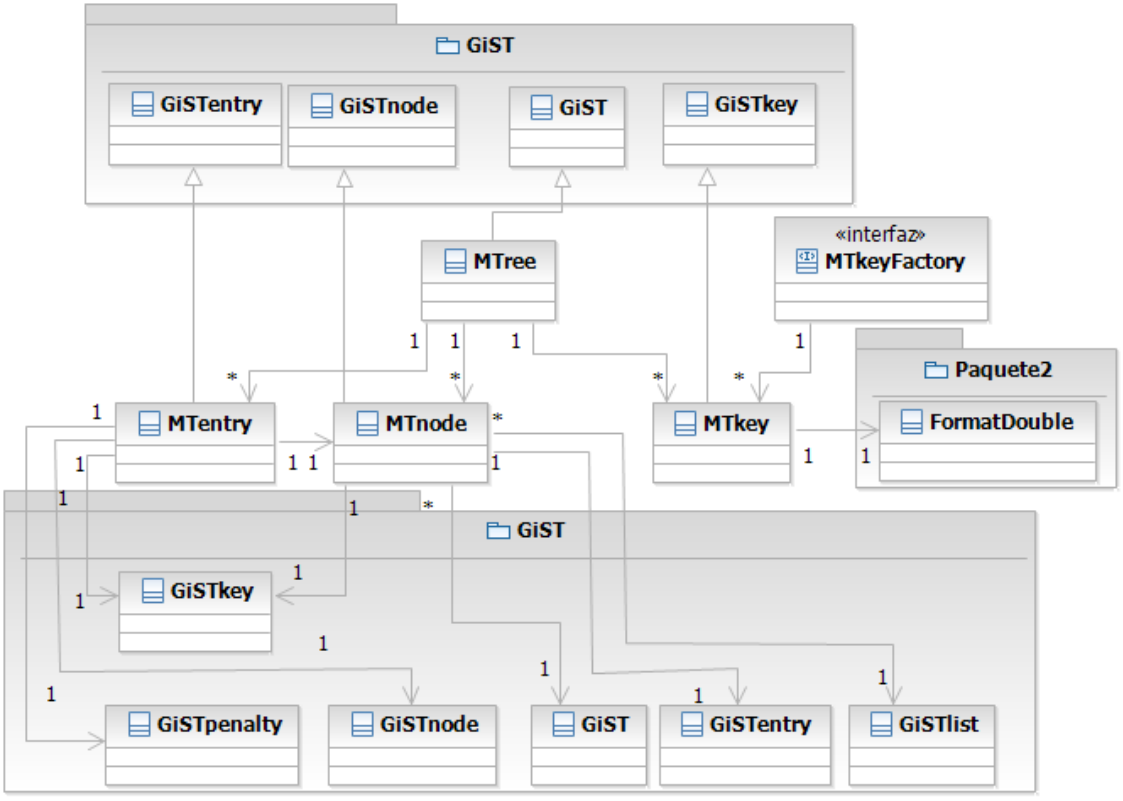


Figura 4.7: Diagrama de clases – Librería Mtree.

4.2.1 Librería GiST

El Árbol de Búsqueda Generalizado (*GiST: Generalizad Search Tree*) fue creado con la finalidad de proveer la misma funcionalidad de varios árboles de búsqueda existentes, en un mismo paquete **[Gist07]**. Siendo así una estructura de datos extensible, lo cual permite crear nuevos índices dinámicos sobre cualquier tipo de datos, soportando variados tipos de búsquedas sobre aquella información.

El diagrama de clases del paquete GiST se presenta en la Figura 4.8. Este diagrama fue creado realizando ingeniería a la inversa utilizando el código fuente de la librería.

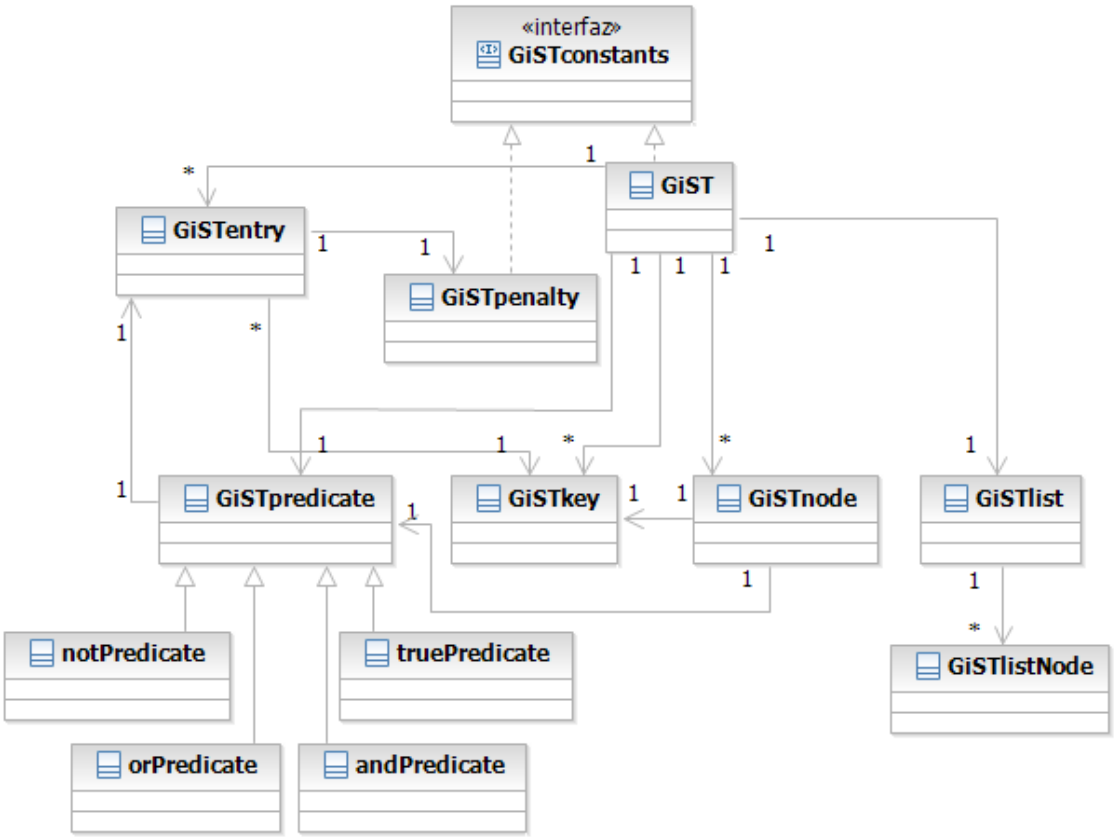


Figura 4.8: Diagrama de clases – Librería GiST.

4.2.2 Descripción de M-Tree.

Existen muchas estructuras de índices que soportan este tipo de búsqueda (búsqueda por rango) para espacios métricos multidimensionales aún más eficazmente. Sin embargo, el

M-Tree posee una particularidad muy especial, es que sólo se necesita conocer una función de distancia, que retorne la distancia entre dos objetos (ver Sección **2.3.1**).

El índice M-Tree también puede ser aplicado con otras estructuras de datos, tal como los string, siendo su función de distancia las distancia de edición, o la distancia de Levenshtein, que también cumple con la desigualdad triangular.

4.2.3 Diagrama de clases paquete ARLib

La Figura 4.9 muestra el diagrama de clases del paquete ARLib, en donde su clase principal *ConjuntoDeMinucias* representa a una huella digital que contiene un conjunto de minucias, siendo éstas cada una de las características representadas por: $\{x, y, \theta, t\}$ (ver Sección **2.1.2)**.

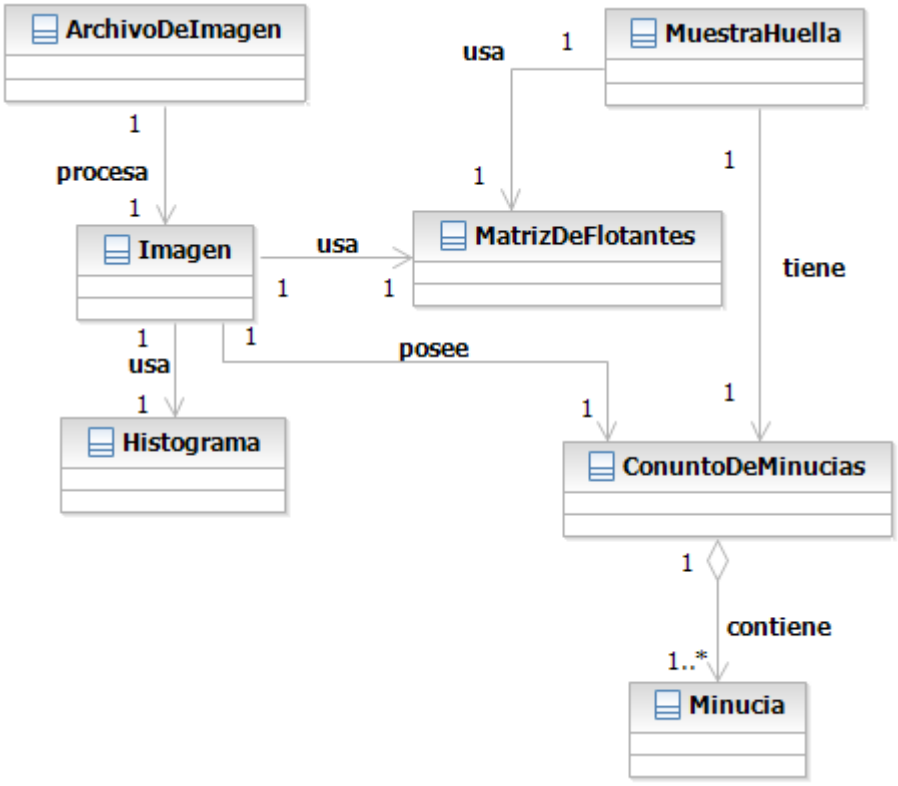


Figura 4.9: Diagrama de clases – Paquete ARLib.

4.2.4 Descripción de clases de la librería DLib

En la Figura 4.10 se muestra el diagrama de clases de la librería DLib, en donde la clase *Object2D* es quien representa al objeto de la clase persona, que posee los datos personales y la representación de la huella digital de una persona, dentro del índice métrico, y realiza el cálculo de distancia existente entre dos huellas a través del método *computeDistance()*.

En la Tabla 4.1 se muestra una breve descripción de cada clase.

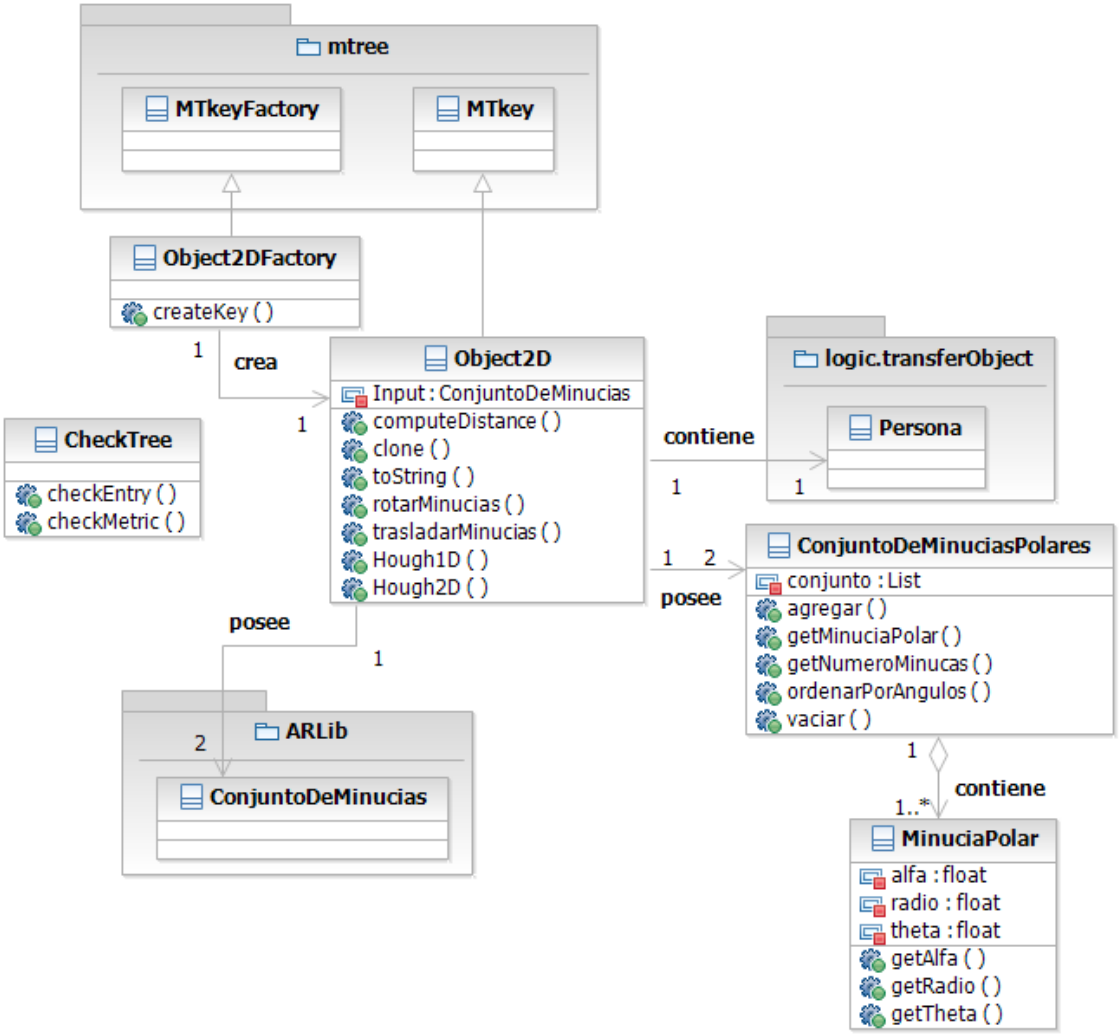


Figura 4.10: Diagrama de clases completo – Paquete DLib.

Nombre	Descripción
Object2DFactory	Esta clase hereda de *MTkeyFactory* de la librería Mtree, en donde implementa el método abstracto *createKey()* que es necesario para crear una instancia de la clase *Object2D* con un objeto de la clase Persona. Y también es necesaria esta clase para crear una instancia de M-Tree la cual es la encargada de especificar el tipo de objeto que se va a insertar dentro del índice.
Object2D	Esta clase es la principal dentro de la librería DLib, la cual hereda de *MTkey* de la librería M-Tree. Esta clase es la que implementa el método *computeDistance()* el que obtiene la distancia entre dos objetos (huellas) utilizado directamente por el método rangeSearch() de la clase *MTnode*. El método *computeDistance()* es implementado a través del matching de huellas digitales descrito en la Sección **4.1**.
CheckTree	Esta clase implementa el método *checkMetric()*, el cual es necesario para verificar si el árbol se cumple con todas las restricciones descritas en la Sección **2.5**.
MinuciaPolar	Es la clase que contiene los atributos necesarios para convertir una minucia en minucia polar, como el radio, el theta, y el alfa.
ConjuntoDeminuciasPolares	Esta es la clase que representa a una huella digital transformada a minucias polares, posee un vector que contiene los objetos de la clase *MinuciaPolar*, y el método ordenar vector por ángulo alfa, necesario para la comparación de minucias utilizando coordenadas polares.

Tabla 4.1: Descripción de las clases de la librería DLib.

4.2.5 Pasos a seguir para su implementación

4.2.5.1 Creación del índice

El primer paso es crear una instancia de M-Tree, es decir, inicializar el índice métrico en donde serán insertados los objetos, a través de la siguiente línea de código:

Mtree tree = new Mtree(x, y, new Objetct2DFactory);

donde *x*, *y* representan la mínima y máxima cantidad de nodos por sub-árbol, respectivamente. *new Object2DFactory* se envía como parámetro real para especificarle al índice qué tipos de objetos serán insertados en él.

4.2.5.2 Inserción en el índice

Luego es necesario insertar los objetos en el índice anteriormente creado, este paso se realiza simplemente con la siguiente línea de código:

tree.insert(persona);

donde *persona*, en este caso, es un objeto que contiene los datos personales de un individuo, junto con la información de las minucias de una de sus huellas digitales. Para la inserción de varios objetos a la vez, sólo basta iterar sobre la anterior línea de código la cantidad de veces necesaria según los objetos a insertar.

4.2.5.3 Chequeo del estado del índice

Luego de cada inserción, o un conjunto de inserciones es necesario chequear si el índice se encuentra en perfectas condiciones, es decir, si cumple con las especificaciones descritas para ser un árbol métrico M-Tree, esencialmente verifica si las distancias entre nodos padres e hijos están calculados de forma correcta. Esto se realiza con la siguiente línea de código:

CheckTree.checkMetrics((MTnode) tree.getRoot());

4.2.5.4 Búsqueda por rango

Luego de realizar los pasos antes citados, es posible realizar búsquedas por rango utilizando el índice creado. Este método retorna una lista (*gistList*) de todos los objetos similares, que estén dentro del radio cobertor indicado. Para este proyecto se consideran entre 10 y 12 minucias o más, es decir retornaría todas aquellas huellas que tienen 10 o más minucias apareadas o cercanas, cada minucia dentro de un área de tolerancia, a las minucias de la huella de entrada. Para realizar la búsqueda por rango son necesarias las siguientes líneas de código:

```
Object2D object2d = new Object2D(persona);
GiSTlist gistlist = tree.rangeSearch(object2d, coveringRadius);
Vector resultado = new Vector();
    for (int i = 0; i < gistlist.getNumEntries(); i++) {
        MTentry mtentry = (MTentry) searchResults.getEntry(i);
        Persona persona = (Persona) ((Object2D) mtentry.getKey()). getObject();
        resultado.add(persona);
    }
```

Primero es necesario crear una instancia de la clase *Object2D* con el objeto que contiene la huella digital del individuo a buscar, luego se invoca el método *rangeSearch*() de la clase Mtree con el objeto object2d creado y el radio cobertor 1/10[8], este método retorna un objeto de la clase GistList que contiene todas las personas que tienen sus huellas cercanas a la de entrada.

[8] 1/10, el radio cobertor se denota de esta forma, puesto que la búsqueda es realizada obteniendo los objetos más cercanos, y para este ejercicio las huellas más cercanas son las que tienen más minucias apareadas, por lo tanto, la cantidad mínima de minucias va en el denominador.

4.2.6 Pruebas del sistema

Se ha evaluado el funcionamiento del sistema completo, mediante pruebas de identificación y pruebas de verificación. Las huellas han sido obtenidas del lector de huellas digitales NITGEN modelo OPU01M, que genera imágenes con las siguientes características.

- Tipo de archivo: Mapa de bits (BMP)
- Tamaño: 248×292 [píxeles] (tamaño de la imagen obtenida con el lector de huellas digitales)
- Profundidad de color: 8 bpp

Los patrones empleados en la comparación se han obtenido previamente mediante el proceso de extracción de minucias descrito en **[BP05]**, que incluye varias etapas de proceso para la mejora de imagen. Este proceso aún cuenta con varias deficiencias, pues también es un proyecto pionero en Chile en el estudio de la biometría de huellas digitales. Por lo tanto influye directamente en los resultados arrojados por estas pruebas, puesto que el método de matching de minucias descrito en este proyecto da por supuesto un buen proceso de extracción de minucias.

La comparación se realiza previa alineación de los dos patrones que se van a procesar, y su conversión en dos cadenas de puntos ordenados según sus coordenadas polares. El resultado de la comparación vendrá dado por el valor que toma la función de costo después de comparar todas las parejas de minucias de las dos cadenas.

Como se mencionó anteriormente el proceso de extracción de las minucias de las huellas digitales de **[BP05]** no es muy efectivo, porque no realiza a cabalidad una parte muy importante dentro del proceso, que es el de eliminación de minucias espurias o falsas minucias que se producen también porque la calidad del algoritmo de mejoramiento de la imagen no es de muy buena calidad, esto afecta directamente al proceso de comparación.

Por lo tanto para las pruebas de verificación e identificación fue necesario utilizar imágenes de huellas rotadas manualmente a la izquierda y a la derecha, como se muestra en la Figura 4.11.

(a) (b) (c)

Figura 4.11: Huellas digitales rotadas por un editor de imágenes.

4.2.6.1 Pruebas de verificación

En verificación, una persona accede al sistema identificándose, siendo el sistema quien decide si acepta o rechaza a dicha persona. Para esta prueba se ha utilizado una base de datos con 15 huellas digitales de diferentes individuos, y las comparaciones se han realizado con las 15 imágenes rotadas a la izquierda o derecha (vea Figura 4.5).

En la Tabla 4.2 se muestran los resultados de las pruebas realizadas, indicando en la columna ***normal*** el porcentaje de aceptación o rechazo al comparar la misma imagen de la huela digital almacenada en la base de datos, en la columna ***rotada*** indica el porcentaje de aceptación o rechazo al comparar la misma imagen de la huela digital almacenada en la base de datos pero rotadas de forma aleatoria, y en la columna ***impostor*** se indica el porcentaje de aceptación o rechazo al comparar huellas digitales de distintas personas.

	Rut	Nombre	Apellido	Normal	Rotada	Impostor
1	16221894	Diego	Calderón	Aceptado 76%	Aceptado 72%	Rechazado 6%
2	16447222	Cristina	Lillo	Aceptado 72%	Aceptado 45%	Rechazado 0%
3	15996285	Roberto	Calderón	Aceptado 66%	Aceptado 60%	Rechazado 4%
4	14430395	Eva	Adelba	Aceptado 66%	Aceptado 65%	Rechazado 12%
5	198566661	Ángel	Torres	Aceptado 75%	Rechazado 32%	Rechazado 8%
6	4085884	Fernando	Jiménez	Aceptado 69%	Aceptado 60%	Rechazado 22%
7	10317817	José	Serrano	Aceptado 72%	Aceptado 67%	Rechazado 6%
8	6386114	Francisco	Iglesias	Aceptado 72%	Aceptado 42%	Rechazado 4%
9	18963262	Jaime	Cóndel	Aceptado 78%	Rechazado 40%	Rechazado 18%
10	7723468	Ana	Ruiz	Aceptado 80%	Aceptado 72%	Rechazado 10%
11	18156210	Jaime	Rodríguez	Aceptado 79%	Rechazado 38%	Aceptado 45%
12	13159508	Antonio	Gutiérrez	Aceptado 70%	Aceptado 66%	Rechazado 12%
13	18442539	Nadia	Ortega	Aceptado 67%	Aceptado 72%	Rechazado 15%
14	4196846	Pedro	Domínguez	Aceptado 65%	Aceptado 42%	Rechazado 2%
15	11424132	Eduardo	Martínez	Aceptado 72%	Rechazado 28%	Rechazado 1%

Tabla 4.2: Resultados obtenidos de las pruebas de verificación

Con los resultados obtenidos es posible determinar la tasa de falso rechazo (que es calculada a partir de la cantidad de personas rechazadas siendo las verdaderas) y la tasa falsa aceptación (que es calculada a partir de la cantidad de personas aceptadas siendo impostoras) (ver Sección **2.2.4.1**).

- **Tasa de falso rechazo:** 0,1
- **Tasa de falsa aceptación:** $0,0\overline{6}$

4.2.6.2 Pruebas de identificación

En identificación, un usuario accede al sistema aportando su huella digital, pero sin identificarse. El sistema, después de obtener su patrón biométrico, genera una lista con las identificaciones de los individuos que cuentan con huellas digitales similares a la huella de entrada, ésta búsqueda es realizada mediante un índice métrico M-Tree, que para las pruebas se han creado 3 de ellos que contienen 100, 300 y 600 huellas digitales respectivamente, dentro de su estructura, para las cuales fue necesario generar huellas aleatorias a partir de 30 existentes en la base de datos.

Para realizar un proceso de identificación, como se indica más arriba, sólo se debe aportar una huella digital al sistema, pero para esta prueba a modo de demostración se ha

considerado además el radio cobertor (ver Sección **2.5.3.3**) para la búsqueda dentro del índice M-Tree.

La Tabla 4.3 muestra los resultados de las pruebas realizadas, donde *BD* es el número de objetos almacenados, *q* la cantidad de objetos devueltos, y *t* el tiempo de respuesta.

Estos resultados se ven reflejados en los gráficos de las Figuras 4.12, 4.13 y 4.14.

	Radio cobertor						
BD	**30**		**50**		**70**		**Tiempo de inserción**
	q	t	q	t	q	t	
50	21	2484	4	1291	2	887	12:25
100	12	3860	5	1580	2	1157	1:54:32
300	25	4297	7	1950	4	1591	2:24:33
600	28	4657	15	2169	12	1884	5:45:34

Tabla 4.3: Resultados obtenidos en las pruebas de identificación

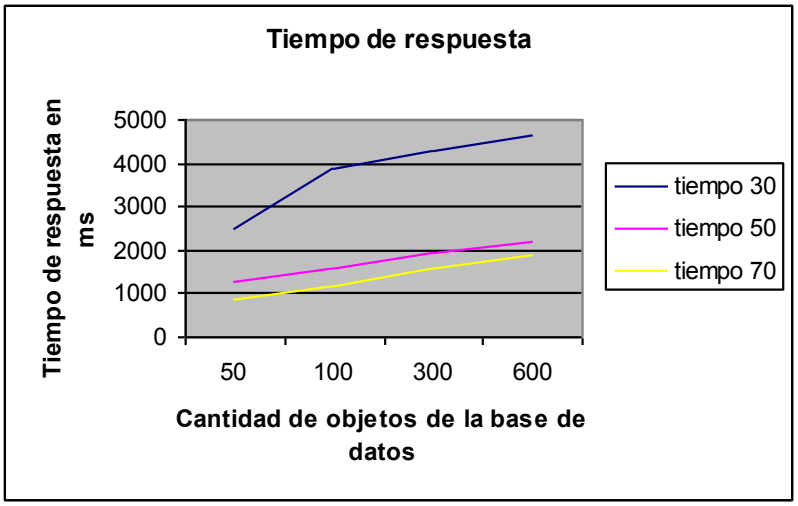


Figura 4.12: Tiempos de respuesta representados en ms

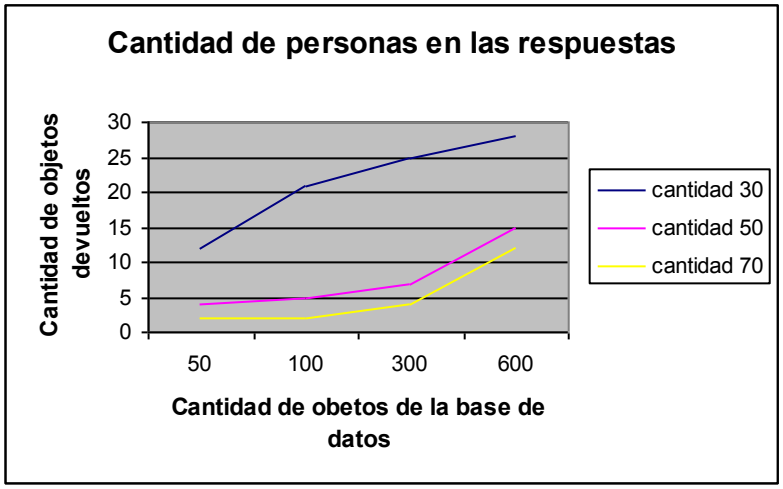


Figura 4.13: Cantidad de personas devueltas

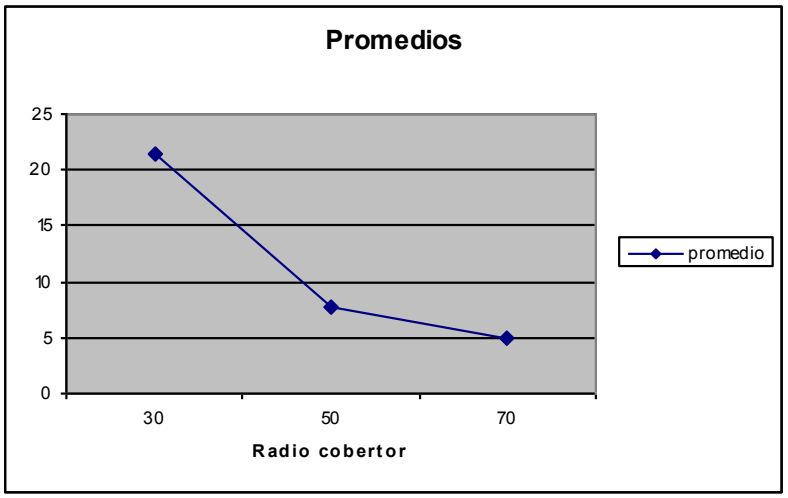


Figura 4.14: Promedio de personas devueltas

4.3 Limitaciones del sistema

- La eliminación de objetos no está implementada. Esta característica no está especificada en el M-Tree original, aunque es necesaria en algunos índices, para este proyecto no representa una necesidad, pues para el uso que se le da a la identificación de huellas digitales siempre es necesario mantener almacenadas todas las huellas, aunque el individuo fallezca.

- La búsqueda del vecino más cercano no está implementada. Este método cuenta con una gran dificultad en su implementación, pero también como la característica anterior, ésta tampoco representa una necesidad en este proyecto, pues como se ha mencionado anteriormente, la búsqueda de los vecinos más cercanos obtiene los k objetos más cercanos a uno en específico, siendo estos vecinos en algunas circunstancias muy lejanos a la huella de búsqueda.

- El árbol no opera en memoria secundaria. Sólo puede ser hecho persistente con el mecanismo Java Persistance. Esta implementación de M-Tree en Java no implementa el almacenamiento en disco, a pesar de que esta característica está especificada en el M-Tree original, a través de archivos dinámicos, similar a otros índices métricos. Se estudió otra implementación de M-Tree en Java, la denominada XXL (Librería flexible y extensible), pero su utilización es bastante más complicada que la actual por su complejo diseño, ya que aquella librería implementa diversas estructuras de datos, sería interesante utilizar XXL para dar solución a éste problema, aunque su diseño está más enfocado a su elegancia por sobre la velocidad de inserción y búsqueda.

- A pesar de que en este proyecto se implementó un sistema que realiza búsquedas de huellas digitales de forma muy rápida, el tiempo de inserción de las huellas en el índice es demasiado costoso, por su compleja función de distancia, esto se podría mejorar implementando el sistema utilizando el lenguaje de programación C++, con las librerías originales de M-Tree y GiST.

Conclusiones

El propósito de esta tesis fue conseguir un algoritmo de identificación de huellas digitales que entregara buenos resultados en tiempos razonables. Ahora se presentan, de forma general, las conclusiones en cuanto a lo que se ha logrado y a lo que aún queda por hacer.

Como se ha podido apreciar en el proyecto, los resultados propuestos se pueden aplicar sin ningún problema en imágenes de huellas digitales de alta calidad, pues se tendrían que mejorar o tomar en consideración nuevas vías de trabajo como mejorar el filtrado de las minucias en el post-procesado para mejorar los resultados obtenidos del estudio en imágenes de menor calidad.

El sistema propuesto es apto para realizar una verificación automática de identidad utilizando huellas digitales (suponiendo un buen algoritmo de extracción de minucias), sin embargo para la identificación de huellas digitales hasta el momento se ha propuesto una mejora en el tiempo de respuesta, pues los resultados arrojados por las pruebas indican que todavía es necesaria la participación de un experto para autentificar los resultados obtenidos, pues para obtener un resultado óptimo es necesario realizar una búsqueda exhaustiva.

Como se mencionó anteriormente, el algoritmo de inserción aún es muy costoso, es posible mejorar aquel problema implementando una mejora a la hora de realizar las divisiones (split) del índice, de manera que no se realicen demasiados cálculos de distancia, en **[CPZ97]** se propone una matriz de distancia que minimiza estos cálculos, en la librería de M-Tree utilizada en este proyecto no se utiliza dicha matriz.

Este trabajo representa una iniciativa para la utilización de nuevas técnicas de indexación sobre estructuras de datos, o espacios métricos, como es en este caso. A pesar de la dificultad que se presenta a la hora de la recopilación de información necesaria, ésta área de

estudio está en constante investigación, por lo tanto, cualquier propuesta es totalmente válida, y sirve como base para futuros estudios o mejoras a lo propuesto inicialmente.

Para concluir se indica que este proyecto representa los primeros pasos de investigación en identificación de huellas digitales, utilizando búsquedas por proximidad sobre espacios métricos, por lo tanto todo lo descrito anteriormente se enfoca a una nueva propuesta para solucionar el problema antes descrito.

Referencias bibliográficas

[Baez05] Baez, L. 2005. Extracción de características de Galton de Huellas Dactilares por procesamiento digital de la imagen.

[Ben75] J. Bentley. Multidimensional binary search trees used for associative searching. Comm. of the ACM, 18(9):509–517, 1975.

[BFPR06] N. Brisaboa, A. Fariña, O. Pedreira, N. Reyes. 2006. Indexación dinámica para la recuperación de información basada en búsqueda por similitud.

[BK73] W. Burkhard and R. Keller. Some approaches to best-match file searching. Comm. of the ACM, 16(4):230–236, 1973.

[BKK96] S. Berchtold, D. Keim, and H. Kriegel. The X-tree: an index structure for high-dimensional data. In Proc. 22nd Conference on Very Large Databases (VLDB'96), pages 28–39, 1996.

[BO97] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In Proc. ACM Conference on Management of Data (ACM SIGMOD'97), pages 357–368, 1997. Sigmod Record 26(2).

[BP05] Briones, R. y Paredes, A. 2005. "Análisis de Identidad Mediante el Estudio de Huellas Digitales Preprocesadas, Memoria de Ingeniero de Ejecución en Computación e Informática. Chillán, Universidad del Bío-Bío, Facultad de Ciencias Empresariales. 95p.

[Bri95] S. Brin. Near neighbor search in large metric spaces. In Proc. 21st Conference on Very Large Databases (VLDB'95), pages 574–584, 1995.

[BY97] R. Baeza-Yates. Searching: an algorithmic tour. In A. Kent and J. Williams, editors, Encyclopedia of Computer Science and Technology, volume 37, pages 331–359. Marcel Dekker Inc., 1997.

[BYCMW94] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In Proc. 5th Combinatorial Pattern Matching (CPM'94), LNCS 807, pages 198–212, 1994.

[BYN98] R. Baeza-Yates and G. Navarro. Fast approximate string matching in a dictionary. In Proc. 5th South American Symposium on String Processing and Information Retrieval (SPIRE'98), pages 14–22. IEEE CS Press, 1998.

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. 1999. Modern Information Retrieval. Addison-Wesley.

[CBYN05] V. Cuquejo, R. Baeza-Yates, G. Navarro. 2005. Algoritmos y estructuras de datos para búsqueda de objetos similares.

[Cha94] B. Chazelle. Computational geometry: a retrospective. In Proc. of the 26th ACM Symposium on the Theory of Computing (STOC'94), pages 75–94, 1994.

[Chi94] T. Chiueh. Content-based image indexing. In Proc. of the 20th Conference on Very Large Databases (VLDB'94), pages 582–593, 1994.

[CMBY99] E. Chávez, J. Marroquín, and R. Baeza-Yates. Spaghettis: an array based algorithm for similarity queries in metric spaces. In Proc. 6th International Symposium on String Processing and Information Retrieval (SPIRE'99), pages 38–46. IEEE CS Press, 1999.

[CMN01] E. Chávez, J. Marroquín, and G. Navarro. Fixed queries array: A fast and economical data structure for proximity searching. Multimedia Tools and Applications, 14(2):113–135, 2001.

[CN00a] E. Chávez and G. Navarro. An effective clustering algorithm to index high dimensional metric spaces. In Proceedings of the 7th International Symposium on String Processing and Information Retrieval (SPIRE'2000), pages 75–86. IEEE CS Press, 2000.

[CNBYM01] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. ACM Computing Surveys, 33(3):273–321, September 2001.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: an efficient access method for similarity search in metric spaces. In Proc. of the 23rd Conference on Very Large Databases (VLDB'97), Athens, Greece. pages 426–435, 1997.

[Cra99] Larman Craig. 1999. "UML y Patrones", primera edición, México, Pretince may Hispanoamericana, 507p.

[DN87] F. Dehne and H. Noltemeier. Voronoi trees and clustering problems. Information Systems, 12(2):171–175, 1987.

[DR99] D. Morales L. Javier Ruiz-del-Solar, 1999. "Sistemas biométricos: Matching de huellas dactilares mediante Transformada de Hough Generalizada".

[Ele73] M. Eleccion, 1973. "Automatic Fingerprint Identification", IEEE Spectrum, vol. 10, 36-45.

[FEF+94] C. Faloutsos, W. Equitz, M. Flickner, W. Niblack, D. Petkovic, and R. Barber. July 1994. Efficient and effective querying by image content. J. of Intell. Inf. Sys., 3(3/4):231–262.

[FL95] C. Faloutsos and K.-I. Lin. Fastmap: a fast algorithm for indexing of traditional and multimedia databases. In SIGMOD Conf. pages 163-174. ACM, 1995.

[Gab46] Gabor, D. 1946. "Theory of Communication", en Journal of Institute for Electrical Engineering, vol. 93, part III, Nº 26. p. 429-457

[Gal07] Galvis, C. 2007. "Introducción a la Biometría". [en línea] <http://www.monografias.com/trabajos43/biometria/biometria.shtml> [consulta: 30 de Agosto de 2007].

[Gist07] The GiST Indexing Project [en línea] <http://gist.cs.berkeley.edu/> [consulta: 21 de Octubre de 2007].

[Gut84] A. Guttman. June 1984. R-trees: A dynamic index structure for spatial searching. ACM SIGMOD International Conference on Management of Data, pages. 47-57, Boston, MA.

[GW93] Gonzalez, R. y Woods, R., 1993, Digital Image Processing, Addison Wesley.

[HHD07] "La Historia de las Huellas Digitales. 2007" [en línea] <http://babelfish.altavista.com/babelfish/trurl_pagecontent?lp=en_es&trurl=http%3a%2f%2fonin.com%2ffp%2ffphistory.html> [consulta: 29 de Septiembre de 2007].

[HJ98] L. Hong and A. Jain, 1998. "Integrating Faces and Fingerprints for Personal Identification", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 12, pp. 1295-1307.

[Hong98] Hong, L., et al, 1998. Fingerprint Image Enhancement: Algorithm and Performance.

[IAFIS07] FBI. Integrated Automated Fingerprint Identification System. [en línea] <http://www.fbi.gov/hq/cjisd/iafis.htm> [consulta: 15 de Octubre de 2007].

[IG07] Iglesias, R. y Gómez, I. 2007. "Autenticación basada en huella digital".

[JR97] A. Jain and R. Bolle. "On-Line Fingerprint Verification", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 4, 302-313, 1997.

[KM83] I. Kalantari and G. McDonald. A data structure and an algorithm for the nearest point problem. IEEE Transactions on Software Engineering, 9(5), 1983.

[Lan08] M. Lantadilla. 2008. Algoritmos Biométricos. Red en Acción. (16): 13-15

[Lar03] "Procedimientos para comparar huellas digitales" Larcher, P. "et al" 2003. España patente ES 2 161 800 T3 (0 680 044).

[Lop04] López A. et al., 2004. Fingerprint Recognition.

[Lor04] Lorda, M. 2004. "Análisis del algoritmo de Maio para la extracción de huellas dactilares". Ingeniería Técnica Industrial Especialidad Electrónica Industrial. Tarragona, Universitat Rovira i Vergili. 129p.

[Mal03] Maltoni, D. et al., 2003. Handbook of Fingerprint Recognition. Bolonia, Italia. 353p.

[Mas02] Maschner, C. 2002. "Mtree Tester Applet". [en línea] <http://www.cmarschner.net/mtree.html> [consulta: 20 de Octubre de 2007].

[Mill94] B. Miller, 1994. "Vital Signs of Identity", IEEE Spectrum, vol. 31, no. 2, 22-30.

[MJ03] Mainguet, Jean-Franc_ois, R. 2003. "Sistema y procedimiento de lectura de huellas digitales". España patente ES 2 191 816 T3 (0 813 164).

[MOC96] L. Micó, J. Oncina, and R. Carrasco. 1996. A fast branch and bound nearest neighbour classifier in metric spaces. Pattern Recognition Letters.

[MOV94] L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (AESA) with linear preprocessing-time and memory requirements. Pattern Recognition Letters, 15:9–17, 1994.

[Nav01] G. Navarro. A guided tour to approximate string matching. ACM Computing Surveys, 33(1):31–88, 2001.

[Nav99] G. Navarro. Searching in metric spaces by spatial approximation. In Proc. String Processing and Information Retrieval (SPIRE'99), pages 141–148. IEEE CS Press, 1999.

[NN97] S. Nene and S. Nayar. A simple algorithm for nearest neighbor search in high dimensions. IEEE Trans. on Pattern Analysis and Machine Intelligence, 19(9):989–1003, 1997.

[Nol89] H. Noltemeier. Voronoi trees and applications. In Proc. International Workshop on Discrete Algorithms and Complexity, pages 69–74, Fukuoka Recent Hotel, Fukuoka, Japan, 1989.

[NVZ92] H. Noltemeier, K. Verbarg, and C. Zirkelbach. Monotonous Bisector

[Pat98] Patella, M. 1998. "The M-tree Project". [en línea] <http://www-db.deis.unibo.it/Mtree/> [consulta: 10 de Septiembre].

[RCJ95] Nalini K. Ratha, Shaoyun Chen and Anil K. Jain, 1995. "Adaptive flow orientation based feature extraction in fingerprint images". Michigan State University. 32p

[RJ92] A. Rao y R. Jain, 1992. "Computerized Flow Field Analysis: Oriented Texture Fields".

[RKCJ96] N. Ratha, K. Karu, S. Chen and A. Jain, 1996. "A Real-Time Matching System for Large Fingerprint Databases", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 18, no. 8, pp. 799-813.

[Rod07] Rodríguez F. 2007. Dactiloscopía Computerizada. OCCAM'S RAZOR 1(2):26-29

[Sam84] H. Samet. The quadtree and related hierarchical data structures. ACM Computing Surveys, 16(2):187–260, 1984.

[Sha77] M. Shapiro. The choice of reference points in best-match file searching. Comm. of the ACM, 20(5):339–343, 1977.

[Tha03] Thai, R., 2003. "Fingerprint image enhancement and minutiae extraction". University of Western Australia.

[Uhl91a] J. Uhlmann. Implementing metric trees to satisfy general proximity/similarity queries. Manuscript, 1991.

[Uhl91b] J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. Information Processing Letters, 40:175–179, 1991.

[Vid86] E. Vidal. An algorithm for finding nearest neighbors in (approximately) constant average time. Pattern Recognition Letters, 4:145–157, 1986.

[Yia00] P. Yianilos. Locally lifting the curse of dimensionality for nearest neighbor search. In Proc. 11th ACM-SIAM Symposium on Discrete Algorithms (SODA'00), 2000. To appear.

[Yia93] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93), pages 311– 321, 1993.

[Yia99] P. Yianilos. Excluded middle vantage point forests for nearest neighbor search. In DIMACS Implementation Challenge, ALENEX'99, Baltimore, MD, 1999.

[ZCR96] P. Zezula, P. Ciaccia, and F. Rabitti, 1996. "Mtree: A dynamic index for similarity queries in multimedia databases". TR 7, HERMES ESPRIT LTR Project, Available at URL http://www.ced.tuc.gr/hermes/.