UNIVERSIDAD DEL BÍO-BÍO
FACULTAD DE CIENCIAS EMPRESARIALES
MAGÍSTER EN CIENCIAS DE LA COMPUTACIÓN

Algoritmos para calcular la Distancia de Hausdorff sobre conjuntos de puntos representados mediante k^2-tree

Por
Fernando Javier Dominguez Pacheco

Tesis para optar al grado de Magíster
en Ciencias de la Computación

Dirigido por:
Dr. Gilberto Gutiérrez
Universidad del Bío-Bío, Chillán, Chile

Co-Dirigido por:
Dr. Miguel Romero
Universidad del Bío-Bío, Chillán, Chile

2018 - II

Resumen

La Distancia de Hausdorff entre dos conjuntos de puntos A y B corresponde a la mayor de las distancias entre cada objeto $x \in A$ y su vecino más cercano en B. La Distancia de Hausdorff tiene múltiples aplicaciones como por ejemplo la comparación de imágenes médicas o la comparación entre dos rutas de transporte. Existen múltiples algoritmos para calcular la Distancia de Hausdorff, algunos operan con los conjuntos en memoria principal y otros en memoria secundaria. Por otro lado, para afrontar el desafío de indexar grandes conjuntos de puntos en memoria principal, existen estructuras de datos compactas como el k^2-tree la cual, minimizando el almacenamiento, permite ser consultada eficientemente. En este artículo se presentan dos nuevos algoritmos (HDKP1 y HDKP2) para calcular la Distancia de Hausdorff en la estructura compacta k^2-tree, logrando una solución eficiente tanto en tiempo como espacio. Mediante una serie de experimentos con datos reales y sintéticos se evaluó el rendimiento de nuestros algoritmos con otro propuesto en la literatura bajo condiciones similares. Los resultados permiten concluir que HDKP2 es más eficiente en tiempo de ejecución que dicho algoritmo en alrededor de entre uno y tres órdenes de magnitud, esto dependiendo de la distribución de los puntos.

Índice

Parte I

Motivación

Capítulo 1

Introducción

El cálculo de la Distancia de Hausdorff es un problema antiguo, tratado especialmente en el ámbito de la topología [19]. En Ciencias de la Computación ha sido tratado a través de los últimos treinta años. En efecto, el primer trabajo encontrado data del año 1983 donde los autores [2] proponen un algoritmo para el cálculo de Distancia de Hausdorff entre polígonos convexos.

La Distancia de Hausdorff es principalmente una medida de similitud entre dos conjuntos de puntos A y B y, por lo tanto, puede ser utilizada para comparar formas geométricas tales como polilíneas, polígonos convexos [2] o poliedros generales representados como mallas triangulares [3]. Esto tiene muchas aplicaciones prácticas debido a que es posible modelar varias situaciones del mundo real como un conjunto de puntos o bien como figuras geométricas. Por ejemplo, esta medida, es usada en el área de la topología para la comparación de planos [19]. También es utilizada para comparar imágenes, por ejemplo es utilizada en el área de la dermatología para identificar la similitud entre imágenes de lesiones cutáneas de un paciente y otro [32]; o utilizada para conocer la evolución del tamaño de un tumor cerebral a través del tiempo [32]; y para el reconocimiento facial [12, 16, 34].

Formalmente, $\mathsf{HausDist}(A, B)$ es una función donde A y $B \subseteq \mathbb{R}^d$, definida como [33]:

$$\mathsf{HausDist}(A, B) = \max_{x \in A}\{\min_{y \in B}\{\mathsf{Dist}(x, y)\}\}$$

es decir, es la mayor de las distancias entre cada punto $x \in A$ con su vecino más cercano $y \in B$. $\mathsf{HausDist}(A, B)$ también es llamada Distancia directa de Hausdorff. Notar que la Distancia de Hausdorff no es simétrica ($\mathsf{HausDist}(A, B) \neq \mathsf{HausDist}(B, A)$). La Distancia simétrica de Hausdorff se define como [26]:

$$\mathsf{SymHD}(A, B) = \max\{\mathsf{HausDist}(A, B), \mathsf{HausDist}(B, A)\}$$

En Figura 1.1 se presentan dos trayectorias[1] A y B, en este caso la Distancia de Hausdorff se puede considerar como la mayor discrepancia de una trayectoria con otra, donde la noción de discrepancia entre trayectorias es una medida de similitud [36]. De este modo la mayor discre-

[1] Una trayectoria es la secuencia de puntos por los cuales a pasado un objeto móvil. La secuencia de puntos se encuentra ordenada por el tiempo

pancia de A con respecto a B es $\mathsf{HausDist}(A, B) = \mathsf{Dist}(a_4, b_4)$ y la mayor discrepancia de B con respecto a A es $\mathsf{HausDist}(B, A) = \mathsf{Dist}(b_5, a_4)$. Una aplicación práctica de lo anterior, sería por ejemplo, identificar cual es la máxima distancia adicional que tendría que recorrer un pasajero si la autoridad de transporte decidiera reemplazar una ruta de bus (secuencia de paradas) por otra [26], de este modo, la autoridad de transporte podría evaluar diferentes rutas alternativas con la finalidad de escoger la que genere el menor impacto en la población. Hacemos notar que si bien es cierto la Distancia de Hausdorff puede ser calculada en conjuntos estáticos de datos, también puede ser calculada en conjuntos subyacentes filtrados mediante operaciones de conjunto para k^2-tree [27]. Siguiendo el ejemplo anterior, se podría calcular la Distancia de Hausdorff para obtener la distancia adicional que tendría que recorrer un pasajero con respecto a un tramo específico de una ruta de transporte. Además, utilizando operaciones de conjunto, se podrían comparar zonas específicas de imágenes. Notar que cualquier cambio en los conjuntos a comparar, significa tener que calcular nuevamente la Distancia de Hausdorff.

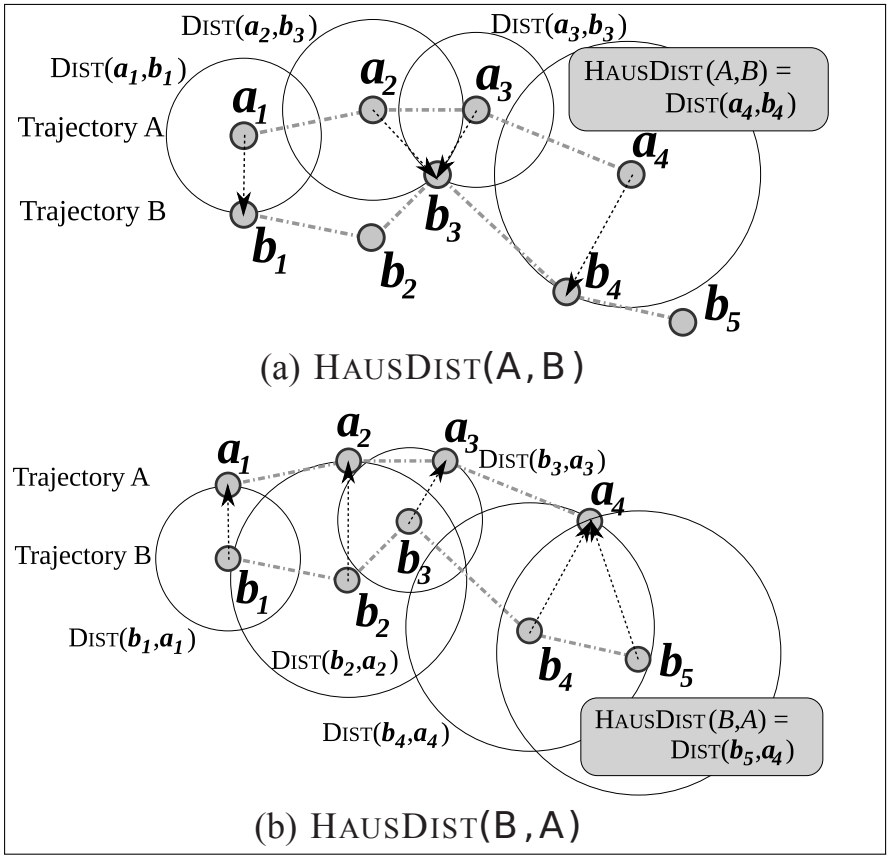


Figura 1.1: Ejemplo de cálculo de distancia de Hausdorff, tanto de A a B como de B a A [26].

Por otro lado, con el creciente aumento en los volúmenes de información, se hace indispensable utilizar eficientemente el almacenamiento para la representación de los datos. El tamaño de los datos, no solo impacta en el costo de almacenamiento, también influye en el costo de transmisión de los datos y en su procesamiento.

Un enfoque interesante para abordar este problema emerge del campo de los sistemas de recuperación de información, las llamadas estructuras de datos compactas [25]. Estas estructuras de datos permiten representar la información de una manera eficiente, utilizando un espacio muy cercano al óptimo teórico según la teoría de la información. Junto con ello, las estructuras compactas mantienen las propiedades de acceso directo a los datos, lo que permite realizar operaciones sobre ellos directamente, sin descompactar toda la estructura.

En los ejemplos descritos anteriormente es fácil visualizar contextos en los cuales existan grandes volúmenes de puntos, por ejemplo en la comparación de imágenes de alta resolución. Dichas imágenes son más costosas de analizar cuando requieren la indexación de los puntos en memoria secundaria que cuando están en memoria principal, simplemente por la localidad de los datos.

En la literatura existen variadas estructuras de datos compactas para representar conjuntos de puntos, entre ella el Wavelet Tree [24] y el k^2-tree [20]. El Wavelet Tree destaca porque permite obtener un almacenamiento teórico óptimo, a diferencia del k^2-tree. Sin embargo, en la práctica con el k^2-tree se puede obtener un menor costo de almacenamiento que con Wavelet Tree cuando los puntos se encuentran agrupados [25]. De acuerdo a la literatura revisada no existe un algoritmo que calcule la Distancia de Hausdorff sobre estructuras de datos compactas.

En este contexto, si alguien decide utilizar una estructura compacta para representar conjuntos de puntos tiene dos alternativas para calcular la Distancia de Hausdorff i) extraer los puntos desde la estructura compacta y luego procesarlo con algún algoritmo conocido que procese los datos sin indexarlos previamente; y (ii) calcular la Distancia de Hausdorff con un algoritmo que procese directamente sobre la estructura compacta.

En esta tesis se proponen dos nuevos algoritmos para calcular la Distancia de Hausdorff considerando que los conjuntos, son puntos de dos dimensiones y están representados mediante k^2-trees. Nuestro algoritmo sigue las ideas planteadas en [26] y [33] y [28], pero adaptándolas a la naturaleza de la estructura. En nuestro algoritmo usamos la distancia euclídea para el cálculo de la Distancia de Hausdorff.

1.1. Hipótesis y objetivos de la investigación

1.1.1. Hipótesis

Es factible conseguir algoritmos que calculen $\mathsf{HausDist}(A, B)$ directamente sobre los puntos representados en los k^2-trees y, que aprovechando las propiedades de esta estructura de datos compacta alcancen un mejor rendimiento que recuperar los puntos de los k^2-trees y aplicar el mejor algoritmo sobre conjuntos no indexados conocido en la literatura.

1.1.2. Objetivo General

Generar algoritmos eficientes, tanto en tiempo como en almacenamiento, para calcular $\mathsf{HausDist}(A, B)$ sobre la estructura de datos compacta k^2-tree.

1.1.3. Objetivos Específicos

1. Diseñar algoritmos para calcular $\mathsf{HausDist}(A, B)$ sobre conjuntos de puntos almacenados en k^2-trees.
2. Implementar los algoritmos para calcular $\mathsf{HausDist}(A, B)$.
3. Estudiar las propiedades de la estructura de datos compacta k^2-tree que puedan ser útiles para el cálculo de la Distancia de Hausdorff.
4. Evaluar de manera teórica y experimental los algoritmos propuesto para evaluar su rendimiento frente al mejor algoritmo propuesto en la literatura.
5. Generar una librería, con fines de investigación, con las implementaciones de los algoritmos.

1.2. Alcance de la investigación

Para el desarrollo de la librería se empleará el lenguaje de programación C. Los algoritmos se evaluarán por medio de una serie de experimentos con datos generados de manera aleatoria siguiendo diferentes distribuciones (gauss, homogénea) y de diferentes tamaños, tanto del espacio como del número de puntos de los conjuntos. Además, se evaluará con un conjunto de datos reales en algún dominio de aplicación.

1.3. Metodología de Trabajo

La Metodología de trabajo utilizada para el desarrollo de esta tesis consta de las siguientes etapas:

1. Realizar una revisión exhaustiva de la literatura con el propósito de estudiar los algoritmos propuestos para calcular la Distancia de Hausdorff. Realizando una división entre aquellos algoritmos que indexan o no los datos, además de aquellos que utilizan memoria principal o memoria secundaria para el procesamiento de los datos. También se analizarán las distintas estructuras de datos compactas con el fin de definir las propiedades que serán utilizadas para la implementación de los distintos algoritmos que contempla esta tesis. En específico, el énfasis estará en las propiedades y limitaciones de la estructura de datos compacta k^2-tree.
2. Diseñar e implementar algoritmos para calcular $\mathsf{HausDist}(A, B)$.
3. Realizar un análisis de complejidad en términos de almacenamiento y tiempo de ejecución de los algoritmos propuestos.
4. Implementar y adaptar el mejor algoritmo propuesto en la literatura (baseline) el cual será comparado con los algoritmos propuestos en esta tesis.
5. Realizar experimentos que consideren datos reales y sintéticos con el propósito de comparar nuestros algoritmos con el baseline. Los dataset a utilizar representaran diversos escenarios dadas ciertas configuraciones como por ejemplo: el tipo de distribución, cantidad de

puntos, tamaño del grid, por nombrar algunos. Estos conjuntos nos permitirán evaluar el rendimiento y escalabilidad de los algoritmos propuestos.

1.4. Contribución

En la presente tesis se presentan las siguientes contribuciones:

1. Se implementa el algoritmo propuesto por [33], el cual es utilizado como baseline de comparación.
2. Se implementan dos algoritmos para calcular la Distancia de Hausdorff que utilizan k^2-trees para almacenar y procesar los puntos. El primer algoritmo realiza podas en un solo k^2-tree, en este caso en el conjunto B. Mientras que el segundo algoritmo realiza podas en ambos k^2-trees.
3. Se realiza experimentación, obteniendo resultados favorables en comparación con el algoritmo baseline. Estos resultados han permitido enviar un articulo y hacer una presentación de nuestro trabajo en la Conferencia Latinoamericana de Computación durante el mes de Octubre del 2018.

1.5. Organización de la Tesis

En el siguiente capítulo se presentan los trabajos relacionados a algoritmos que solucionan el problema del cálculo de la Distancia de Hausdorff, además de trabajos que tratan acerca de la estructura de datos compacta k^2-tree. Este capítulo esta dividido en secciones que tratan de algoritmos que utilizan memoria principal y algoritmos que utilizan memoria secundaria. Al final de este capítulo se presenta una discusión que resume los algoritmos descritos en el estado del arte. En el Capítulo 3 se describe nuestro trabajo, además de la estrategia que se utilizará en el diseño de nuestros algoritmos. Se presentan dos nuevos algoritmos para calcular la Distancia de Hausdorff. El primero de los algoritmos realiza podas en el conjunto B de datos, mientras que el segundo algoritmo realiza podas en ambos conjuntos. En el Capítulo 4 presentamos los experimentos realizados. Se realiza una medición del tiempo de ejecución tanto para datos sintéticos como datos reales. Finalmente, las conclusiones y el trabajo futuro son presentados en el Capítulo 5.

Parte II

Estado del Arte

Capítulo 2

Estado del Arte

En este capítulo se discuten los algoritmos propuestos en la literatura que son utilizados para calcular la Distancia de Hausdorff. Estos algoritmos son divididos en algoritmos que utilizan memoria principal y memoria secundaria para el procesamiento de los datos. Adicionalmente se describen estructuras de datos compactas, haciendo el énfasis en la estructura k^2-tree, estructura utilizada para la implementación de los algoritmos presentados en esta tesis.

2.1. Distancia de Hausdorff en memoria principal

El primer algoritmo encontrado en la literatura para calcular la Distancia de Hausdorff en memoria principal es el propuesto por Atallah [2]. Este algoritmo es utilizado para calcular la Distancia de Hausdorff entre dos polígonos convexos. Dicho algoritmo es de complejidad temporal $\mathcal{O}(m+n)$, donde m y n corresponden al número de vértices exteriores de cada polígono convexo. Este algoritmo tiene la limitante que la intersección de los polígonos debe ser vacía. En la misma línea de comparar formas poligonales, Tang *et. al.* [35] presentan un algoritmo para calcular la Distancia de Hausdorff entre modelos poligonales complejos, el cual a diferencia del algoritmo anterior no requiere condiciones especiales sobre el conjunto de puntos. El algoritmo propuesto utiliza una subdivisión de Voronoi para dividir el modelo poligonal en triángulos más pequeños para calcular la Distancia de Hausdorff entre los vértices de dichos triángulos. El problema de este algoritmo es que entrega un resultado aproximado. Helmut *et. al.* en [1] presentan algoritmos que también utilizan diagrama de Voronoi para calcular la Distancia de Hausdorff cuya complejidad temporal es $\mathcal{O}((n+m)\log(n+m))$.

Por otro lado, Taha *et. al.* [33] proponen un algoritmo que busca evitar tener que evaluar todo el conjunto B de datos. En la primera iteración del proceso se toma un punto $p \in A$, se calcula la distancia d a todos los puntos de B. La menor distancia sera la Distancia de Hausdorff candidata, la cual es almacenada en la variable $cmax$. En la siguiente iteración, si al calcular la distancia d de un punto p_1 a cualquier punto de B, esta es menor que $cmax$, se termina de buscar el vecino más cercano de p_1, debido a que no es posible mejorar la solución candidata ($cmax$), ya que en el mejor de los casos esta distancia d ya es un mínimo local (línea 11 del Algoritmo 1). A esto el autor lo denomina *"rompimiento temprano"*. Esto ocurre por la definición implícita del cálculo de la Distancia de Hausdorff, la cual señala que se deben maximizar las menores distancias

Algorithm 1 EARLYBREAK. Calcula la Distancia de Hausdorff usando un rompimiento temprano

Sea A,B dos conjuntos de puntos finitos
$cmax \leftarrow 0$
$E \leftarrow A \backslash (A \cup B)$
$E_r \leftarrow randomize(E)$
$B_r \leftarrow randomize(B)$
for cada entrada $x \in E_r$ **do**
    $cmin \leftarrow \emptyset$
    for cada entrada $y \in B_r$ **do**
        $d \leftarrow \| x, y \|$
        if $d \leq cmax$ **then**
            break ▷ Rompimiento Temprano
        end if
        if $d \leq cmin$ **then**
            $cmin \leftarrow d$
        end if
    end for
    if $cmin \geq cmax$ **then**
        $cmax \leftarrow cmin$
    end if
end for
return $cmax$

encontradas. Esto se puede apreciar en Figura 2.1. Este rompimiento temprano del algoritmo le

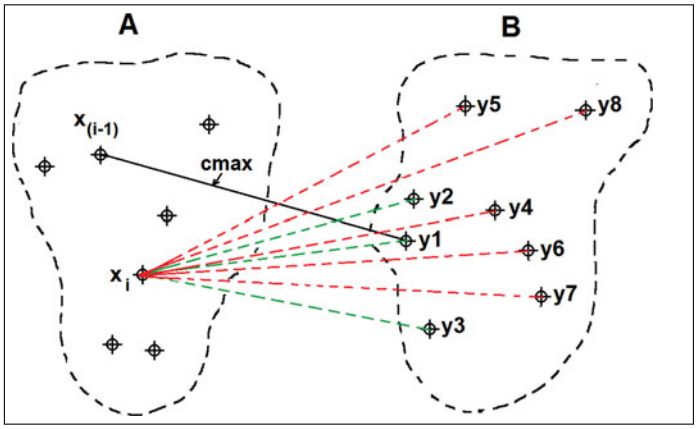


Figura 2.1: Ejemplo de cálculo de distancia de Hausdorff utilizando el algoritmo propuesto por [33].

permite mejorar su eficiencia en cuanto al tiempo de ejecución evitando eventualmente recorrer todos los puntos de B. Cuando los puntos son obtenidos desde imágenes, el escaneo horizontal o vertical de los puntos se realiza en forma secuencial lo que impide que el *"rompimiento temprano"* sea efectivo. Para enfrentar este problema Taha [33] propone randomizar los puntos antes de calcular la Distancia de Hausdorff (líneas 4 y 5 en algoritmo 1). Este algoritmo presenta una complejidad temporal $\mathcal{O}(m)$ para su mejor caso y $\mathcal{O}(mn)$ en su peor caso.

Otro algoritmo que utiliza esta técnica de *"rompimiento temprano"* es el propuesto por Chen *et. al.* [10], quienes presentan un algoritmo llamado Local Start Search (LSS). El *"rompimiento temprano"* de este algoritmo proviene de la probabilidad de que los puntos estén en zonas de densidad. Estos *"rompimiento temprano"* son realizados, al igual que en [33], en los bucles de escaneo de puntos. Además, el algoritmo utiliza un mecanismo para registrar la posición del último rompimiento como una posición de inicio, la cual es utilizada como centro de búsqueda del siguiente bucle, explorando los puntos cercanos alrededor de este centro. Para ello define el concepto de puntos vecinos, que son aquellos puntos ordenados en secuencia que se encuentran adyacentes a un punto determinado (ver Figura 2.2).

En otra línea, Guthe et. al. [14] proponen un algoritmo para calcular la Distancia de Hausdorff en el contexto de mallas geométricas. Este algoritmo para evitar revisar todos los puntos saca ventajas de las características específicas que poseen las mallas geométricas. El algoritmo solo revisa aquellas áreas donde existen distancias máximas entre los triángulos de cada malla. Para ello construye una grid a través de una estructura de datos Octree [22] en la cual almacena cada superficie, calculando las distancias mínimas y máximas entre ellas. Notar que en un Octree cada nodo del octágono tiene ocho hijos o ninguno, mientras que el nodo raíz describe un cuadro de límites cúbicos que encapsula todos los puntos hijos. Además, al igual que [26], utiliza una cola de prioridad, en la cual almacena las celdas ya procesadas, las que ordena por su distancia geométrica máxima. En esta línea Kang et. al. [18] presentan un algoritmo para calcular la Distancia de Hausdorff entre una malla geométrica triangular y una malla cuádruple, en la cual,

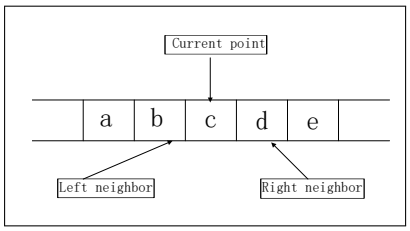


Figura 2.2: Puntos vecinos definidos en [10]

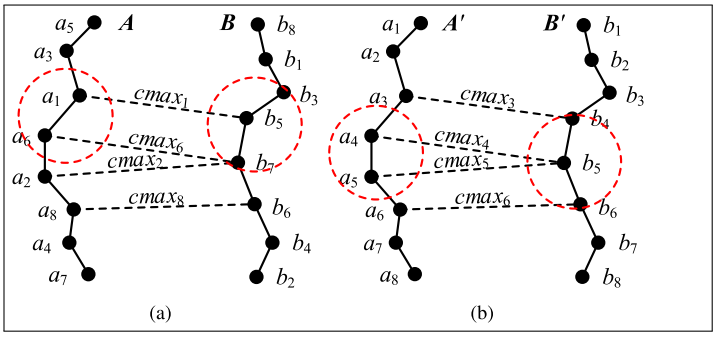


Figura 2.3: Búsqueda de difusión propuesto en [37].

buscan disminuir el uso de memoria al eliminar el almacenamiento de todas las combinaciones, evitando así tener que procesar todo el conjunto de información. Esto lo realiza utilizando un límite inferior y superior para el procesamiento de información.

En la línea de comparar modelos tri-dimensionales y utilizando las ideas presentadas en [33], Zhang et. al. [37] presentan algoritmos que utilizando lo que los autores llaman *"búsqueda de difusión"* buscan optimizar el cálculo de la Distancia de Hausdorff, para ello utiliza, al igual que [14], una estructura de datos OctTree. La idea principal de estos algoritmos es valerse de la probabilidad de que encontrando un *"rompimiento temprano"* definido en [33], la próxima solución se encuentra en puntos cercanos a esta primera solución candidata (ver círculos rojos en Figura 2.3). Esta idea es similar a la presentada en [10]. Otro punto interesante de estos algoritmos es que toma en cuenta la distribución y densidad de los puntos. Cuando se trata de conjunto de puntos dispersos, el autor propone un algoritmo que revisa los puntos del segundo conjunto en Z-orden, para ello convierte estos conjuntos utilizando código Morton, la cual es una función que mapea datos multidimensionales a una sola dimensión, preservando la localidad de los puntos [23]. Ahora bien, si se trata de un conjunto denso de puntos, el autor propone tomar el segundo conjunto y convertirlo en un OctTree.

Para comparar imágenes, Huttenlocher *et. al.* [15], proponen dos algoritmos, uno para objetos en $\mathbb{R}^2$ con complejidad $\mathcal{O}(mn \log mn)$ y otro algoritmo para objetos en $\mathbb{R}^3$ con complejidad $\mathcal{O}(m^2n^2 \propto (mn))$, donde m y n es el número de puntos de cada conjunto y $\propto (mn)$ corresponde a la función de traslación de dichos conjuntos. Estos algoritmos calculan la Distancia de Hausdorff a través de los polígonos generados por los puntos exteriores de los conjuntos de datos, siguiendo la misma idea presentada en [2]. Utilizando este mismo método de traslación, pero específicamente

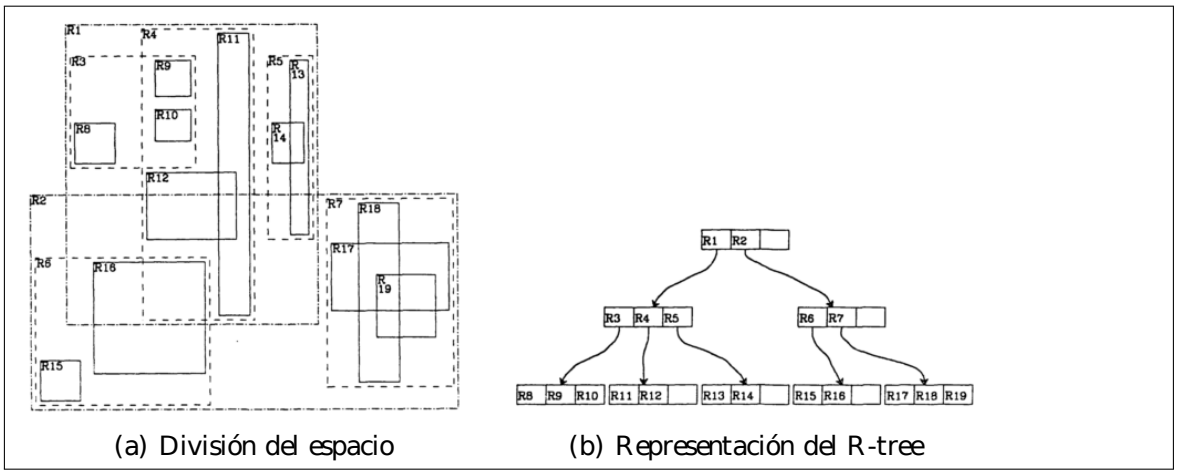


Figura 2.4: R-tree [4].

para el cálculo de la Distancia de Hausdorff entre dos líneas, Rote [29] y Li *et. al.* [21] proponen algoritmos que realizan algunas mejoras al algoritmo presentado en [15]. La complejidad de estos algoritmos es $\mathcal{O}((n+m)\log(n+m))$. Ahora bien, el problema de estas técnicas de traslación es que, dependiendo de la cantidad de puntos de cada conjunto, requieren de un excesivo tiempo de cómputo. Para disminuir este tiempo Chang *et. al.* [8] proponen un algoritmo que utiliza un filtro de Kalman [17] para filtrar puntos y con ello mejorar el tiempo de cómputo en la comparación de dichos conjuntos.

La Distancia de Hausdorff también puede ser calculada entre curvas. Un algoritmo para esto es el presentado por Chen *et. al.* [9]. Este algoritmo utiliza un método heurístico para dividir las curvas en sub-intervalos a través de curvas B-spline. Además, este algoritmo utiliza técnicas de poda geométrica para eliminar aquellos sub-intervalos que no aportan a la solución final.

2.2. Cálculo de la Distancia de Hausdorff en memoria secundaria

Nutanong *et. al.* [26] proponen algoritmos que utilizan estructuras de datos R-tree [4], las cuales son almacenadas en memoria secundaria, ver 2.4. El uso de estas estructuras R-tree permite la indexación de los conjuntos A y B, con lo cual se puede realizar podas de aquellos puntos que no aportan a la solución. Los algoritmos presentados requieren de algunas definiciones previas. Una de ella es la definición de un límite inferior, el cual es calculado como la mínima distancia entre la cara del rectángulo generado por el MBR(Minimum Bounding Rectangle) de A que está frente a la cara del rectángulo generado por el MBR de B, esto puede ser visto en Figura 2.5. Este límite inferior determina una cota inferior para el cálculo de la Distancia de Hausdorff.

Además, los autores definen un límite superior, el cual es calculado como la máxima distancia entre la cara del MBR del conjunto A, con respecto al MBR del conjunto B, tal como lo muestra la Figura 2.6. Este límite superior determina una cota superior para el cálculo de la Distancia de

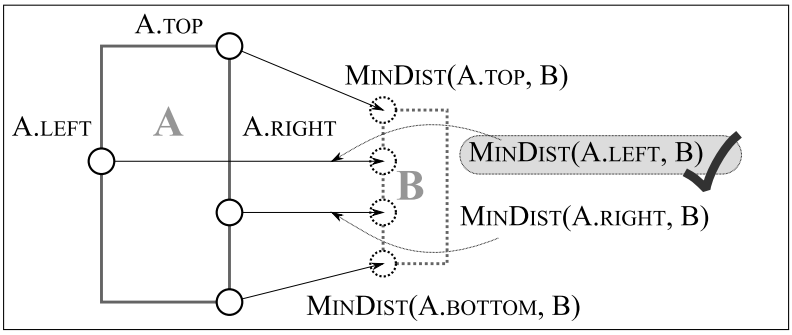


Figura 2.5: Límite inferior definido por Nutanong [26].

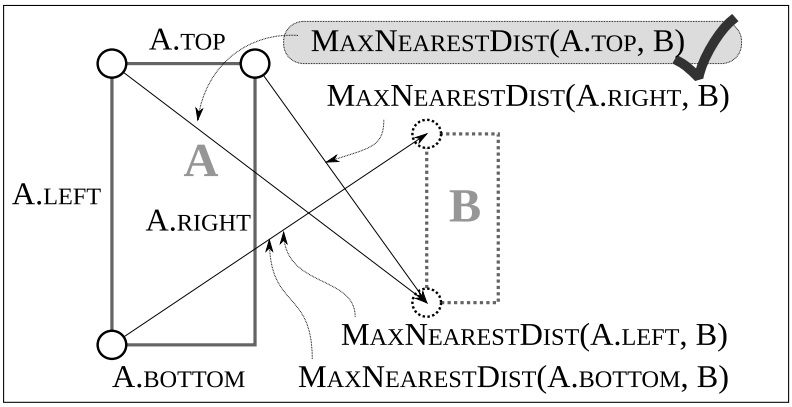


Figura 2.6: Límite superior definido por Nutanong. [26].

Hausdorff.

El algoritmo principal presentado por Nutanong utiliza colas de prioridad. Utiliza una cola de principal (MainPQ) y múltiples colas de prioridad secundarias (SecPQs) asociadas a cada nodo de MainPQ. MainPQ controla el orden en que se exploran los nodos en el primer conjunto. Para cada entrada de MainPQ, su SecPQ correspondiente controla el orden en que se exploran los nodos en el segundo conjunto. Las entradas en MainPQ se organizan en orden descendente y las entradas en un SecPQ se organizan en orden ascendente, como se puede ver, este ordenamiento corresponde a la naturaleza de la Distancia de Hausdorff , por lo que al finalizar el procesamiento de los datos, la Distancia de Hausdorff final puede ser encontrada en el tope de MainPQ. Esto puede ser visto en 2.7. El algoritmo propuesto realiza un control balanceado de los nodos escaneados, esto lo realiza utilizando los límites superiores e inferiores definidos en el punto anterior, los cuales va optimizando en cada cálculo de distancia entre los nodos de ambos conjuntos.

2.3. Estructuras de Datos Compactas

Las estructuras datos compactas son estructuras que permiten representar datos utilizando de forma eficiente el espacio, mientras que todavía son capaces de resolver de forma eficiente

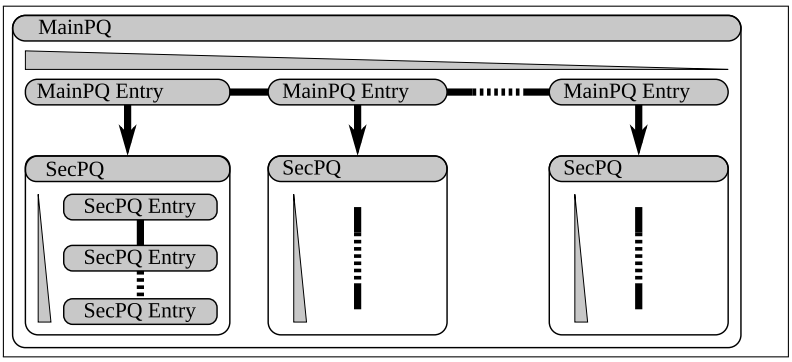


Figura 2.7: Estructuras utilizadas por Nutanong. [26].

operaciones requeridas sobre los datos [20, 25]. Existen estructuras de datos compactas para: secuencias binarias (bitmaps), secuencias de símbolos, árboles cardinales, permutaciones, grafos, indexación de rangos, entre otras.

Entre las estructuras de datos compactas podemos encontrar el k^2-tree, ver Figura 2.8, el cual es definido por Brisaboa et. al [7] como una estructura de datos compacta basada en QuadTree [31]. Esta estructura también puede verse como una relación binaria representada por un árbol k^2-ario de altura $h = \lceil \log_k n \rceil$, el cual representa una matriz de adyacencia binaria de rango $n \times n$ de manera compacta. Puede ser usada por ejemplo para representar grafos [7], datos RDF (Resource Description Framework), o un grid bi-dimensional de puntos [5].

Con respecto a la estructura del árbol conceptual (Figura 2.8) debemos decir que cada nodo del árbol está etiquetado con un bit 1 si la sub-matriz correspondiente contiene al menos un uno, o con un bit 0 si la sub-matriz está llena de ceros. Así el proceso continúa recursivamente sólo con aquellas sub-matrices que contienen al menos un 1. La subdivisión recursiva de la matriz se detiene cuando la sub-matriz actual está llena de ceros o cuando se completan las celdas de la matriz de adyacencia original. Esto puede verse en la Figura 2.8. Finalmente, lo que se almacena en el k^2-tree es la secuencia de bits resultantes de recorrer el árbol conceptual en anchura, guardando separadamente el último nivel (bitmap L) del resto del árbol (bitmap T). Para la Figura 2.8, los bitmaps que son almacenados son $T = 1001$ y $L = 11011110$.

El ahorro de espacio en un k^2-tree, está dado por la existencia de cuadrantes que se encuentran llenos de ceros y, por lo tanto, no son subdivididos. Además, no se almacena explícitamente la topología del árbol, ahorrando el espacio requerido por los punteros y nodos de una representación clásica. Algunas operaciones que se pueden realizar en un k^2-tree, son la recuperación de los vecinos directos y reversos, recuperaciones de celdas, consultas por rango.

Una variante del k^2-tree es el enfoque Híbrido, en el cual busca mejorar los tiempos de consulta. Para ello mantiene dos valores de k, uno valor de k más grande para los niveles superiores, y uno valor de k menor para los niveles inferiores, con esto se logra disminuir la altura del árbol.

Otra versión especial del k^2-tree es aquella que utiliza compresión de unos. Esta estructura busca obtener una representación eficiente para matrices binarias que contienen grandes grupos de ceros y unos. Un ejemplo de estas matrices son aquellas que representen imágenes binarias. La principal diferencia con el k^2-tree original es la filosofía de división de las sub-matrices. En esta

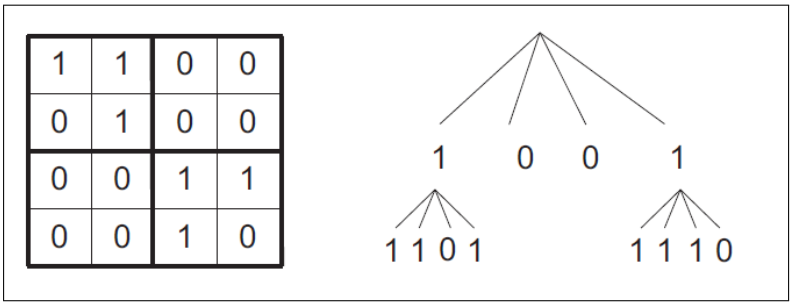


Figura 2.8: Representación de una matriz de adyacencia en un k^2-tree [7].

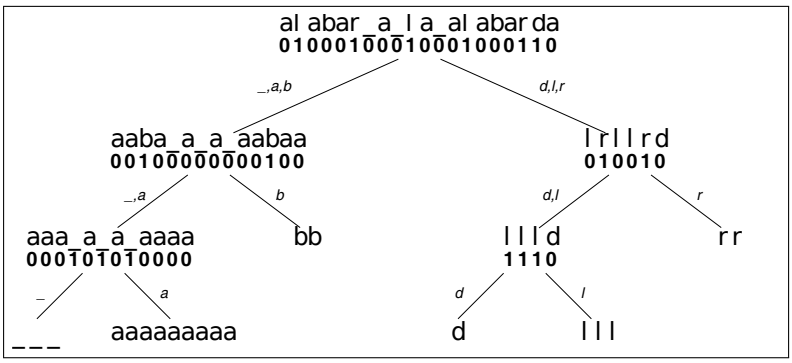


Figura 2.9: Wavelet Tree definido en [13].

estructura la descomposición original se detendrá cuando se encuentren regiones uniformes, ya sea de ceros o unos. Esto tiene como ventaja reducir el número de nodos en el árbol conceptual cuando grandes regiones de unos aparecen en la matriz binaria.

Otra variante es el Dk^2-tree el cual es una versión dinámica del k^2-tree. En esta estructura al cambiar una celda de la matriz de adyacencia dinámicamente se hace la modificación en el árbol. Esto permite hacer modificaciones posteriores a la construcción inicial del árbol.

El almacenamiento requerido por un k^2-tree para almacenar un conjunto de puntos ha sido estudiado en diversos trabajos tales como en [5,6,28] los que en general concluyen que el k^2-tree es eficiente para representar puntos, en especial si los puntos se encuentran concentrados en regiones del espacio.

Si bien no son utilizadas en esta tesis, podemos destacar un tipo estructura de datos sucinta llamada Wavelet Tree, presentada en [13] para la representación de secuencias de texto. En termino generales un wavelet tree es un árbol, donde cada nodo es representado por un bitmap. Como se puede apreciar en 2.9, el bitmap que representa el nodo raíz tiene tantos bits como posee la secuencia de texto. Otra característica de este árbol es que cada nodo a su vez posee dos hijos, los cuales están relacionados con su padre a través de un elemento definido. Sea un alfabeto α, la altura puede ser calculada como $\lceil \log \alpha \rceil$ y el tamaño total de almacenamiento de la estructura es de $n\lceil \log \alpha \rceil$.

2.3.1. Discusión

Como se ha señalado, existen variados contextos en los cuales la Distancia de Hausdorff puede ser calculada, tanto para memoria principal como para memoria secundaria. En memoria principal, podemos destacar el algoritmo propuesto por Taha [33], el cual utiliza lo que el autor denomina un *"rompimiento temprano"*, para evitar escanear todos los puntos del segundo conjunto B. De acuerdo a la revisión del estado del arte y a nuestro propio conocimiento, podemos decir que el algoritmo propuesto por Taha, es el de mejor rendimiento en cuanto a tiempo de ejecución. El principal problema de este algoritmo es que solo realiza podas en el conjunto B, teniendo que escanear exhaustivamente los puntos del conjunto A. Otro algoritmo que sigue la idea de este *"rompimiento temprano"* es el presentado en Zhang et. al. [37], pero mantiene el problema de tener que revisar todos los puntos del conjunto A.

Cuando se trata de memoria secundaria existe el algoritmo propuesto por Nutanong *et. al.* [26], el cual mediante el uso de un R-Tree y sus propiedades, busca evitar escanear todos los puntos en los conjuntos A y B. Concretamente utiliza las propiedades de los MBRs, de las cuales se vale para definir cotas superior e inferior para evitar escanear exhaustivamente los puntos de ambos conjuntos. Notar que este algoritmo también utiliza estructuras adicionales para obtener la Distancia de Hausdorff.

Señalar además que se puede calcular la Distancia de Hausdorff para distintas dimensiones. En $\mathbb{R}^3$, destacan los algoritmos propuestos en [15]. Otro punto interesante es que existen algoritmos [29] [21] [15] que utilizan técnicas de translación antes de calcular la Distancia de Hausdorff. Esta técnica permite buscar el mayor grado de similitud entre dos conjuntos de datos y es principalmente usado para la comparación de imágenes. El problema de estos algoritmos es que dependiendo del tamaño de los conjuntos pueden tener un excesivo tiempo de ejecución.

Con respecto a la estructuras de datos compactas, nos hemos centrado principalmente en las características del k^2-tree en sus distintas versiones. Para el desarrollo de esta tesis se ha tomado la versión del k^2-tree definida en [7], la cual permite el procesamiento de datos haciendo un uso eficiente del almacenamiento.

Parte III

Algoritmos Propuestos

Capítulo 3

Algoritmos Propuestos

Como se ha señalado existen diversos algoritmos que resuelven de distintas formas y en variados escenarios el cálculo de la Distancia de Hausdorff. Para la implementación de nuestros algoritmos pusimos nuestra atención en dos [33] [26], porque poseen estrategias de poda útiles que se pueden reusar en nuestra implementación. En el caso de lo propuesto en [33], aprovechando las cualidades de la estructura de datos compacta k^2-tree, nuestros algoritmos utilizan el principio básico del *"rompimiento temprano"*, pero haciendo el descarte de un grupo de puntos en ambos conjuntos.

En el caso de lo propuesto en [26], si bien es cierto la estructura de datos k^2-tree no posee MBRs, si se puede utilizar la idea general para eliminar conjuntos de puntos que no aporten a la solución. Esto se realiza a través del cálculo de las mayores distancias, ya sea desde un punto del conjunto A a un vértice de una región del conjunto B (ver Figura 3.1), o la mayor distancia entre áreas (ver Figura 3.2). Para el cálculo de la mayor distancia entre áreas, se debe calcular la mayor distancia entre cada vértice de un área del conjunto A con respecto a cada vértice de un área del conjunto B (ver Figura 3.2).

Notar que se puede seguir la estrategia presentada en [26] en cuanto a utilizar estructuras adicionales para el cálculo de la Distancia de Hausdorff, sin embargo al no existir las propiedades de los MBRs en los k^2-tree la poda es ineficiente. Como se puede ver en Figura 2.7, cuando el algoritmo finaliza, es decir termina de ordenar cada cola de prioridad, en el tope de $MainPq()$ se encuentra la solución para la Distancia de Hausdorff. Esto se da por la naturaleza MAX-MIN propia de la Distancia de Hausdorff. Esto no se puede asegurar al utilizar k^2-tree, ya que el ordenamiento de prioridades solo puede ser realizado por las distancias máximas entre cada cuadrante de ambos conjuntos. Lo cual no asegura que en el tope de la cola de prioridad exista una solución real, por lo que tendríamos que volver a escanear cada cola para saber si existe una Distancia de Hausdorff válida. Además, en esta tesis evitamos usar estructuras adicionales, ya que uno de los objetivos de las estructuras de datos compactas es minimizar el uso de memoria principal.

A continuación se presentan nuestros dos algoritmos (HDKP1 y HDKP2), los cuales se basan en lo anteriormente propuesto. Recordar que HDKP1 realiza podas solo en el conjunto B, mientras que HDKP2, busca realizar podas en los conjuntos A y B.

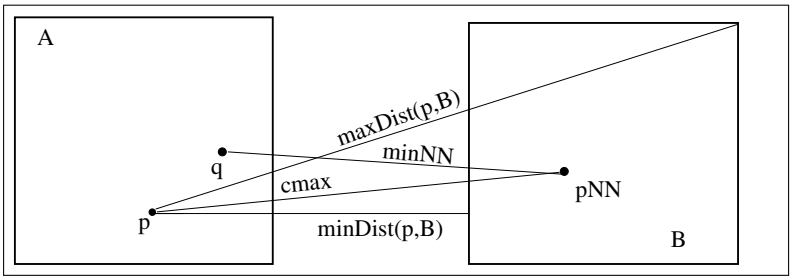


Figura 3.1: Métricas utilizadas en las reglas de poda 1, 2 y 3

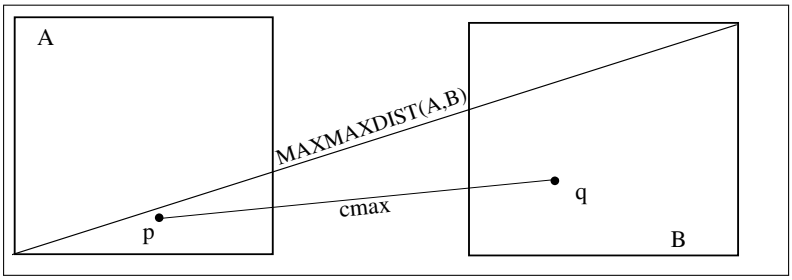


Figura 3.2: Métrica utilizada en la regla de poda 4

3.1. Algoritmo HDKP1

El algoritmo asume que los conjuntos de puntos A y B se encuentran almacenados en los k^2-trees K_A y K_B respectivamente. El algoritmo HDKP1 explora exhaustivamente los puntos en K_A y por cada punto del conjunto A encuentra su vecino más cercano en K_B, siempre y cuando dicho vecino sea una posible solución.

La estrategia seguida por el algoritmo es de ramificación y poda. Concretamente usa tres reglas de poda basadas en la distancia euclídea entre dos puntos y de un punto a un rectángulo.

El algoritmo hace uso de las funciones $Dist(p_1, p_2)$ para obtener la distancia entre dos puntos y las función $minDist(p, B)$ y $maxDist(p, B)$ para obtener la menor y la mayor distancia entre el punto p y el cuadrante B respectivamente (ver Figura 3.1).

La función $minDist(.,.)$ se encuentra formalmente definida en [30]. Por su parte la función $maxDist(.,.)$ corresponde a una caso especial de la función $MAXMAXDIST(S, R)$ con R y S dos rectángulos definida en [11] en que S o R es un rectángulo definido por solo un punto y corresponde a q.

A continuación se explica en detalle en que consisten las tres reglas de podas que utiliza nuestro primer algoritmo.

3.1.1. Reglas de poda

Dado un punto $p \in A$, los valores de las funciones $minDist(p, B)$ y $maxDist(p, B)$ representan los límites inferior y superior respectivamente del intervalo en el cual se encuentra la distancia

del vecino más cercano de p dentro del rectángulo B.

Asumimos que $cmax$ representa una solución parcial (solución candidata) de la Distancia de Hausdorff y cuyo valor se obtuvo entre $Dist(p, pNN)$ (ver Figura 3.1) con $p \in A$ y $pNN \in B$. Las reglas son las siguientes:

1. **Regla de poda 1:** Si durante la exploración de K_B se encuentra un punto $pNN \in B$ tal que $minNN = Dist(q, pNN) \leq cmax$, con $q \in A$, entonces ya no es necesario seguir explorando K_B, pues el vecino más cercano de q no mejorará la solución mantenida en $cmax$, ya que si $t \neq pNN \in B$ es el vecino más cercano de q, entonces $Dist(q, t) \leq Dist(q, pNN)$.

2. **Regla de poda 2:** Esta segunda regla se basa en la distancia $maxDist(.,.)$ (ver Figura 3.1). Considere que está buscando en K_B el vecino más cercano para un punto $q \in A$, y R es la región del espacio representada por K_B, si durante la búsqueda ocurre que $maxDist(q, R) \leq cmax$, entonces ya no es necesario seguir explorando K_B, pues el vecino más cercano de q se encontrará a una distancia menor o igual que $maxDist(q, R)$ y por lo tanto el valor de la Distancia de Hausdorff mantenida hasta el momento (valor de $cmax$) no cambiará.

3. **Regla de poda 3:** Adicional a las dos reglas anteriores, el algoritmo utiliza la regla de poda basada en $minDist(.,.)$ y que típicamente usan los algoritmos conocidos (por ejemplo el propuesto en [30]) para calcular el vecino más cercano. Dicha regla descarta regiones o zonas delimitadas por un rectángulo R generadas por K_B. Específicamente, si $q \in A$, $minNN$ la mejor distancia conocida para el vecino más cercano de q y $minDist(q, R) > minNN$, entonces la región R no necesita ser explorada.

3.1.2. Descripción del algoritmo HDKP1

En Algorithm 2 se muestra la parte principal del algoritmo HDKP1. Se asume que la función $getRoot(.)$ obtiene el nodo raíz del k^2-tree (línea 3). Esencialmente el tamaño de la matriz y el nivel del nodo. De la misma manera $getRegion(.)$ permite obtener la región del espacio cubierta por K_B representada mediante un rectángulo definido por dos puntos extremos opuestos (línea 4). Tal como indicamos en la sección anterior, $cmax$ es una variable utilizada para mantener la solución de la Distancia de Hausdorff, la cual se maximiza conforme el algoritmo progresa. Luego HDKP1 por cada punto $p \in A$ almacenado en K_A verifica, mediante el algoritmo $NNMax$ (ver Algorithm 3), si la distancia entre p y su vecino más cercano mejora la Distancia de Hausdorff mantenida en $cmax$ (líneas 5-10 de Algorithm 2). Notar que en la línea 7 se aplica la regla de poda 1. Se asume que $pNN \in B$ es un punto que optimizó la variable $cmax$ para un punto en A procesado antes que el punto $p \in A$. En esta etapa del algoritmo, la distancia entre p y pNN ($Dist(p, pNN)$) constituye una cota superior para el vecino más cercano de p; y por lo tanto se ejecuta el algoritmo $NNMax$ solamente si la distancia entre p y su vecino más cercano mejora la solución mantenida en $cmax$. Para mayor claridad de la aplicación de la regla de poda 1 podemos ver la Figura 3.1 en donde asumiendo que p fue procesado antes que q, es claro que si $minNN = Dist(q, pNN) \leq cmax$ la distancia entre q y pNN no mejorará el valor de $cmax$.

El algoritmo $NNMax$ (ver Algorithm 3), en general permite calcular el vecino más cercano de un punto $q \in A$ sobre el conjunto de puntos B representado por el k^2-tree K_B; con el propósito

de mejorar la Distancia de Hausdorff. Eventualmente $NNMax$, mediante las reglas de poda definidas previamente puede evitar calcular el vecino más cercano de q y decidir tempranamente que la distancia entre q y su vecino más cercano no mejorará la solución.

$NNMax$ usa una cola de prioridad representada por un heap pQ (ver Algorithm 3, línea 1). Cada entrada de pQ tiene la estructura $\langle n, r, d\rangle$ con n el nodo del k^2-tree, r la región del nodo n y d la distancia máxima ($maxDist(q, r)$) entre q y r. pQ está organizado por d, es decir, el menor valor de d se encuentra en la raíz del heap.

Algorithm 2 Algoritmo HDKP1 para calcular la Distancia de Hausdorff sobre dos conjuntos de puntos almacenados en k^2-tree

HDKP1(K_A, K_B)
input: Dos k^2-Tree K_A y K_B.
output: La Distancia de Hausdorff.

```
1:  pNN = (∞, ∞)
2:  cmax = 0                                   ▷ cmax Distancia de Hausdorff
3:  root =getRoot(K_B)
4:  r = getRegion(root)
5:  for Cada punto p en K_A do
6:      minNN = Dist(p, pNN)
7:      if minNN > cmax then                                   ▷ regla 1
8:          cmax = NNMax(root, r, p, cmax, minNN)
9:      end if
10: end for
11: return cmax
```

El algoritmo $NNMax$ (Algorithm 3) inicialmente (línea 3) crea e inserta una entrada en la cola de prioridad pQ. Tal entrada considera la raíz de K_B, la región representada por el K_B y la distancia ($maxDist(.,.)$) de q a dicha región. Luego el algoritmo ejecuta un ciclo (líneas 4-29) procesando las entradas de pQ para calcular el vecino más cercano de q que eventualmente mejorará la solución. La función $Extract - Min(.)$ (línea 5) recupera y elimina la entrada de pQ con el menor valor de d.

En las líneas 6-13 se procesan las entradas e de pQ que corresponden a puntos de B los que se encuentran en las hojas de K_B y que se ubican en el último nivel. Los nodos hojas que no se encuentran en el último nivel representan regiones en las cuales no existen puntos (zonas vacías) y por lo tanto son descartadas por el algoritmo. En este tipo de entradas $e.r$ representa las coordenadas del punto de B almacenado en la hoja y por lo tanto $e.d$ ($maxDist(q, r)$) corresponde a la distancia entre los puntos q y $e.r$. Notar que en las líneas 7-9 se aplica la regla de poda 1. En este punto del algoritmo, como $e.d$ representa una distancia real entre q y $e.r$ y si dicha distancia es menor o igual que la candidata en $cmax$, entonces ya no es necesario continuar explorando K_B pues el vecino más cercano real de q será menor o igual a $e.d$ y por lo tanto ya no mejorará el valor de $cmax$, y el algoritmo termina.

Entre las líneas 15-28 se procesan las entradas e de pQ correspondientes a nodos intermedios (no hojas) de K_B. Dicho procesamiento consiste básicamente en expandir el nodo $e.n$ de K_B e

Algorithm 3 (**NNMax**)Maximización de la Distancia de Hausdorff

NNMax($rootNode, Region, q, cmax, minNN$)
input: Nodo raíz de k^2-tree K_B, el punto $q \in A$, la Distancia de Hausdorff $cmax$ candidata y $minNN$ la cota superior para la distancia entre q y su vecino más cercano.
output: La Distancia de Hausdorff $cmax$ actualizada.

```
1: pQ =createMin-heap( )
2: d = maxDist(q, Region)
3: Insert(pQ, ⟨rootNode, Region, d⟩)
4: while (Not Empty(pQ)) do
5:     e = Extract-Min(pQ)
6:     if isLeaf(e.n) then
7:         if (e.d ≤ cmax) then                                  ▷ regla 1
8:             return cmax
9:         end if
10:        if e.d < minNN then
11:            minNN = e.d
12:            pNN1 = e.r
13:        end if
14:    else
15:        for all Nodo h hijo de e.n do
16:            if (hasChildren(h) then
17:                r = getRegion(h)
18:                mayDist = maxDist(q, e.r)
19:                if (mayDist ≤ cmax) then                      ▷ regla 2
20:                    return cmax
21:                end if
22:                menDist = minDist(q, e.r)
23:                if (menDist < minNN) then                     ▷ regla 3
24:                    Insert(pQ, ⟨h, r, mayorDis⟩)
25:                end if
26:            end if
27:        end for
28:    end if
29: end while
30: pNN = pNN1
31: return minNN
```

insertar los nodos hijos no vacíos de $e.n$ en pQ. Sobre estos nodos se aplica la regla de poda 2 (líneas 19-21). Para ello se calcula la cota superior para el vecino más cercano de q ($maxDist(q, r)$) y que se encuentra dentro de la región r de h. Es claro que si dicha cota está por debajo de $cmax$ el vecino más cercano de q en B no mejorará la solución, y por lo tanto ya no es necesario seguir explorando K_B, el algoritmo termina.

Antes de insertar una entrada en pQ se aplica la regla 3. Es decir, se calcula la cota inferior de la distancia del vecino más cercano de q en r ($minDist(q, r)$). Si el valor de esta cota está por sobre $minNN$, es decir, la distancia candidata entre q y su vecino más cercano descubierto hasta este punto del algoritmo, entonces la entrada correspondiente al hijo h de $e.n$ no se agrega a la cola de prioridad pQ, pues no minimizará el valor de $minNN$.

3.2. Algoritmo HDKP2

El algoritmo HDKP2 (Algorithm 4) al igual que el primer algoritmo HDKP1 asume que los conjuntos de puntos A y B se encuentran almacenados en los k^2-trees K_A y K_B respectivamente. A diferencia del primer algoritmo HDKP1, el algoritmo HDKP2 busca evitar explorar exhaustivamente los puntos tanto en K_A como en K_B.

La estrategia seguida por el algoritmo HDKP2, al igual que en el primer algoritmo, es de ramificación y poda, pero ahora aplicándola en K_A. Este algoritmo usa una nueva regla de poda, la cual está basada en la distancia euclídea máxima entre regiones de ambos k^2-trees (ver Figura 3.2). Además, utiliza el algoritmo NNMax (ver Algorithm 3) para aplicar las reglas de poda 2 y 3 en K_B (ver Algorithm 5, línea 9).

El algoritmo hace uso de la función $MAXMAXDIST(A, B)$ para obtener la mayor distancia entre el rectángulo A y el rectángulo B (ver Figura 3.2), definida en [11].

A continuación se explica en que consiste la regla de poda utilizada en nuestro segundo algoritmo.

3.2.1. Regla de poda

Dadas dos regiones $R \subseteq A$ y $S \subseteq B$, el valor de la función $MAXMAXDIST(R, S)$ representa el límite superior del intervalo en el cual se encuentra la distancia del vecino más cercano de cualquier punto de R respecto de la región S.

Asumimos que $cmax$ representa una solución parcial (solución candidata) de la Distancia de Hausdorff y cuyo valor se obtuvo entre $Dist(q, B)$ con $q \in A$. La regla es la siguiente:

1. **Regla de poda 4:** La regla utilizada para realizar podas en el segundo algoritmo se basa en la distancia $MAXMAXDIST(.,.)$ (ver Figura 3.2). Sean R y S dos regiones de K_A y K_B respectivamente. Si durante la búsqueda ocurre que $MAXMAXDIST(R, S) \leq cmax$, entonces ya no es necesario seguir explorando los puntos de la región R, pues el vecino más cercano de cualquier punto de esta región se encontrará a una distancia menor o igual que $MAXMAXDIST(R, S)$, por lo tanto el valor de la Distancia de Hausdorff mantenida hasta el momento (valor de $cmax$) no cambiará.

3.2.2. Descripción del algoritmo HDKP2

En Algorithm 4 se muestra la parte principal del algoritmo HDKP2. En la línea 2 obtenemos cualquier punto $p \in A$, para luego calcular su vecino más cercano en B. Este valor es asignado a la variable $cmax$ (línea 3). Al inicializar esta variable con un candidato valido aumenta las

probabilidades de realizar podas en el primer ciclo de la ejecución. Tal como se ha indicado, *cmax* es una variable utilizada para mantener una solución candidata de la Distancia de Hausdorff, la cual se maximiza conforme el algoritmo progresa. En las líneas 4 y 5 se asume que la función *getRoot*(.) obtiene el nodo raíz de ambos k^2-trees. Luego HDKP2 realiza la llamada al algoritmo $HDKPrunning$ (ver Algorithm 5).

El algoritmo $HDKPrunning$ (ver Algorithm 5), a través de la regla de poda 4 busca evitar escanear todo el conjunto de puntos en K_A. Además, utiliza el algoritmo $NNMax$ anteriormente descrito para realizar las reglas de podas 2 y 3 en K_B. No se utiliza la regla de poda 1, ya que de acuerdo a la experimentación realizada no mejora el rendimiento del algoritmo HDKP2. El algoritmo $HDKPrunning$ es el encargado de mejorar la variable *cmax*, para luego entregar una solución final para la Distancia de Hausdorff.

$HDKPruning$ al igual que $NNMax$ utiliza una cola de prioridad, ahora representada por un max-heap *pQ* (ver Algorithm 5, línea 1). Cada entrada de *pQ* tiene la región del nodo n y una distancia d, que es calculada por el algoritmo *isCandidate*, y se inicializa con valor 0. *pQ* está organizado por d, es decir, el mayor valor de d se encuentra en la raíz del heap. Este heap es utilizado para recorrer las regiones de K_A. El algoritmo $HDKPruning$ recibe como parámetros el nodo raíz de K_A y K_B, además de la variable *cmax*, la cual es optimizada para determinar la Distancia de Hausdorff.

El algoritmo $HDKPruning$ (Algorithm 5) inicialmente (líneas 1, 2 y 3) crea e inserta una entrada en la cola de prioridad *pQ*. Luego el algoritmo ejecuta un ciclo (líneas 4-23) procesando las entradas de *pQ* con el fin de recorrer las regiones de K_A. El objetivo de este ciclo es podar aquellas regiones de puntos que no mejorarán la solución. Para aquellos puntos que no son podados, se calcula su vecino más cercano con respecto a K_B, para ello se utiliza el algoritmo anteriormente descrito $NNMax$ (Algorithm 3).

En la línea 6 se extraen las entradas *aa* de *pQ*, que corresponde a regiones o puntos de K_A. En la línea 6 si se trata de un nodo hoja, es decir si se trata de un punto de K_A, es necesario calcular su vecino más cercano con respecto a K_B. Notar que si se ha alcanzado un nodo hoja, es porque estos puntos no han sido podados, es decir podrían mejorar la solución hasta ahora almacenada en *cmax*, por lo que deben ser revisados. En la línea 9 se hace la llamada al algoritmo $NNMax$, el cual a su vez realiza las podas 2 y 3 en K_B, descritas anteriormente. Si la solución entregada por $NNMax$ mejora el valor de *cmax* esta es actualizada, en caso contrario es descartada, manteniendo el valor actual de *cmax*. En la línea 14, en caso de no tratarse de una hoja, el algoritmo sigue revisando las regiones de K_A. Recordar que los nodos hojas que no se encuentran en el último nivel representan regiones en las cuales no existen puntos (zonas vacías) y por lo tanto son descartadas por el algoritmo.

Entre las líneas 14 y 21 se procesan las entradas *aa* de *pQ* correspondientes a nodos internos (no hojas) de K_A. En la línea 16 se obtiene una región de K_A, la cual es evaluada por el algoritmo *isCandidate* (Algorithm 6), el cual es el encargado de definir si esa región contiene puntos candidatos o debe ser podada aplicando la regla de poda 4. El algoritmo *isCandidate* entrega como respuesta la distancia obtenida al calcular $MAXMAXDIST(.,.)$ entre las regiones $R \subseteq K_A$ y $S \subseteq K_B$. Si la región de K_A tiene puntos candidatos, es puesta en la cola de prioridad *pQ* de acuerdo a la distancia entregada por el algoritmo *isCandidate*. Notar que en este ciclo todas las regiones son revisadas para determinar si contiene puntos candidatos. Ahora bien, si una región

Algorithm 4 Algoritmo HDKP2 para calcular la distancia de Hausdorff sobre dos conjuntos de puntos almacenados en k^2-tree

HDKP2(K_A, K_B)
input: Dos k^2-Tree K_A y K_B.
output: La Distancia de Hausdorff.

```
1: cmax = 0                                          ▷ cmax Distancia de Hausdorff
2: Obtenemos un punto p de K_A                       ▷ Cualquier punto
3: cmax = NearestNeighbor(p, K_B)
4: root_A = getRoot(K_A)
5: root_B = getRoot(K_B)
6: hd = HDKPruning(root_A, root_B, cmax)
7: return hd
```

es podada, también son podados sus nodos internos y sus nodos hojas, sin necesidad de ser revisados. El algoritmo finaliza entregando la variable *cmax* optimizada, la cual es la Distancia de Hausdorff definitiva.

El objetivo del algoritmo *isCandidate* (Algorithm 6) como se ha señalado es determinar que regiones de K_A tienen puntos candidatos, es decir puntos que puedan mejorar la solución hasta ahora almacenada en *cmax*. Para ello al igual que los algoritmos anteriores utiliza una cola de prioridad, en este caso, un min-heap. Esta es una estructura en la cual el valor menor se encuentra en el tope del heap. Dicha entrada considera la raíz de K_B, la región representada por K_B y la distancia ($MAXMAXDIST(.,.)$) entre la región $R \subseteq K_A$ y un punto de la región $S \subseteq K_B$. El algoritmo *isCandidate* recibe como parámetros una región de K_A para ser analizada, el nodo raíz de K_B, y la variable *cmax*, la que es utilizada para aplicar la regla de poda 4 ya descrita.

Inicialmente (línea 1, 2 y 3) se crea e inserta una entrada en la cola de prioridad *pQ*. Luego el algoritmo ejecuta un ciclo (líneas 4-21) procesando las entradas de *pQ* para determinar si la región $R \subseteq K_A$ ingresada como parámetro contiene puntos candidatos que permitan mejorar la variable *cmax*.

En línea 7, si se trata de un nodo hoja de K_B, es decir se ha llegado a un punto, quiere decir que se han escaneado todas las regiones de K_B siendo imposible aplicar la regla de poda 4. Por lo tanto, esa región R si contiene puntos candidatos que podrían optimizar *cmax*. En este caso el algoritmo retorna la distancia d obtenida entre la región $R \subseteq K_A$ y un punto $q \in K_B$ (línea 8).

Entre las líneas 10 y 22, se procesan las entradas *bb* de *pQ* correspondientes a nodos internos (no hojas) de K_B. En línea 15 se aplica la regla de poda 4, es decir si calculamos la mayor distancia entre ($MAXMAXDIST(.,.)$) de la región $R \subseteq K_A$ y una región $S \subseteq K_B$, y esta es menor que la variable *cmax*, sabemos que cualquier vecino más cercano de los puntos de la región $R \subseteq K_A$ serán menor a *cmax*, por ende ninguno de esos puntos mejorarán la variable *cmax*. En este caso retornamos -1, indicando que esa región $R \subseteq K_A$ debe ser podada. En caso que $MAXMAXDIST(.,.)$ sea mayor que *cmax*, se inserta la región R en *pQ*, para que siga siendo procesada (línea 18).

Algorithm 5 Algoritmo HDKPruning para calcular la distancia de Hausdorff sobre dos conjuntos de puntos almacenados en k^2-tree, descartando cuadrantes en K_A

HDKPruning($root_A$, $root_B$, $cmax$)
input: Nodo raíz de K_A, Nodo raíz de K_B, la Distancia de Hausdorff $cmax$ candidata.
output: El valor de $cmax$, la cual es la Distancia de Hausdorff.

```
1: pQ = CreateMax-Heap()
2: aa = getRegion(root_A)
3: d = 0;
4: Insert(pQ, ⟨aa, d⟩)
5: while (Not Empty(pQ)) do
6:     aa = Extract-Max(pQ)
7:     if isLeaf(aa.n) then
8:         p = aa.n
9:         nn = NNMax(root_B, r_B, p, cmax, ∞)
10:        if nn ≥ cmax then
11:            cmax = nn
12:        end if
13:    else
14:        for all Nodo h hijo de aa.n do
15:            if (h tiene hijos) then
16:                r = getRegion(h)
17:                if (isCandidate(r, B, cmax) ≠ −1) then
18:                    Insert(pQ, h)
19:                end if
20:            end if
21:        end for
22:    end if
23: end while
24: return cmax
```

Algorithm 6 (**IsCandidate**)Determina si el cuadrante del conjunto A tiene puntos candidatos para ser escaneados

IsCandidate($R, rootNodeB, cmax$)
input: Región a explorar de puntos de K_A, Nodo raíz de K_B, la Distancia de Hausdorff $cmax$ candidata.
output: La distancia $MAXMAXDIST(.,.)$ entre la región $R \subseteq K_A$ y una punto de la región $S \subseteq K_B$

```
1: pQ = CreateMin-Heap()
2: bb = getRegion(K_B)
3: d = MAXMAXDIST(R, S)
4: Insert(pQ, bb)
5: while (Not Empty(pQ)) do
6:     bb = Extract-Min(pQ)
7:     if isLeaf(bb.n) then
8:         return bb.d
9:     else
10:        for all Nodo h hijo de bb.n do
11:            if (hasChildren(h) then
12:                S = getRegion(h)
13:                maxDist = MAXMAXDIST(R, S)
14:                h.d = maxDist
15:                if (maxDist ≤ cmax) then                ▷ Regla de Poda 4
16:                    return −1
17:                end if
18:                Insert(pQ, h)
19:            end if
20:        end for
21:    end if
22: end while
```

Parte IV

Experimentos

Capítulo 4

Experimentación

Con el propósito de evaluar los algoritmos propuestos en esta tesis, HDKP1 y HDKP2, se realizaron una serie de experimentos. Uno de los objetivos de la experimentación fue comparar los resultados de nuestros algoritmos con, según nuestro conocimiento, el mejor algoritmo descrito en la literatura que calcula la Distancia de Hausdorff. Este algoritmo es el propuesto por [33] (que identificamos como Taha). Hacemos notar que para el algoritmo Taha es necesario, previamente, recuperar los puntos desde los k^2-trees en arreglos diferentes, y a partir de allí calcular la Distancia de Hausdorff. Esta situación es realista, pues al no existir un algoritmo que resuelva el problema de la Distancia de Hausdorff directamente sobre los k^2-tree, Taha es la opción más directa. Recordar que una forma básica de obtener la Distancia de Hausdorff es realizar una búsqueda exhaustiva de cada uno de los vecinos más cercanos de cada punto de B, con respecto a cada punto de A, para luego maximizar dichas soluciones.

4.1. Entorno de Pruebas

Los experimentos fueron realizados en un computador con procesador Intel(R) Xeon(R) CPU E3-1220 V2 @ 3.10GHz, Memoria RAM 23 GB y Sistema Operativo Linux Debian 3.2. En los experimentos medimos el rendimiento mediante el tiempo de ejecución.

El escenario para realizar los experimentos consideró que los conjuntos de puntos se encuentran almacenados en dos k^2-trees. El tiempo promedio calculado incluye el tiempo necesario para extraer los puntos desde los k^2-trees y el requerido para calcular la Distancia de Hausdorff. Los algoritmos fueron implementados en el lenguaje de programación C.

En la experimentación propia del algoritmo HDKP2, la estructura max-heap (línea 1, del algoritmo 5) fue reemplazada por un min-heap, por una pila y una fila. Esto fue realizado solo con fines de analizar el comportamiento del algoritmo al cambiar el orden en el cual se recorren las regiones de K_A.

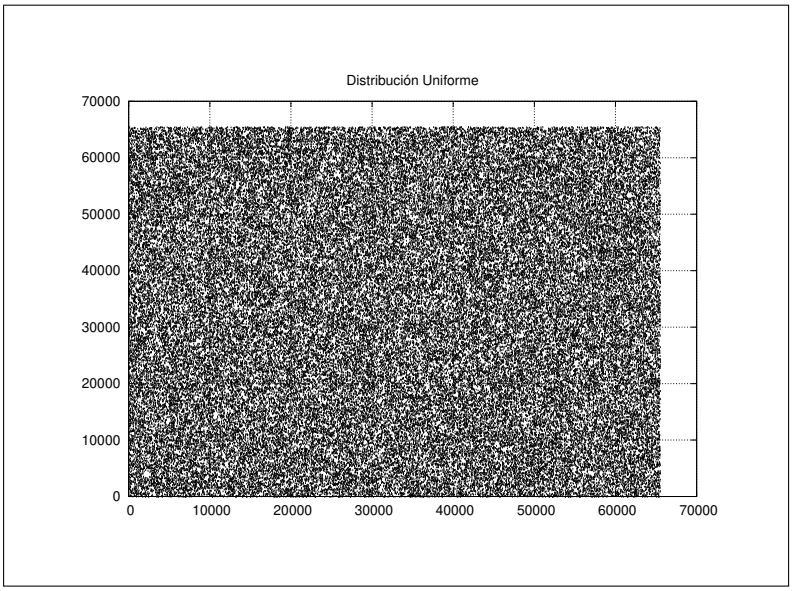


Figura 4.1: Distribución uniforme

4.2. Conjunto de datos

4.2.1. Datos Sintéticos

Se realizaron experimentos con puntos generados de manera aleatoria siguiendo las distribuciones uniforme y gaussiana. Además se generó un conjunto mixto de puntos bajo una distribución uniforme / gaussiana. Los puntos fueron generados dentro de un espacio 2D de rango $2^{16} \times 2^{16}$. El tamaño de los conjuntos fue de 10.000, 100.000, 1.000.000 y 10.000.000 de puntos. Cada distribución puede ser vista en su forma gráfica en: Uniforme (Figura 4.1), Gaussiana (Figura 4.2) y Mixta (Figura 4.3).

Para analizar el tiempo de ejecución, por cada prueba se generaron seis pares de conjuntos de datos y se calculó el tiempo promedio utilizado para calcular la Distancia de Hausdorff por cada algoritmo. Es importante señalar que debido al excesivo tiempo de ejecución del algoritmo de Taha, solo se contabilizaron tres pares de conjuntos de datos de 10 millones de puntos para el cálculo del promedio, esto bajo las distribuciones Uniforme y Gaussiana. Hacemos notar que el algoritmo de Taha para estas distribuciones con 10 millones de puntos, tomaba alrededor de 40 horas, para entregar un resultado por cada conjunto de datos.

4.2.2. Datos Reales

Para el conjunto de datos reales se utilizaron ocho conjuntos de datos que representan los enlaces que hay entre páginas web del Reino Unido. Estos enlaces son tomados como representaciones gráficas de sus coordenadas, esto con el fin de mostrar distribuciones distintas a las analizadas anteriormente. Un ejemplo práctico para el análisis de estos conjuntos de datos, es que se podría

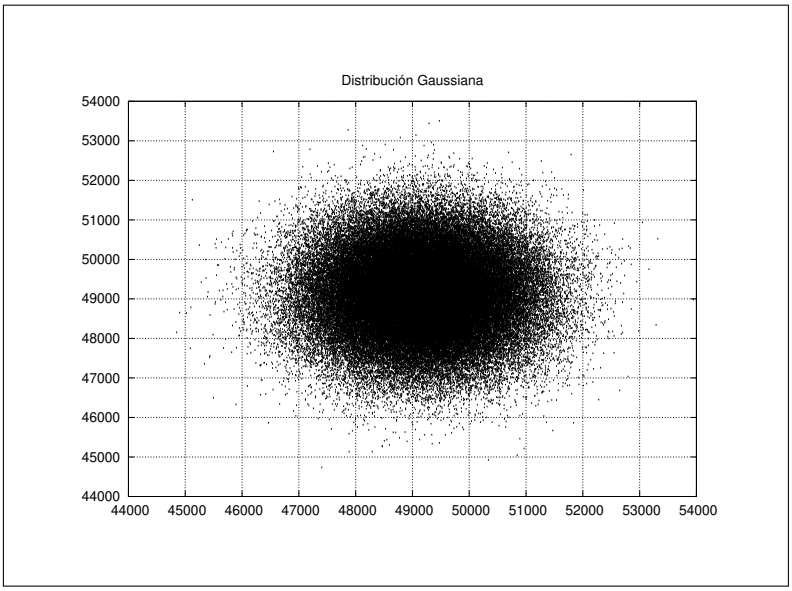


Figura 4.2: Distribución gaussiana

estudiar que tanto cambia la locación de estos enlaces a través de los años. Estos ocho conjuntos de datos fueron divididos en cinco pruebas. El tamaño de los conjuntos por cada prueba puede ser visto en la Tabla 4.1. Gráficamente los conjuntos de datos comparados por cada prueba, pueden ser revisados en: Prueba 1 (Figura 4.4), prueba 2 (Figura 4.5), prueba 3 (Figura 4.6), prueba 4 (Figura 4.7) y prueba 5 (Figura 4.8).

El objetivo de las pruebas 1, 2 y 3, es analizar como la distribución de los puntos afecta el rendimiento de los distintos algoritmos. La prueba 4 busca ver que sucede con el rendimiento de los algoritmos cuando el tamaño de los conjuntos A y B es distinto. La prueba 5 tiene como objetivo analizar el rendimiento cuando los conjuntos A y B son iguales, es decir la Distancia de Hausdorff es igual a 0. Para ello se compara el primer conjunto consigo mismo.

Conjunto de datos reales		
	Nº de puntos A	Nº de puntos B
Prueba 1	1048575	1048575
Prueba 2	1048575	1048575
Prueba 3	1048575	1048575
Prueba 4	772030	767234
Prueba 5	1048575	1048575

Tabla 4.1: Conjunto de Datos Reales

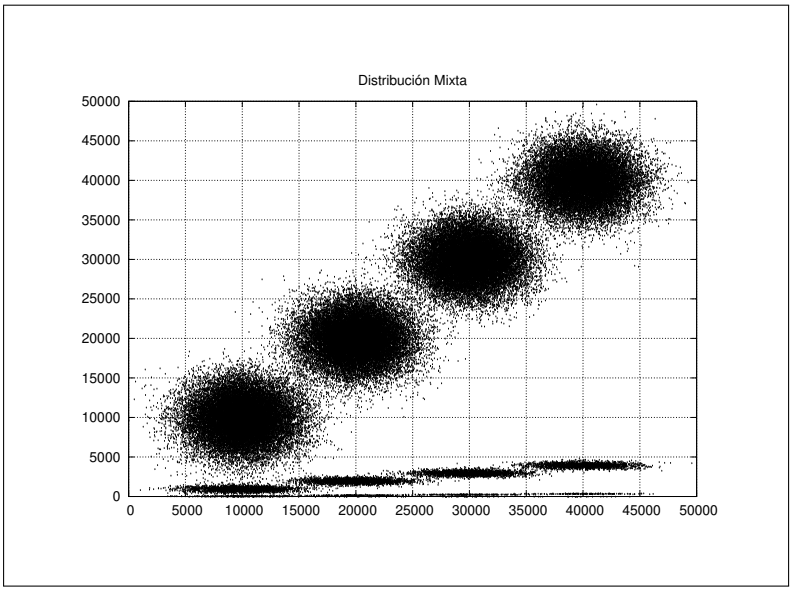


Figura 4.3: Distribución mixta

4.3. Resultados

4.3.1. Comparación de los algoritmos HDKP1, HDKP2 y Taha

En el primer experimento se comparó el tiempo de ejecución de los tres algoritmos, HDKP1, HDKP2 y Taha para el conjunto de datos tanto sintético como real.

En la Figura 4.9 (a) podemos ver el comportamiento de los tres algoritmos para los diferentes conjuntos de datos considerando distribución uniforme. Notar que cuando se trata de conjuntos pequeños de puntos, menos de 10.000 puntos, no existe mayor diferencia entre los algoritmos HDKP1, HDKP2 y Taha, siendo este ultimo el que presenta un mejor rendimiento. Sin embargo, a partir de los 100.000 puntos, se puede ver que los algoritmos HDKP1 y HDKP2 presentan un mejor

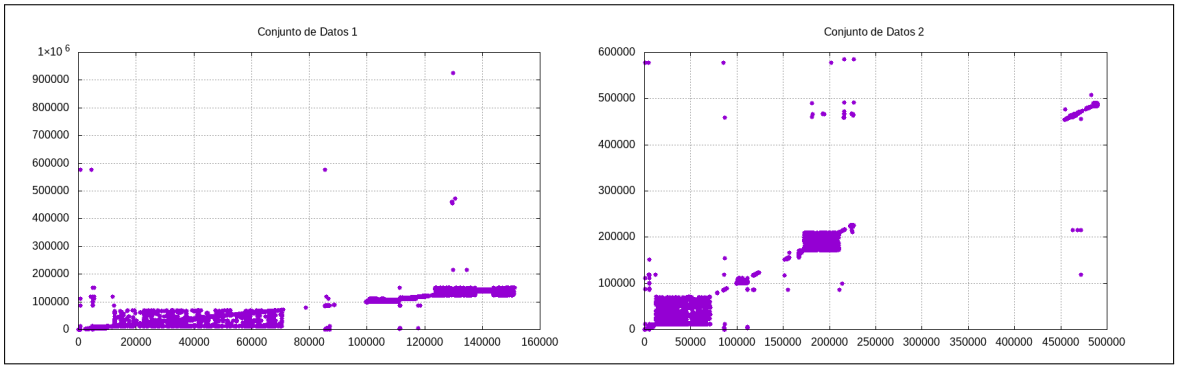


Figura 4.4: Prueba 1

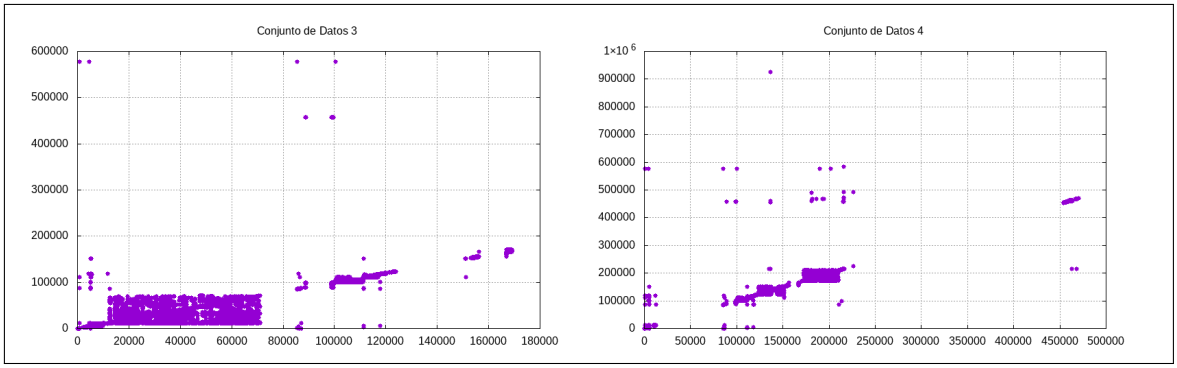


Figura 4.5: Prueba 2

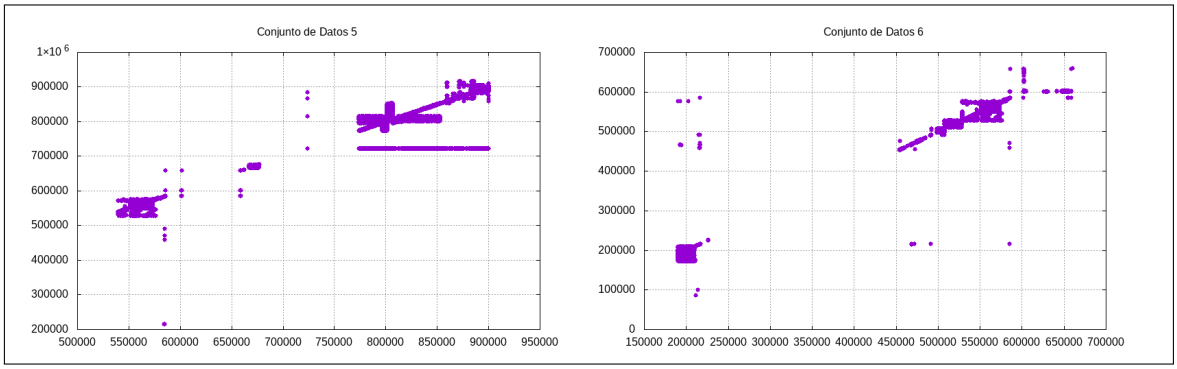


Figura 4.6: Prueba 3

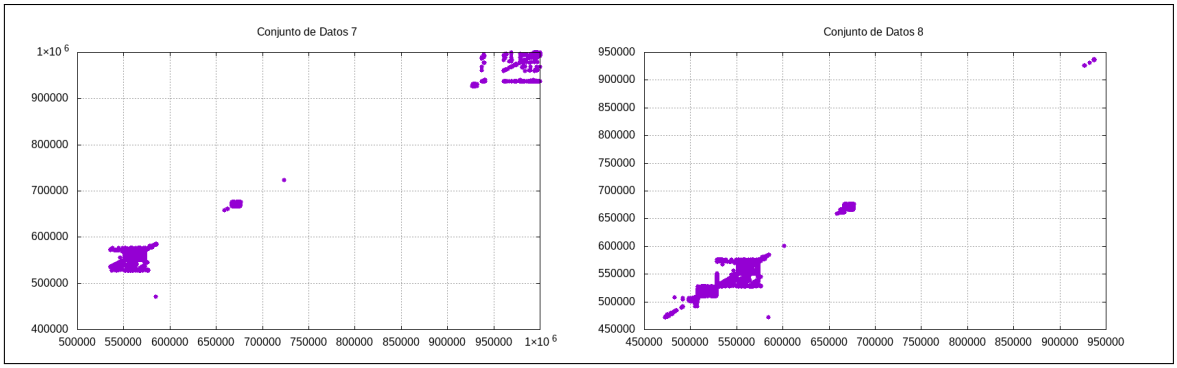


Figura 4.7: Prueba 4

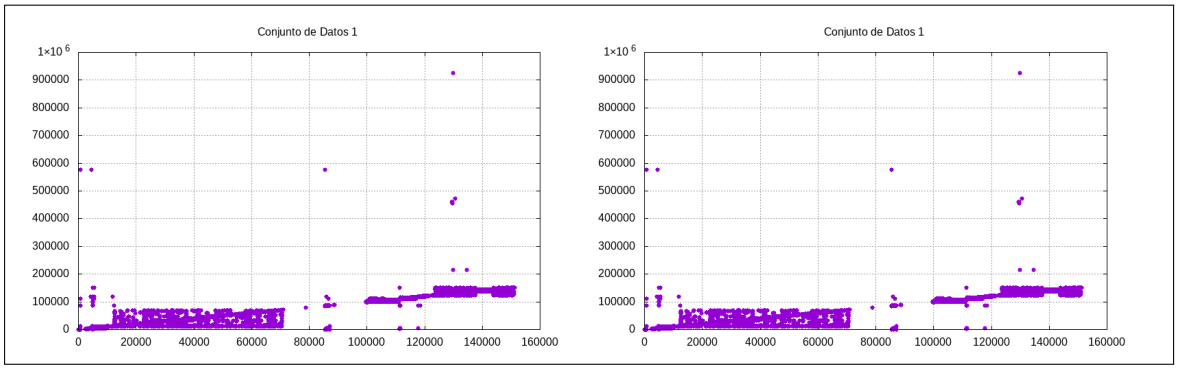


Figura 4.8: Prueba 5

rendimiento que el algoritmo de Taha, llegando a superarlo por tres órdenes de magnitud (ver Figura 4.9 (a)). Si comparamos el rendimiento de los algoritmos HDKP1 y HDKP2, podemos decir que tienen un rendimiento muy similar, siendo más eficiente el algoritmo HDKP1 en conjuntos de menos de 10 millones de puntos. A partir de los 10 millones el algoritmo HDKP2 tiene mejor rendimiento. Esto se debe a que el algoritmo HDKP2, como ya hemos señalado, con el fin de realizar podas en el primer conjunto debe calcular $MAXMAXDIST(.,.)$ entre regiones de ambos conjuntos, lo que le implica tener que calcular la distancia entre cada vértice de la región R del primer conjunto A, con respecto cada vértice de una región S del conjunto B. Notar que cada punto que se poda en el conjunto A, evita tener que calcular el vecino más cercano de dicho punto con respecto a todo el conjunto B. Por lo que la regla de poda 4 utilizada por el algoritmo HDKP2, es eficiente cuando el número de puntos podados en el primer conjunto A es mayor que el costo que le implica tener que calcular $MAXMAXDIST(.,.)$ entre regiones de ambos conjuntos. Al considerar conjuntos con distribución gaussiana (Figura 4.9 (b)) es posible apreciar una ventaja de HDKP2 con respecto a los algoritmos HDKP1 y Taha, aunque esta es menor que la que se produce en el escenario con distribución uniforme. Es posible ver en la Figura 4.9 (b) que a diferencia de lo que sucede cuando se trata de una distribución uniforme, a partir de los 10.000 puntos, HDKP2 supera a HDKP1 en un orden de magnitud. Esto sucede porque bajo una distribución gaussiana el conjunto de puntos se encuentra agrupado, lo cual permite que la regla de poda 4, utilizada por HDKP2, tenga un mayor impacto al permitirle podar un número mayor de puntos en el primer conjunto. Bajo una distribución mixta de puntos (ver Figura 4.9 (c)), el rendimiento del algoritmo HDKP2 es similar al obtenido bajo una distribución uniforme, superando por tres ordenes de magnitud a los algoritmos HDKP1 y Taha. Cabe señalar que bajo este tipo de distribución, el algoritmo de Taha tiene un mejor rendimiento que el algoritmo HDKP1.

Como se puede apreciar en Figura 4.10, cuando se trata de un conjunto de datos reales, el rendimiento de los distintos algoritmos depende de la distribución de los puntos. En las pruebas 1, 3 y 4, el algoritmo con mejor rendimiento es el algoritmo HDKP2. Esto se debe a que existe un mayor número de puntos agrupados en ambos conjuntos, tal como se puede apreciar gráficamente en las figuras 4.4, 4.6, 4.7. Mientras mayor dispersión exista en la distribución de puntos (ver Figura 4.6), el rendimiento de los tres algoritmos decae, como se puede apreciar en la prueba 2

(ver Figura 4.10). De igual forma, a mayor dispersión de los puntos peor rendimiento tiene el algoritmo de Taha, esto ya ha sido mostrado en los experimentos bajo una distribución uniforme (Figura 4.10). Otro detalle interesante es que el algoritmo HDKP1 es el de peor rendimiento en las pruebas 1,3 y 4, sin embargo en la prueba 2, presenta el mejor rendimiento, lo cual demuestra como la distribución de los puntos afecta el rendimiento de los algoritmos. En la prueba 5, cuando los conjuntos de datos son iguales, es decir la Distancia de Hausdorff es igual a 0, el algoritmo con mejor rendimiento es HDKP1. En este escenario el rendimiento del algoritmo de Taha decae, esto se debe a que el *"rompimiento temprano"* que utiliza este algoritmo es ineficiente al no poder mejorar de forma rápida el valor de *cmax* (ver Algorithm 1, línea 10), teniendo eventualmente que escanear una gran cantidad de puntos antes de realizar este *"rompimiento temprano"*. Hacemos notar que *cmax*, cuando se trata de conjuntos iguales de datos, siempre será 0.

4.3.2. Impacto Poda 2, en algoritmo HDKP1

Con el propósito de evaluar el efecto de las reglas de poda en nuestro algoritmo HDKP1 (ver Sección 3.2.2) se realizó una serie de experimentos en dos escenarios. El primero (reglas de poda 1, 2 y 3) consistió en ejecutar nuestro algoritmo incluyendo todas las reglas de poda; y el segundo (reglas de poda 1 y 3) consistió en ejecutar HDKP1 sin la regla 2. En Figura 4.11 podemos ver el comportamiento de HDKP1 en cada escenario para los diferentes tamaños de conjuntos. Podemos notar que el uso de la regla de poda 2 tiene un importante impacto en el tiempo de ejecución ya sea se trate de datos sintéticos o reales. Ahora bien, entre las distintas distribuciones de datos sintéticos el mayor impacto de la regla de poda 2 ocurre bajo una distribución mixta. Recordar que la regla de poda 2 es un símil a la técnica de *"rompimiento temprano"* que utiliza el algoritmo de Taha [33].

Para datos reales, como se puede apreciar en Figura 4.12, proporcionalmente el impacto de la regla de poda 2 siempre es similar, teniendo un menor impacto en la prueba 5. Como se ha señalado, la regla de poda 2 es un símil al *"rompimiento temprano"* utilizado por el algoritmo Taha. Esta regla de poda 2 no es eficiente cuando no le es posible mejorar rápidamente el valor de *cmax*. Recordar que el valor de *cmax* siempre es una Distancia de Hausdorff valida, la cual siempre tiene valor 0 cuando se trata de conjuntos de datos iguales, haciendo imposible que el valor de esta variable pueda ser maximizada.

4.3.3. Evaluación estrategia de exploración en HDKP2

Otro experimento realizado fue comparar el rendimiento del algoritmo HDKP2 en distintas implementaciones. Para esto se cambió la estructura max-heap (línea 1, del algoritmo 5) reemplazándola por un min-heap, por una pila y una fila. Esto se realizó con el objetivo de analizar el comportamiento del algoritmo al cambiar el orden de como se recorren las regiones de K_A. Hacemos notar que cuando se utiliza un max-heap o min-heap, K_A se recorre por un candidato prometedor. Cuando se utiliza una pila, K_A es recorrido en profundidad, y cuando se utiliza una fila, K_A es revisado en amplitud. Recordar que esta estructura es utilizada para saber que región debe ser podada en el primer conjunto K_A. Como se puede ver en la Figura 4.13 el peor rendimiento siempre se obtiene al utilizar una fila, independiente del tipo de distribución o si se trata de datos reales o sintéticos. Cuando se trata de una distribución uniforme (ver Figura 4.13

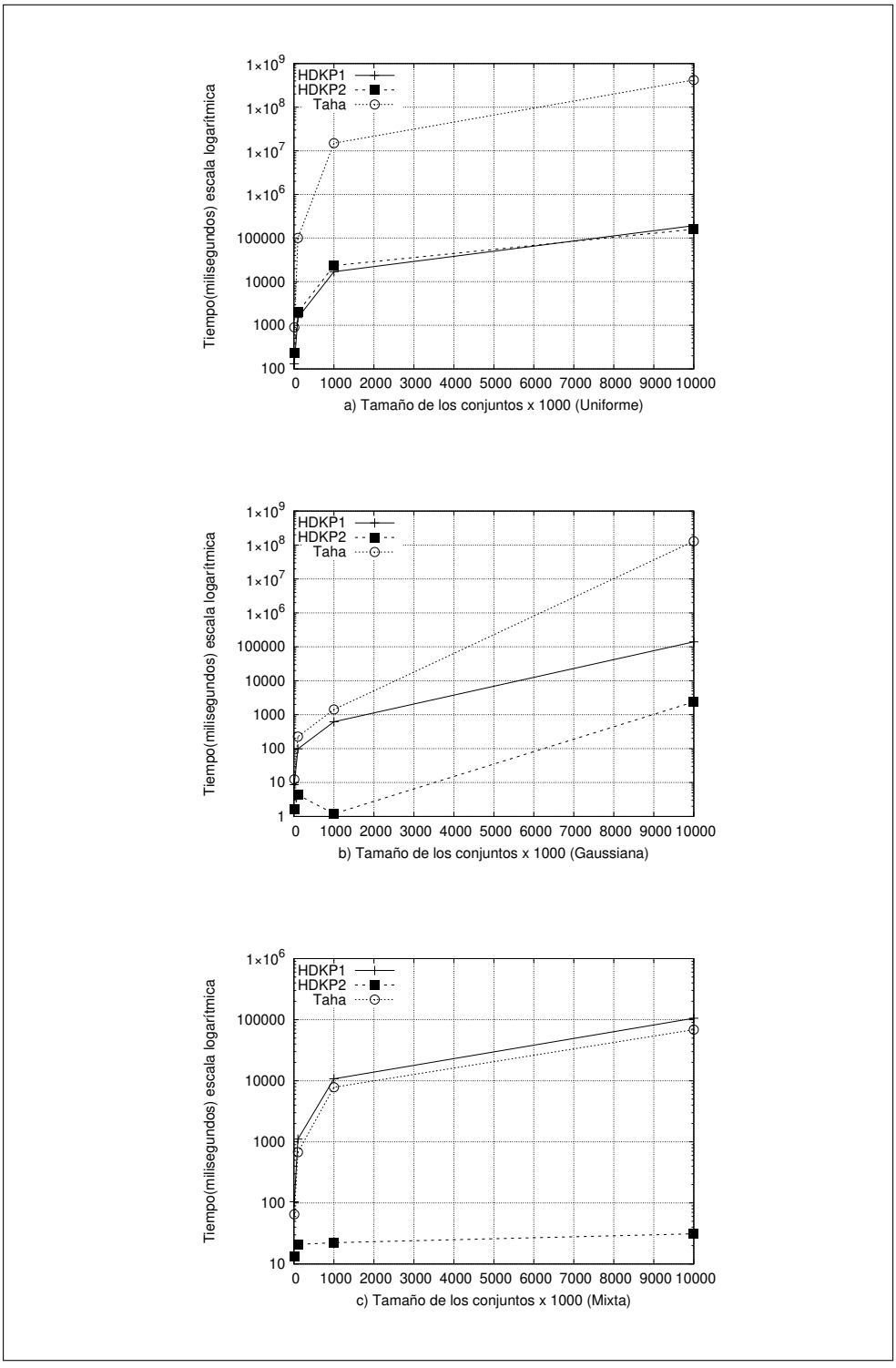


Figura 4.9: Tiempos de ejecución de los tres algoritmos para conjuntos de datos sintéticos

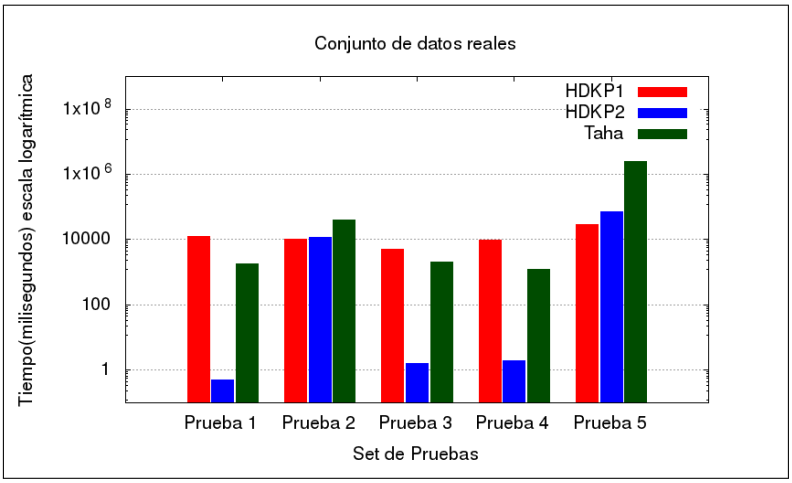


Figura 4.10: Tiempos de ejecución de los tres algoritmos para datos reales

(a)), la implementación con un mejor rendimiento es la que utiliza un min-heap, pero sin mayor diferencia con respecto a la implementación de HDKP2 que usa una pila. Un rendimiento menor presenta la implementación que utiliza un max-heap, pero no hay mayor diferencia. Cuando se trata de una distribución gaussiana (ver Figura 4.13 (b)), la implementación con un mejor rendimiento es la que utiliza un max-heap. Esta supera por un orden de magnitud que al utilizar una pila, y en alrededor de dos órdenes de magnitud que al usar un min-heap. Cuando se trata de una distribución mixta de datos (ver Figura 4.13 (c)), la implementación con mejor rendimiento es la que utiliza una pila. Con un rendimiento un tanto menor está el uso de un max-heap. En esta distribución el uso de min-heap tiene un rendimiento menor en casi tres órdenes de magnitud con respecto a la utilización de un max-heap o una pila.

Cuando se trata de datos reales, como se ha señalado, el uso de una fila es aquella con peor rendimiento. El uso de un max-heap o una pila tienen, con pequeñas diferencias, siempre el mejor rendimiento. El uso de un min-heap aumenta el tiempo de ejecución, siendo en la prueba 3 cuando presenta un peor rendimiento con respecto al uso de un max-heap o de una pila. Como se ha analizado anteriormente, cuando la dispersión de puntos aumenta, el rendimiento del algoritmo HDKP2 decae, sin importar la implementación que se este utilizando. En la prueba 5, el rendimiento de las distintas implementaciones de HDKP2 decae. Como se ha señalado, el valor de *cmax* cuando ambos conjuntos son iguales siempre es 0, esto impide que el resultado de calcular $MAXMAXDIST(.,.)$ sea superior a *cmax* (ver Algorithm 6, línea 15), lo cual impide podar regiones de puntos en el conjunto A.

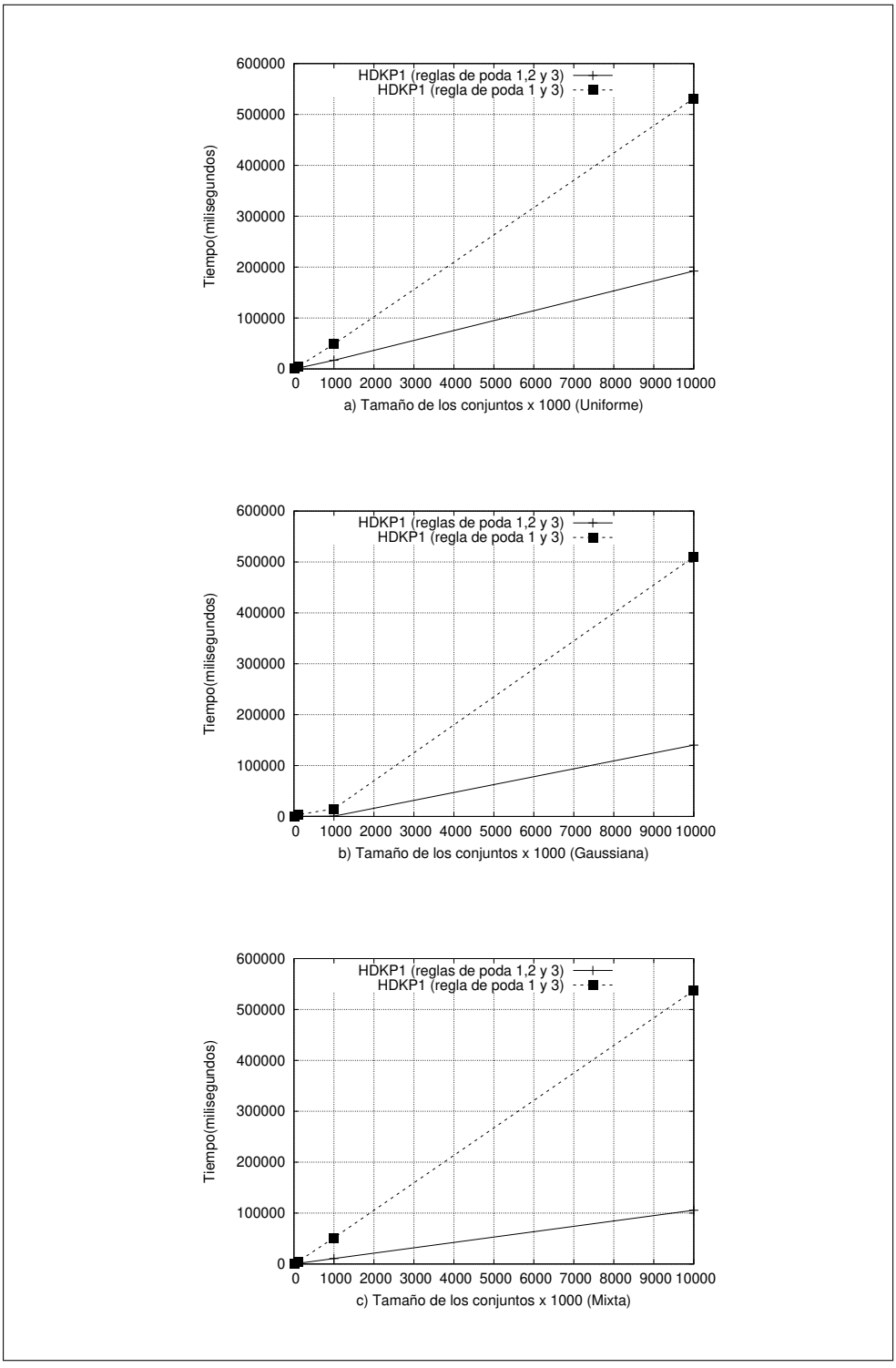


Figura 4.11: Tiempo de ejecución de HDKP1 con y sin regla de poda 2 con datos sintéticos

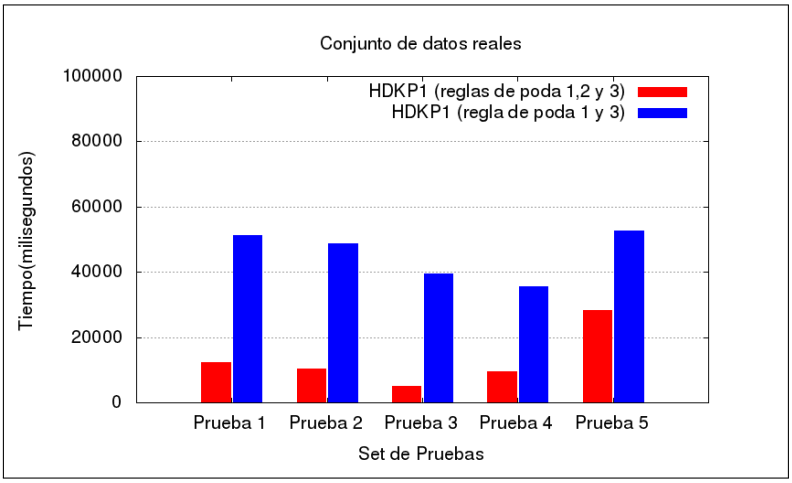
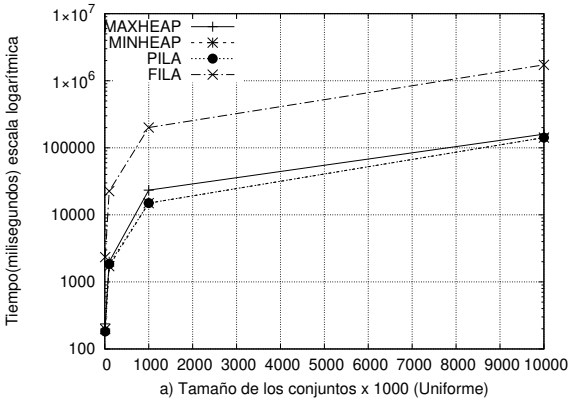
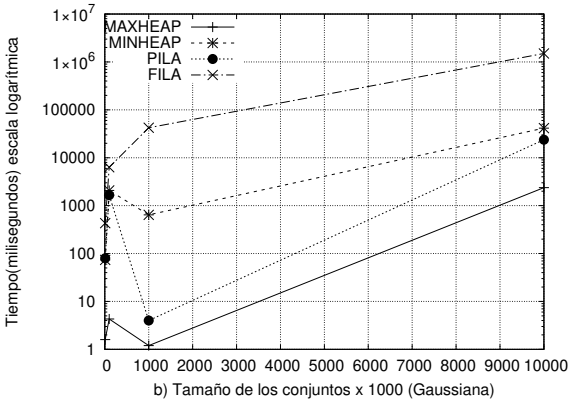


Figura 4.12: Tiempo de ejecución de HDKP1 con y sin regla de poda 2 con datos reales





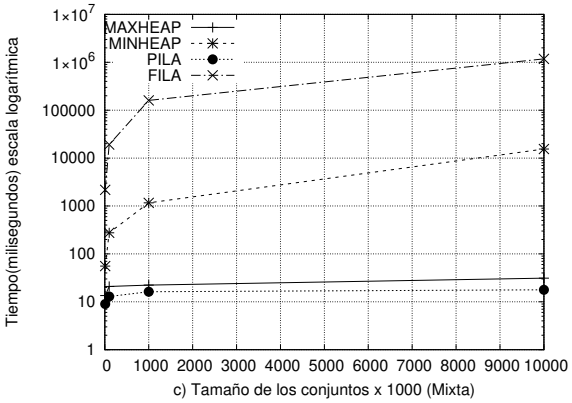


Figura 4.13: Tiempo de ejecución de HDKP2 en sus distintas estrategias de exploración con datos sintéticos

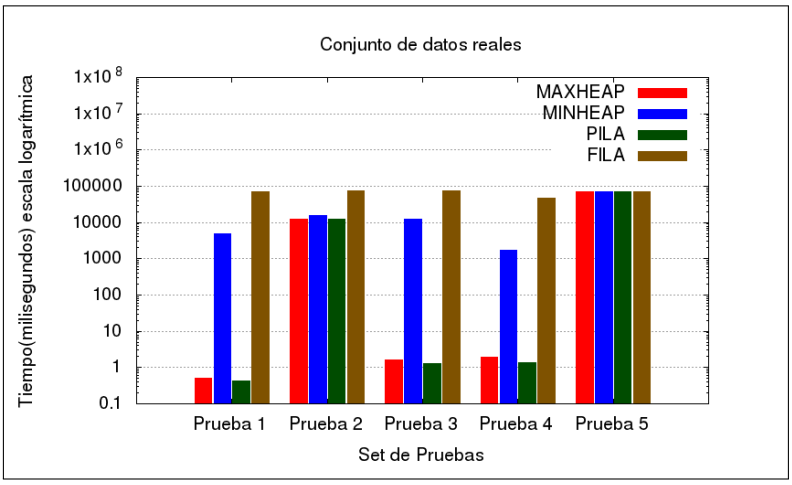


Figura 4.14: Tiempo de ejecución de HDKP2 en sus distintas estrategias de exploración con datos reales

Parte V

Conclusiones y Trabajo Futuro

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

En esta tesis se presentan dos nuevos algoritmos (HDKP1 y HDKP2) para calcular la Distancia de Hausdorff los cuales asumen que cada conjunto de puntos se encuentra representado en la estructura de datos compacta k^2-tree. Para calcular la Distancia de Hausdorff $\mathsf{HausDist}(A, B)$ el algoritmo HDKP1 por cada punto de A, eventualmente, calcula la distancia de su vecino más cercano en B. Para evitar procesar todos los puntos de B el algoritmo usa tres reglas de poda las que permiten reducir el tiempo de ejecución. Por su parte el algoritmo HDKP2, utiliza una nueva regla de poda para evitar tener que escanear todos los puntos en A. Además utiliza el algoritmo NNMax (ver Algoritmo 3) para evitar escanear todos los puntos en B. De acuerdo a la revisión del estado del arte y a nuestro propio conocimiento se puede concluir que actualmente no existe en la literatura un algoritmo para calcular la Distancia de Hausdorff considerando que los puntos están almacenados en un k^2-tree.

Con el propósito de evaluar el rendimiento de HDKP1 y HDKP2 se realizaron una serie de experimentos, los que consideraron datos reales, además de datos sintéticos con distribución uniforme, gaussiana y una distribución mixta basada en datos generados de forma gaussiana / uniforme. Nuestros algoritmos se compararon con el algoritmo de Taha [33] (Taha). Los resultados experimentales permiten concluir que, a partir de 10.000 puntos, tanto el algoritmo HDKP1 y HDKP2 superan al algoritmo de Taha en tiempo de ejecución entre 1 y 3 órdenes de magnitud, esto cuando se trata de distribución gaussiana o uniforme. Esta diferencia en el rendimiento del algoritmo está determinada por la distribución de los puntos, obteniéndose una mayor diferencia en el tiempo de ejecución con respecto al algoritmo de Taha bajo una distribución uniforme. Cuando se trata de una distribución mixta, nuestro algoritmo HDKP2, tiene un mejor rendimiento en tres órdenes de magnitud con respecto a los algoritmos HDKP1 y Taha. Al tratarse de datos reales, el algortimo HDKP2, tiene mejor rendimiento en las pruebas 1, 3 y 4. En las pruebas 2 y 5 el algoritmo de mejor rendimiento es HDPK1. Esto demuestra que la distribución de los puntos es determinante en el tiempo de ejecución de los distintos algoritmos.

También se evaluó el efecto de la regla de poda 2 sobre el tiempo de ejecución del HDKP1. Los experimentos mostraron que tal regla permite reducir el tiempo de ejecución de HDKP1 en cerca de un 0,6 % para distribución uniforme, gaussiana y mixta, para conjuntos de tamaño por

sobre el millón de puntos. Cuando se trata de datos reales, al analizar los resultados de la prueba 5, se puede concluir que la regla de poda 2 es ineficiente cuando de trata de conjuntos iguales, esto se da por la imposibilidad de maximizar el valor de la variable $cmax$, lo que impide realizar podas eficientes en el conjunto B. Recordar que $cmax$ siempre tiene como valor una Distancia de Hausdorff candidata válida, la cual cuando ambos conjuntos iguales siempre tiene valor 0.

Otro experimento realizado consistió en medir distintas estrategias de exploración del algoritmo HDKP2 con respecto a como se recorre el conjunto A. Como se puede apreciar en dicho experimento, la distribución de los puntos y la estrategia de exploración afecta el rendimiento del algoritmo HDKP2. En esta tesis se decidió usar la implementación de un max-heap, el cual si bien es cierto en algunas distribuciones de puntos presenta un menor rendimiento, si se mantiene estable en los distintos escenarios probados. Destacar que bajo una distribución gaussiana el uso de un max-heap le da un margen importante en el tiempo de ejecución con respecto al uso de las otras implementaciones. Cuando se trata de datos reales, el rendimiento de HDKP2 se mantiene favorable en la mayoría de los escenarios, teniendo el peor rendimiento en la prueba 5, cuando se trata de conjuntos de datos iguales. Como se ha señalado, esto sucede porque la regla de poda 4, al igual que la regla de poda 2, es ineficiente ante la imposibilidad de poder maximizar la variable $cmax$, la cual siempre tiene valor 0 al tratarse de conjuntos iguales.

Dados los experimentos realizados, se puede apreciar que existen dos grandes factores que afectan el rendimiento de los distintos algoritmos. Uno de ellos es la distribución de los puntos, mientras más agrupados estén los puntos mejor es el rendimiento en términos generales de los tres algoritmos. A medida que los puntos se van dispersando, el rendimiento de los tres algoritmos decae, esto básicamente ocurre porque el impacto de la poda 2 no es eficiente al tener datos dispersos. Recordar que la poda 2, es un símil al *"rompimiento temprano"* que utiliza el algoritmo de Taha. Otro factor que afecta el rendimiento de los algoritmos es el número de puntos, cuando se trata de pocos puntos (menos de 10 mil puntos) el algoritmo de Taha tiene un rendimiento superior al obtenido por HDKP1 y HDKP2. Esto sucede porque al no existir una posibilidad de podar un gran número de puntos, el cálculo de distancia que utilizan la reglas de podas usadas por los algoritmos presentados en esta tesis, es mayor que el costo del cálculo de distancia que se evita al tener que calcular el vecino más cercano de los puntos podados.

Como conclusión final decir que en términos generales el algoritmo HDPK2 tiene un mejor rendimiento que el algoritmo de Taha a partir de los 10 mil puntos, sin importar la distribución de estos. Ahora bien, el mayor problema que presenta el algoritmo HDKP2, es que su rendimiento decae cuando debe calcular $MAXMAXDIST(.,.)$ repetidamente sin lograr descartar un gran número de puntos en el primer conjunto A.

5.2. Trabajo Futuro

Como se ha planteado en esta Tesis, el cálculo de Distancia de Hausdorff tiene un amplio espectro de usos, que van desde la comparación de imágenes hasta la comparación de trayectorias. Por otro lado, el aumento de los flujos de información presenta un gran desafío a la hora de utilizar estructuras que disminuyan el uso en el almacenamiento de memoria principal, sin perder capacidad de procesamiento. Dicho esto, aún existe trabajo a desarrollar en este ámbito, esto con el fin de obtener soluciones más eficientes tanto en tiempo como en espacio. Como trabajo futuro

se plantea lo siguiente:

- Según lo observado en la fase de experimentación, el desafío para mejorar los tiempos de procesos esta determinado en evitar escanear el mayor número de puntos, y por ende evitar el cálculo de distancias entre los puntos. Por esto se propone mejorar las métricas utilizadas en los procesos de poda de los algoritmos, con el fin de evitar escanear un mayor número de puntos en ambos conjuntos de puntos A y B.

- Otro punto interesante es que los algoritmos propuestos no realizan un descenso coordinado en ambos k^2-trees, es decir, para obtener el vecino más cercano de un punto de K_A necesariamente se debe llegar a una hoja de este k^2-tree. Una idea en este sentido es que se pueda realizar una revisión y poda balanceada, recorriendo coordinadamente ambos k^2-trees. Una idea similar es presentada en [26].

- Como se ha analizado en esta Tesis, la organización de los puntos a escanear tiene impacto en el tiempo de proceso para la obtención de una solución. Es por esto, que se propone utilizar otras estructuras de datos compactas que permitan realizar un escaneo más eficiente de los datos en un ámbito determinado, como por ejemplo la comparación de imágenes. En este campo específico se propone utilizar una versión especial de k^2-tree que utiliza una compresión de 1, con el fin de conocer si se puede obtener soluciones más eficientes, tanto en tiempo como en espacio.

- Diseñar un algoritmo que utilice la estructura k^2-tree para calcular la Distancia de Hausdorff simétrica. Recordar que para calcular $\mathsf{SymHausDist}(A, B)$ es necesario maximizar la solución entre el cálculo de $\mathsf{HausDist}(A, B)$ y el cálculo de $\mathsf{HausDist}(B, A)$. Este nuevo algoritmo debe contemplar una solución que evite escanear todos los puntos y el cálculo de ambas distancias.

- Diseñar un algoritmo que calcule $\mathsf{HausDist}(A, B)$, pero bajo conjuntos de k^2-tree que puedan utilizar técnicas de traslación, siguiendo las ideas presentadas en el algoritmo en [15]. Esto seria útil en el campo de comparación de imágenes, donde es necesario hacer que las imágenes calcen con el fin de establecer el mayor grado de similitud.

Bibliografía

[1] Alt, H., Behrends, B., Blömer, J.: Approximate matching of polygonal shapes. In: Proceedings of the seventh annual symposium on Computational geometry. pp. 186–193. ACM (1991)

[2] Atallah, M.J.: A linear time algorithm for the hausdorff distance between convex polygons. Information processing letters 17(4), 207–209 (1983)

[3] Bartoň, M., Hanniel, I., Elber, G., Kim, M.S.: Precise hausdorff distance computation between polygonal meshes. Computer Aided Geometric Design 27(8), 580–591 (2010)

[4] Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The r*-tree: an efficient and robust access method for points and rectangles. In: ACM Sigmod Record. vol. 19, pp. 322–331. Acm (1990)

[5] de Bernardo Roca, G.: New data structures and algorithms for the efficient management of large spatial datasets. Ph.D. thesis, Universidade da Coruna (2014)

[6] Brisaboa, N.R., Ladra, S., Navarro, G.: Compact representation of web graphs with extended functionality. Information Systems pp. 152–174 (2014)

[7] Brisaboa, N.R., Ladra, S., Navarro, G.: k2-trees for compact web graph representation. In: International Symposium on String Processing and Information Retrieval. pp. 18–30. Springer (2009)

[8] Chang, F., Chen, Z., Wang, W., Wang, L.: The hausdorff distance template matching algorithm based on kalman filter for target tracking. In: Automation and Logistics, 2009. ICAL'09. IEEE International Conference on. pp. 836–840. IEEE (2009)

[9] Chen, X.D., Ma, W., Xu, G., Paul, J.C.: Computing the hausdorff distance between two b-spline curves. Computer-Aided Design 42(12), 1197–1206 (2010)

[10] Chen, Y., He, F., Wu, Y., Hou, N.: A local start search algorithm to compute exact hausdorff distance for arbitrary point sets. Pattern Recognition 67, 139–148 (2017)

[11] Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M.: Closest pair queries in spatial databases. SIGMOD Rec. 29(2), 189–200 (May 2000), `http://doi.acm.org/10.1145/335191.335414`

[12] Gao, Y.: Efficiently comparing face images using a modified hausdorff distance. IEE Proceedings-Vision, Image and Signal Processing 150(6), 346–350 (2003)

[13] Grossi, R., Gupta, A., Vitter, J.S.: High-order entropy-compressed text indexes. In: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms. pp. 841–850. Society for Industrial and Applied Mathematics (2003)

[14] Guthe, M., Borodin, P., Klein, R.: Fast and accurate hausdorff distance calculation between meshes (2005)

[15] Huttenlocher, D.P., Kedem, K.: Computing the minimum hausdorff distance for point sets under translation. In: Proceedings of the sixth annual symposium on Computational geometry. pp. 340–349. ACM (1990)

[16] Jesorsky, O., Kirchberg, K.J., Frischholz, R.W.: Robust face detection using the hausdorff distance. In: International Conference on Audio-and Video-Based Biometric Person Authentication. pp. 90–95. Springer (2001)

[17] Kalman, R.E.: A new approach to linear filtering and prediction problems. Journal of basic Engineering 82(1), 35–45 (1960)

[18] Kang, Y., Kyung, M.H., Yoon, S.H., Kim, M.S.: Fast and robust hausdorff distance computation from triangle mesh to quad mesh in near-zero cases. Computer Aided Geometric Design 62, 91–103 (2018)

[19] Kelley, J.L.: General topology. Courier Dover Publications (2017)

[20] Ladra González, S.: Algorithms and compressed data structures for information retrieval. Ph.D. thesis, Universidade da Coruna (2011)

[21] Li, B., Shen, Y., Li, B.: A new algorithm for computing the minimum hausdorff distance between two point sets on a line under translation. Information Processing Letters 106(2), 52–58 (2008)

[22] Meagher, D.: Geometric modeling using octree encoding. Computer graphics and image processing 19(2), 129–147 (1982)

[23] Morton, G.M.: A computer oriented geodetic data base and a new technique in file sequencing. Tech. rep., IBM Ltd, Ottawa, Canada (1966)

[24] Navarro, G.: Wavelet trees for all. In: Annual Symposium on Combinatorial Pattern Matching. pp. 2–26. Springer (2012)

[25] Navarro, G.: Compact Data Structures: A Practical Approach. Cambridge University Press, New York, NY, USA, 1st edn. (2016)

[26] Nutanong, S., Jacox, E.H., Samet, H.: An incremental hausdorff distance calculation algorithm. Proceedings of the VLDB Endowment 4(8), 506–517 (2011)

[27] Quijada-Fuentes, C., Penabad, M.R., Ladra, S., Gutiérrez, G.: Set operations over compressed binary relations. Information Systems 80, 76–90 (2019)

[28] Romero, M.: Un índice espacio temporal para puntos móviles basado en estructuras de datos compactas. Ph.D. thesis, Universidade da Coruña (2017), http://hdl.handle.net/2183/19303

[29] Rote, G.: Computing the minimum hausdorff distance between two point sets on a line under translation. Information Processing Letters 38(3), 123–127 (1991)

[30] Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. SIGMOD Rec. 24(2), 71–79 (May 1995), http://doi.acm.org/10.1145/568271.223794

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

[32] Spyridonos, P., Gaitanis, G., Bassukas, I.D., Tzaphlidou, M.: Gray hausdorff distance measure for medical image comparison in dermatology: Evaluation of treatment effectiveness by image similarity. Skin Research and Technology 19(1) (2013)

[33] Taha, A.A., Hanbury, A.: An efficient algorithm for calculating the exact hausdorff distance. IEEE transactions on pattern analysis and machine intelligence 37(11), 2153–2163 (2015)

[34] Takacs, B.: Comparing face images using the modified hausdorff distance. Pattern recognition 31(12), 1873–1881 (1998)

[35] Tang, M., Lee, M., Kim, Y.J.: Interactive hausdorff distance computation for general polygonal models. ACM Transactions on Graphics (TOG) 28(3), 74 (2009)

[36] Trajcevski, G., Ding, H., Scheuermann, P., Tamassia, R., Vaccaro, D.: Dynamics-aware similarity of moving objects trajectories. In: Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems. p. 11. ACM (2007)

[37] Zhang, D., Zou, L., Chen, Y., He, F.: Efficient and accurate hausdorff distance computation based on diffusion search. IEEE Access 6, 1350–1361 (2018)