UNIVERSIDAD NACIONAL DEL ALTIPLANO

Facultad de Ingeniería Mecánica Eléctrica, Electrónica y Sistemas

E.P. de Ingeniería de Sistemas

ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE ESTRUCTURAS DE DATOS MÉTRICAS PARA LA BÚSQUEDA POR SIMILITUD EN ESPACIOS DISCRETOS Y CONTINUOS

MONOGRAFÍA

SIS220 - Estructuras de Datos Avanzadas

Presentado por:
Brayan Luis Calderon Calderon
Código: 232036

Docente:
Ing. Miguel Romilio Aceituno Rojo

Fecha de entrega:
26 de febrero de 2026

PUNO - PERÚ
2026

Índice general

Capítulo 1

Introducción

1.1 Contextualización de las Estructuras de Datos Avanzadas

En la era del Big Data, la naturaleza de la información ha mutado de datos estructurados simples (enteros, cadenas cortas) a objetos complejos de alta dimensionalidad, como vectores de características de imágenes, secuencias genómicas y nubes de puntos 3D. Las estructuras de datos clásicas, como los árboles binarios de búsqueda (BST) o las tablas hash, fallan ante estos datos porque dependen de un orden total o de una función de hash que no preserva la noción de cercanía.

Las Estructuras de Datos Métricas surgen como la solución especializada para organizar objetos donde lo único que conocemos es la distancia entre ellos.

1.2 El problema de la búsqueda por similitud (Nearest Neighbor Search)

El núcleo de esta investigación es el *Nearest Neighbor Search* (NNS). Formalmente, dado un conjunto de objetos S en un espacio métrico y una consulta q, el objetivo es encontrar el objeto $u \in S$ tal que la distancia $d(q, u)$ sea mínima.

- **Búsqueda por Rango:** Encontrar todos los objetos a una distancia r de q.
- **K-NN:** Encontrar los k objetos más cercanos.

El reto no es solo encontrar el resultado, sino hacerlo evitando la búsqueda exhaustiva ($O(n)$), la cual es computacionalmente prohibitiva en conjuntos de datos de escala masiva.

1.3 La "maldición de la dimensionalidad" en espacios métricos

A medida que aumenta la dimensionalidad de los datos, el volumen del espacio crece exponencialmente y los datos se vuelven dispersos. En dimensiones altas, la diferencia entre la distancia al vecino más cercano y al más lejano tiende a cero, lo que invalida muchas estrategias de particionamiento espacial.

Esta monografía analiza cómo estructuras como el VPT y el M-Tree intentan mitigar este efecto mediante el uso eficiente de la desigualdad triangular para podar o descartar regiones del espacio que no contienen resultados relevantes, reduciendo drásticamente el número de cálculos de distancia.

1.4 Objetivos de la monografía

El presente trabajo se propone:

- Analizar los fundamentos matemáticos que permiten la búsqueda en espacios métricos.
- Explorar el funcionamiento del Burkhard-Keller Tree (BKT) como referente para distancias discretas.
- Evaluar las estructuras de puntos de ventaja (VPT, MVPT, VPF) y árboles métricos jerárquicos (MST, MT) para entornos de datos continuos.
- Establecer criterios técnicos para la selección de la estructura óptima según el tipo de dato y la métrica de distancia empleada.

Capítulo 2

Fundamentos de Espacios Métricos

El éxito de las estructuras como BKT o VPT no reside en el orden de los elementos (como en un AVL), sino en las propiedades geométricas del espacio donde habitan los datos.

2.1 Definición formal de espacio métrico

Un Espacio Métrico es un par ordenado $(\mathbb{X}, d)$, donde $\mathbb{X}$ es un conjunto no vacío (el universo de nuestros datos) y d es una función de distancia (métrica) $d : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ que asigna un valor real a cualquier par de objetos.

Para que d sea considerada una métrica válida, debe satisfacer cuatro axiomas fundamentales para todo $x, y, z \in \mathbb{X}$:

- **No negatividad:** $d(x, y) \geq 0$.
- **Identidad de los indiscernibles:** $d(x, y) = 0 \iff x = y$.
- **Simetría:** $d(x, y) = d(y, x)$.
- **Desigualdad Triangular:** $d(x, z) \leq d(x, y) + d(y, z)$.

2.2 La desigualdad triangular: el motor de la poda

Este es el concepto más crítico de la monografía. En estructuras de datos, usamos esta propiedad para descartar regiones enteras del espacio sin calcular distancias reales.

Principio de poda

Si conocemos la distancia entre una consulta q y un punto de referencia (pivote) p, y conocemos la distancia entre p y un objeto almacenado u, podemos establecer límites inferiores y superiores para la distancia $d(q, u)$ sin calcularla directamente:

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

Si el rango de búsqueda de nuestra consulta no se solapa con estos límites, el objeto u (y, potencialmente, todo el subárbol que cuelga de él) puede ser ignorado.

2.3 Clasificación de espacios de búsqueda

Dependiendo de la naturaleza de $\mathbb{X}$ y d, las estructuras de datos se dividen en dos grandes grupos:

Espacios discretos

Aquí, el rango de la función de distancia es un conjunto de valores enteros o finitos (ej. $\{0, 1, 2, \dots\}$).

Ejemplo clásico: La Distancia de Levenshtein (o de edición), que cuenta el número mínimo de operaciones para transformar una palabra en otra.

Estructura ideal: BKT (Burkhard-Keller Tree), que utiliza estas distancias discretas para crear ramas específicas para cada valor entero.

Espacios continuos

El rango de la distancia es $\mathbb{R}^+$ y los datos suelen ser vectores en $\mathbb{R}^n$.

Ejemplo clásico: Distancia Euclidiana (L_2) o Manhattan (L_1).

Estructura ideal: VPT (Vantage Point Tree), que debe usar umbrales (radios) para dividir el espacio en dentro de la bola o fuera de la bola, ya que no existen valores discretos para cada rama.

2.4 Tipos de distancias (métricas) comunes

Para la monografía, es vital mencionar que la estructura elegida depende de la métrica:

- **Minkowski (L_p):** Generalización que incluye a la Euclidiana ($p = 2$) y Manhattan ($p = 1$).
- **Distancia de Coseno:** Usada en procesamiento de lenguaje natural (NLP) para medir la similitud angular entre vectores de palabras. Técnicamente no es una métrica porque no cumple la desigualdad triangular, pero se puede transformar (por ejemplo, usando la distancia angular) para usarse en estas estructuras.
- **Distancia de Jaccard:** Usada para comparar la similitud entre conjuntos de datos.

Capítulo 3

Estructuras para Espacios Discretos: Burkhard-Keller Tree (BKT)

En este capítulo, entramos de lleno en la primera estructura de datos avanzada del estudio. El *Burkhard-Keller Tree* (BKT) es una joya de la ingeniería de algoritmos diseñada específicamente para espacios donde las distancias son valores discretos (números enteros).

El BKT fue propuesto en 1973 por Walter A. Burkhard y Robert M. Keller. Su diseño aprovecha que, en ciertos dominios, la distancia entre dos objetos siempre resulta en un entero, lo que permite una ramificación exacta y una poda muy eficiente.

3.1 Origen y motivación

En estructuras de búsqueda tradicionales para texto, como los *Tries* o *Suffix Trees*, la búsqueda es exacta. Sin embargo, en problemas como la corrección ortográfica o la identificación de huellas genéticas, necesitamos encontrar elementos cercanos.

El BKT resuelve esto organizando los elementos no por su contenido, sino por su distancia relativa a un nodo raíz.

3.2 Algoritmo de construcción

La construcción del árbol es incremental y sigue estas reglas:

- **Selección de la raíz:** El primer elemento insertado se convierte en la raíz del árbol.
- **Inserción de nodos:** Para insertar un nuevo elemento u:
  - Se calcula la distancia $d = \text{dist}(\text{raz}, u)$.
  - Si la raíz ya tiene un hijo con la arista de valor d, se repite el proceso recursivamente con ese hijo.
  - Si no existe un hijo con esa distancia, u se convierte en el nuevo hijo de la raíz, conectado por una arista etiquetada con el valor d.

3.3 Algoritmo de búsqueda por rango (querying)

Supongamos que queremos encontrar todos los elementos u en el árbol tales que $\text{dist}(q, u) \leq r$, donde q es nuestra consulta y r es el radio de tolerancia (por ejemplo, buscar palabras con máximo 2 errores de escritura).

1. Se calcula $D = \text{dist}(q, \text{raz})$.
2. Se añade la raíz a los resultados si $D \leq r$.
3. **Criterio de poda (pruning):** Solo exploramos los hijos cuya arista d_h cumpla:

$$D - r \leq d_h \leq D + r$$

Este rango se deriva directamente de la desigualdad triangular. Cualquier subárbol fuera de este rango no puede contener físicamente ningún elemento que esté a distancia r de q.

3.4 Análisis de eficiencia

- **Complejidad de espacio:** $O(n)$, donde n es el número de elementos.
- **Complejidad de búsqueda:** En el peor de los casos es $O(n)$, pero en la práctica, para radios de búsqueda pequeños, se comporta de manera logarítmica.
- **Limitación principal:** Solo es eficiente si el diámetro del espacio métrico (la distancia máxima posible) es pequeño. Si las distancias son muy variadas o continuas, el árbol tendría demasiados hijos por nodo, degradando el rendimiento.

3.5 Casos de uso reales

- Correctores ortográficos: usando la distancia de Levenshtein.
- Sistemas de recomendación de amigos o contactos: basados en intereses comunes discretos.
- Bioinformática: comparación de cadenas de nucleótidos (ADN).

Capítulo 4

Estructuras para Espacios Continuos: La Familia Vantage Point

En espacios métricos continuos (como el espacio euclidiano $\mathbb{R}^n$), no podemos crear una rama para cada valor de distancia. La familia de árboles de Puntos de Ventaja (*Vantage Point*) soluciona esto mediante un particionamiento binario del espacio basado en umbrales esféricos.

4.1 Vantage Point Tree (VPT)

Propuesto por Peter Yianilos en 1993, el VPT es la base de la búsqueda por similitud en dimensiones altas.

- **Selección del punto de ventaja (p):** Se elige un elemento del conjunto para que actúe como pivote o punto de ventaja. Idealmente, este punto debe estar en un extremo del conjunto de datos para maximizar la diferenciación.
- **Radio de la mediana (μ):** Se calculan las distancias de todos los demás puntos hacia p. Se determina la mediana de estas distancias (μ).
- **Partición binaria:**
  - **Hijo izquierdo (interior):** Contiene todos los puntos u tales que $d(p, u) \leq \mu$ (dentro de la bola).
  - **Hijo derecho (exterior):** Contiene todos los puntos u tales que $d(p, u) > \mu$ (fuera de la bola).

Búsqueda y poda

Al buscar una consulta q con radio r, decidimos si entrar a una rama basándonos en si la zona de búsqueda solapa los límites de la mediana:

$$\text{Si } d(q, p) - r \leq \mu, \text{ exploramos el hijo izquierdo.}$$

$$\text{Si } d(q, p) + r > \mu, \text{ exploramos el hijo derecho.}$$

4.2 Multi-Vantage Point Tree (MVPT)

El MVPT es una evolución diseñada para optimizar el recurso más crítico: el número de cálculos de distancia.

- **Múltiples pivotes:** En lugar de un solo punto de ventaja por nivel, el MVPT utiliza dos o más puntos de ventaja en un solo nodo.
- **Uso de memoria (filtrado de distancias):** Almacena distancias calculadas previamente en los niveles superiores para evitar recalcularlas en niveles inferiores. Esto permite que un solo cálculo de distancia descarte subárboles en múltiples niveles de profundidad, mejorando drásticamente el rendimiento en conjuntos de datos masivos.

4.3 Vantage Point Forest (VPF)

Cuando los datos son extremadamente complejos o de muy alta dimensionalidad, un solo árbol puede volverse ineficiente debido al solapamiento de regiones.

- **Estructura de bosque:** El VPF utiliza múltiples árboles independientes (un bosque) construidos con diferentes puntos de ventaja iniciales.
- **Búsqueda paralela/probabilística:** Permite realizar búsquedas en paralelo y, en aplicaciones de tiempo real (como visión artificial), puede detenerse tras encontrar una buena aproximación, sacrificando exactitud por velocidad extrema.

Capítulo 5

Otras Estructuras Métricas Relevantes: MST y M-Tree

En esta sección exploramos cómo las estructuras métricas evolucionan para manejar grandes volúmenes de datos que cambian con el tiempo (inserciones y borrados) y que no caben en la memoria RAM.

5.1 Metric Space Tree (MST)

El término MST, en el contexto de métricas avanzadas, se refiere a jerarquías de particionamiento del espacio basadas en distancias. A menudo se utiliza como un modelo conceptual para entender cómo se agrupan los puntos en clústeres métricos.

- **Organización:** Los puntos se agrupan en torno a centros de clúster. Cada nodo define una región del espacio mediante un centro y un radio de cobertura.
- **Limitación:** Muchos MST tradicionales son estáticos; si los datos cambian, la estructura pierde balanceo y eficiencia rápidamente.

5.1.1 Bisector Tree (BST)

El Bisector Tree es una de las estructuras pioneras para el particionamiento de espacios métricos. A diferencia del VPT, que utiliza un solo punto de ventaja y un radio, el BST utiliza dos puntos de ventaja (pivotes) p_1 y p_2.

- **Particionamiento:** El espacio se divide mediante un hiperplano implícito definido por todos los puntos que están más cerca de p_1 que de p_2.
- **Criterio de cobertura:** Cada nodo almacena los radios r_1 y r_2, que representan la distancia máxima desde el pivote respectivo a cualquier punto en su subárbol.
- **Poda:** Durante la búsqueda, si la bola de consulta no intersecta la región del bisector, se descarta el subárbol completo.

5.2 Metric Tree (MT / M-Tree)

El M-Tree (propuesto por Ciaccia, Patella y Zezula en 1997) es probablemente la estructura más robusta para funciones continuas, ya que fue diseñada para comportarse como un B-Tree pero en espacios métricos.

Características dinámicas

- A diferencia del VPT, el M-Tree se construye de abajo hacia arriba (*bottom-up*).
- **Inserciones:** Cuando un nuevo dato entra, se busca el nodo hoja cuya región métrica requiera la menor expansión para incluirlo.
- **Balanceo (split):** Si un nodo excede su capacidad, se divide en dos, eligiendo dos nuevos centros y redistribuyendo los objetos. Esto mantiene el árbol balanceado automáticamente.

Componentes del nodo

Cada nodo en un M-Tree almacena Objetos de Enrutamiento (*Routing Objects*). Para cada objeto O_r, se guarda:

- El radio de cobertura ($r(O_r)$): la distancia máxima desde O_r hasta cualquier objeto en su subárbol.
- La distancia al padre: para acelerar las consultas usando la desigualdad triangular sin cálculos adicionales.

Optimización para memoria secundaria (disco)

Esta es la mayor ventaja del M-Tree. Al tener un alto factor de ramificación (muchos hijos por nodo), el árbol es chato o de poca altura. Esto minimiza las operaciones de lectura/escritura en disco, lo que lo hace ideal para sistemas de gestión de bases de datos (DBMS).

5.3 Comparación crítica: VPT vs. M-Tree

Característica	**Vantage Point Tree (VPT)**	**M-Tree (MT)**
Construcción	Top-Down (Estático)	Bottom-Up (Dinámico)
Balanceo	Difícil tras inserciones	Automático (vía *split*)
Uso de memoria	Optimizado para RAM	Optimizado para Disco (I/O)
Criterio de división	Mediana de distancias	Algoritmos de clustering

Capítulo 6

Análisis Comparativo y Complejidad

Llegamos al Capítulo VI, la sección de análisis crítico. Aquí se evalúa no solo cómo funcionan las estructuras, sino cuándo y por qué elegir una sobre otra basándose en su rendimiento computacional.

En este capítulo, comparamos el rendimiento de las estructuras analizadas bajo tres métricas fundamentales: el tiempo de construcción, el costo de búsqueda y la eficiencia en el uso de la memoria.

6.1 Análisis de complejidad temporal

La complejidad en espacios métricos se mide principalmente en el número de evaluaciones de la función de distancia (d), ya que esta suele ser mucho más costosa que cualquier operación aritmética simple.

Estructura	**Construcción (Promedio)**	**Búsqueda (Caso Promedio)**	**Búsqueda (Peor Caso)**
Burkhard-Keller Tree (BKT)	$O(n \log n)$	$O(\log n)$	$O(n)$
Vantage Point Tree (VPT)	$O(n \log n)$	$O(\log n)$	$O(n)$
Multi-Vantage Point Tree (MVPT)	$O(n \log n)$	$O(\log n)^*$	$O(n)$
M-Tree	$O(n \log n)$	$O(\log n)$	$O(n)$

*El MVPT reduce la constante oculta en la notación Big O mediante la reutilización de cálculos de distancia previos.

Nota: El MVPT reduce la constante oculta en la notación Big O al reutilizar cálculos de distancia previos.

6.2 Comparación de eficiencia por tipo de espacio

No todas las estructuras se comportan igual ante diferentes naturalezas de datos:

Espacios discretos (BKT)

Su eficiencia depende del alfabeto de distancias. Si el rango de distancias es pequeño (por ejemplo, distancias de edición entre 0 y 10), el árbol es muy ancho y poco profundo, lo que acelera la búsqueda.

Espacios continuos (VPT / M-Tree)

- El VPT es superior en memoria RAM gracias a su estructura binaria compacta.
- El M-Tree es el ganador absoluto cuando los datos no caben en memoria y deben ser paginados en disco (bases de datos vectoriales).

6.3 El impacto de la dimensionalidad

Un punto vital es mencionar cómo la eficiencia de búsqueda se degrada cuando la dimensión (D) aumenta.

- En dimensiones bajas ($D < 10$), estas estructuras son extremadamente rápidas.
- En dimensiones muy altas ($D > 100$), ocurre el fenómeno de la superposición de regiones, donde casi todas las ramas del árbol deben ser exploradas, y el rendimiento tiende a la búsqueda lineal ($O(n)$).

6.4 Resumen de trade-offs (compromisos)

Escenario de Aplicación	**Estructura Óptima**	**Justificación Técnica**
Datos estáticos en RAM	Vantage Point Tree (VPT)	Estructura binaria compacta que maximiza la poda mediante particionamiento por medianas. Ideal para consultas en conjuntos inmutables.
Mínima evaluación de distancia	Multi-Vantage Point Tree (MVPT)	Reutiliza distancias calculadas en niveles superiores, reduciendo drásticamente el número de llamadas a la función métrica.
Datos dinámicos en disco	M-Tree	Balanceo automático y organización por páginas que minimiza las operaciones de E/S en almacenamiento secundario.
Datos discretos (texto, ADN)	Burkhard-Keller Tree (BKT)	Aprovecha distancias enteras para crear ramificación exacta, permitiendo poda óptima en espacios discretos.

Capítulo 7

Implementación Práctica y Algoritmos

En este capítulo se presentan implementaciones académicas simplificadas de las principales estructuras métricas estudiadas. El objetivo es ilustrar su funcionamiento algorítmico, manteniendo coherencia con los fundamentos matemáticos desarrollados en capítulos anteriores.

7.1 Burkhard-Keller Tree (BKT)

El BKT es especialmente adecuado para espacios métricos discretos, donde la función de distancia devuelve valores enteros.

Pseudocódigo de búsqueda

```
Función Buscar(nodo, consulta, radio, resultados):
    Si nodo es Nulo: Retornar

    d = calcular_distancia(nodo.valor, consulta)

    Si d <= radio:
        Agregar nodo.valor a resultados

    Para cada hijo en nodo.hijos:
        Si (d - radio) <= hijo.distancia_al_padre <= (d + radio):
            Buscar(hijo, consulta, radio, resultados)
```

Implementación en C++

```
#include <iostream>
#include <vector>
#include <map>
#include <string>
#include <algorithm>

using namespace std;

// Distancia de Levenshtein simplificada
int levenshteinDistance(const string& s1, const string& s2) {
    int m = s1.length(), n = s2.length();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1));
```

```
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0) dp[i][j] = j;
            else if (j == 0) dp[i][j] = i;
            else if (s1[i - 1] == s2[j - 1])
                dp[i][j] = dp[i - 1][j - 1];
            else
                dp[i][j] = 1 + min({dp[i - 1][j],
                                    dp[i][j - 1],
                                    dp[i - 1][j - 1]});
        }
    }
    return dp[m][n];
}

struct Node {
    string word;
    map<int, Node*> children;

    Node(const string& w) : word(w) {}
};

void insert(Node* root, const string& word) {
    if (!root) return;

    int d = levenshteinDistance(root->word, word);

    if (root->children.find(d) == root->children.end()) {
        root->children[d] = new Node(word);
    } else {
        insert(root->children[d], word);
    }
}

void query(Node* root, const string& word,
            int r, vector<string>& res) {

    if (!root) return;

    int d = levenshteinDistance(root->word, word);

    if (d <= r)
        res.push_back(root->word);

    for (int dist = d - r; dist <= d + r; dist++) {
        if (root->children.count(dist)) {
            query(root->children[dist], word, r, res);
        }
```

```
    }
}
```

Nota técnica: En implementaciones reales se recomienda el uso de `smart pointers` o destructores adecuados para evitar fugas de memoria.

7.2 Vantage Point Tree (VPT)

El VPT es adecuado para espacios métricos continuos, especialmente en memoria RAM.

Pseudocódigo de construcción

```
Función ConstruirVPT(S):
    Si S está vacío: Retornar Nulo

    p = elegir_punto_ventaja(S)
    calcular distancias desde p
    mu = mediana(distancias)

    S_izq = {u | dist(p,u) <= mu}
    S_der = {u | dist(p,u) > mu}

    Retornar Nodo(p, mu,
                  ConstruirVPT(S_izq),
                  ConstruirVPT(S_der))
```

Implementación en C++

```
#include <cmath>
#include <vector>

using namespace std;

struct Point {
    double x, y;
};

double euclidean(const Point& a, const Point& b) {
    return sqrt(pow(a.x - b.x, 2) +
                pow(a.y - b.y, 2));
}

struct VPNode {
    Point vp;
    double mu;
    VPNode *left, *right;
```

```
    VPNode(Point p)
        : vp(p), mu(0),
          left(nullptr), right(nullptr) {}
};

void search(VPNode* node,
            const Point& q,
            double r,
            vector<Point>& res) {

    if (!node) return;

    double d = euclidean(node->vp, q);

    if (d <= r)
        res.push_back(node->vp);

    if (d - r <= node->mu)
        search(node->left, q, r, res);

    if (d + r > node->mu)
        search(node->right, q, r, res);
}
```

Nota técnica: La construcción completa requiere cálculo eficiente de la mediana y particionamiento del conjunto; se omite aquí por claridad académica.

7.3 Bisector Tree (BST)

El Bisector Tree divide el espacio utilizando dos pivotes y asigna cada punto al pivote más cercano.

Pseudocódigo de construcción

```
Función ConstruirBST(S):
    Si |S| < 2:
        Retornar NodoHoja(S)

    Seleccionar pivotes p1, p2

    S1 = {u | dist(p1,u) <= dist(p2,u)}
    S2 = {u | dist(p2,u) < dist(p1,u)}

    r1 = max dist(p1, elementos en S1)
    r2 = max dist(p2, elementos en S2)

    Retornar Nodo(p1, p2, r1, r2,
```

```
                ConstruirBST(S1),
                ConstruirBST(S2))
```

Implementación en C++

```
struct BSTNode {
    Point p1, p2;
    double r1, r2;
    BSTNode *left, *right;

    BSTNode(Point a, Point b,
            double rad1, double rad2)
        : p1(a), p2(b),
          r1(rad1), r2(rad2),
          left(nullptr), right(nullptr) {}
};

void searchBST(BSTNode* node,
               const Point& q,
               double r,
               vector<Point>& res) {

    if (!node) return;

    double d1 = euclidean(q, node->p1);
    double d2 = euclidean(q, node->p2);

    if (d1 <= r) res.push_back(node->p1);
    if (d2 <= r) res.push_back(node->p2);

    if (d1 - r <= node->r1)
        searchBST(node->left, q, r, res);

    if (d2 - r <= node->r2)
        searchBST(node->right, q, r, res);
}
```

Consideraciones de implementación

Las implementaciones presentadas son versiones académicas simplificadas. En entornos reales deben considerarse:

- Gestión segura de memoria (smart pointers).
- Optimización del cálculo de distancias.
- Estrategias de selección óptima de pivotes.

- Paralelización de consultas.
- Estructuras cache-friendly para alto rendimiento.

Estas mejoras son esenciales cuando las estructuras métricas se aplican en sistemas modernos de recuperación de información y bases de datos vectoriales.

Capítulo 8

Conclusiones

Llegamos al capítulo final de la monografía. En esta sección se sintetizan los hallazgos técnicos y se proyecta la importancia de estas estructuras en el panorama tecnológico actual (2026), especialmente con el auge de la Inteligencia Artificial.

8.1 Síntesis de la investigación

A lo largo de este trabajo, se ha demostrado que la búsqueda por similitud en espacios métricos trasciende las capacidades de las estructuras de datos tradicionales. Mientras que el BKT se consolida como la solución óptima para dominios discretos (como la lingüística computacional), la familia de árboles *Vantage Point* y el M-Tree ofrecen un marco robusto para manejar la complejidad de los datos continuos y de alta dimensionalidad.

8.2 Eficiencia y el factor de distancia

Una conclusión fundamental es que la eficiencia en estas estructuras no se mide por la cantidad de nodos visitados, sino por la minimización de las llamadas a la función de distancia.

La desigualdad triangular no es solo una propiedad matemática, sino la herramienta de ingeniería que permite que sistemas masivos (como la búsqueda de imágenes por contenido) sean viables en tiempo real.

8.3 Adaptabilidad según el entorno

Se concluye que la elección de la estructura debe estar estrictamente alineada con el entorno de ejecución:

- Para aplicaciones en memoria con datos estáticos, el VPT ofrece el mejor compromiso entre simplicidad y velocidad.
- Para sistemas de bases de datos escalables que requieren inserciones constantes, el M-Tree es la arquitectura de referencia debido a su naturaleza dinámica y optimización para almacenamiento persistente.

8.4 El futuro: bases de datos vectoriales e IA

En la actualidad, estas estructuras de datos avanzadas forman el núcleo de las Bases de Datos Vectoriales (*Vector Databases*). Con el crecimiento de los Modelos de Lenguaje

(LLMs) y los *embeddings*, la capacidad de realizar búsquedas rápidas en espacios métricos de cientos de dimensiones es más crítica que nunca.

Las técnicas de particionamiento analizadas en esta monografía constituyen la base sobre la cual se construyen los sistemas de recuperación de información de próxima generación.

Bibliografía

[1] Burkhard, W. A., & Keller, R. M. (1973). *Some approaches to best-match file searching*. Communications of the ACM.

[2] Yianilos, P. N. (1993). *Data structures and algorithms for nearest neighbor search in general metric spaces*. Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA).

[3] Ciaccia, P., Patella, M., & Zezula, P. (1997). *M-tree: An efficient access method for similarity search in metric spaces*. Proceedings of the International Conference on Very Large Data Bases (VLDB).

[4] Chávez, E., Navarro, G., Baeza-Yates, R., & Marroquín, J. L. (2001). *Searching in metric spaces*. ACM Computing Surveys.

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