UNIVERSIDAD DEL BÍO-BÍO

Facultad de Ciencias Empresariales
Departamento de Ciencias de la Computación y
Tecnologías de la Información

Algoritmos de separabilidad de objetos en grandes conjuntos de datos espaciales

Por
Claudio Andrés Torres Fonseca

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

Dirigido por:
Dr. Gilberto Gutiérrez

Co-Dirigido por:
Dr. Pablo Pérez-Lantero

Marzo 2016

Estudios de Postgrado financiados por beca CONICYT-PCHA/MagisterNacional/2015 - 22151665.

en memoria de mi mami María,
quien es la luz
que ilumina mi camino

Agradecimientos

En esta etapa de mi vida que termina, debo dar las gracias a todas las personas que han participado en ella, que de una u otra forma han estado presente y han influido en forma positiva en mi vida y me han ayudado a sacar adelante este proyecto. En especial debo dar las gracias a

A Dios, por acompañarme en cada camino de mi vida. Por permitirme conocer buenas personas y darme la posibilidad de compartir gratos momentos y experiencias junto a ellos.

A mi Familia, en especial a mi madre y mi hermana, por todo el apoyo brindado en esta etapa de mi vida, quienes me han dado ánimo para terminar este proyecto.

A Fabián, por su amistad, por todos los buenos momentos que hemos vivido y por ser un importante apoyo en los malos momentos que me ha tocado vivir. Eres como un verdadero hermano para mi.

A Gilberto, por compartir sus conocimientos y experiencias conmigo. Ayudandome a mejorar continuamente. Por sus consejos, que me han ayudado a tomar grandes decisiones.

A Pablo, por todo su apoyo durante esta investigación. Por todo su conocimiento entregado, enseñanzas y correciones realizadas.

Finalmente, agradecer a <`select * from mis_amigos`> por ser una parte fundamental de mi vida, con quienes comparto mi felicidad y tristeza. En especial agradecer a Fernando, Nicole, Matías, Carlos, Claudia, Brenda, Juan, por estar presentes y acompañarme cuando más lo necesitaba.

A todos, muchísimas gracias.

Resumen

Sea S un conjunto de puntos en $\mathbb{R}^2$, el cual está formado por los conjuntos R (puntos rojos) y B (puntos azules), tal que $S = R \cup B$. Los conjuntos R y B se encuentran almacenados en R-trees distintos. El Problema de la Separabilidad Lineal consiste en encontrar una recta ℓ (si existe), que divida el espacio en dos regiones, de tal forma que los puntos rojos y azules están separados, es decir, cada región contiene puntos de un sólo color.

En este trabajo, se resuelve el Problema de la Separabilidad Lineal en el contexto de las Bases de Datos Espaciales, es decir, considerando grandes volúmenes de datos en memoria secundaria lo que provoca que sea inviable utilizar los algoritmos disponibles en la Geometría Computacional. Debido a lo costoso del proceso de lectura desde memoria secundaria y al gran volumen de datos que deben ser procesados. En concreto, se estudia la Separabilidad Lineal de dos conjuntos de puntos (R, conjunto de puntos rojos, y B, conjunto de puntos azules) en $\mathbb{R}^2$, los cuales están almacenados en R-trees distintos.

En esta tesis se proponen dos algoritmos de separabilidad. El primer algoritmo (de ramificación y poda) descarta ramas enteras del árbol con el fin de recuperar solo los puntos relevantes de R y B. Una vez realizado esto, los puntos recuperados son procesados usando un algoritmo de Geometría Computacional. Los resultados experimentales muestran que este método debe acceder solamente un 17 % de los nodos de los R-trees, y debe procesar un 0,21 % de los puntos usando el algoritmo de Geometría Computacional.

Nuestro segundo algoritmo calcula pseudo cerraduras convexas de los conjuntos R y B, con lo cual en cada iteración se obtiene una aproximación cada vez más exacta de las cerraduras convexas reales de R y de B. Resultados experimentales muestran que este método necesita acceder sólo a un 2,1 % de los nodos de los R-tree y sólo necesita almacenar las cerraduras convexas de los conjuntos en memoria principal.

A través de los métodos propuestos es posible calcular la separabilidad lineal para dos conjuntos almacenados en R-trees distintos sin la necesidad de leer todos los nodos de los R-trees. Esto disminuye la cantidad de memoria usada y el tiempo de cálculo, comparado con recuperar todos los puntos y procesarlos en memoria principal usando un algoritmo de Geometría Computacional.

Índice

Parte I

Motivación

Capítulo 1

Introducción

1.1. Motivación

La Geometría Computacional (GC), inicialmente propuesta por Shamos [38], es una rama de las Ciencias de la Computación que se enfoca en diseñar algoritmos y estructuras de datos eficientes, para resolver problemas que pueden ser modelados como problemas geométricos [20].

Algunos problemas que se han estudiado en la GC son: búsqueda de los k-vecinos más cercanos de un punto q [29], búsqueda de la cerradura convexa (convex hull) de un conjuntos de puntos [30, 38], obtención del diagrama de Voronoi [30, 38], separabilidad bicromática [14, 36], entre otros.

Debido a la necesidad de almacenar y procesar grandes volúmenes de datos espaciales, han surgido las Bases de Datos Espaciales (BDEs), que son sistemas de bases de datos que poseen soporte para tipos de datos espaciales [33]. Las BDEs han extendido las capacidades de los Sistemas de Bases de Datos Relacionales, a través de estructuras de datos, índices y métodos de acceso espaciales, entre otros [21, 40]. Adicionalmente, ha sido necesario resolver nuevos tipos de consultas. Por ejemplo, búsqueda por rango (Range Query)[21, 33], intersección (Intersection Query)[33, 40], reunión espacial (Spatial Join) [33, 40] y operaciones de agregación espacial (Spatial Aggregate) [40]. Muchas de estas consultas fueron problemas propuestos desde la GC y fue necesario proponer y diseñar algoritmos para resolverlos en el dominio de las BDEs [33]. Por ejemplo, para el problema de los k vecinos más cercanos Roussopoulos et al. [35] proponen un algoritmo de ramificación y poda para resolver el problema considerando que los puntos se encuentran almacenados en un R-tree (memoria secundaria) [22]. Para el problema de la cerradura convexa, Böhm et al [8] proponen algoritmos que aprovechan las propiedades de los índices espaciales jerárquicos.

La separabilidad de objetos es un área de estudio dentro de la GC que considera problemas como el siguiente [24]: supongamos que tenemos dos conjuntos de puntos en el espacio, clasificados como puntos rojos y puntos azules. ¿Existe cierto objeto geométrico con determinadas propiedades que separe los puntos rojos de los puntos azules?. La noción más natural de separabilidad es por una línea recta (o un hiperplano), o Separabilidad Lineal [24] (Fig. 1.1 a), surgiendo variantes [36] como la separabilidad por banda (Fig. 1.1 c), por cuña (Fig. 1.1 d), por doble cuña y por figuras geométricas convexas, como círculos [15], cajas (i.e., rectángulos o hiperrectángulos con los lados

paralelos a los ejes coordenados) [14] (Fig. 1.1 b), cuadrados[15], entre otros. Los algoritmos de separabilidad estudiados por la GC consideran que los objetos están almacenados en memoria principal. El problema de la separabilidad de objetos posee aplicaciones en diversos campos, por ejemplo: análisis de imágenes, estadísticas y campos en los cuales la discriminación y/o clasificación es requerida [36].

Algunos ejemplos del uso de la separabilidad de objetos son los siguientes:

El parque $P1$ posee una alta población de osos. Los osos se alimentan de un determinado fruto y algunas plantas del lugar. Actualmente, se ha reportado diversos avistamientos de osos en las zonas de camping del parque, debido a ello se está evaluando colocar una valla que separe los árboles y plantas que sirven como alimento de los osos de las zonas de camping. La ubicación geográfica de las zonas de camping y de las plantas de los osos están almacenadas en una Base de Datos Espacial. Los encargados del parque necesitan conocer si es posible instalar una valla que separe los lugares de camping de los lugares donde comen los osos de tal forma que a un lado de la valla se encuentren todas las zonas de camping y al otro lado estén todos los lugares donde se alimentan los osos.

Los viajes entre Concepción y Chillán son muy frecuentes, razón por la cual el Ministerio de Transportes ha decidido implementar un tren de alta velocidad entre ambas ciudades. En la zona geográfica comprendida entre ambas ciudades existe una reserva de la flora y fauna, en la cual se han identificado dos ecosistemas, cuya posición geográfica de las especies que la componen esta almacenada en una base de datos espacial. La normativa ambiental vigente exige que la instalación de una línea de tren de alta velocidad no puede atravezar un ecosistema, debido al terrible impacto ambiental que eso generaría. Debido a ello, se desea saber si es posible que la línea del tren atraviese la reserva.

La ciudad de Chillán ha tenido un gran aumento en la construcción de grandes edificios. La compañía eléctrica de la ciudad está preocupada, ya que la red de electricidad actual no está preparada para los futuros requerimientos energéticos de la ciudad. Es por ello, que planea dividir la ciudad en dos zonas: zona de alto consumo y zona de bajo consumo. La compañía eléctrica desea saber si es posible realizar esta división, de tal forma que solo existan casas en la zona de bajo consumo y solo edificios en la zona de alto consumo, considerando que la posición geográfica de las casas y edificios están almacenadas en una base de datos espacial.

En esta tesis se pretende diseñar y evaluar algoritmos para resolver problemas de separabilidad de un conjunto de puntos en un espacio Euclídeo de dos dimensiones, almacenados en estructuras de datos que residen en memoria secundaria, en el contexto de las BDEs. El resto del presente capítulo se estructura de la siguiente forma: en la Sección 1.2 se presenta la hipótesis y los objetivos de la tesis, a continuación, en la Sección 1.3 se presenta el alcance de la tesis y en la Sección 1.4 se muestra la metodología de trabajo. Finalmente, en la Sección 1.5, se exponen las contribuciones logradas en esta tesis y en la Sección 1.6 se muestra la estructura y contenidos de este informe.

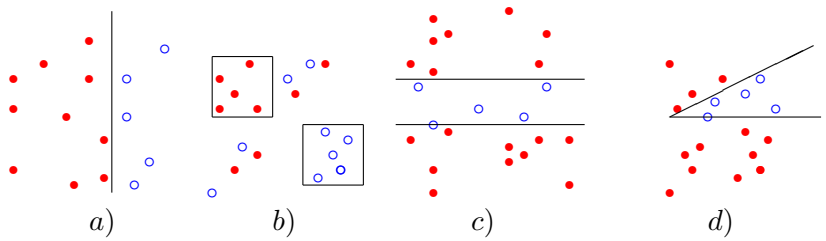


Fig. 1.1: Separabilidad: a)lineal b)por caja c) por banda d) por cuña.

1.2. Hipótesis y objetivos

Consideremos el siguiente problema: sea S un conjunto de puntos en $\mathbb{R}^2$, el cual está formado por los conjuntos R (puntos rojos) y B (puntos azules), tal que $S = R \cup B$. Se desea encontrar un objeto, bajo ciertas propiedades, que divida el espacio en regiones, de tal forma que los puntos rojos y azules están separados, es decir, cada región contiene puntos de un solo color. Este problema se denomina Problema de la Separabilidad Bicromática, a partir del cual nuestra hipótesis y objetivos de trabajo son:

Hipótesis

Es factible diseñar algoritmos eficientes en tiempo y almacenamiento, que permitan resolver el Problema de la Separabilidad Bicromática utilizando algoritmos de ramificación y poda, asumiendo que los puntos se encuentran almacenados en una estructura de datos en memoria secundaria y que logren un mejor rendimiento que resolver el problema recuperando todos los elementos de la estructura de datos y utilizar un algoritmo de Geometría Computacional.

Objetivo general

El objetivo general es desarrollar y evaluar algoritmos eficientes que tomen ventajas de las propiedades de las estructuras de datos en memoria secundaria para resolver el problema de la Separabilidad Bicromática.

Objetivos específicos

- Definir los problemas de Separabilidad Bicrómatica específicos a resolver y las estructuras de datos multidimensionales a utilizar.
- Diseñar, implementar y evaluar algoritmos que resuelvan los problemas especificados y que consideren que los datos están almacenados en estructuras de datos espaciales en memoria secundaria, tales como el R-tree.
- Implementar algoritmos existentes en la literatura que utilizan datos almacenados en memoria principal para resolver los problemas propuestos.

- Comparar los algoritmos existentes en la literatura con los algoritmos propuestos que utilizan memoria secundaria.

1.3. Alcance de la investigación

Esta tesis contempla el Problema de la Separabilidad Bicromática en un espacio Euclídeo de dos dimensiones ($\mathbb{R}^2$). La implementación de los algoritmos se realizará en el lenguaje de programación $C++$ y los algoritmos construidos se publicarán en el repositorio GitHub[1].

1.4. Metodología de trabajo

La metodología de trabajo a aplicar consta de las siguientes etapas:

1. Se realizará una revisión de la literatura, con el propósito de analizar los algoritmos y técnicas propuestos para revolver el problema de separabilidad bicromática.
2. Se realizará una revisión de la literatura con el objetivo de analizar las propiedades de algunas estructuras de datos espaciales en memoria secundaria.
3. Se diseñarán algoritmos que utilicen las propiedades de las estructuras de datos espaciales en memoria secundaria para disminuir la cantidad de datos a ser recuperados durante la ejecución de los algoritmos.
4. Se construirán algoritmos que permitan utilizar las técnicas diseñadas en la etapa anterior y que resuelvan el problema planteado.
5. Se realizarán análisis experimentales, con datos reales y sintéticos, para comparar los algoritmos propuestos con los algoritmos de la literatura.

1.5. Contribuciones de la tesis

Las principales contribuciones de esta tesis son:

- Se resuelve el problema de la Separabilidad de objetos, desarrollado y solucionado en el dominio de la Geometría Computacional, en el dominio de las Bases de Datos Espaciales.
- Se presentan dos nuevos métodos para calcular la separabilidad lineal de conjuntos almacenados en R-trees distintos
- Se implementaron los algoritmos propuestos en el lenguaje $C++$.

Como resultado de esta tesis se han generado las siguientes publicaciones en conferencias nacionales:

[1] `https://github.com/ctorresf/libSLR`

- Claudio Torres, Gilberto Gutiérrez, Pablo Pérez-Lantero. Algoritmos de separabilidad de objetos espaciales almacenados en R-tree. Encuentro Chileno de Computación 2015, Santiago (Chile), 9 de Noviembre del 2015.
- Claudio Torres, Gilberto Gutiérrez, Pablo Pérez-Lantero. Algoritmos de separabilidad de objetos espaciales almacenados en R-tree. Encuentro de Tesistas 2015, Santiago (Chile), 10 de Noviembre del 2015.
- Claudio Torres, Gilberto Gutiérrez, Pablo Pérez-Lantero. Algoritmos de separabilidad de objetos espaciales almacenados en R-tree. IV Encuentro de Investigación de Estudiantes de Postgrado UBB 2015, Concepción (Chile), 19 y 20 de Noviembre del 2015.

Finalmente, se envió un artículo a la revista GeoInformatica[2].

- Claudio Torres, Gilberto Gutiérrez, Pablo Pérez-Lantero. Linear Separability in Spatial Databases.

Además, este artículo se publicó en el repositorio electrónico para prepublicaciones arXiv. El artículo se encuentra disponible en `http://arxiv.org/abs/1602.04399`.

1.6. Estructura y contenidos de la tesis

Esta tesis contempla 4 partes y 7 capítulos. En la primera parte se realiza una motivación al tema de tesis, exponiendo la hipótesis y los objetivos de la tesis.

En la segunda parte se realiza una revisión de la literatura, empezando en el Capítulo 2, en el cual se realiza una descripción de las BDEs, operadores y consultas espaciales, junto a métodos de acceso, como lo es el R-tree. En el Capítulo 3 se desarrolla el tema de las separabilidad bicrómatica, mirado desde el punto de vista de la Geometría Computacional. En este capítulo se define la separabilidad bicromática, se exponen las principales formas de separabilidad y se muestran algoritmos que permiten comprobar la separabilidad bicromática para dos conjuntos de puntos.

La tercera parte describe los métodos propuestos para determinar la separabilidad lineal de dos conjuntos de puntos almacenados en R-tree distintos, que han sido el resultado de esta tesis. El Capítulo 4 se centra en describir un conjunto de conceptos y propiedades que son utilizados por los métodos propuestos. El Capítulo 5 describe nuestra primera propuesta, basada en los algoritmos de ramificación y poda. El Capítulo 6 describe la segunda propuesta, que se basa en calcular cerraduras convexas para determinar la separabilidad lineal de los conjuntos.

La cuarta parte se centra en analizar y discutir los resultados obtenidos en esta tesis. En el Capítulo 7 se discuten las conclusiones y se proponen problemas que pueden ampliar el trabajo de esta tesis.

[2] `http://link.springer.com/journal/10707`

Parte II

Trabajo relacionado

Capítulo 2

Bases de Datos Espaciales

2.1. Introducción

Las Bases de Datos Espaciales (BDEs) han surgido como una extensión de las Bases de Datos Relacionales [21], debido a la complejidad de los datos espaciales [42]. Las BDEs contemplan el diseño de nuevas estructuras de datos para almacenar la información espacial, índices, nuevos tipos de consulta, entre otros [21]. Para la implementación de índices espaciales se han diseñado estructuras de datos para indexar objetos espaciales, como puntos, líneas y regiones. Ejemplos de estas estructuras son: Grid Files, VA-Files, K-D-B-Tree, R-Tree, R^+-Tree, R^*-Tree, entre otros.

Las BDEs deben procesar nuevas consultas sobre los datos espaciales [21, 42], tales como consulta de coincidencia exacta (Exact Match Query), consulta por rango (Range Query), intersección (Interseccion Query), consulta de adyacencia (Adjacency Query). Muchas de estas consultas fueron propuestas desde la GC, por lo que fue necesario diseñar algoritmos que las resolvieran en el dominio de las BDEs [40].

Las BDEs son fundamentales en los Sistemas de Información Geográfica y en sistemas que deben almacenar y procesar grandes conjuntos de datos espaciales.

El presente capítulo se estructura de la siguiente forma. En la Sección 2.2 se muestran los tipos de datos espaciales básicos y en la Sección 2.3 se presentan las principales operaciones que se realizan sobre objetos espaciales. En la Sección 2.4 se describen las consultas comunes que debe responder una BDE, en la Sección 2.5 se describe el proceso para dar respuestas a las consultas espaciales. Finalmente, en la Sección 2.6 se explican los métodos de acceso espacial usados para almacenar los datos espaciales.

2.2. Tipos de datos espaciales

Dentro de una BDE, considerando que el espacio se modela mediante el modelo de entidad-relación, un objeto es modelado mediante diversos atributos no espaciales (cadenas de caracteres, números enteros, etc.) y un atributo espacial. Para el modelado de los atributos espaciales, se utilizan tres abstracciones elementales: punto, línea y región o polígono (ver Fig. 2.1). Se considera como un punto a un objeto espacial del cual sólo interesa su posición en el espacio. Por ejemplo,

edificios, árboles, antenas de transmisión, etc.

Una línea es una abstracción que nos permite modelar objetos como carreteras, ríos, caminos, etc. Una región nos permite modelar objetos que poseen una cobertura espacial, tales como bosques, ciudades, etc [40].

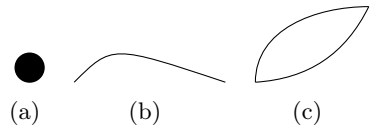


Fig. 2.1: Datos espaciales: a) punto b) línea c) región.

2.3. Operaciones sobre objetos espaciales

Las operaciones que se pueden realizar sobre objetos espaciales están determinadas de acuerdo a la forma en que los objetos interactúan entre sí. En esta sección describiremos los distintos tipos de operaciones/relaciones que poseen los objetos espaciales [40].

Orientación

Las relaciones más simples y generales entre objetos espaciales son denominadas de orientación [40]. Las relaciones básicas que se pueden encontrar en esta categoría son: unión, intersección, contención y afiliación.

Relaciones topológicas

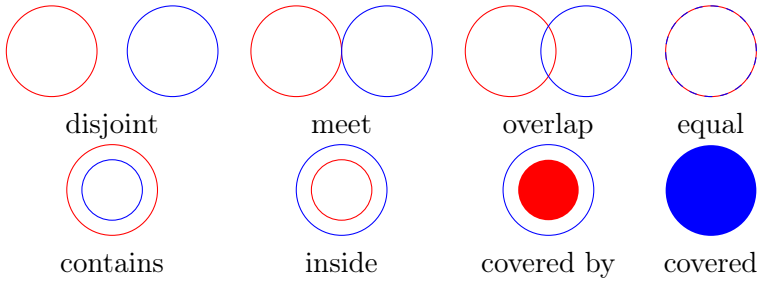
Las operaciones topológicas son aquellas que establecen relaciones entre objetos espaciales, las cuales se mantienen ante operaciones de rotación y traslación [21].

Desde un punto de vista de las bases de datos espaciales/sistemas geográficos, es más común que un usuario consulte sobre una relación topológica en una consulta a la base de datos. Las consultas típicas en bases de datos espaciales se dividen en dos tipos:

- Encontrar todos los objetos que tienen una relación topológica R con un objeto determinado.
- ¿Cual es la relación topológica entre los objetos A y B?

Para objetos de tipo polígono, se pueden establecer 8 relaciones topológicas entre ellos, la cuales son [17, 40]: *disjoint*, *meet*, *overlap*, *equal*, *contains*, *inside* y *covered by*, las cuales se pueden apreciar en la Fig. 2.2.

Fig. 2.2: Relaciones topológicas



Relaciones de dirección

Las relaciones de dirección se clasifican en tres tipos: absoluta, relativa a un objeto o basada en el espectador [40]. Las relaciones absolutas se definen en el contexto de un sistema de referencia global, por ejemplo: norte, sur, este, oeste, etc. En cambio, las relaciones relativas a un objeto se definen en base a un objeto dado, por ejemplo, arriba, derecha, atrás, etc. Las relaciones de dirección basadas en un espectador están definidas con respecto a un objeto de referencia especial designado por el espectador.

Operaciones métricas

Las operaciones o relaciones métricas permiten obtener atributos numéricos de un objeto espacial o valores entre ellos que los relacionan [40]. Por ejemplo, la distancia.

2.4. Consultas espaciales

En las bases de datos relacionales, la mayoría de las consultas están compuestas por un conjunto fijo de operaciones básicas. Estas operaciones básicas forman los bloques de construcción para la composición de consultas más complejas [40]. En cambio, en el contexto de las BDEs, debido al amplio dominio de las aplicaciones de las bases de datos espaciales, no se ha podido llegar a un consenso sobre el conjunto de operadores básicos que puedan cubrir todos los casos. Además, los algoritmos para calcular los operadores espaciales son complejos, requieren un uso intenso de CPU (Central Processing Unit) y lectura/escritura de disco. Esto hace que la construcción de las consultas espaciales sea un proceso complejo.

En esta sección se explicarán un conjunto de consultas catalogadas en la literatura como "fundamentales"[21]. Además, se muestra una clasificación de las principales operaciones que debe soportar un sistema de gestión de bases de datos espaciales.

2.4.1. Consultas fundamentales

A continuación se definen las consultas, las cuales se consideran fundamentales [19]. Para referirse al atributo espacial de un objeto o se utiliza la notación $o.G \subseteq \mathbb{R}^2$.

- Exact Match query (EMQ): Dado un objeto o' encontrar otro objeto o cuyo atributo espacial es igual al de o'.

$$EMQ(o') = \{o|o'.G = o.G\}$$

- Point Query (PQ): Dado un punto $p \in \mathbb{R}^n$, encontrar todos los objetos o que cubren a p:

$$PQ(p) = \{o|p \cap o.G = p\}$$

- Windows Query/Range Query (WQ/RQ): Dado un rango $q \subseteq \mathbb{R}^n$, encontrar todos los objetos o que tienen al menos un punto en común con q:

$$WQ(q) = \{o|q \cap o.G \neq \emptyset\}$$

  El rango q es iso-orientado, es decir, las caras que definen a q son paralelas a uno de los ejes.

- Intersection Query (IQ): Dado un objeto o' con atributo espacial $o'.G \subseteq \mathbb{R}^n$, encontrar todos los objetos o que tienen al menos un punto en común con o':

$$IQ(o') = \{o|o'.G \cap o.G \neq \emptyset\}$$

- Enclosure Query (EQ): Dado un objeto o' con atributo espacial $o'.G \subseteq \mathbb{R}^n$, encontrar todos los objetos o que lo encierran o contienen completamente:

$$EQ(o') = \{o|o'.G \cap o.G = o'.G\}$$

- Containment Query (CQ): Dado un objeto o' con atributo espacial $o'.G \subseteq \mathbb{R}^n$, encontrar todos los objetos o encerrados por o':

$$CQ(o') = |o'.G \cap o.G = o.G\}$$

- AQ (Adjacency Query): Dado un objeto o' con atributo espacial $o'.G \subseteq \mathbb{R}^n$, encontrar todos los objetos o adyacentes a o':

$$AQ(o') = \{o|o.G \cap o'.G \neq \emptyset \wedge o'.G^0 \cap o.G^0 = \emptyset\}$$

  Aquí $o'.G^0$ y $o.G^0$ denotan el interior de las figuras geométricas formadas por los atributos espaciales $o.G$ y $o'.G$, respectivamente.

- Nearest-Neighbor Query (NNQ): Dado un objeto o' con atributo espacial $o'.G \subseteq \mathbb{R}^n$, encontrar todos los objetos o que tienen la mínima distancia a o':

$$NNQ(o') = \{o|\forall o'' : dist(o.G, o'.G) \leq dist(o'.G, o''.G)\}$$

En este caso se define *dist* de dos atributos espaciales como la distancia (Euclídea) entre sus puntos más cercanos.

- Spatial Join (SJ): Dadas dos colecciones de objetos R y S y un predicado binario espacial θ, encontrar todos los pares de objetos $(o, o') \in R \times S$ donde $\theta(o.G, o'.G)$ es verdadero:

$$R \bowtie S = \{(o, o') | o \in R \wedge o' \in S \wedge \theta(o.G, o'.G)\}$$

El predicado θ puede ser:

- intersect
- contains
- is_enclosed_by
- distance
- northwest
- adjacent
- meets
- overlap

2.4.2. Clasificación

Las operaciones espaciales se pueden clasificar en cuatro categorías [40]:

- *Operaciones de actualización:* operaciones comunes de una base de datos, tales como modificar, crear, etc.
- *Operaciones de selección:* pueden ser de dos tipos:
  1. Point Query.
  2. Range Query.
- *Reunión espacial (Spatial Join).*
- *Agregación Espacial:* usualmente son variantes del problema del Nearest-Neighbor Query. Un ejemplo sería, "encontrar el río más cercano a un campamento".

2.5. Procesamiento de consultas espaciales

El procesamiento de los datos espaciales necesario para responder una consulta consta de una serie de actividades (ver Fig. 2.3) divididas en dos etapas: *filtrado* y *refinamiento*.

En la etapa de filtrado se realiza un preprocesamiento de la consulta utilizando las propiedades del índice espacial, si es que el conjunto de datos dispone de uno, con lo que se obtiene un conjunto de objetos candidatos, para los cuales es posible que se cumpla con el predicado de la consulta. Todos los objetos candidados pasan a la etapa de refinamiento.

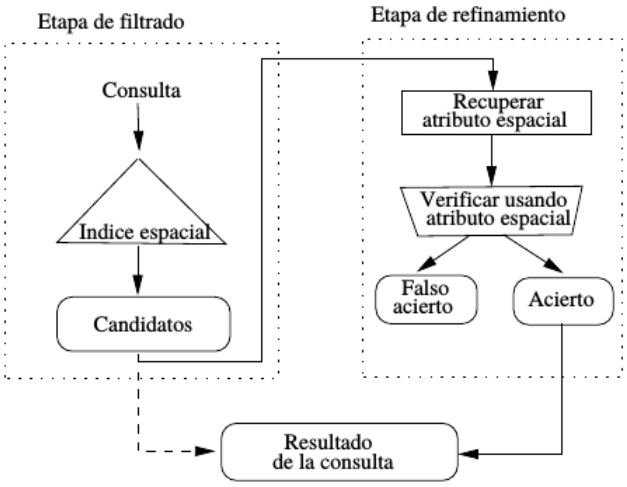


Fig. 2.3: Procesamiento de consultas espaciales usando un índice espacial, extraído de [21].

En la etapa de refinamiento se comprueba que los objetos candidatos realmente cumplan con el predicado de la consulta, para lo cual es necesario recuperar el atributo espacial del objeto y verificar si éste cumple con el predicado.

Debido a que los datos espaciales poseen una estructura compleja y un gran tamaño, la etapa de refinamiento puede realizar un uso intenso de la CPU, siendo una componente no despreciable del costo total de procesado de la consulta. Es por esta razón que se invierte bastante esfuerzo en lograr que el conjunto candidato obtenido de la etapa de filtrado sea lo más cercano posible a la respuesta definitiva, con el fin de disminuir al máximo el procesamiento que se realiza en la etapa de refinamiento.

2.6. Métodos de acceso espacial

La información espacial posee una estructura compleja, ya que los datos espaciales tienen una alta dimensionalidad. Para almacenar y procesar grandes volúmenes de datos espaciales se requieren estructuras de índices multidimensionales, que sean capaces de gestionar y responder en forma eficiente consultas espaciales. Los métodos unidimensionales, como el B-tree o sus variantes, son incapaces de responder en forma eficiente a estos requerimientos, es por ello que se han diseñado nuevos métodos que dan soporte a las necesidades de almacenamiento y procesado de información espacial. En esta sección se presentan los principales métodos de acceso espacial [13, 21, 40]: Grid File, K-D-B-tree, R-tree, R^+-tree y R^*-tree. Se asume que el lector tiene conocimientos de B-tree [5].

Dado que en esta tesis se utiliza el R-tree como estructura de datos para almacenar los conjuntos de puntos, en la Sección 2.7 se realizará un analisis más detallado de esta estructura.

Grid File

Un Grid File [28] realiza la división del espacio mediante una rejilla de k-dimensiones, cuyas celdas no siempre tendrán el mismo tamaño. Está formada por *páginas de datos* (almacenadas en disco), el *directorio* (compuesto por las celdas k-dimensionales, relacionando las celdas con las páginas de datos) y las *escalas* (mantienen información de las divisiones que se han realizado del espacio en cada dimensión). Las páginas de datos y el directorio se mantienen en memoria secundaria, mientras que las escalas se almacenan en memoria principal para garantizar un rendimiento óptimo sobre consultas con varias claves de búsqueda.

En Fig. 2.4 se observa la división del espacio generada por un Grid File. En ella se puede observar que cada celda contiene a lo más dos puntos y que el tamaño de las celdas está determinado por la cantidad de puntos que posee, por lo que las celdas no cubren necesariamente la misma área.

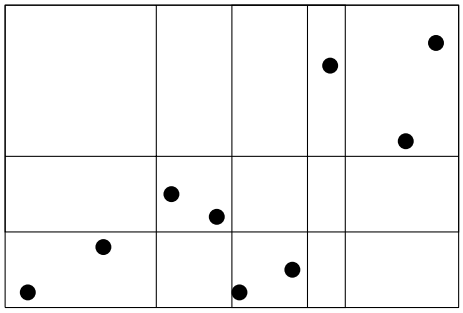


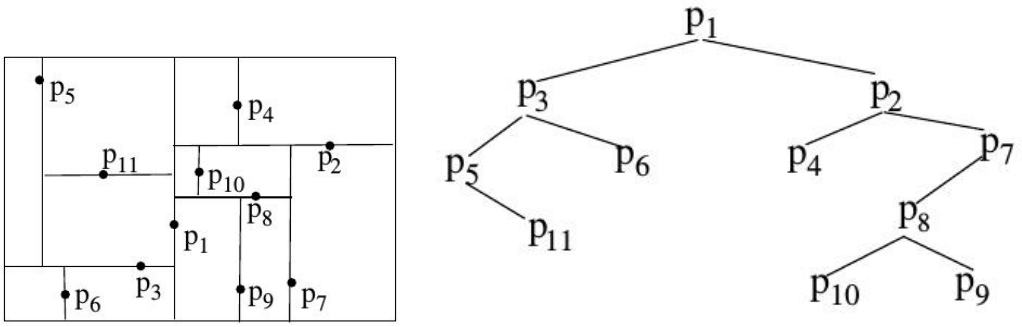
Fig. 2.4: Ejemplo de un Grid-File

K-D-B-tree

El K-D-tree [7] es una estructura multidimensional de memoria principal, la cual está diseñada para indexar conjuntos de objetos de tipo *punto*. Divide el espacio multidimensional en forma recursiva utilizando hiperplanos de $(n-1)$-dimensiones. Los hiperplanos son iso-orientados y sus direcciones alternan entre las n posibilidades. En la Fig. 2.5 (a) observamos la forma en la que divide el espacio un K-D-tree. En la Fig. 2.5 (b) se muestra el K-D-tree del conjunto de puntos de la Fig. 2.5 (a).

El K-D-B-tree [34] es la combinación del K-D-tree y el B-tree. Es un árbol perfectamente balanceado, cada nodo se almacena en una página o bloque de disco. Al igual que un B-tree, el K-D-B-Tree considera dos clases de nodos: nodos internos y nodos hojas. Los nodos internos representan regiones del espacio, almacenan tuplas de la forma <región, ref>, donde región representa el subespacio y ref es una referencia al nodo hijo que contiene todas las subregiones contenidas en la región de la entrada. Los nodos de un mismo nivel representan regiones mutuamente disjuntas y su unión constituye el universo original. Los nodos hojas almacenan los datos (puntos) que están ubicados en la misma región, almacenan la información en forma de tuplas del tipo <punto,oid >, donde oid es la dirección de la tupla o registro dentro de la base de datos.

En la Fig. 2.6 se observa un K-D-B-tree. Notar que los nodos internos dividen el espacio en regiones disjuntas. Además, se observa que los puntos están almacenados en los nodos hojas.



(a) División del espacio generada por un k-d-tree.

(b) k-d-tree del conjunto de puntos de (a).

Fig. 2.5: Ejemplo de un K-D-tree, extraído de [21].

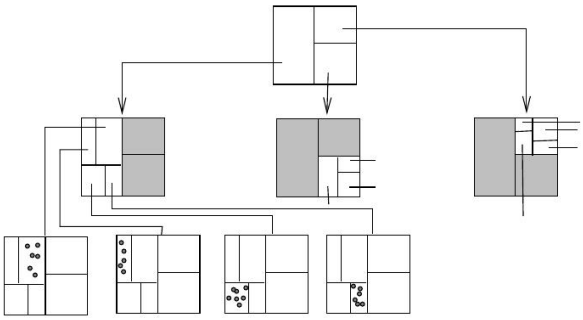


Fig. 2.6: Ejemplo de un K-D-B-tree, extraído de [21].

R-tree

Un *R*-tree [22] es una extensión del *B*-tree, diseñado para almacenar objetos espaciales. El *R*-tree es un árbol multidimensional balanceado en altura, en forma similar al *B*-tree, que representa una jerarquía de regiones rectangulares minimales (MBRs, del inglés Minimun Bound Rectangle). Cada nodo del *R*-tree tiene asociado un MBR, en cambio el *B*-tree tiene asociado un rango. Además, cada nodo representa un bloque de disco o página de datos. Los nodos hojas almacenan un identificador del objeto y el MBR que lo contiene. Los nodos internos almacenan un conjunto de nodos del nivel inferior y el MBR que los contiene.

La Fig. 2.7 (a) muestra las particiones del espacio generadas por un R-tree a diferentes niveles. La Fig. 2.7 (b) muestra el conjunto de rectángulos de la Fig. 2.7 (a) organizados en un R-tree.

En la Sección 2.7 se realizará un analisis más detallado del R-tree.

R^+-tree y R^*-tree

Uno de los principales problemas que posee el R-tree es que para realizar una búsqueda es probable que se deba seguir más de un camino, desde los nodos internos a las hojas, para llegar a la solución ya que las hojas contienen los objetos espaciales. Esto se debe a que, por lo general, los MBRs de los nodos están intersecados. Para intentar evitar este problema se han creado variantes del R-tree, como el R^+-tree [37] y el R^*-tree [6].

El R^+-tree establece que los MBRs de los nodos no se intersecan, con lo cual al insertar un objeto, este puede ser divido en diversos nodos de ser necesario. Debido a que no existe intersección entre los MBRs de los nodos, las consultas del tipo **Point Query** (búsqueda de un punto) son muy veloces, superando al R-tree. Sin embargo, otras consultas, como la **Windows Query**, requieren recorrer varios caminos, obteniendo un rendimiento similar al R-tree. Las principales desventajas del R^+-tree es que la inserción y eliminación de objetos es muy costosa, mayor al R-tree. Además, debido a lo complicado de la inserción, los MBRs de los datos están muy fragmentados.

El R^*-tree mejora al R-tree en el proceso de inserción, ya que este proceso es crítico en la etapa de búsqueda, utilizando la reinserción forzada. El proceso de inserción forzada consiste en no dividir un nodo cuando esté lleno, se propone eliminar p entradas del nodo y reinsertarlas en el árbol. Los algoritmo del R^*-tree toman en cuenta la optimización de:

- El área de intersección entre MBRs de un mismo nivel del árbol debe ser minimizada.
- Se debe minimizar los perímetros de los MBRs. La forma preferida es el cuadrado, debido a que es la representación rectangular más compacta.
- Se debe maximizar la utilización del almacenamiento.

Estos cambios permiten mejorar el rendimiento del R^*-tree en hasta un 50 % comparado con el R-tree. Con respecto a la búsqueda e inserción en el R^*-tree, estas son idénticas a las utilizadas por el R-tree.

2.7. El R-tree, un método de acceso espacial

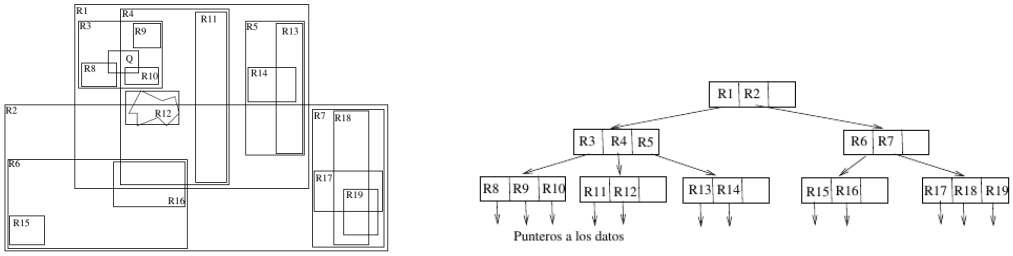
El R-tree [22] es una de las estructuras de datos multidimensional más estudiada y que ha sido adoptado por diversos productos de Bases de Datos, tales como Oracle y Postgres. Es por ello que en esta tesis se utilizará esta estructura para almacenar los conjuntos de puntos.

Cada nodo corresponde a un bloque de disco. Los nodos hojas contienen entradas de la forma <MBR, *oid*>, donde MBR es un rectángulo mulidimensional que corresponde al mínimo rectángulo que encierra al objeto espacial y *oid* es el identificador del objeto espacial en la Base de Datos. Los nodos internos (nodos no-hojas) contienen entradas de la forma <MBR, *ref*>, donde MBR es el rectángulo mínimo que contiene a todos los rectángulos definidos en las entradas del nodo hijo y ref es la dirección del correspondiente nodo hijo en el R-tree [21].

2.7.1. Propiedades

Sea M el número máximo de entradas que puede almacenar un nodo y sea m el número mínimo de entradas de un nodo, un R-tree satisface las siguientes propiedades [22]:

- El número mínimo de entradas es $m < \frac{M}{2}$.
- Cada nodo hoja contiene entre m y M entradas, a menos que sea la raíz.
- Para cada entrada <MBR, oid >, MBR es el mínimo rectángulo que contiene (espacialmente) al objeto espacial.
- Cada nodo interno contiene entre m y M hijos, a menos que sea la raíz.
- Para cada entrada <MBR, ref > de un nodo interno, MBR es el mínimo rectángulo que contiene (espacialmente) a los rectángulos definidos en los nodos hijos.
- El nodo raíz contiene al menos dos hijos, a menos que sea una hoja.
- Todas las hojas se encuentran en el mismo nivel.



(a) Agrupaciones de MBRs generadas por un R-tree.

(b) R-tree de los rectángulos de (a).

Fig. 2.7: Ejemplo de un R-tree, extraído de [21].

2.7.2. Consultas básicas en un R-tree

En la presente sección mostraremos cómo el R-tree da respuesta a algunas consultas básicas de una BDE, como son inserción, eliminación y búsqueda. Esta sección está basada en el trabajo de Gutiérrez [21], los algoritmos han sido extraídos del mismo trabajo.

Inserción

La inserción de un objeto o en un R-tree consiste en insertar el objeto en las hojas. Si el nodo excede su capacidad máxima de almacenamiento, éste se divide, enviando un nuevo elemento al padre. Si el padre excede su capacidad máxima de almacenamiento, se dividirá, enviando un

nuevo elemento a su padre, este proceso se repitirá recursivamente hasta llegar al nodo raíz. Si el nodo raíz se divide, se generará una nueva raíz. En Alg. 2.1 se observa el pseudocódigo del algoritmo de inserción del R-tree.

```
1  Alg.: InsercionEnRtree(T,o)
   Data: T: nodo del R-tree, o: objeto a insertar.
2  L = SeleccionarHoja(T,o);
3  if L tiene suficiente espacio para almacenar a o then
4  |  insertar o en L, se termina el algoritmo;
5  else
6  |  LL = dividirNodo(L);
7  |  AjustarArbol(L, LL);
8  |  if el ajuste del árbol provoca dividir la raíz then
9  |  |  crear una nueva raíz, sus nodos hijos son los nodos generados en la división de la
   |  |  raíz.
10 |  end
11 end
```

Alg. 2.1: Algoritmo de inserción de un R-tree.

Eliminación

La eliminación de un objeto o de un R-tree consiste en realizar una búsqueda del nodo hoja que contiene el objeto. Luego, se elimina la entrada del nodo hoja, si dicho nodo queda con menos entradas que las permitidas, las entradas se reubican en los nodos hoja restantes. La eliminación se propaga desde las hojas hasta la raíz del árbol, de ser necesario se ajustan los MBRs de los nodos. En Alg. 2.2 se observa el pseudocódigo del algoritmo de eliminación del R-tree.

```
1  Alg.: EliminacionEnRtree(T,o)
   Data: T: nodo del R-tree, o: objeto a eliminar.
2  L = SeleccionarHoja(T,o);
3  if L = ∅ then
4  |  no se encontró el objeto o en el R-tree, se lanza un error;
5  else
6  |  remover o de L;
7  |  CondensarRtree(L);
8  |  if después de condensar el R-tree, la raíz tiene un sólo nodo hijo then
9  |  |  el nodo hijo se convierte en la nueva raíz.
10 |  end
11 end
```

Alg. 2.2: Algoritmo de eliminación de un R-tree.

Búsqueda

La búsqueda en un R-tree se inicia en la raíz del árbol, descendiendo hasta alcanzar las hojas, en forma similar al algoritmo realizado por un B-tree.

Para una consulta Windows Query(WQ), en los nodos internos se decide por cuáles hijos continuar la búsqueda (debido a que la partición del espacio no es disjunta, es posible seguir más de un camino desde un nodo interno a las hojas), la decisión consiste en verificar si el MBR del nodo hijo interseca con el objeto de búsqueda q. En Alg. 2.3 se observa el algoritmo en pseudocódigo.

```
1  Alg.: BusquedaEnRtree(T,q)
   Data: T: nodo del R-tree, q: objeto de búsqueda.
   Result: L: conjunto de punteros a objetos cuyos MBRs se intersecan con q.
2  L = ∅;
3  for cada entrada e ∈ T do
4  |  if MBR(e) ∩ q ≠ ∅ then
5  |  |  if T es una hoja then
6  |  |  |  L = L ∪ {e};
7  |  |  end
8  |  else
9  |  |  L = L∪ BusquedaEnRtree(e, q);
10 |  end
11 end
12 return L;
```

Alg. 2.3: Algoritmo de búsqueda de un R-tree.

Capítulo 3

Separabilidad bicromática

3.1. Introducción

Como indicamos la separabilidad bicromática de objetos consiste en dar solución al siguiente problema [24]: supongamos dos conjuntos de puntos en el espacio, clasificados como puntos rojos y puntos azules. ¿Existe cierto objeto geométrico con determinadas propiedades que separe los puntos rojos de los puntos azules?

Este problema posee diversas variantes, dependiendo del objeto geométrico que se utilice (una línea recta, una cuña, un disco, un cono, etc.) y si los conjuntos resultantes de la separación son monocromáticos o bicromáticos.

Los problemas de separabilidad de objetos poseen aplicaciones en diversos campos, por ejemplo: análisis de imágenes, estadísticas y campos en los cuales la discriminación y/o clasificación es requerida [36].

En el presente capítulo se abordará la separabilidad de conjuntos de puntos, los criterios descritos pueden ser usados en otros tipos de conjuntos. Debido a que en esta tesis se consideran puntos en un espacio de dos dimensiones, este capítulo está enfocado en problemas de separabilidad en $\mathbb{R}^2$. Si el lector desea revisar la separabilidad de objetos en dimensiones mayores, se sugiere visitar las referencias de la Sección 3.4, donde se mencionan problemas en $\mathbb{R}^3$.

El presente capítulo está estructurado de la siguiente forma. En la Sección 3.2 se define la separabilidad bicromática y se exponen algunos tipos de separabilidad. En la Sección 3.3 se discute sobre los problemas de Optimización Geométrica y finalmente, en la Sección 3.4 se muestran algoritmos de Separabilidad que han sido estudiados en el contexto de la GC.

3.2. Separabilidad bicromática

En la presente sección se definirá la separabilidad bicromática de objetos y se expondrán los tipos de separabilidad existentes [36].

Definición 3.1. *Sea S un conjunto de puntos en $\mathbb{R}^d$, con $d \geq 1$, el cual está formado por R (conjunto de puntos rojos) y B (conjunto de puntos azules), tal que $S = R \cup B$. La* ***separabilidad bicromática*** *de objetos busca separar el conjunto rojo del conjunto azul utilizando un objeto*

geométrico que posea determinadas propiedades [36].

Principalmente, se distinguen dos grandes tipos de separabilidad que son la separabilidad estricta y la separabilidad débil. Si luego de realizar la separación, los conjuntos resultantes son monocromáticos, se habla de separabilidad estricta, en caso contrario se trata de separabilidad débil.

Definición 3.2. *Sea R y B dos conjuntos de objetos y o un objeto geométrico. Sí o separa a R y B de tal forma que los conjuntos resultantes son monocromáticos, se habla de* ***separabilidad estricta.***

Definición 3.3. *Sea R y B dos conjuntos de objetos y o un objeto geométrico. Sí o separa a R y B de tal forma que los conjuntos resultantes son bicromáticos, se habla de* ***separabilidad débil.***

A continuación se describirán algunos de los principales problemas de separabilidad bicromática.

3.2.1. Separabilidad lineal

La noción más básica de separabilidad es a través de una línea recta.

Definición 3.4. *Sea R y B dos conjuntos de objetos y ℓ una línea recta (o hiperplano). Si ℓ separa a R y B en forma estricta, se dice que los conjuntos son linealmente separables.*

R y B son linealmente separables sí y solo sí sus cerraduras convexas no se intersecan [41]. Este tipo de problemas, denominados de separabilidad lineal, se puede resolver en tiempo lineal para dimensiones bajas o constantes [23]. En Fig. 3.1 se observa un ejemplo de separabilidad lineal.

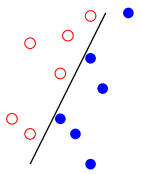


Fig. 3.1: Separabilidad lineal.

3.2.2. Separabilidad por cuña y doble cuña

Una cuña es la unión de dos rayos con un origen común.

Definición 3.5. *Dos conjuntos de objetos R y B son separables por cuña si existe una cuña que contenga solo los objetos de uno de los conjuntos [36].*

En Fig. 3.2 (a) se puede apreciar un ejemplo de separabilidad por cuña. Si el ángulo que forman los rayos de la cuña tiene una amplitud de π, entonces los conjuntos son linealmente separables.

La separabilidad por doble cuña consiste en encontrar dos líneas que intersequen en un punto p, el vértice de la doble cuña, de tal forma que particionen a R en R_1 y R_2 y a B en B_1 y B_2, como se muestra en Fig. 3.2 (b).

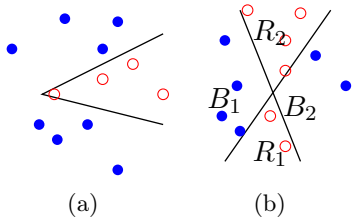


Fig. 3.2: Separabilidad por: (a) cuña y b) doble cuña.

3.2.3. Separabilidad por banda

Una banda se define como la unión de dos líneas paralelas [36].

Definición 3.6. *Dos conjuntos de objetos R y B son separables por banda si existen dos líneas paralelas que contengan entre ellas solo a todos los objetos de uno de los conjuntos [36].*

En la Fig. 3.3 vemos un ejemplo de separabilidad por banda, donde el conjunto de puntos rojos se encuentra entre las dos líneas que definen la banda. La separabilidad por banda requiere que una de las cerraduras convexas de uno de los conjuntos sea monocromática, en el ejemplo de la Fig. 3.3 es claro que se cumple esta condición.

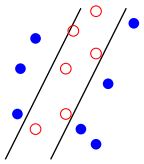


Fig. 3.3: Separabilidad por banda.

3.2.4. Separabilidad por figuras geométricas convexas

Dado dos conjuntos de objetos, R y B, otro críterio de separabilidad es a través de figuras geométricas convexas (círculos, polígonos convexos, etc).

Un ejemplo es la separabilidad circular, la cual consiste en encerrar uno de los conjuntos (R o B) en un disco, cuyo interior no sea intersecado por el otro conjunto [9]. Este tipo de separabilidad se muestra en Fig. 3.4.

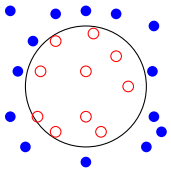


Fig. 3.4: Separabilidad circular.

3.3. Optimización Geométrica

Existen problemas en los cuales no es posible conseguir una separación estricta con algún objeto geométrico, por lo que se busca maximizar el número de puntos cubiertos por los objetos. Estos problemas se denominan problemas de *Optimización Geométrica*, ya que existe una función objetivo a optimizar, en este caso el número de puntos que pueden ser separados. Este tipo de problemas pueden ser interpretados de la siguiente forma: "Eliminar el menor número posible de puntos, de tal forma que los puntos restantes sean separables según el criterio (objeto geométrico) seleccionado". Los puntos eliminados no se consideran en el proceso de separación.

El Problema de la Caja Máxima [2], consiste en encontrar una caja isométrica H, tal que $H \cap R = \emptyset$ y la cardinalidad de $H \cap B$ sea maximizada. Este problema se puede ampliar al uso de dos cajas, cuyo objetivo es encontrar una caja para cada color, las cuales deben maximizar la cantidad de puntos cubiertos [14]. Cortés et al. [31] estudian criterios de separabilidad bicromática con dos cajas. En Fig. 3.5 se observa un ejemplo de separabilidad bicromática usando dos cajas.

El problema de los cuadrados unitarios, consiste en encontrar dos cuadrados isométricos unitarios, los cuales cubran la máxima cantidad de puntos. Mahapatra et al. [26] estudia el problema de encontrar cuadrados disjuntos, cuadrados no disjuntos y encontrar los k cuadrados disjuntos.

En la literatura se encuentran problemas que no requieren una separabilidad estricta, o que sea posible eliminar puntos del conjunto con el objetivo de que los restantes sean separables según un determinado criterio. Tal es el caso del problema de los dos discos disjuntos propuesto por Díaz-Bánez et al. [15], el cual consiste en encontrar la posición de dos discos disjuntos C_R y C_B, coloreados rojo y azul respectivamente y que maximice la cantidad de puntos rojos cubiertos por C_R más la cantidad de puntos azules cubiertos por C_B. En el mismo trabajo se presenta un problema similar; el problema de los dos cuadrados unitarios alineados a los ejes, en el cual, en forma análoga al problema de los discos, se deben localizar dos cuadrados, S_R y S_B, con el objetivo de maximizar la cantidad de puntos rojos cubiertos por S_R más la cantidad de puntos azules cubiertos por S_B. En un trabajo posterior [11], se presentó una mejora a las soluciones de estos problemas y se presentó un problema similar al de los cuadrados eliminando la restricción de que estos deban ser alineados a los ejes.

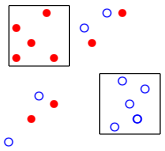


Fig. 3.5: Separabilidad por caja.

3.4. Algoritmos de separabilidad

En la literatura se encuentran diversas técnicas para resolver el problema de la separabilidad de objetos. En la Tabla 3.1 se muestran algunos de los principales algoritmos de separabilidad en GC, los cuales consideran que todos los objetos están en memoria principal. Podemos observar que al aumentar en una dimensión el espacio (de $\mathbb{R}^2$ a $\mathbb{R}^3$), los algoritmos de separabilidad aumentan el orden de su complejidad temporal.

De acuerdo a la revisión de la literatura realizada, no existen algoritmos que resuelvan el problema de separabilidad de objetos que consideren que los objetos se encuentran almacenados en memoria secundaria.

Debido a que en esta tesis se aborda el problema de la separabilidad lineal, en esta sección se expondrán los algoritmos que ya han sido propuestos en la GC.

3.4.1. Algoritmos de separabilidad lineal

En esta sección se describe el modelado del problema de separabilidad lineal como un problema de Programación Lineal (PL), basado en el trabajo desarrollado por Houle [24]. El problema de PL se puede resolver en tiempo $O(n)$, según lo propuesto por Meggido [27]. El método propuesto por Meggido no se explicará en esta sección, debido a que escapa del ámbito de esta tesis.

Sea $P = \{p_1, p_2, ..., p_n\}$ un conjunto de n puntos etiquetados en $\mathbb{R}^d$ y sea w_i un peso asociado con p_i, para todos los $i = 1, 2, ..., n$. Sea la (d+1)-tupla $h = \tilde{h}, h_{d+1} \in \mathbb{R}^{d+1}$ es el hiperplano en $\mathbb{R}^d$ descrito por $\{x \in \mathbb{R}^d \mid h \cdot (x, 1) = 0\}$, usando coordenadas cartesianas, donde $\tilde{h}$ es el vector normal no cero (h_1, h_2, ..., h_d). Dado alguna (d+1)-tupla z, nosotros decimos que $\tilde{z}$ es la tupla formada al tomar las primeras d coordenadas de z. Si dos (d+1)-tuplas α y β representan el mismo hiperplano, debemos decir que α y β son *equivalentes* y denotar esto por $\alpha \equiv \beta$. La siguiente sencilla observación ilustra el grado de libertad en la selección de la (d+1)-tupla para representar un hiperplano dado:

Observación 3.1. *Sea α un hiperplano en $\mathbb{R}^d$.*

1. *La (d+1)-tupla β es equivalente a α sí y solo sí existe algún $t \neq 0$ tal que $\beta = t\alpha$, y*
2. *Dado algún $k > 0$, existe alguna (d+1)-tupla β tal que $\alpha \equiv \beta$ y $||\tilde{\beta}1|| = k$.*

Algoritmo	**Complejidad Temporal**	**Ref.**
Separación lineal	$O(n)$	[23, 36]
Separación por cuña	$O(n \log n)$	[36]
Separación por banda	$O(n \log n)$	[36]
Separación por doble cuña	$O(n \log n)$	[25, 36]
Separación por doble banda	$O(n \log n)$	[36]
Separabilidad circular	$O(n \log n)$	[36]
Separación bicromática por cajas	$O(n^2 \log n)$	[32]
Encontrar 3 cajas para separar 3 conjuntos de puntos en $\mathbb{R}^2$	$O(n \log n)$	[32]
Encontrar 3 cajas para separar 3 conjuntos de puntos en $\mathbb{R}^3$	$O(n^4 \log n)$	[32]
Problema de la caja máxima	$O(n \log^3 n)$	[3]
Problema de la máxima suma	$O(n^2)$	[4]
Separación por prisma convexo en $\mathbb{R}^3$	$O(n \log^4(n))$	[1]
Separación por slab en $\mathbb{R}^3$	$O(n^2 \log(n))$	[1]
Separación por cuña en $\mathbb{R}^3$	$O(n^2 polylog(n))$	[1]
Separación por doble cuña en $\mathbb{R}^3$	$O(n^3 polylog(n))$	[1]
El problema de las dos monedas	$O(n^{(8/3)} \log^2 n)$	[11]
El problema de los dos cuadrados unitarios disjuntos alineados a los ejes	$O(n \log n)$	[11]
El problema de los dos cuadrados unitarios disjuntos arbitrariamente orientados	$O(n^3 \log n)$	[11]
Separabilidad por 3 líneas paralelas	$O(n \log n)$	[36]
Separabilidad por polígono	$O(n \log n)$	[36]
Separación por L-shapes monocromáticos	$O(n^2 \log n)$	[39]
Separación por dos círculos	$O(n^2)$	[18]

Tabla 3.1: Algoritmos de Separabilidad en GC.

La distancia ortogonal Euclídea entre un punto $x \in \mathbb{R}^d$ y un hiperplano h es dada por [10]

$$\sigma(x,h) = \frac{|(x,1)\cdot h|}{\|\tilde{h}\|}$$

Si un punto p tiene un peso $\omega > 0$ asignado a él, entonces la distancia ortogonal Euclídea con peso entre x y h está dada por

$$\omega\sigma(x,h) = \omega\frac{|(x,1)\cdot h|}{\|\tilde{h}\|}$$

La expresión $\omega\frac{|(x,1)\cdot h|}{\|\tilde{h}\|}$ tiene la misma magnitud que $\omega\sigma(x,h)$, pero el signo depende sobre cuál semi-espacio abierto de h contiene a x.

Sea $P = \{p_1, p_2, ..., p_n\}$ un conjunto de n puntos etiquetados en $\mathbb{R}^d$ y sea w_i un peso asociado con p_i. Sea R el conjunto de puntos rojos y B el conjunto de puntos azules. Usando una técnica usada en reconocimiento de patrones para obtener funciones de discriminación lineal [16], podemos eliminar lo necesario para distinguir entre R y B en la descripción matemática del problema. Esto se logra transformando P en un conjunto de (d+1)-tuplas $\mu(P) = \{\mu(p_1), \mu(p_2), ..., \mu(p_n)\}$, donde $\mu(p_i) = \chi(p_i)\omega_i(p_i, 1)$. Claramente, si la inecuación

$$\frac{\mu(p_i)\cdot h}{\|\tilde{h}\|} > 0$$

es verdadera para todos los $i = 1, 2, ..., n$, entonces h es un separador lineal estricto para R y B. A la inversa, si R y B son linealmente separables, entonces su separador tiene alguna parametrización h que satisface la inecuación para todos los i.

El problema de encontrar un separador lineal estricto para P es entonces reducido al problema de encontrar una (d+1)-tupla h que satisfaga

$$\text{maximizar } \min_i \frac{\mu(p_i)\cdot h}{\|\tilde{h}\|} \tag{3.1}$$

Si el valor óptimo de este problema es negativo o cero, entonces los puntos de P no pueden ser separados por un hiperplano. En otro caso, el valor óptimo es la distancia ortogonal mínima desde h a los puntos de P.

Ahora, establecemos una correspondencia entre 3.1 y el siguiente problema de minimización cuadrático con n rectricciones y $d+1$ variables:

$$\begin{aligned} &\text{minimiza } \|\tilde{h}\|^2 \\ &\text{sujeto a } \mu(p_i)\cdot h \geq 1 \end{aligned}$$

Los problemas de minimización cuadráticos con m variables y n restricciones pueden ser resueltos en tiempo y espacio $O(n)$, asumiendo que m es fijo [27].

Parte III

Separabilidad lineal sobre objetos almacenados en R-trees distintos

Capítulo 4

Separabilidad lineal en conjuntos almacenados en R-trees distintos

4.1. Introducción

La separabilidad lineal constituye la forma más intuitiva y simple de separabilidad, usando una línea recta (o hiperplano) como objeto de separación. Como se mencionó en la Sección 3.4, en la literatura revisada no se han encontrado algoritmos que consideren que los objetos (puntos en nuestro caso), estén almacenados en memoria secundaria, es por ello que nuestra investigación consiste en determinar la separabilidad lineal en conjuntos almacenados en memoria secundaria.

La estructura de este capítulo es la siguiente. En la Sección 4.2 definiremos el problema de separabilidad que abordaremos en esta tesis. Luego, en la Sección 4.3 expondremos algunos conceptos y métodos que son la base de nuestras propuestas.

4.2. Definición del problema

El problema que abordamos es el siguiente: sea S un conjunto de puntos en $\mathbb{R}^2$, el cual está formado por los conjuntos R (puntos rojos) y B (puntos azules), tal que $R \cup B = S$. Los conjuntos R y B se encuentran almacenados en R-trees distintos. Se desea encontrar una recta ℓ, que divida el espacio en 2 regiones, de tal forma que los puntos rojos y azules estén separados, es decir, cada región contiene puntos de un solo color. Llamaremos a este problema "Problema de Separabilidad Lineal en R-trees distintos" (PSLR).

4.3. Preliminares

En esta sección se expondrán algunos conceptos que son la base para los algoritmos propuestos.

Definición 4.1. *Sea $MBR(N)$ el MBR (Minimun Bound Rectangle) que envuelve al conjunto N.*

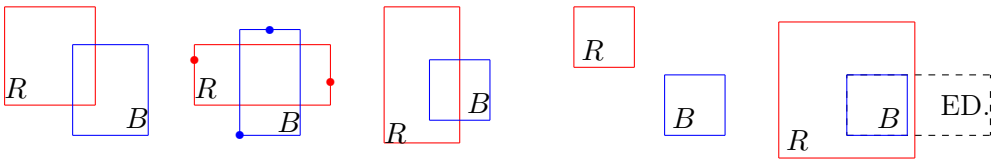
4.3.1. Intersección de los conjuntos

Debido a que los conjuntos de puntos rojos y azules están almacenados en R-trees, R y B respectivamente, podemos observar que existen 5 posibles relaciones entre $MBR(R)$ y $MBR(B)$: corner, sandwich, semi-disjunto, disjunto e intersección completa (ver Fig. 4.1) [14].

Definición 4.2. *Sea $MBR(R)$ y $MBR(B)$, podemos encontrar las siguientes relaciones entre ellos:*

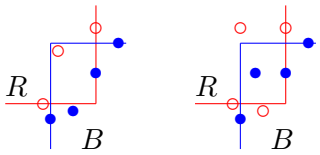
- *$MBR(R)$ y $MBR(B)$ poseen una intersección corner si cada MBR contiene exactamente un vértice del otro MBR, como se aprecia en la Fig. 4.1 (a), o uno de los MBR está contenido en otro y comparten un solo vértice.*
- *$MBR(R)$ y $MBR(B)$ poseen una intersección sandwich si el interior de los MBR no es disjunto y ningún MBR contiene en su interior un vértice del otro MBR, como se aprecia en la Fig. 4.1 (b).*
- *$MBR(R)$ y $MBR(B)$ poseen una intersección semi-disjunta si uno de los MBR contiene exactamente dos vértices del otro MBR, como se aprecia en la Fig. 4.1 (c).*
- *$MBR(R)$ y $MBR(B)$ poseen una intersección disjunta si los MBRs no se intersecan, como se aprecia en la Fig. 4.1 (d).*
- *$MBR(R)$ y $MBR(B)$ poseen una intersección completa si un MBR contiene los cuatro vértices del otro MBR, como se aprecia en la Fig. 4.1 (e).*

Sí $MBR(R) \cap MBR(B) = \emptyset$, entonces los conjuntos son fácilmente separables, por lo que la solución es trivial. En el caso de la intersección tipo sandwich, podemos observar que los conjuntos no son linealmente separables, ya que como se observa en la Fig. 4.1 (b) debe existir a lo menos un punto en cada arista de los MBRs, por lo que no es factible establecer una línea que divida ambos conjuntos de forma tal que cada región contenga puntos de un solo color. En el caso de la intersección completa (Fig. 4.1 (e)), si expandimos el MBR del conjunto B a la derecha, por ejemplo en Fig. 4.1 (e) el rectángulo ED es la expansión a la derecha del MBR del conjunto B, obtenemos una intersección semi-disjunta, por lo que para analizar este caso trabajaremos las cuatro posibilidades de intersección semi-disjunta (expansión a la izquierda, derecha, arriba y abajo del MBR interior) por separado. En las intersecciones de tipo corner y semi-disjunta no es posible decidir si los conjuntos son linealmente separables utilizando solo los MBRs, por lo que es necesario descender por los árboles y recuperar sólo aquellos puntos que inciden en la decisión final. Como se aprecia en Fig. 4.2 (a), los conjuntos poseen una intersección corner. Si examinamos los puntos nos damos cuenta que los conjuntos sí son linealmente separables. En Fig. 4.2 (b) vemos otro ejemplo de intersección corner, pero en este caso los conjuntos no son linealmente separables.



(a) Corner. (b) Sandwich. (c) Semi-Disjunto. (d) Disjunto. (e) Intersección completa.

Fig. 4.1: Tipos de intersección de rectángulos.



(a) (b)

Fig. 4.2: Conjuntos (a) linealmente separable y (b) no linealmente separable.

4.3.2. Problema de Separabilidad Lineal

Como se mencionó en el Capítulo 3, en la GC existen algoritmos que resuelven el Problema de Separabilidad Lineal, los cuales usan diversas técnicas, tales como:

- Modelado del problema como un problema de programación lineal.
- Cálculo de la intersección de las cerraduras convexas de ambos conjuntos.

Si consideramos el conjunto R de tamaño n y el conjunto B de tamaño m, los algoritmos anteriormente descritos resuelven el Problema de Separabilidad Lineal en tiempo $O(n+m)$ para dimensiones bajas. Estos algoritmos requieren que todos los puntos estén almacenados en memoria principal, por lo que un primer algoritmo para resolver el PSLR es llevar todos los puntos a memoria principal y utilizar uno de estos Algoritmos de Geometría Computacional (AGC). Como se puede preveer, este algoritmo *ingenuo* es ineficiente, ya que leer todos los R-tree es costoso en término de tiempo y espacio y en conjuntos muy grandes es inviable, dado que el espacio en memoria principal no puede almacenar todos los conjuntos. Es por ello que en los siguientes capítulos se proponen 2 algoritmos que utilizan las propiedades del R-tree para determinar la separabilidad lineal de dos conjuntos, sin la necesidad de leer todos los bloques (nodos) del R-tree.

Capítulo 5

Algoritmo de ramificación y poda

5.1. Introducción

Los algoritmos de ramificación y poda son usados para explorar grafos dirigidos (por ejemplo, un árbol) con el fin de buscar una solución óptima a un problema determinado sin tener que analizar todas las soluciones posibles [12]. En el presente capítulo se presenta un método para solucionar el PSLR, que tiene como objetivo realizar un filtrado de los puntos, para luego calcular la separabilidad de los conjuntos utilizando un AGC y los puntos no filtrados.

Este capítulo se estructura de la siguiente forma: en la Sección 5.2 se describe el algoritmo propuesto. Luego, en la Sección 5.3 se explica el pseudocódigo del algoritmo. En la Sección 5.4 se detallan las pruebas experimentales y, finalmente, en la Sección 5.5 se exponen las conclusiones de los resultados obtenidos.

5.2. Descripción

En forma general, nuestro algoritmo de ramificación y poda accede a una pequeña fracción de los nodos del R-tree, descartando la mayor cantidad de puntos posibles, para luego procesar los puntos no descartados en memoria principal con un AGC. Como una primera aproximación a la idea detrás de nuestro algoritmo, analicemos los conjuntos R, puntos rojos, y B, puntos azules, mostrados en la Fig. 5.1 (a), los cuales están en un espacio de una dimensión. Para realizar un filtrado de los puntos, debemos identificar la relación existente entre los conjuntos, en el caso de los conjuntos mostrados en Fig. 5.1 (a), los puntos rojos están ubicados a la izquierda de los puntos azules, por lo tanto, la recta ℓ que separa ambos conjuntos debería estar a la derecha del conjunto rojo y a la izquierda del conjunto azul, como se observa en Fig. 5.1 (b). Si observamos la Fig. 5.1 (b), notamos que los puntos p_1 y p_2 son los puntos más cercanos al otro conjunto, y la posición de estos afecta la ubicación de ℓ, por lo que los denominamos **puntos relevantes**. Finalmente, si solo consideramos los puntos relevantes, unido a la relación entre los conjuntos, podemos encontrar ℓ (o descubrir que los conjuntos no son separables), sin necesidad de analizar todos los puntos de los conjuntos, tal como se aprecia en Fig. 5.1 (c). Mediante la utilización de los puntos relevantes (p_1 y p_2) podemos determinar que ℓ existe y se encuentra a la derecha de

p_1 y a la izquierda de p_2.

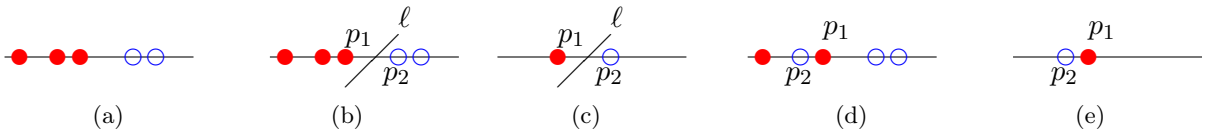


Fig. 5.1: Ejemplo.

En la Fig. 5.1 (d) observamos un conjunto que no es linealmente separable. Los puntos relevantes son p_1 y p_2, y el conjunto rojo está a la izquierda del azul. Si filtramos los puntos y solo analizamos los puntos relevantes, como se aprecia en Fig. 5.1 (e), los conjuntos no son separables, ya que la recta que separa los puntos pasa a la derecha del punto azul, lo cual no es factible dado que el conjunto rojo está a la izquierda del conjunto azul.

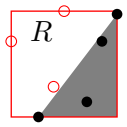
Definición 5.1. *El conjunto de vértices V_R del conjunto R corresponde a un subconjunto de los vértices del $MBR(R)$, es decir, $V_R \subseteq vertices(MBR(R))$. En forma análoga, el conjunto de vértices V_B del conjunto B corresponde a un subconjunto de los vértices del $MBR(B)$, es decir, $V_B \subseteq vertices(MBR(B))$.*

Si nuestro problema lo trasladamos a un espacio de dos dimensiones, es decir $R \cup B \in \mathbb{R}^2$, debemos analizar el tipo de intersección que poseen $MBR(R)$ y $MBR(B)$. Según lo descrito en la Seción 4.3.1 solo nos interesan los casos en los cuales $MBR(R)$ y $MBR(B)$ poseen una intersección corner o semi-disjunta. Una vez determinado el tipo de intersección que poseen $MBR(R)$ y $MBR(B)$, se debe definir el conjunto de vértices V_R y V_B y dos cotas (superior e inferior) para cada conjunto, con los cuales se forma la *zona relevante*, en la cual están contenidos los *puntos relevantes*, necesarios para calcular la separabilidad lineal de los conjuntos. Las cotas corresponden a puntos ubicados en las aristas de los MBR y son los puntos más cercanos a alguno de los vértices de los conjuntos V_R y V_B que cumplan ciertas reglas explicadas en la subsecciones siguientes. Debido a que el MBR es el mínimo rectángulo que encierra al conjunto de puntos, su definición garantiza la existencia de por lo menos un punto en cada arista del MBR. Para determinar la cota superior e inferior basta con realizar un recorrido hasta las hojas de cada R-tree.

Definición 5.2. *La zona relevante (Z_r) del conjunto R (resp. B) corresponde a la zona cubierta por el polígono formado por el conjunto de vértices V_R (resp. V_B) y las cotas superior e inferior del conjunto. En Fig. 5.2 la zona sombreada corresponde a Z_r para el conjunto R con intersección corner.*

Definición 5.3. *El conjunto de puntos relevantes (P_r) corresponde a todos los puntos p tal que $p \cap Z_r \neq \emptyset$. Además, $P_r \subseteq N$, siendo $N = R \cup B$. En Fig. 5.2 los puntos negros corresponden al conjunto P_r para el conjunto R con intersección corner.*

Los puntos relevantes corresponden a los puntos no descartados por nuestro algoritmo y, junto con el tipo de intersección de los MBR, son utilizados para calcular la separabilidad lineal de los conjuntos por medio de un AGC.



(a)

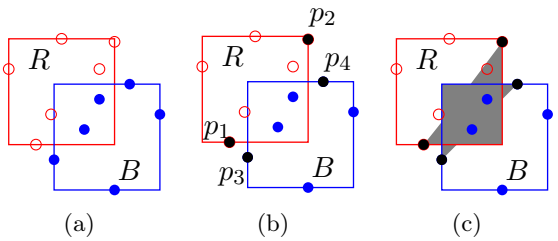
Fig. 5.2: Z_r (zona sombrada) y P_r (puntos negros) del conjunto R en una intersección tipo corner.

5.2.1. Intersección tipo corner

En esta Sección explicaremos cómo calcular Z_r para $MBR(R)$ y $MBR(B)$ con intersección corner. A modo de ejemplo utilizaremos los conjuntos R y B de la Fig. 5.3 (a); otros casos se resuelven en forma análoga. Para el conjunto R, la cota superior es el punto más abajo que interseca la arista derecha (en nuestro ejemplo corresponde al punto p_2) y la cota inferior es el punto más a la izquierda que interseca la arista inferior (en nuestro ejemplo corresponde al punto p_1) de $MBR(R)$ (ver Fig.5.3 (b)). El conjunto V_R está formado por el vértice inferior derecho. La zona relevante del conjunto R corresponde al triángulo formado por las cotas (superior e inferior) y el vértice inferior derecho.

En el conjunto B, la cota superior es el punto más a la izquierda que interseca la arista superior del MBR (en nuestro ejemplo corresponde al punto p_4) y la cota inferior es el punto más arriba y que interseca la arista izquierda de $MBR(B)$ (en nuestro ejemplo corresponde al punto p_3) (ver Fig.5.3 (b). El conjunto V_B está formado por el vértice superior izquierdo. La zona relevante corresponde al triángulo formado por las cotas (superior e inferior) y el vértice superior izquierdo.

La zona relevante de ambos conjuntos se puede apreciar en la Fig. 5.3 (c).



(a) (b) (c)

Fig. 5.3: Intersección corner.

5.2.2. Intersección tipo semi-disjunta

En esta Sección explicaremos cómo calcular Z_r para $MBR(R)$ y $MBR(B)$ con intersección semi-disjunta, a modo de ejemplo utilizaremos los conjuntos R y B de la Fig. 5.4 (a). Otros casos se resuelven en forma análoga. La cota superior del conjunto R corresponde al punto más a la derecha que interseca a la arista superior (en nuestro ejemplo corresponde al punto p_2) y la cota inferior es el punto más a la derecha que interseca a la arista inferior (en nuestro ejemplo corresponde al punto p_1) de $MBR(R)$, como se aprecia en Fig. 5.4 (b). El conjunto V_R está formado por el vértice superior derecho y el vértice inferior derecho. La zona relevante corresponde al polígono formado por las cotas (superior e inferior) y los vértices del V_R. En forma análoga, la cota superior del conjunto B corresponde al punto más a la izquierda que interseca a la arista superior (en nuestro ejemplo corresponde al punto p_4) y la cota inferior es el punto más a la izquierda que interseca a la arista inferior (en nuestro ejemplo corresponde al punto p_3) de $MBR(B)$, como se aprecia en la Fig. 5.4 (b). El conjunto V_R está formado por el vértice superior izquierdo y el vértice inferior izquierdo. La zona relevante corresponde al polígono formado por las cotas (superior e inferior) y los vértices del V_R.

La zona relevante de ambos conjuntos se puede apreciar en la Fig. 5.4 (c).

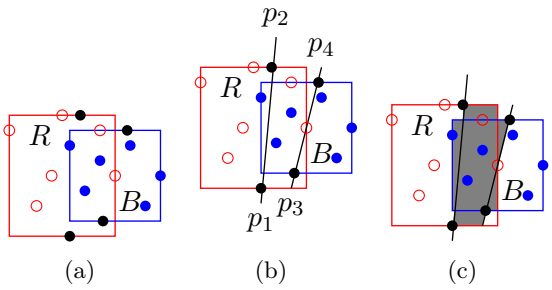


Fig. 5.4: Intersección semi-disjunto.

5.2.3. Cálculo de separabilidad lineal

Una vez calculada Z_R, si el MBR representa a un nodo interno del R-tree, los MBRs hijos que intersequen a Z_R son nuevamente procesados utilizando el proceso de filtrado, es decir, se calcula nuevamente Z_R en forma recursiva en los hijos hasta llegar a los nodos hoja. Notar que el tipo de intersección es obtenido solo al inicio del algoritmo, en el descenso por el árbol se utiliza la intersección original para calcular Z_R.

Si el MBR está formado por puntos, se almacenan en memoria principal el conjunto de puntos P_r, el cual es procesado con un AGC, por ejemplo, utilizar el algoritmo de Programación Lineal propuesto por Meggido [27], el cual calcula la separabilidad lineal en tiempo lineal.

5.3. Algoritmo

En Alg. 5.1 podemos ver el algoritmo de separación. En las líneas 2 y 3 se obtienen los MBR de los nodos R_R y R_B respectivamente. En la línea 4 se obtiene el tipo de intersección, de acuerdo a lo descrito en la Sección 4.3.1. Si el tipo de intersección es disjunta, se puede obtener una recta que separe los conjuntos (líneas 5 y 6). Si la intersección es del tipo corner o semi-disjunta debemos aplicar el algoritmo de filtrado (líneas 8, 9 y 10), para finalmente usar el AGC con los puntos relevantes encontrados (línea 25). Si la intersección es completa, debemos expandir el conjunto interior a la izquierda (línea 13) para obtener una intersección semi-disjunta, ejecutando nuevamente el algoritmo en forma recursiva (línea 13). Si no se encuentra una recta, se debe expandir el conjunto a la derecha (línea 18) y ejecutar nuevamente el algoritmo de separabilidad (línea 18). Si la intersección es sandwich, los conjuntos no son separables (línea 22).

El algoritmo de filtrado para un R-tree se muestra en Alg. 5.2. En la línea 2 obtenemos las cotas (superior e inferior) del MBR, con las cuales generamos la zona relevante (línea 3). Para cada entrada del nodo (líneas 4 a la 17), verificamos si es un nodo hoja (línea 5). En este caso para cada punto contenido en la hoja (líneas 6 a la 11), accedemos al punto (línea 7) y si éste interseca a la zona relevante (línea 8) es ingresado al conjunto de los puntos relevantes (línea 9). Si la entrada es un nodo interno, verificamos si interseca a la zona relevante (línea 13). En tal caso ejecutamos en forma recursiva el algoritmo de filtrado (línea 14) hasta llegar a las hojas del árbol.

5.4. Experimentación

En esta sección se describe la ejecución de un conjunto de experimentos, que tienen como objetivo comprobar el rendimiento del algoritmo.

Los experimentos consideran conjuntos de datos sintéticos de 1, 2, 5, 10 millones de puntos por cada conjunto (árbol de puntos rojos y árbol de puntos azules). Se considera una intersección de las zonas o regiones cubiertas por los conjuntos de un 1, 2, 5 y 50 %. Los puntos se distribuyen uniformemente en el espacio $[0,1{,}5] \times [0,1]$. La cantidad de bloques de disco que poseen los R-trees se muestra en la Tabla 5.1. Las mediciones realizadas fueron: cantidad de nodos accedidos, cantidad de puntos recuperados y el porcentaje de filtrado.

5.4.1. Implementación

El algoritmo fue implementado en el lenguaje C++, utilizando la librería LibSpatialIndex (`http://libspatialindex.org/`) que implementa el R-tree. Los experimentos se ejecutaron en un computador Lenovo ThinkPad x240 (8GB de memoria RAM, procesador Intel Core i5 4300U).

5.4.2. Discusión de los resultados

Durante las pruebas se realizaron mediciones del porcentaje de nodos accedidos por el algoritmo (Fig. 6.8 (a)), porcentaje de puntos relevantes (no filtrados) (Fig. 6.8 (b)) y la cantidad de puntos filtrados (Fig. 6.8 (c)).

```
1  Alg.: AlgSep(R_R,R_B)
   Data: R_R: R-Tree de puntos rojos, R_B: R-Tree de puntos azules.
   Result: recta que separa los conjunto o null si los conjuntos no son linealmente
           separables.
2  MBR_R = MBR(R_R)
3  MBR_B = MBR(R_B)
4  tipoInterseccion = getInterseccion(MBR_R,MBR_B)
5  if tipoInterseccion == disjunta then
6  |  obtenerSeparacion(R_R,R_B)
7  end
8  if tipoInterseccion == corner OR tipoInterseccion == semi-disjunta then
9  |  filtradoPuntosRojos(R_R, tipoInterseccion, solucion)
10 |  filtradoPuntosAzules(R_B, tipoInterseccion, solucion)
11 else
12 |  if tipoInterseccion == completa then
13 |  |  R_BexI = expandirIzquierda(R_B)
14 |  |  resultado = AlgoritmoDeSeparacion(R_BexI, R_A )
15 |  |  if resultado ≠ null then
16 |  |  |  return resultado
17 |  |  else
18 |  |  |  R_BexD = expandirDerecha(R_B);
19 |  |  |  return AlgoritmoDeSeparacion(R_BexD, R_A )
20 |  |  end
21 |  else
22 |  |  return null
23 |  end
24 end
25 return obtenerSeparabilidadLineal(solucion);
```

Alg. 5.1: Algoritmo de separación.

```
1  Alg.: filtradoPuntosRojos(nodo, tipoInterseccion, solucion)
   Data: nodo: nodo de R-tree, tipoInterseccion: tipo de intersección de los conjuntos,
         solucion: conjunto de puntos no filtrados.
   Result: conjunto de puntos a ser procesados con un algoritmo de Geometría
           Computacional.
2  cotas= getCotas(nodo, tipoInterseccion)
3  ZonaRelevante = generarZonaRelevante( cotas, MBR(nodo) ))
4  forall the entrada e del nodo do
5  |  if esHoja(e) then
6  |  |  forall the entrada e del nodo do
7  |  |  |  punto = getPunto(e)
8  |  |  |  if punto ∩ ZonaRelevante then
9  |  |  |  |  solucion = punto ∪ solucion)
10 |  |  |  end
11 |  |  end
12 |  else
13 |  |  if MBR(e) ∩ ZonaRelevante then
14 |  |  |  filtradoPuntosRojos(hijo(e), tipoInterseccion, solucion)
15 |  |  end
16 |  end
17 end
```

Alg. 5.2: Algoritmo de filtrado.

Tamaño (en mill. de puntos)	Bloques rojos	Bloques azules	Total
1	61.386	61.388	122.774
2	122.210	122.226	244.436
5	304.494	304.858	609.352
10	608.671	609.749	1.218.420

Tabla 5.1: Bloques utilizados por los R-tree.

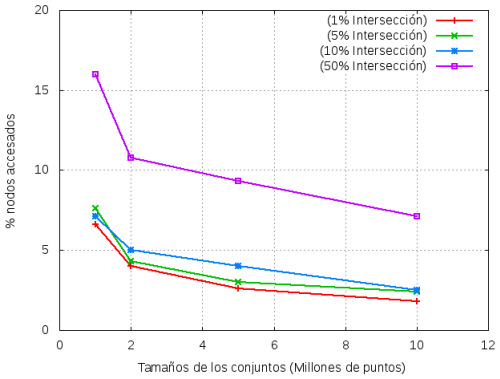
Los resultados indican que al aumentar el porcentaje de intersección de los conjuntos es necesario acceder a una mayor cantidad de nodos del R-tree, alcanzando este paramétro a un 17 % (Fig. 5.5 (a)). Además, se observa que al aumentar la cantidad de puntos para un mismo porcentaje de intersección, disminuye la proporción de nodos del R-tree que es necesario acceder. Por ejemplo, para 10 millones de puntos y un 50 % de intersección es necesario acceder a una pequeña fracción de los nodos del R-tree (alrededor de un 7 %). Notar que un algoritmo que no filtra, requiere acceder todos los nodos del R-tree para recuperar todos los puntos y procesarlos con un AGC.

Con respecto a los puntos relevantes, los que deben ser procesados con el AGC, ocurre un fenómeno similar al anterior. Se observa en Fig. 5.5 (b) y Fig. 5.5 (c) que al aumentar el porcentaje de intersección de las zonas, aumenta la cantidad de puntos recuperados, llegando a un máximo de 0,21 % del total de puntos. Además, al aumentar el tamaño del conjunto de puntos, aumenta la cantidad de puntos relevantes, pero el porcentaje de puntos con respecto al tamaño total tiende a mantenerse, no sufriendo grandes variaciones, es decir, el tamaño del conjunto de puntos relevantes crece en forma casi lineal en la medida que aumenta el tamaño de los conjuntos de puntos.

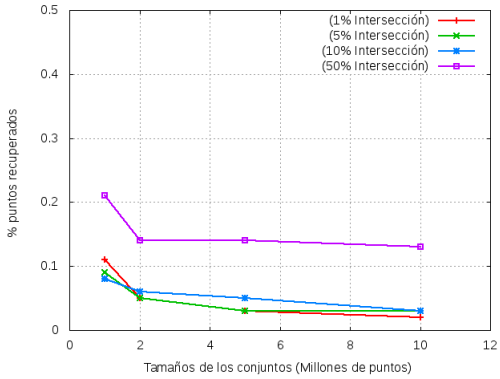
5.5. Conclusiones

En el presente capítulo se propone un método de filtrado que tiene como objetivo disminuir la cantidad de nodos que son accedidos para demostrar si dos conjuntos son linealmente separables cuando estos conjuntos se encuentran almacenados en R-trees distintos. Las pruebas muestran que con nuestro método solo es necesario acceder como máximo a un 17 % de los nodos, dejando sin filtrar un 0,21 % de los puntos, lo que representa un significativo ahorro en términos de accesos a disco, memoria utilizada y tiempo de cálculo, si lo comparamos con tener que acceder a todos los nodos de los R-trees, recuperando y procesando todos los puntos de los conjuntos.

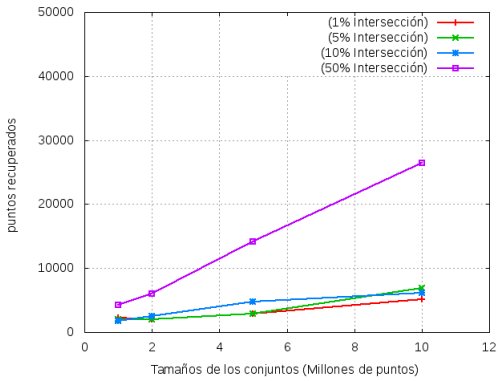
La principal desventaja de este algoritmo es que siempre debe alcanzar a las hojas de los R-trees para calcular la separabilidad.



(a) Porcentaje de nodos accedidos.



(b) Porcentaje de puntos recuperados.



(c) Puntos recuperados.

Fig. 5.5: Gráficos de resultados.

Capítulo 6

Algoritmo Convex Hull de separabilidad lineal

6.1. Introducción

En el presente capítulo se presenta un nuevo método para calcular la separabilidad lineal, el cual explota el uso de las cerraduras convexas (Convex Hull) para determinarla. Este nuevo algoritmo calcula pseudo cerraduras convexas a medida que desciende por los R-trees, obteniendo resultados parciales con los cuales es posible decidir la separabilidad de los conjuntos sin haber alcanzado necesariamente las hojas de los R-trees. Este método es una mejora del algoritmo presentado en el Capítulo 5, por lo que se realizará un análisis más detallado.

Este capítulo se estructura de la siguiente forma. En la Sección 6.2 se describe el algoritmo. En la Sección 6.3 se explica el pseudocódigo del algoritmo. En la Sección 6.4 se detallan las pruebas experimentales y finalmente, en la Sección 6.5 se exponen las conclusiones de los resultados obtenidos.

6.2. Descripción

Definición 6.1. *Sea $conv(N)$ la cerradura convexa del conjunto N. $conv(N \cup X)$ corresponde a la cerradura convexa de la unión de los conjuntos N y X.*

Supongamos que tenemos almacenado en memoria principal un conjunto N_R de MBRs del R-tree de R de modo que cubra todo el conjunto R y un conjunto de MBRs N_B del R-tree de B de modo que cubra todo el conjunto B. Si la cerradura convexa $conv(N_R)$ de los MBRs de N_R no interseca la cerradura convexa $conv(N_B)$ de los MBRs de N_B, entonces $conv(R)$ y $conv(B)$ son disjuntos ya que $conv(R) \subseteq conv(N_R)$ y $conv(B) \subseteq conv(N_B)$, por lo tanto R y B son linealmente separables.

Una idea más concreta es la siguiente. De acuerdo con las posiciones relativas de $MBR(R)$ y $MBR(B)$, nosotros elegimos de $MBR(R)$ un conjunto V_R de al menos tres vértices del $MBR(R)$ y un conjunto similar V_B de $MBR(B)$. La idea de elegir V_R y V_B es que $conv(R)$ y $conv(B)$ sean disjuntos sí y solamente sí $conv(R \cup V_R)$ y $conv(B \cup V_B)$ son disjuntos.

Partimos con un N_R inicial, el cual contiene el conjunto de MBRs almacenados en el nodo raíz del R-tree de R y un N_B inicial que contiene el conjunto de MBRs almacenados en el nodo raíz del R-tree de B. Luego, iteramos de la siguiente forma: si $conv(N_R \cup V_R)$ y $conv(N_B \cup V_B)$ son disjuntos, entonces los conjuntos sí son separables, por lo que respondemos con un 'si' y calculamos la línea de separación. En otro caso, para cada MBR de N_R tomamos el conjunto de puntos que puede asegurarse que pertenezca a $conv(R \cup V_R)$ y formamos el conjunto N'_R. Un conjunto similar N'_B es formado desde N_B. Si $conv(N'_R \cup V_R)$ y $conv(N'_B \cup V_B)$ no son disjuntos, indica que R y B no son linealmente separables, por lo que respondemos con 'no'. Si ninguna de las condiciones anteriores se cumple, filtramos el conjunto N_R de tal forma que el nuevo N_R contenga solamente los MBRs (o puntos) que pertenezcan a las cerraduras convexas de R y B y 'refinamos' $conv(N_R \cup V_R)$ reemplazando cada MBR en N_R por sus respectivos MBRs hijos (o puntos) [8]. Ejecutamos un procedimiento similar con N_B y continuamos la iteración. Si en algún momento de la iteración ambos conjuntos, N_R y N_B están formados sólo por puntos, entonces la respuesta es dada por la intersección de $conv(N_R \cup V_R)$ y $conv(N_B N \cup V_B)$, terminando el algoritmo.

Definición 6.2. *Si $MBR(R)$ y $MBR(B)$ tienen una intersección corner, entonces V_R denota el conjunto de los vértices superior derecho, superior izquierdo e inferior izquierdo del $MBR(R)$ y V_B denota el conjunto de vértices superior derecho, inferior izquierdo e inferior derecho de $MBR(B)$.*

Si $MBR(R)$ y $MBR(B)$ tienen una intersección semi-disjunta, entonces V_R denota el conjunto de vértices superior izquierdo y inferior izquierdo de $MBR(R)$ y V_B denota el conjunto de vértices superior derecho y inferior derecho del $MBR(B)$ (ver las Fig. 6.1(a) y 6.1(b)).

La idea detrás de la definición de V_R y V_B es entregada en los siguientes lemas:

Lema 6.1. *Las cerraduras convexas $conv(R)$ y $conv(B)$ son disjuntas sí y solamente sí las cerraduras convexas $conv(R \cup V_R)$ y $conv(B \cup V_B)$ son disjuntas.*

Demostración. Si $conv(R \cup V_R)$ y $conv(B \cup V_B)$ son disjuntas, entonces $conv(R)$ y $conv(B)$ también son disjuntas porque $conv(R) \subseteq conv(R \cup V_R)$ y $conv(B) \subseteq conv(B \cup V_B)$. Supongamos que $conv(R)$ y $conv(B)$ son disjuntas (ver la Fig. 6.1(c)). Sea p un punto de $conv(R \cup V_R) \setminus conv(R)$. Si p no pertenece a $MBR(B)$, entonces p no está contenido en $conv(B \cup V_B)$. Ahora, asumamos que p pertenece a la intersección $MBR(R) \cap MBR(B)$. Sea h_p una semirecta con origen en p y orientada hacia la derecha o hacia abajo, tal que h_p contenga un punto q perteneciente a $conv(R) \cap MBR(B)$. Sabemos que h_p siempre existe dado p y podemos asumir sin pérdida de generalidad que h_p es horizontal. El caso cuando h_p es vertical (el cual aparece solamente en el caso de una intersección corner) es análogo. Ya que q no pertenece a $conv(B)$ porque $conv(R)$ y $conv(B)$ son disjuntos y q está en el interior de $MBR(R) \cap MBR(B)$, q está a la izquierda y encima de $conv(B)$ en el caso de una intersección corner, y a la izquierda de $conv(B)$ en el caso de una intersección semi-disjunta. De esta forma, q no pertenece a $conv(B \cup V_B) \setminus conv(B)$ por la definición de V_B. Por lo tanto, p no está en $conv(B \cup V_B) \setminus conv(B)$ por la convexidad de $conv(B)$ y p está a la izquierda de q. De manera similar y usando argumentos simétricos se muestra que si un punto p' pertenece a $conv(B \cup V_B) \setminus conv(B)$, entonces p' no está en $conv(R \cup V_R) \setminus conv(R)$. Todas estas observaciones implican que $conv(R \cup V_R)$ y $conv(B \cup V_B)$ son disjuntos. □

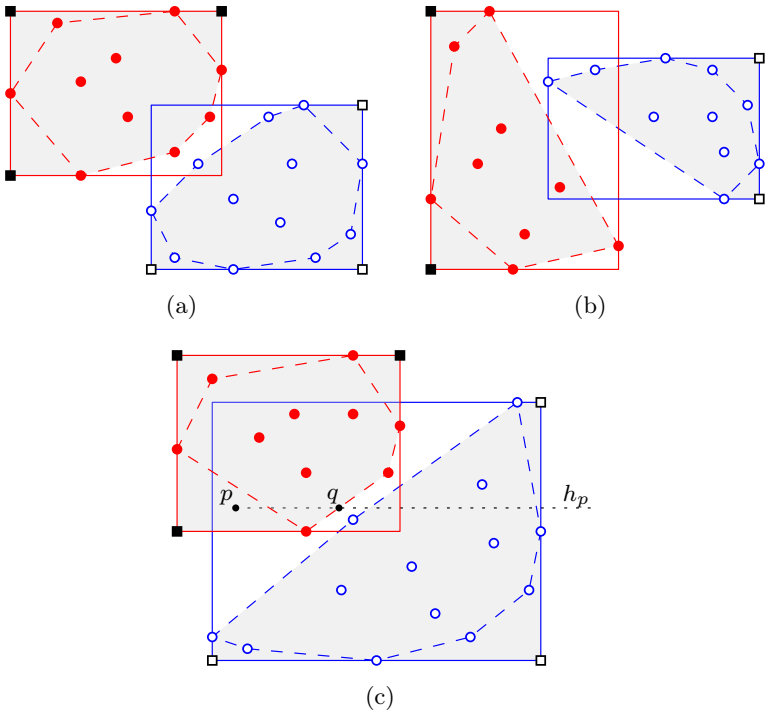
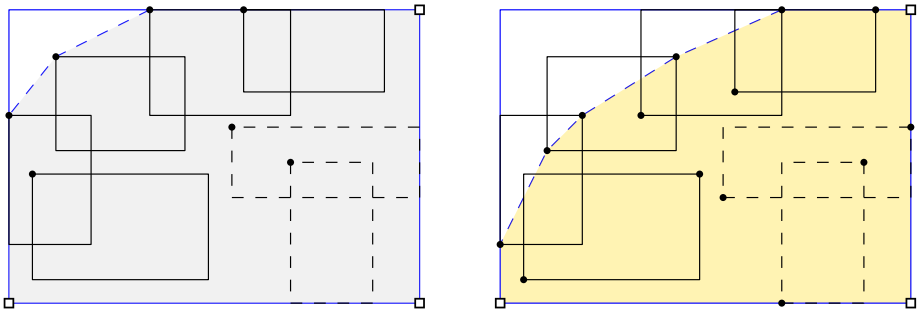


Fig. 6.1: Los conjuntos de vértices V_R y V_B para: (a) intersección corner; (b) intersección semi-disjunta. Los vértices V_R son denotados como cuadrados rellenos, y los vértices de V_B como cuadrados vacíos. (c) Demostración del Lema 6.1. En cada figura, las cerraduras convexas $conv(R \cup V_R)$ y $conv(B \cup V_B)$ son denotadas como regiones sombreadas.

Lema 6.2. *Las cerraduras convexas $conv(R \cup V_R)$ y $conv(B \cup V_B)$ tienen una intersección no vacía sí y solamente sí uno de ellos contiene un vértice perteneciente al otro. Además, los vértices de $conv(R \cup V_R)$ contenidos en $conv(B \cup V_B)$ son todos consecutivos y los vértices $conv(B \cup V_B)$ contenidos en $conv(R \cup V_R)$ son todos consecutivos.*

Demostración. Si $conv(R \cup V_R)$ ó $conv(B \cup V_B)$ contiene un vértice del otro, entonces ellos no son disjuntos. Supongamos ahora que $conv(R \cup V_R)$ y $conv(B \cup V_B)$ no son disjuntos. Sea X un conjunto de puntos rojos e Y un conjunto de puntos azules. Si $conv(X)$ y $conv(Y)$ son no disjuntos y ninguno de ellos contiene un vértice del otro (por ejemplo, ver la Figura 4.1 (c) con intersección semi-disjunta), entonces la cerradura convexa $conv(X \cup Y)$ contiene al menos cuatro aristas bicromáticas (es decir, aristas conectando puntos de diferentes colores). Para $X = R \cup V_R$ y $Y = B \cup V_B$, $conv(X \cup Y)$ contiene solamente dos aristas bicromáticas, dadas las posiciones relativas de $MBR(R)$ y $MBR(B)$ y las deficiones de V_R y V_B. Por lo tanto, $conv(R \cup V_R)$ ó $conv(B \cup V_B)$ debe contener un vértice del otro. Además, el hecho que $conv(X \cup Y)$ contiene solamente dos aristas bicromáticas implica la segunda parte del lema. □



(a) (b)

Fig. 6.2: Un conjunto N_B de MBRs del R-tree de B, en el caso donde $MBR(R)$ y $MBR(B)$ tienen una intersección corner. (a) La Cerradura Convexa Optimista $\mathsf{opt}(N_B)$. (b) La Cerradura Convexa Pesimista $\mathsf{pess}(N_B)$. Los MBRs en líneas discontinuas pueden ser removidos de N_B ya que están contenidos en $\mathsf{pess}(N_B)$.

6.2.1. Cerradura Convexa Optimista y Pesimista

Supongamos que tenemos almacenado un conjunto de MBRs del R-tree de R y un conjunto de MBRs del R-tree de B. En esta sección explicamos una forma para determinar que de estos dos conjuntos sí tenemos suficiente información para decidir que $conv(R \cup V_R)$ y $conv(B \cup V_B)$ son disjuntos, sin tener que descender en los R-trees y recuperar más MBRs o puntos.

Definición 6.3. *Sea N_R un conjunto de MBRs del R-tree de R y N_B un conjunto de MBRs del R-tree de B, los cuales cumplen las siguientes propiedades:*

(1) $conv(R \cup V_R) \subseteq conv(N_R \cup R)$.

(2) $conv(B \cup V_B) \subseteq conv(N_B \cup B)$.

Para un ejemplo de la Definición 6.3, ver las Fig. 6.2 y 6.3. Un punto puede ser visto como un MBR de perímetro nulo, extendemos las definiciones de N_R y N_B para que estos puedan estar formados de puntos.

Definición 6.4. *Dado los conjuntos N_R y N_B, la cerradura convexa optimista de $R \cup V_R$ es el conjunto* $\mathsf{opt}(N_R) = conv(N_R \cup V_R)$ *el cual contiene $conv(R \cup V_R)$, la cerradura convexa optimista de $B \cup V_B$ es el conjunto* $\mathsf{opt}(N_B) = conv(N_B \cup V_B)$ *el cual contiene $conv(B \cup V_B)$ (ver Figura 6.2(a) y Figura 6.3(a)).*

La idea de definir la cerradura convexa optimista se basa en la siguiente observación: Si $\mathsf{opt}(N_R)$ y $\mathsf{opt}(N_B)$ son disjuntos, entonces podemos asegurar que $conv(R \cup V_R)$ y $conv(B \cup V_B)$ son disjuntos y que R y B son linealmente separables por el Lema 6.1. Además, $\mathsf{opt}(N_R)$ y $\mathsf{opt}(N_B)$ son aproximaciones de $conv(R \cup V_R)$ y $conv(B \cup V_B)$, y para calcularlos no necesitamos los puntos cubiertos por estos MBRs, los cuales están situados en las hojas de los R-trees.

También necesitamos un método para determinar si con N_R y N_B tenemos la información suficiente para decidir que $conv(R \cup V_R)$ y $conv(B \cup V_B)$ no son disjuntos. Esto es explicado a continuación.

Sea N un MBR: $\texttt{NE}(N)$ es el triángulo nordeste de N, es decir, el subconjunto de puntos de N que están sobre o por debajo de la diagonal que conecta los vértices superior izquierdo e inferior derecho. Similarmente, $\texttt{NW}(N)$ es el subconjunto de N en o por debajo de la diagonal que conecta los vértices superior derecho e inferior izquierdo; y $\texttt{SE}(N)$ es el subconjunto de puntos de N en o por debajo de la diagonal que conecta los vértices superior izquierdo e inferior derecho.

Supongamos que $MBR(R)$ y $MBR(B)$ tienen una intersección corner y sea N un MBR de N_R. Un MBR, por ser el rectángulo de cobertura miníma de un conjunto, posee puntos del conjunto en sus aristas, por lo que el conjunto $\texttt{NW}(N)$ está contenido en la cerradura convexa $conv(R \cup V_R)$. Similarmente, si N es un MBR de N_B, entonces el conjunto $\texttt{SE}(N)$ está contenido en la cerradura convexa $conv(B \cup V_B)$. Usamos estas observaciones para definir la cerradura convexa pesimista, la cual está siempre contenida en nuestro objetivo $conv(R \cup V_R)$ y $conv(B \cup V_B)$, que son las cerraduras convexas que nos interesan.

Definición 6.5. *Sean R y B conjuntos de puntos rojos y azules respectivamente, tal que $MBR(R)$ y $MBR(B)$ tienen una intersección corner. La Cerradura Convexa Pesimista de $R \cup V_R$ es el conjunto $\mathsf{pess}(N_R) = conv(N'_R \cup V_R)$ el cual está contenido en $conv(R \cup V_R)$, donde $N'_R = \{\texttt{NW}(N) \mid N \in N_R\}$. La Cerradura Convexa Pesimista de $B \cup V_B$ es el conjunto $\mathsf{pess}(N_B) = conv(N'_B \cup V_B)$ el cual está contenido en $conv(B \cup V_B)$, donde $N'_B = \{\texttt{SE}(N) \mid N \in N_B\}$ (ver Fig. 6.2(b)).*

Definición 6.6. *Sean R y B conjuntos de puntos rojos y azules respectivamente, tal que $MBR(R)$ y $MBR(B)$ tienen una intersección semi-disjunta. La cerradura convexa pesimista $R \cup V_R$ es el conjunto $\mathsf{pess}(N_R) = conv(N'_R \cup V_R \cup \{u_1\}) \cap conv(N''_R \cup V_R \cup \{u_2\})$, donde $N'_R = \{\texttt{NW}(N) \mid N \in N_R\}$, $N''_R = \{\texttt{SW}(N) \mid N \in N_R\}$, y u_1 y u_2 son los vértices superior derecho y inferior derecho de $MBR(R)$, respectivamente. La cerradura convexa pesimista de $B \cup V_B$ es el conjunto $\mathsf{pess}(N_B) = conv(N'_B \cup V_B \cup \{v_1\}) \cap conv(N''_B \cup V_B \cup \{v_2\})$, donde $N'_B = \{\texttt{SE}(N) \mid N \in N_B\}$, $N''_B = \{\texttt{NE}(N) \mid N \in N_B\}$, y v_1 y v_2 son los vértices inferior-izquierdo y superior izquierdo de $MBR(B)$, respectivamente. (ver Figura 6.3).*

Cuando $MBR(R)$ y $MBR(B)$ tienen una intersección corner, las cerraduras convexas $\mathsf{pess}(N_R)$ y $\mathsf{pess}(N_B)$ no son disjuntas, por lo que podemos asegurar que $conv(R \cup V_R)$ y $conv(B \cup V_B)$ no son disjuntos y R y B no son linealmente separables por el Lema 6.1. Cuando $MBR(R)$ y $MBR(B)$ tienen una intersección semi-disjunta, las mismas cerraduras convexas están demostradas en el Lema 6.3 y podemos asegurar que $conv(R \cup V_R)$ y $conv(B \cup V_B)$ no son disjuntos si $\mathsf{pess}(N_R)$ y $\mathsf{pess}(N_B)$ no lo son.

Lema 6.3. *Sean R y B conjuntos de puntos rojos y azules, respectivamente, tal que $MBR(R)$ y $MBR(B)$ tienen una intersección semi-disjunta. Las cerraduras convexas $\mathsf{pess}(N_R)$ y $\mathsf{pess}(N_B)$ están contenidas en $conv(R \cup V_R)$ y $conv(B \cup V_B)$, respectivamente.*

Demostración. Demostraremos que $\mathsf{pess}(N_B) \subseteq conv(B \cup V_B)$. Los argumentos para demostrar $\mathsf{pess}(N_R) \subseteq conv(R \cup V_R)$ son análogos. Consideremos los MBRs de N_B cuyos lados superiores están alineados con la parte superior de $MBR(B)$, sea t_1 el vértice más a la izquierda de los vértices superior derechos (ver la Fig. 6.4). Consideremos los MBRs de N_B cuyos lados inferiores están alineados con el lado inferior de $MBR(B)$, sea b_1 el vértice más a la izquierda de los vértices

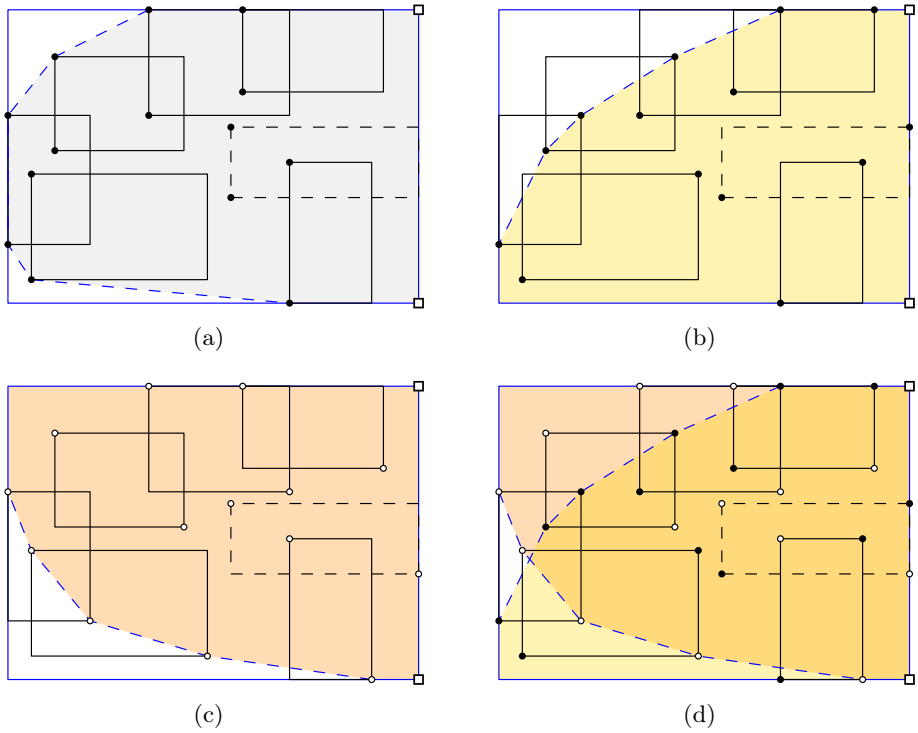


Fig. 6.3: Un conjunto N_B de rectángulos del R-tree de B, cuando $MBR(R)$ y $MBR(B)$ tienen una intersección semi-disjunta. (a) La cerradura convexa optimista $\mathsf{opt}(N_B)$. (b) La cerradura convexa $conv(N'_B \cup V_B \cup \{v_1\})$. (c) La cerradura convexa $conv(N''_B \cup V_B \cup \{v_2\})$. (d) La cerradura convexa pesimista $\mathsf{pess}(N_B) = conv(N'_B \cup V_B \cup \{v_1\}) \cap conv(N''_B \cup V_B \cup \{v_2\})$. El rectángulo en líneas discontinuas puede ser removido de N_B ya que está contenido en $\mathsf{pess}(N_B)$.

inferior derechos. Considerar los MBRs de N_B cuyos lados izquierdos están alineados con el lado izquierdo de $MBR(B)$, sea ℓ_1 el vértice más bajo de los vértices superior-izquierdos y ℓ_2 el vértice superior (de más arriba) de los vértices inferior izquierdos. Observar que ℓ_2 y t_1 son vértices de $conv(N'_B \cup V_B \cup \{v_1\})$ y que b_1 y ℓ_1 son vértices de $conv(N''_B \cup V_B \cup \{v_2\})$. Sea U la ruta de conexión de ℓ_2 con t_1 a lo largo del borde de $conv(N'_B \cup V_B \cup \{v_1\})$ en el sentido de las manecillas del reloj y L denota la ruta de conexión de b_1 con ℓ_1 a lo largo de $conv(N''_B \cup V_B \cup \{v_2\})$ en el sentido de las manecillas del reloj. Notar que existen los siguientes puntos azules de B: un punto azul t más a la izquierda en el segmento que conecta v_2 y t_1, un punto b más a la izquierda en el segmento que conecta v_1 y b_1, y un punto azul ℓ más alto y un punto azul más bajo ℓ' (posiblemente igual a ℓ) en el borde que conecta a v_1 con v_2. Por las definiciones de ℓ_1 y ℓ_2, tenemos que ℓ pertenece al segmento que conecta a ℓ_2 con v_2, y ℓ' pertenece al segmento que conecta a ℓ_1 con v_1. La observación clave es que b, ℓ', ℓ, y t son todos vértices de $conv(B \cup V_B)$. Además, el camino en el sentido de las manecillas del reloj a lo largo del borde de $conv(B \cup V_B)$ que conecta b y ℓ'' está por debajo de la trayectoria de L y el camino similar que conecta ℓ con t está por encima de la trayectoria U. Todas estas observaciones implican que $\mathsf{pess}(N_B) \subseteq conv(B \cup V_B)$. □

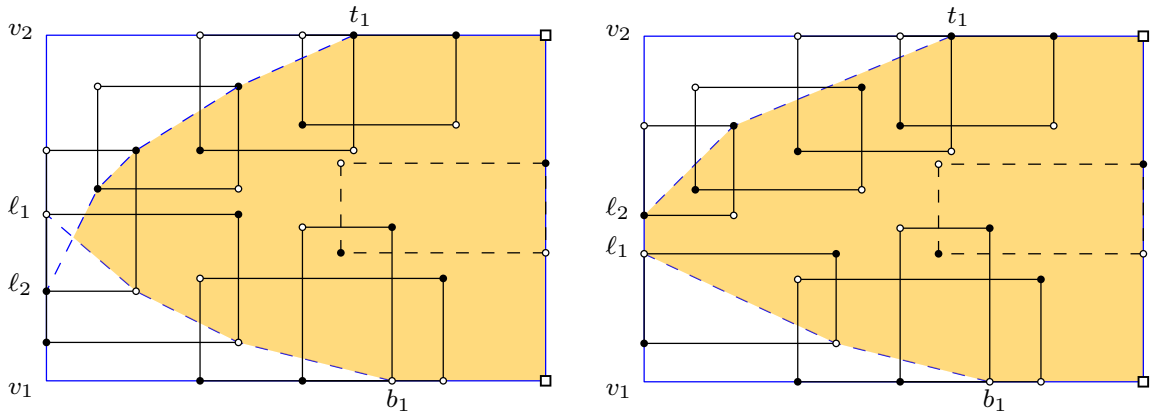


Fig. 6.4: Prueba del Lema 6.3.

6.3. Algoritmo

En esta sección, presentamos nuestro algoritmo para probar la separabilidad de los conjuntos R y B almacenados en R-trees distintos. Iniciamos presentando los procedimientos que forman parte del algoritmo de la separabilidad: cálculo de las cerraduras convexas optimistas y pesimistas (Sección 6.3.1), decidir si las cerraduras convexas son disjuntas y encontrar la línea de separación en el caso positivo (Sección 6.3.2), y filtrar los conjuntos de MBRs N_R y N_B (Sección 6.3.3). Luego, mostramos el algoritmo de separabilidad (Sección 6.3.4). Finalmente, mostramos cómo las técnicas que previamente definimos pueden ser usadas para calcular la cerradura convexa de un conjunto de puntos dados en un R-tree (Sección 6.3.5).

6.3.1. Cálculo de la cerradura convexa

En esta Sección explicamos cómo calcular las cerraduras convexas optimistas y pesimistas para N_B, cuando posee intersección corner y semi-disjunta con N_R. El algoritmo para calcular estas cerraduras convexas para N_R es análogo por simetría.

Sea t_1 (resp. t_2) el vértice más a la izquierda con respecto a los vértices superior derecho (resp. superior izquierdo) de los MBRs de N_B cuyos lados superiores están alineados con la parte superior de $MBR(B)$; b_1 (resp. b_2) el vértice más a la izquierda de los vértices inferiores derechos (resp. inferior izquierdo) de los rectángulos de N_B cuyos lados inferiores están alineados con el lado inferior de $MBR(B)$; ℓ_1 (resp. ℓ'_1) el vértice más abajo de los vértices superior izquierdos (resp. inferior izquierdo) de los rectángulos de N_B cuyos lados izquierdos están alineados con el lado izquierdo de $MBR(B)$; y ℓ_2 (resp. ℓ'_2) el vértice más alto de los vértices inferior izquierdos (resp. superior izquierdo) de los rectángulos de N_B cuyo lado izquierdo está alineado con el lado izquierdo de $MBR(B)$. Todos estos puntos pueden ser encontrados en tiempo $O(|N_B|)$, con una simple pasada por los elementos de N_B (ver Figura 6.5).

Supongamos que $MBR(R)$ y $MBR(B)$ tienen una intersección corner. Para calcular $\mathsf{opt}(N_B)$,

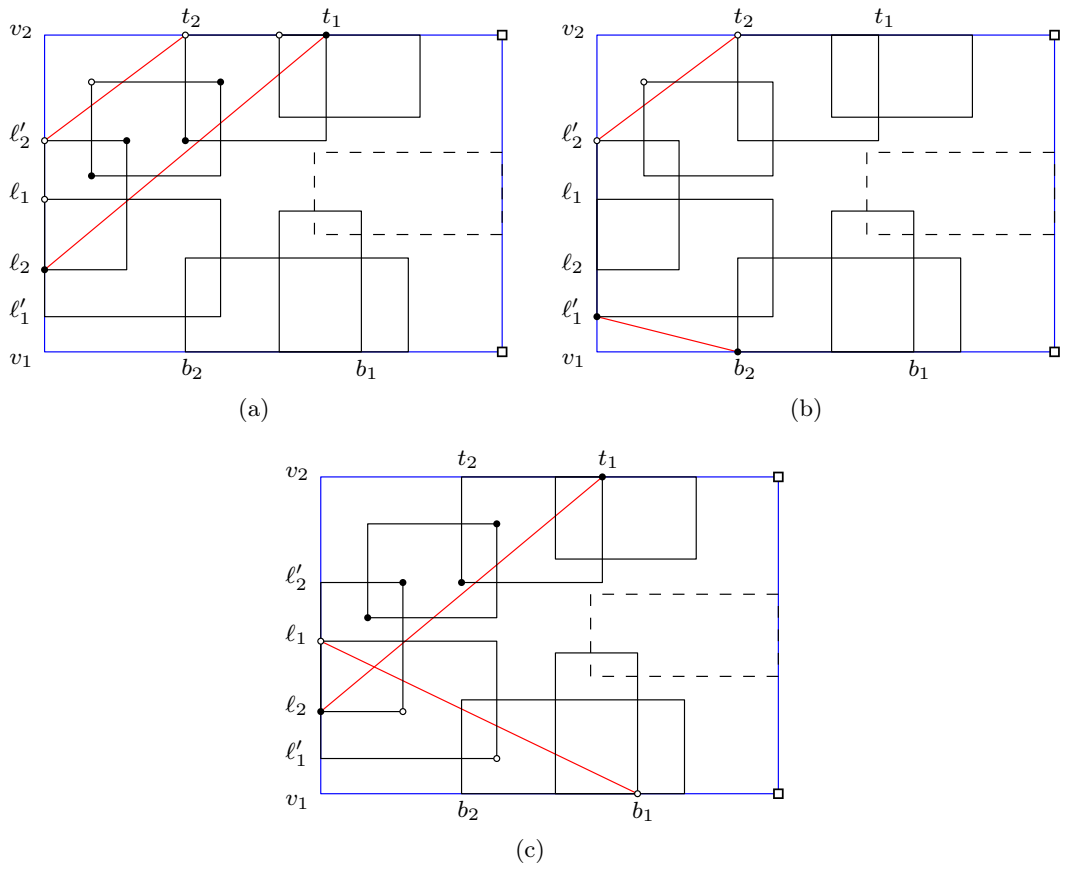


Fig. 6.5: Algoritmo para calcular $\mathsf{opt}(N_B)$ y $\mathsf{pess}(N_B)$: (a) Los vértices llenos son los puntos de S y los puntos vacíos son los vértices de S'. (b) Los vértices del conjunto S_0. (c) Los vértices llenos son los puntos de S_1 los puntos vacíos son los vértices de S_2.

necesitamos calcular la cerradura convexa de los vértices superior izquierdo de los rectángulos de N_B y del conjunto V_B. Observar que aquellos vértices que no están en el triángulo T formado por los vértices $\{v_2, \ell'_2, t_2\}$ no pueden ser vértices de $\mathsf{opt}(N_B)$ (ver Figura 6.5(a)). De esta forma, primero encontramos en tiempo $O(|N_B|)$ el conjunto S de los vértices superior izquierdo de los MBRs de N_B que pertenecen a T y luego calculamos $\mathsf{opt}(N_B) = conv(S \cup V_B)$ en tiempo $O(|S| \log |S|)$, usando un algoritmo estándar para calcular la cerradura convexa. Haciendo esto, aplicamos el algoritmo de cerradura convexa solamente en el conjunto de puntos relevantes. Una idea similar que puede ser usada es la que sigue. Para calcular $\mathsf{pess}(N_B)$, encontramos en tiempo $O(|N_B|)$ el conjunto S' de los vértices inferior izquierdo y superior derecho de los MBRs de N_B que pertenecen al triángulo de vértices $\{v_2, \ell_2, t_1\}$ (ver Figura 6.5(a)), y luego calcular $\mathsf{pess}(N_B) = conv(S' \cup V_B)$ en tiempo $O(|S'| \log |S'|)$.

En el caso de una intersección semi-disjunta entre $MBR(R)$ y $MBR(B)$, se procede como sigue: Para calcular $\mathsf{opt}(N_B)$, calculamos en tiempo $O(|N_B|)$ el conjunto de puntos S_0 el cual consiste en los vértices superior izquierdo de los MBRs de N_B que pertenecen al triángulo con vértices

$\{v_2, \ell'_2, t_2\}$ y los vértices inferior izquierdo que pertenecen al triángulo de vértices $\{v_1, \ell'_1, b_2\}$ (ver Figura 6.5(b)). Luego, calculamos $\mathsf{opt}(N_B) = conv(S_0 \cup V_B)$ en tiempo $O(|S_0| \log |S_0|)$. Para calcular $\mathsf{pess}(N_B)$, primero encontramos en tiempo $O(|N_B|)$ el conjunto S_1 que contiene los vértices inferior izquierdo y superior derecho de los MBRs en N_B que pertenecen al triángulo con vértices $\{v_2, \ell_2, t_1\}$ y el conjunto S_2 que contiene los vértices superior izquierdo y inferior derecho de los rectángulos en N_B que pertenecen al triángulo con vértices $\{v_1, \ell_1, b_1\}$ (ver Figura 6.5(c)). Después de esto, calculamos las cerraduras convexas $C_1 = conv(N'_B \cup V_B \cup \{v_1\}) = conv(S_1 \cup V_B \cup \{v_1\})$ y $C_2 = conv(N''_B \cup V_B \cup \{v_2\}) = conv(S_2 \cup V_B \cup \{v_2\})$ en tiempo $O(|S_1| \log |S_1|)$ y $O(|S_2| \log |S_2|)$, respectivamente. Finalmente, calculamos $\mathsf{pess}(N_B)$ como la intersección $C_1 \cap C_2$. Si $|C_1|$ y $|C_2|$ denota el número de vértices de C_1 y C_2, respectivamente, una representación de $C_1 \cap C_2$ se puede calcular en tiempo $O(\log |C_1| \cdot \log |C_2|)$ en el peor caso.

6.3.2. Decidiendo la intersección de la cerradura convexa

En esta sección explicamos como decidir si $\mathsf{opt}(N_R)$ y $\mathsf{opt}(N_B)$ son disjuntos. Un procedimiento similar puede ser aplicado para $\mathsf{pess}(N_R)$ y $\mathsf{pess}(N_B)$.

Sea $C_1 = \mathsf{opt}(N_R)$ y $C_2 = \mathsf{opt}(N_B)$. Supongamos que $MBR(R)$ y $MBR(B)$ tienen una intersección corner. Por el Lema 6.2, podemos orientar en el sentido de las manecillas del reloj las aristas de C_2 y considerar la secuencia S de aristas consecutivas que se inicia con el vértice inferior izquierdo de $MBR(B)$ y termina con el vértice superior derecho de $MBR(B)$. Si la primera arista de S no posee un punto de C_1, entonces C_1 y C_2 son disjuntos. En general, dada alguna arista de C_2, la pregunta de si ésta pertenece a un intervalo de S se puede realizar a través de una búsqueda binaria en las aristas de C_1 desde el vértice superior derecho hasta el vértice inferior izquierdo, en tiempo $O(\log |C_1|)$. Además, decidir si existe una arista de S que tiene al menos un punto en el interior de C_1 se puede realizar con una búsqueda binaria realizando $O(\log |C_2|)$ consultas, cada consulta costará un tiempo de $O(\log |C_1|)$. Por lo tanto, decidir si C_1 y C_2 son disjuntos puede realizarse en tiempo $O(\log |C_1| \cdot \log |C_2|)$.

Cuando C_1 y C_2 son disjuntos, para encontrar la línea que los separa es necesario encontrar la primera arista e en S que no apunte a C_1. Esta arista puede ser encontrada con un procedimiento similar al explicado anteriormente, en tiempo $O(\log |C_1| \cdot \log |C_2|)$. Sea $\overline{e}$, la misma arista e, pero orientada en dirección contraria. Si $\overline{e}$ no apunta a C_1, entonces la línea recta que pasa a través de e es una línea de separación. En otro caso, en tiempo $O(\log |C_1|)$ podemos encontrar la arista e' de C_1 apuntada por $\overline{e}$ y la línea a través de e' es una línea de separación.

Cuando $MBR(R)$ y $MBR(B)$ tienen una intersección semi-disjunta, se debe considerar una secuencia S, la cual es similar a la anterior. En este caso, S es la secuencia de aristas que inician con la arista iniciada en el vértice inferior derecho del $MBR(B)$ y finalizada con la arista terminada en el vértice superior izquierdo de $MBR(B)$. Decidir si C_1 y C_2 son disjuntos se puede realizar en tiempo $O(\log |C_1| \cdot \log |C_2|)$.

6.3.3. Filtrando MBRs

El filtrado de los MBRs de N_R y N_B, consiste en refinar estos conjuntos removiendo algunos elementos, de tal forma que los nuevos N_R y N_B todavía satisfagan las propiedades (1) y (2) de la Definición 6.3. Supongamos que queremos remover los MBRs de N_B. Una forma natural de

remover los MBRs consiste en eliminar todos los MBRs que están completamente contenidos en la cerradura convexa pesimista (ver los MBRs con líneas discontinuas en la Figura 6.3(d) y la Figura 6.4 (b)). Si el MBR tiene una parte afuera de la cerradura convexa pesimista, entonces no puede ser removido porque una parte puede contener puntos azules que son vértices de $conv(B \cup V_B)$.

Para determinar si un MBR N de N_B está contenido en $\mathsf{pess}(N_B)$ debemos consultar si alguno de los vértices de N no se encuentra contenido en $\mathsf{pess}(N_B)$. Cada consulta se ejecuta en un tiempo de $O(\log k)$, donde k es el número de vértices de $\mathsf{pess}(N_B)$. El tiempo de ejecución del algoritmo de filtrado para N_B es $O(|N_B| \cdot \log k) = O(|N_B| \cdot \log |N_B|)$. En forma análoga, para determinar si un MBR N de N_R está contenido en $\mathsf{pess}(N_R)$ debemos consultar si alguno de los vértices de N no se encuentra contenido en $\mathsf{pess}(N_R)$, por lo que usamos criterios y un algoritmo similar al explicado anteriormente. El tiempo de ejecución del algoritmo de filtrado para N_R es $O(|N_R| \cdot \log |N_R|)$, para el caso de N_R.

6.3.4. Algoritmo de separabilidad

El algoritmo consiste en un procedimiento principal y en un prodecimiento interno. El procedimiento principal (ver el pseudocodigo `DecideSeparability` de la Figura 6.1) recibe R y B como entrada, ambos representados en un R-tree y decide la separabilidad lineal de R y B. También retorna una línea de separación en el caso positivo. En este procedimiento, primero se inicializa el conjunto de MBRs N_R como los MBRs contenidos en el nodo raíz del R-tree de R y el conjunto de rectángulos N_B como los rectángulos contenidos en el nodo raíz del R-tree de B. Esto permite calcular $MBR(R) = MBR(N_R)$ y $MBR(B) = MBR(N_B)$. Luego, procedemos como sigue: Si la intersección entre $MBR(R)$ y $MBR(B)$ es vacía, entonces retornamos 'sí' como respuesta junto con una línea alineada a uno de los ejes conteniendo un borde del $MBR(R)$ que separa R y B. Si $MBR(R)$ y $MBR(B)$ tienen una intersección sandwich, entonces retornamos 'no' como respuesta. De otra manera, si $MBR(R)$ y $MBR(B)$ tienen una intersección completa, corner o semi-disjunta, entonces se llama al procedimiento interno (ver el pseudocódigo `DecideSeparabilityCS` de la Figura 6.2). Este procedimiento decide la separabilidad lineal cuando $MBR(R)$ y $MBR(B)$ tienen una intersección corner o semi-disjunta. Cuando la intersección entre $MBR(R)$ y $MBR(B)$ es completa se llama nuevamente al procedimiento. El problema se reduce a resolver (a lo sumo) cuatro instancias del problema en el cual $MBR(R)$ y $MBR(B)$ tienen una intersección corner (ver Sección 4.3.1). Esto se logra extendiendo el MBR interior para contener un vértice del MBR exterior. El procedimiento interno es el siguiente: Inicialmente se calcula el conjunto de vértices V_R y V_B, de acuerdo a las posiciones relativas de $MBR(R)$ y $MBR(B)$, los cuales son necesarios para que el algoritmo distinga entre una interseción corner y una semi-disjunta. Luego, se realizan las siguientes acciones con N_R y N_B. Calculamos $\mathsf{opt}(N_R)$ y $\mathsf{opt}(N_B)$ (ver Sección 6.3.1) y probamos si $\mathsf{opt}(N_R)$ y $\mathsf{opt}(N_B)$ son disjuntos (ver Sección 6.3.2). Si lo son, entonces reportamos una respuesta 'si' y encontramos una línea de separación (ver Sección 6.3.2). De otra manera, si $\mathsf{opt}(N_R)$ y $\mathsf{opt}(N_B)$ no son disjuntos, continuamos como sigue. Calculamos $\mathsf{pess}(N_R)$ y $\mathsf{pess}(N_B)$ y decidimos si ellos son disjuntos (ver secciones 6.3.1 y 6.3.2). Si no son disjuntos, reportamos una respuesta 'no'. En otro caso, para cada conjunto de puntos $X \in \{R, B\}$ tal que N_X está formado de MBRs, filtramos N_X (ver Sección 6.3.3) y reemplazamos cada MBR en N_X por sus hijos, MBRs o puntos, en el R-tree correspondiente. Observar que los

nuevos MBRs de N_X están todos un nivel abajo de los MBRs formados en N_X. Si al menos uno de los nuevos N_R y N_B están formados de MBRs (puede ocurrir que N_R o N_B esté formado de MBRs y el otro esté formado de puntos, debido a que los R-trees de R y de B tienen distintas alturas), entonces repetimos todas las acciones con estos nuevos N_R y N_B. En otro caso, si N_R o N_B están formados de puntos, probamos si $\mathsf{opt}(N_R) = conv(R)$ y $\mathsf{opt}(N_B) = conv(B)$ son disjuntos y en el caso positivo se obtiene una línea de separación, para finalmente decidir si R y B son linealmente separables.

A continuación, analizamos el tiempo de ejecución asintótico del algoritmo en el peor caso. Sean $m = |R|$ y $n = |B|$, y $r = O(\log m)$ y $b = O(\log n)$ las alturas del R-trees de R y del R-tree de B, respectivamente. Asumimos, sin pérdida de generalidad, que $r \leq b$. Para $i = 0, 1, \ldots, r$, m_i que denota el número de MBRs en el i-ésimo nivel del R-tree de R y $N_R^{(i)}$ el conjunto de MBRs de N_R cuando N_R es formada por MBRs del i-ésimo nivel, donde $|N_R^{(i)}| \leq m_i$ y $m_r = m$. Similarmente, para $j = 0, 1, \ldots, b$, n_j denota el número de MBRs en el j-ésimo nivel del R-tree de B y $N_B^{(j)}$ el conjunto de MBRs de N_B cuando N_B está formado por MBRs del j-ésimo nivel, donde $|N_B^{(j)}| \leq n_j$ y $n_b = n$. Para el nivel $i = 0, 1, \ldots, r$, el algoritmo en el peor caso:

- calcula $\mathsf{opt}(N_R^{(i)})$ y $\mathsf{pess}(N_R^{(i)})$ en tiempo $O(|N_R^{(i)}| \cdot \log |N_R^{(i)}|)$;
- calcula $\mathsf{opt}(N_B^{(i)})$ y $\mathsf{pess}(N_B^{(i)})$ en tiempo $O(|N_B^{(i)}| \cdot \log |N_B^{(i)}|)$;
- decide $\mathsf{opt}(N_R^{(i)}) \cap \mathsf{opt}(N_B^{(i)}) = \emptyset$ en tiempo $O(\log |N_R^{(i)}| \cdot \log |N_B^{(i)}|)$;
- decide $\mathsf{pess}(N_R^{(i)}) \cap \mathsf{pess}(N_B^{(i)}) = \emptyset$ en tiempo $O(\log |N_R^{(i)}| \cdot \log |N_B^{(i)}|)$;
- filtra $N_R^{(i)}$ en tiempo $O(|N_R^{(i)}| \cdot \log |N_R^{(i)}|)$, para $i < r$; y
- filtra $N_B^{(i)}$ en tiempo $O(|N_B^{(i)}| \cdot \log |N_B^{(i)}|)$, para $i < b$.

Para el nivel $j = r + 1, \ldots, b$, el algoritmo en el peor caso:

- calcula $\mathsf{opt}(N_B^{(j)})$ y $\mathsf{pess}(N_B^{(j)})$ en tiempo $O(|N_B^{(j)}| \cdot \log |N_B^{(j)}|)$;
- decide $\mathsf{opt}(N_R^{(r)}) \cap \mathsf{opt}(N_B^{(j)}) = \emptyset$ en tiempo $O(\log |N_R^{(r)}| \cdot \log |N_B^{(j)}|)$;
- decide $\mathsf{pess}(N_R^{(r)}) \cap \mathsf{pess}(N_B^{(j)}) = \emptyset$ en tiempo $O(\log |N_R^{(r)}| \cdot \log |N_B^{(j)}|)$; y
- filtra $N_B^{(j)}$ en tiempo $O(|N_B^{(j)}| \cdot \log |N_B^{(j)}|)$, para $j < b$.

```
1  Alg.: DecideSeparability(R, B)
   Data: R: R-tree de puntos rojos, B: R-tree de puntos azules.
2  N_R ← el conjunto de MBRs en el nodo raíz del R-tree R
3  N_B ← el conjunto de MBRs en el nodo raíz del R-tree B
4  MBR(R) ← MBR(N_R)
5  MBR(B) ← MBR(N_B)
6  if MBR(R) ∩ MBR(B) es vacío then
7  |  return 'si'
8  else
9  |  if MBR(R) y MBR(B) tienen una intersección sandwich then
10 |  |  return 'no'
11 |  else
12 |  |  if MBR(R) y MBR(B) tienen una intersección completa then
13 |  |  |  if MBR(B) ⊂ MBR(R) then
14 |  |  |  |  foreach vértice v de MBR(R) do
   |  |  |  |  |  /* MBR(R) y MBR(MBR(B) ∪ {v}) tienen una intersección
   |  |  |  |  |     corner                                              */
15 |  |  |  |  |  if DecideSeparabilityCS(N_R, N_B, MBR(R), MBR(MBR(B) ∪ {v}))
   |  |  |  |  |  then
16 |  |  |  |  |  |  return 'si'
17 |  |  |  |  |  end
18 |  |  |  |  end
19 |  |  |  |  return 'no'
20 |  |  |  else
21 |  |  |  |  foreach vértice v de MBR(B) do
   |  |  |  |  |  /* MBR(MBR(R) ∪ {v}) y MBR(B) tiene una intersección
   |  |  |  |  |     corner                                              */
22 |  |  |  |  |  if DecideSeparabilityCS(N_R, N_B, MBR(MBR(R) ∪ {v}), MBR(B))
   |  |  |  |  |  then
23 |  |  |  |  |  |  return 'si'
24 |  |  |  |  |  end
25 |  |  |  |  end
26 |  |  |  |  return 'no'
27 |  |  |  end
28 |  |  else
   |  |  |  /* MBR(R) y MBR(B) tienen una intersección corner o
   |  |  |     semi-disjunta                                              */
29 |  |  |  return DecideSeparabilityCS(N_R, N_B, MBR(R), MBR(B))
30 |  |  end
31 |  end
32 end
```

Alg. 6.1: Algoritmo para calcular la separabilidad lineal de R y B.

```
1  Alg.: DecideSeparabilityCS(N_R, N_B, MBR(R), MBR(B))
2  Calcular V_R y V_B acorde a MBR(R) y MBR(B)
3  opt_R ← OptimisticConvexHull(N_R, V_R)
4  opt_B ← OptimisticConvexHull(N_B, V_B)
5  while true do
6      if opt_R y opt_B son disjuntos then
7          return 'si'
8      else
9          if (N_R contiene puntos) AND (N_B contiene puntos) then
10             return 'no'
11         else
12             if N_R contiene MBRs then
13                 pess_R ← PessimisticConvexHull(N_R, V_R)
14             else
15                 pess_R ← opt_R
16             end
17             if N_B contiene MBRs then
18                 pess_B ← PessimisticConvexHull(N_B, V_B)
19             else
20                 pess_B ← opt_B
21             end
22             if pess_R y pess_B no son disjuntos then
23                 return 'no'
24             else
25                 if N_R contiene MBRs then
26                     N_R ← Filter(N_R)
27                     N_R ← ⋃_{N∈N_R} children(N)
28                     opt_R ← OptimisticConvexHull(N_R, V_R)
29                 end
30                 if N_B contiene MBRs then
31                     N_B ← Filter(N_B)
32                     N_B ← ⋃_{N∈N_B} children(N)
33                     opt_B ← OptimisticConvexHull(N_B, V_B)
34                 end
35             end
36         end
37     end
38 end
```

Alg. 6.2: Algoritmo para cacular la separabilidad lineal de R y B cuando $MBR(R)$ y $MBR(B)$ tienen una intersección corner o semi-disjunta: OptimisticConvexHull$(\cdot)$ retorna la cerradura convexa optimista; PessimisticConvexHull$(\cdot)$ retorna la cerradura convexa pesimista; Filter$(\cdot)$ remueve del argumento los rectángulos contenidos en la cerradura convexa pesimista; y children(N) retorna los MBRs (o puntos) que son hijos del MBR N en el correspondiente R-tree.

Resumiendo, el tiempo de ejecución en el peor caso es:

$$
\begin{aligned}
& \sum_{i=0}^{r} \left(O\left(|N_R^{(i)}| \cdot \log |N_R^{(i)}|\right) + O\left(|N_B^{(i)}| \cdot \log |N_B^{(i)}|\right) + O\left(\log |N_R^{(i)}| \cdot \log |N_B^{(i)}|\right)\right) + \\
& \sum_{i=r+1}^{b} \left(O\left(|N_B^{(i)}| \cdot \log |N_B^{(i)}|\right) + O\left(\log |N_R^{(r)}| \cdot \log |N_B^{(i)}|\right)\right) \\
= & \sum_{i=0}^{r} \Big(O(m_i \log m_i) + O(n_i \log n_i) + O(\log m_i \log n_i) \Big) + \sum_{i=r+1}^{b} \Big(O(n_i \log n_i) + O(\log m \log n_i) \Big) \\
= & \sum_{i=0}^{r} \Big(O(m_i \log m_i) + O(n_i \log n_i) \Big) + \sum_{i=r+1}^{b} \Big(O(n_i \log n_i) + O(\log m \log n_i) \Big) \\
= & O\left(\sum_{i=0}^{r} m_i \log m_i + \sum_{i=0}^{b} n_i \log n_i + \sum_{i=r+1}^{b} \log m \log n_i \right) \\
= & O\left(\log m \cdot \sum_{i=0}^{r} m_i + \log n \cdot \sum_{i=0}^{b} n_i + (b-r) \log m \log n \right) \\
= & O\left(\log m \cdot \sum_{i=0}^{r} m_i + \log n \cdot \sum_{i=0}^{b} n_i + \log m \log^2 n \right).
\end{aligned}
$$

Dado que el R-tree tiene la propiedad que cada nodo tiene al menos dos nodos hijos, tenemos que $m_r = m$, $m_{r-1} \leq m/2$, $m_{r-2} \leq m/4$, $m_{r-3} \leq m/8$ y así sucesivamente. Esto es, $m_i \leq m/2^{r-i}$ para $i = 0, 1, \ldots, r$. Similarmente, $n_j \leq n/2^{b-j}$ para $j = 0, 1, \ldots, b$. El tiempo de ejecución total entonces es:

$$
\begin{aligned}
O\left(\log m \cdot \sum_{i=0}^{r} m/2^i + \log n \cdot \sum_{i=0}^{b} n/2^i + \log m \log^2 n \right) &= O\left(m \log m + n \log n + \log m \log^2 n \right) \\
&= O(m \log m + n \log n).
\end{aligned}
$$

El peor caso de nuestro algoritmo ocurre cuando todos los nodos de los dos R-trees, y todos los MBRs y puntos, necesitan ser almacenados en memoria para decidir la separabilidad lineal de R y B. Esto ocurre en el siguiente ejemplo. Supongamos que todos los elementos de B pertenecen a la línea $y = x$, por ejemplo, $B = \{(i, i) : i = 1, 2, \ldots, n\}$, y que $|R| = |B| = n$ con $R = \{(i - \varepsilon, i + \varepsilon) : i = 1, 2, \ldots, n\}$ para $\varepsilon = 1/2$. En este caso, R y B son linealmente separables y $MBR(R)$ y $MBR(B)$ tienen una intersección corner, donde $MBR(R)$ contiene el vértice superior izquierdo de $MBR(B)$ y $MBR(B)$ contiene el vértice inferior derecho de $MBR(R)$. Consideramos cualquier paso de nuestro algoritmo (referencia para la Figura 6.2), con N_R y N_B representando los MBRs de los R-trees de R y B, respectivamente. Observar que en cada MBR de N_B la diagonal que conecta el vértice inferior izquierdo con el vértice superior derecho está contenido en la línea $y = x$. Esto implica que la cerradura convexa pesimista $\mathsf{pess}(N_B)$ es igual al triángulo con vértices $(1, 1)$, $(n, 1)$ y (n, n), y que ningún rectángulo de N_B está contenido en $\mathsf{pess}(N_B)$. Una situación

similar ocurre con N_R: la cerradura convexa pesimista $\mathsf{pess}(N_R)$ es igual al triángulo con vértices $(1-\varepsilon, 1+\varepsilon)$, $(1-\varepsilon, n+\varepsilon)$, y $(n-\varepsilon, n+\varepsilon)$ y ningún MBR de N_R está contenido en $\mathsf{pess}(N_R)$. Entonces, ningún MBR de N_R ó N_B puede ser descartado en algún paso del algoritmo. Además, $\mathsf{pess}(N_R)$ y $\mathsf{pess}(N_B)$ son disjuntos, mientras que las cerraduras convexas optimistas $\mathsf{opt}(N_R)$ y $\mathsf{opt}(N_B)$ no son disjuntas (debido a la manera que elegimos ε). Todas estas observaciones implican que el algoritmo se detendrá despúes de cargar todos los puntos, leyendo todos los nodos y MBRs de ambos R-trees.

6.3.5. La cerradura convexa de un conjunto de puntos

Sea P un conjunto finito de puntos en el plano, almacenados en un R-tree. En esta sección, presentamos un algoritmo para calcular $conv(P)$. Sea N_P un conjunto de MBRs del R-tree P tal que N_P satisface $conv(P) \subseteq conv(N_P)$ (similar a Definición 6.3). Para calcular $conv(P)$, primero descartamos los MBRs N de N_P de tal manera que todos los puntos de P contenidos en N no son vértices de $conv(P)$. Luego, reemplazamos cada MBR N que quede en N_P por sus MBR hijos (o puntos) en el R-tree. Nos detenemos cuando todos los elementos de N_P son puntos y retornamos $conv(N_P)$. Sean v_1, v_2, v_3 y v_4 los vértices superior izquierdo, inferior izquierdo, inferior derecho y superior derecho de $MBR(P)$, respectivamente. Para descartar MBRs de N_P, calculamos las siguientes cerraduras convexas por las siguientes ideas similares a las dadas en la Sección 6.3.1: $C_1 = conv(\{\mathtt{SE}(N) \mid N \in N_P\} \cup \{v_2, v_3, v_4\})$, $C_2 = conv(\{\mathtt{NE}(N) \mid N \in N_P\} \cup \{v_1, v_3, v_4\})$, $C_3 = conv(\{\mathtt{NW}(N) \mid N \in N_P\} \cup \{v_1, v_2, v_4\})$ y $C_4 = conv(\{\mathtt{SW}(N) \mid N \in N_P\} \cup \{v_1, v_2, v_3\})$. Dado un MBR N de N_P, si N está contenido en la intersección $C_1 \cap C_2 \cap C_3 \cap C_4$, entonces no hay un punto de P contenido en N que pueda ser un vértice de $conv(P)$. Además, N está contenido en $C_1 \cap C_2 \cap C_3 \cap C_4$ sí y solamente sí los vértices superior izquierdo, inferior izquierdo, inferior derecho y superior derecho de N están contenidos en C_1, C_2, C_3 y C_4, respectivamente. Una vez que dichas cuatro cerraduras convexas son calculadas, estas cuatro decisiones pueden ser tomadas en tiempo $O(\log |C_1|)$, $O(\log |C_2|)$, $O(\log |C_3|)$ y $O(\log |C_4|)$, respectivamente. El algoritmo para calcular $conv(P)$ es descrito en el pseudocódigo de Alg. 6.3. El tiempo de ejecución es $O(n \log n)$, donde n es el número de puntos.

```
1  Alg.: ConvexHull(P)
2  N_P ← el conjunto de MBRs en el nodo raíz del R-tree P
3  MBR(P) ← MBR(N_P)
4  repeat
5  |  C_1 ← conv({SE(N) | N ∈ N_P} ∪ {v_2, v_3, v_4})
6  |  C_2 ← conv({NE(N) | N ∈ N_P} ∪ {v_1, v_3, v_4})
7  |  C_3 ← conv({NW(N) | N ∈ N_P} ∪ {v_1, v_2, v_4})
8  |  C_4 ← conv({SW(N) | N ∈ N_P} ∪ {v_1, v_2, v_3})
9  |  N_P ← N_P \ {N ∈ N_P | N ⊂ C_1 ∩ C_2 ∩ C_3 ∩ C_4}
10 |  N_P ← ⋃_{N∈N_P} children(N)
11 until N_P contiene puntos;
```

Alg. 6.3: Algoritmo para calcular la cerradura convexa de un conjunto P almacenado en un R-tree.

6.4. Experimentación

En esta sección se describe una serie de experimentos y sus resultados que tienen por objeto evaluar el rendimiento de nuestro algoritmo en términos de tiempo (indirectamente por medio de accesos a bloques del R-tree) y de almacenamiento.

6.4.1. Implementación

Para la ejecución de los experimentos se utilizó un laptop Lenovo ThinkPad x240 (8GB de memoria RAM, procesador Intel Core i5 4300U). El algoritmo fue implementado en el lenguaje $C++$, utilizando la implementación del R-tree de la librería LibSpatialIndex (`http://libspatialindex.org/`). Se considera 1 kilobyte como tamaño de bloque para el R-tree.

En los experimentos se consideran conjuntos de datos reales y datos sintéticos.

6.4.2. Datos reales

Los datos reales están formados por conjuntos de 200 mil y 2,2 millones de objetos (ver Tabla 6.1). Ambos conjuntos de datos representan MBRs de objetos espaciales, de Carreteras de California (en adelante puntos rojos) y de los ríos de Iowa, Kansas, Missouri y Nebraska (en adelante puntos azules) respectivamente[1]. Los conjuntos fueron transformados en conjuntos de puntos, tomando el punto medio de cada MBR y representándolos (normalizándolos) en el espacio [0,1]×[0,1]. Cada conjunto fue almacenado en un R-tree distinto. En la Tabla 6.1 se observa la cantidad de bloques utilizados por los R-tree de cada conjunto.

En Fig. 6.6 (a) observamos ambos conjuntos de datos[2]. En Fig. 6.6 (b) podemos observar que los conjuntos poseen una intersección tipo *semi-disjunta* con una intersección superior al 98 %. En la Tabla 6.1 observamos los resultados de las pruebas realizadas. Como se observa en la Tabla 6.1, se deben acceder un 2,79 % de los bloques del R-tree rojo y un 1,36 % de los bloques del R-tree azul. Para resolver esta instancia del problema, el algoritmo requiere de solo 40 kilobytes de memoria principal.

	Tamaño	% nodos/bloques accedidos	tamaño R-tree (bloques)
rojo	200.000	2,79 %	12.178
azul	2.200.000	1,36 %	35.965

Tabla 6.1: Resultados de pruebas con datos reales.

6.4.3. Datos sintéticos

Usamos datos sintéticos formados por conjuntos de 1, 2, 5 y 10 millones de puntos por cada conjunto (rojo y azul). Se considera una intersección de las zonas o regiones cubiertas por los conjuntos de un 1, 2, 5 y 50 %. Los puntos se normalizaron en el espacio $[0, 1] \times [0, 1]$ con distribución uniforme y distribución Gaussiana.

[1] conjuntos extraídos de `http://rtreeportal.org`

[2] Solo se grafica aproximadamente un 10 % de los puntos de cada conjunto.

Tamaño (en mill. de puntos)	Árbol rojo (bloques de disco)	Árbol azul (bloques de disco)	Total (bloques (bloques de disco)
1	61.386	61.388	122.774
2	122.210	122.226	244.436
5	304.494	304.858	609.352
10	608.671	609.749	1.218.420

Tabla 6.2: Bloques utilizados por los R-tree.

tamaño	tipo de intersección							
	corner				semi-disjunta			
	intersección (%)				intersección (%)			
	1	5	10	50	1	5	10	50
1	0,54	0,31	0,41	0,34	0,34	1,14	1,08	2,11
2	0,23	0,13	0,09	0,35	0,22	0,39	0,58	0,19
5	0,08	0,11	0,03	0,05	0,12	0,36	0,28	0,36
10	0,18	0,03	0,03	0,44	0,09	0,18	0,2	0,27

Tabla 6.3: Porcentaje de nodos accedidos de los R-tree, conjunto de datos con distribución uniforme.

Para la generación de los conjuntos, se definieron dos rectángulos R_1 y R_2 de la misma área dentro del espacio $[0,1]\times[0,1]$, con intersección de un 1, 5, 10 y 50 % del área de R_1. Los puntos se generaron utilizando distribución uniforme y Gaussiana, para después normalizarlos dentro de los rectángulos R_1 y R_2. Por cada rectángulo se generaron conjuntos de 1, 2, 5, 10 millones de puntos. En la Figura 6.7 se observan ejemplos de los conjuntos con distribución Gaussiana e intersección tipo corner.

El tamaño de los R-trees en bloques de disco, en promedio para todas las distribuciones, se muestra en la Tabla 6.2.

Durante las pruebas se realizaron mediciones del porcentaje de nodos accedidos del R-tree por nuestro algoritmo (Fig. 6.8, Tabla 6.3 y Tabla 6.4).

Los resultados indican que, cuando los puntos poseen una distribución uniforme, al aumentar el porcentaje de intersección de R_1 y R_2, es necesario acceder a una mayor cantidad de nodos del árbol (Fig. 6.8 (a) y (b)). Por ejemplo, para 1 millón de puntos y un 50 % de intersección, con intersección semi-disjunta es necesario acceder un 2,11 % de los nodos, en cambio, para conjuntos del mismo tamaño y una intersección de un 1,0 % es necesario leer un 0,34 % de los nodos.

Para los conjuntos de datos con una distribución Gaussiana (Fig. 6.8 (c) y (d)), observamos que el porcentaje de nodos leídos no es influenciado por el porcentaje de intersección de R_1 y R_2. Por ejemplo, para conjuntos de 1 millón de puntos con intersección tipo corner, si estos poseen un 1,0 % y 10,0 % de intersección, es necesario leer un 0,45 % y 0,41 % de los nodos del R-tree, respectivamente, como se muestra en la Fig. 6.8 (a).

tamaño	tipo de intersección							
	corner				semi-disjunta			
	% intersección				% intersección			
	1 %	5 %	10 %	50 %	1 %	5 %	10 %	50 %
1	0,45	0,13	0,41	0,13	0,41	0,16	0,29	0,01
2	0,25	0,24	0,23	0,17	0,18	0,08	0,09	0,19
5	0,03	0,03	0,03	0,03	0,08	0,00	0,01	0,00
10	0,06	0,05	0,06	0,05	0,01	0,00	0,01	0,00

Tabla 6.4: Porcentaje de nodos accedidos de los R-tree, conjunto de datos con distribución Gaussiana.

tamaño	tipo intersección							
	corner				semi-disjunta			
	intersección (%)				intersección (%)			
	1	5	10	50	1	5	10	50
1	24	24	25	26	26	27	27	29
2	42	41	46	46	44	43	43	43
5	09	11	09	11	14	12	14	12
10	17	16	17	21	18	19	18	17

Tabla 6.5: Memoria usada (kilobytes) conjunto de datos con distribución uniforme.

En las Fig. 6.8 (a), 6.8 (b), 6.8 (c) y 6.8 (d) observamos que al aumentar la cantidad de puntos de los conjuntos, el porcentaje de nodos leídos disminuye, siendo este porcentaje menor al 0,1 %.

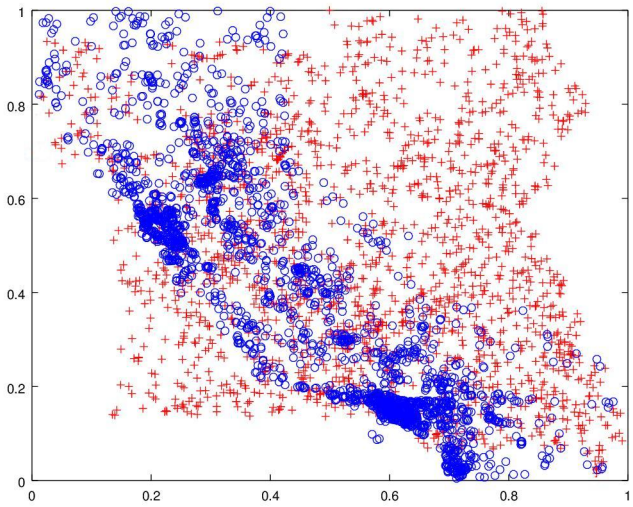
En las Tablas 6.5 y 6.6 se observa la cantidad de memoria requerida por el algoritmo para los conjuntos con distribución uniforme y Gaussiana. Esto incluye la cantidad de memoria usada por la lista de nodos y las cerraduras convexas (optimista y pesimista). Podemos observar que la memoria requerida por el algoritmo varía entre 15 y 47 kilobytes. Además, se observa que ésta no se ve afectada por el aumento del tamaño de los conjuntos. En la Figura 6.9 podemos observar que la cantidad de memoria requerida por el algoritmo posee una tendencia similar, independiente del tipo de intersección (corner o semi-disjunta) y del tipo de distribución (uniforme o gaussiana).

tamaño	tipo de intersección							
	corner				semi-disjunta			
	% intersección				% intersección			
	1 %	5 %	10 %	50 %	1 %	5 %	10 %	50 %
1	26	23	25	23	33	26	28	25
2	44	44	44	43	47	45	43	43
5	08	08	09	08	15	07	09	07
10	19	17	17	17	19	20	17	14

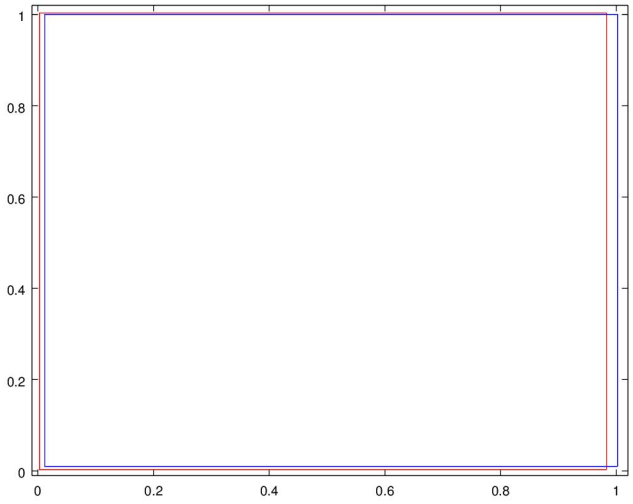
Tabla 6.6: Memoria usada conjunto de datos con distribución Gaussiana.

6.5. Conclusiones

En el presente capítulo se expuso un algoritmo para calcular la separabilidad bicrómatica de dos conjuntos de puntos de cardinalidades n y m respectivamente, almacenados en R-trees distintos. Nuestro algoritmo se beneficia de las propiedades del R-tree, accedienndo a pocos nodos del R-tree y alcanzando una complejidad $O(m \log m + n \log n)$ en el peor caso. Con el propósito de evaluar en la práctica el rendimiento del algoritmo, realizamos una serie de experimentos que consideraron datos sintéticos y datos reales. Los resultados experimentales mostraron que nuestro algoritmo necesitó acceder solo a un 2,1 % de los nodos de los dos R-trees y requiere de aproximadamente 47 kilobytes de memoria principal. Estos resultados representan un ahorro significativo en términos de accesos a disco, memoria utilizada y tiempo de cálculo, si lo comparamos con tener que recuperar todos los puntos de los R-trees, cargarlos en memoria y ejecutar sobre ellos un algoritmo de separabilidad.

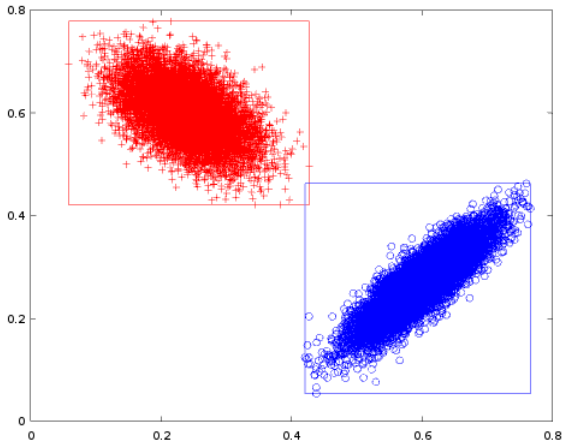


(a)

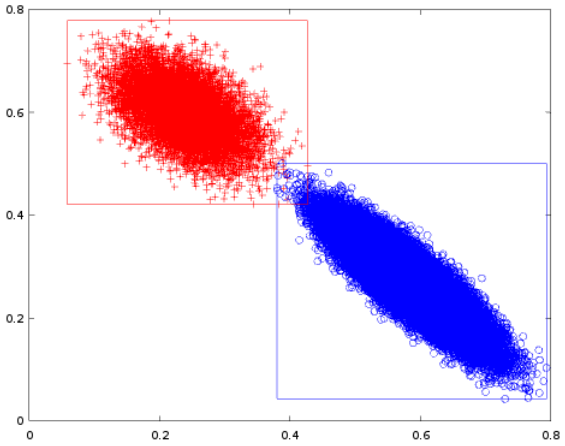


(b)

Fig. 6.6: (a)Conjuntos de datos reales. (b) MBRs de los conjuntos de datos de reales

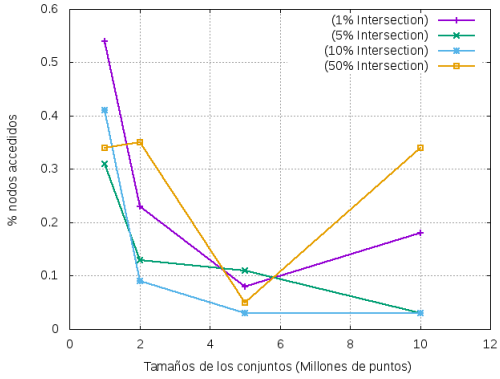


(a) corner, 1 % intersección.

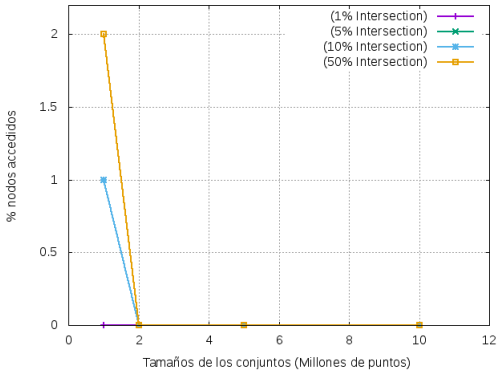


(b) corner, 5 % intersección.

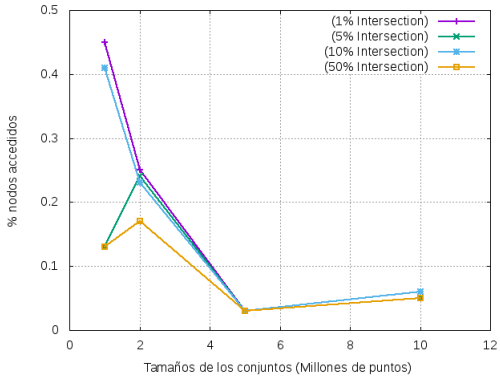
Fig. 6.7: Datos sintéticos con distribución Gaussiana.



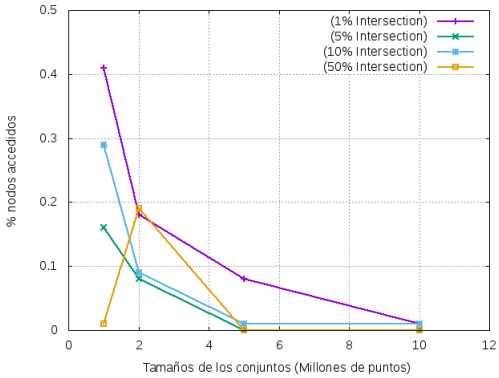
(a) Conjuntos con distribución uniforme, intersección corner.



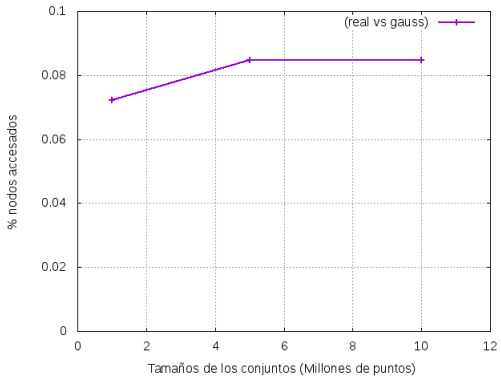
(b) Conjuntos con distribución uniforme, intersección semi-disjunta.



(c) Conjuntos con distribución Gaussiana, intersección corner.

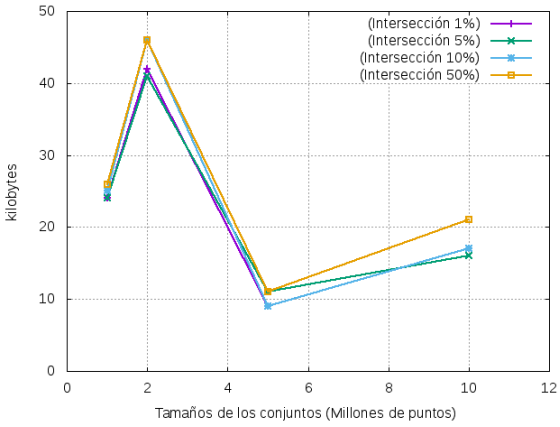


(d) Conjuntos con distribución Gaussiana, intersección semi-disjunta.

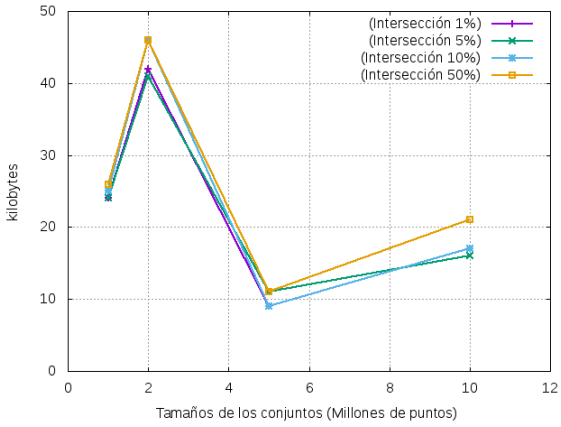


(e) Datos reales vs datos sintéticos con distribución Gaussiana.

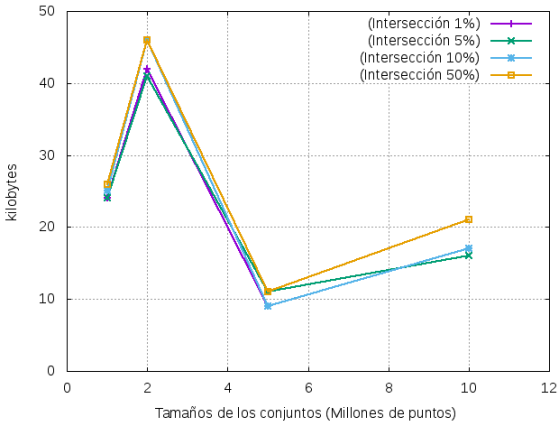
Fig. 6.8: Porcentaje de nodos accedidos.



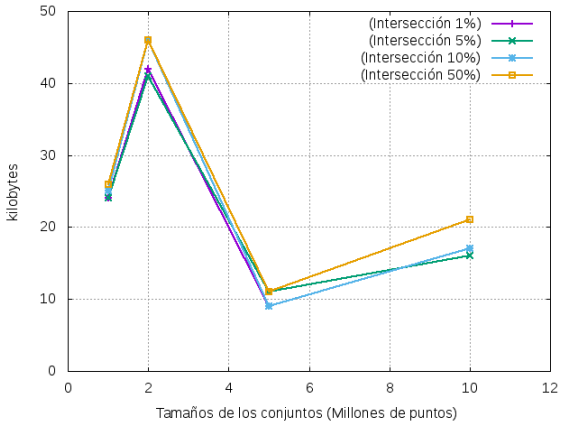
(a) Distribución uniforme intersección tipo corner.



(b) Distribución uniforme intersección tipo semi-disjunta.



(c) Distribución gauss intersección tipo corner.



(d) Distribución gauss intersección tipo semi-disjunta.

Fig. 6.9: Memoria usada.

Parte IV

Conclusiones y trabajo futuro

Capítulo 7

Conclusiones y trabajo futuro

El desarrollo de las BDEs ha motivado el desarrollo de nuevas estructuras de datos y, principalmente, de nuevos algoritmos que permitan dar solución en forma eficiente a los nuevos tipos de consultas que deben dar soporte a estos sistemas. Muchas de estas consultas derivan de problemas propuestos y solucionados en la GC, pero deben ser resueltos en el dominio de las BDEs.

En esta tesis se resuelve el problema de separabilidad lineal en el dominio de las BDEs. En concreto se analizó la separabilidad lineal de dos conjuntos de puntos almacenados en R-trees distintos.

En este capítulo se exponen los resultados de esta tesis y se delínean trabajos futuros interesantes de abordar.

7.1. Conclusiones

En esta tesis se han propuesto dos algoritmos para calcular la separabilidad de dos conjuntos de puntos almacenados en R-trees distintos. Las conclusiones del trabajo realizado son las siguientes:

- De acuerdo a la revisión de la literatura realizada y a nuestro conocimiento, no existen algoritmos que calculen la separabilidad de objetos para conjuntos almacenados en estructuras de datos de memoria secundaria, por lo que nuestros métodos son los primeros en calcular la separabilidad de objetos en el dominio de las BDEs.

- Nuestro primer algoritmo (de ramificación y poda) necesita acceder alrededor de un 17 % de los nodos de los R-trees y requiere procesar un 0,21 % del conjunto total de puntos con el AGC. Esto representa una clara mejora en términos de accesos a disco, memoria utilizada y tiempo de procesamiento, frente a recuperar todos los puntos y procesarlos con el AGC. Esto se debe a que es muy costoso en término de tiempo el acceso a disco, además, el AGC se ejecuta en tiempo $O(n)$, donde n es la cantidad total de puntos de ambos conjuntos, por lo que al disminuir n el tiempo de ejecución también disminuye en forma casi lineal.

- Las desventajas del algoritmo de ramificación son: (1) en todos los casos se necesita descender a las hojas de los R-trees y (2) requiere procesar mediante un AGC los puntos no filtrados para calcular su separabilidad lineal.

- El segundo algoritmo (Convex Hull) se basa en la idea de calcular pseudo cerraduras convexas, que son aproximaciones de la cerradura convexa de cada conjunto, para obtener la separabilidad de los conjuntos. Esto permite evaluar por cada nivel de los R-trees si son linealmente separables los conjuntos, lo que soluciona las desventajas del algoritmo de ramificación y poda, ya que no es necesario descender en todos los casos a las hojas de los R-trees.
- El algoritmo Convex Hull necesita acceder a un 2,1 % aproximadamente de los nodos de los R-trees, requiriendo 47 kilobytes de memoria principal. Esto representa una gran mejora, comparado con el algoritmo de ramificación y poda, al necesitar acceder a una menor cantidad de nodos y no requiere un post-procesado de los puntos.
- La estructura de datos R-tree, se está utilizando en varios productos de Bases de Datos y herramientas software para procesar datos espaciales. Nuestros métodos amplían las capacidades del R-tree, dando soporte a nuevos tipos de consultas espaciales.

7.2. Trabajo futuro

A partir del trabajo realizado en esta tesis, se concluye que se pueden abordar los siguientes problemas:

- Analizar casos en los cuales los conjuntos no son linealmente separables. Se propone calcular una separabilidad lineal no estricta a través de un hiperplano (o línea recta) paralelo a uno de los ejes (x o y), en la cual se definan dos zonas: zona de puntos rojos y zona de puntos azules. La idea es minimizar la cantidad de puntos azules en el sector de los rojos y minimizar la cantidad de puntos rojos en el sector de los azules.
- En la presente tesis se analizó la separabilidad lineal, la cual es una de las formas más básicas de separabilidad. Creemos que este trabajo da las bases para para iniciar el estudio de otros tipos de separabilidad (separabilidad por cuña, por banda, por caja, etc), en conjuntos de puntos almacenados en R-tree distintos. A continuación daremos un ejemplo de uso de la separabilidad por discos. Se desea mantener un control de la población de cóndores y huemules de la Octava Región. Para lo cual se instalarán chips de rastreo a los animales. Para realizar el rastreo, se deben instalar antenas en distintas zonas. Se utilizan dos tipos distintos de antenas, una para los condores y otra para los huemules, las cuales causan interferencias si sus radios de señal se intersecan.
- Finalmente, en esta tesis se abordó el problema de tener los conjuntos almacenados en R-trees distintos. En el contexto de las BDEs los datos pueden estar almacenados en otro tipo de estructuras de datos, tales como el Grid File, K-D-B-tree o incluso en archivos sin índices. Además, los conjuntos pueden estar almacenados en distintas estructuras de datos o estar almacenados en una única estructura. Una interesante extensión de nuestra investigación es estudiar la separabilidad de conjuntos de objetos almacenados en alguna de las configuraciones de estructuras de datos mencionadas.

Bibliografía

[1] Agarwal, P.K., Aronov, B., Koltun, V.: Efficient algorithms for bichromatic separability. ACM Transactions on Algorithms (TALG) 2(2), 209–227 (2006)

[2] Aronov, B., Har-Peled, S.: On approximating the depth and related problems. SIAM Journal on Computing 38(3), 899–921 (2008)

[3] Backer, J., Keil, J.: The mono- and bichromatic empty rectangle and square problems in all dimensions. In: López-Ortiz, A. (ed.) LATIN 2010: Theoretical Informatics, Lecture Notes in Computer Science, vol. 6034, pp. 14–25. Springer Berlin Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-12200-2_3

[4] Barbay, J., Chan, T.M., Navarro, G., Pérez-Lantero, P.: Maximum-weight planar boxes in time (and better). Information Processing Letters 114(8), 437 – 445 (2014), http://www.sciencedirect.com/science/article/pii/S0020019014000507

[5] Bayer, R., McCreight, E.: Organization and maintenance of large ordered indexes. Acta Informatica 1(3), 173–189 (1972), http://dx.doi.org/10.1007/BF00288683

[6] Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles, vol. 19. ACM (1990)

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

[8] Böhm, C., Kriegel, H.P.: Determining the convex hull in large multidimensional databases. In: Data Warehousing and Knowledge Discovery, pp. 294–306. Springer (2001)

[9] Boissonnat, J.D., Czyzowicz, J., Devillers, O., Yvinec, M.: Circular separability of polygons. Algorithmica 30(1), 67–82 (2001)

[10] Borsuk, K.: Multidimensional analytic geometry, vol. 50. PWN-Polish Scientific Publishers (1969)

[11] Cabello, S., Díaz-Báñez, J.M., Seara, C., Sellares, J.A., Urrutia, J., Ventura, I.: Covering point sets with two disjoint disks or squares. Computational Geometry 40(3), 195–206 (2008)

[12] Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M.: Algorithms for processing k-closest-pair queries in spatial databases. Data Knowl. Eng. 49(1), 67–104 (Apr 2004), http://dx.doi.org/10.1016/j.datak.2003.08.007

[13] Corral, A.L.: Algoritmos para el procesamiento de consultas espaciales utilizando R-trees. La consulta de los pares más cercanos y su aplicación en bases de datos espaciales. Ph.D. thesis, Universidad de Almeria (2002)

[14] Cortés, C., Díaz-Báñez, J.M., Pérez-Lantero, P., Seara, C., Urrutia, J., Ventura, I.: Bichromatic separability with two boxes: a general approach. Journal of Algorithms 64(2), 79–88 (2009)

[15] Dıaz-Bánez, J.M., Seara, C., Sellares, J.A., Urrutia, J., Ventura, I.: Covering point sets with two convex objects. 21st European Workshop on Computational Geometry p. 179 (2005)

[16] Duda, R.O., Hart, P.E., et al.: Pattern classification and scene analysis, vol. 3. Wiley New York (1973)

[17] Egenhofer, M.J., Frank, A.U., Jackson, J.P.: A topological data model for spatial databases. Springer (1990)

[18] Farahani, A.H., Bagheri, A.: Finding two maximum separating circles. Advances in Computer Science: an International Journal 3(4), 82–88 (2014)

[19] Gaede, V., Günther, O.: Multidimensional access methods. ACM Comput. Surv. 30(2), 170–231 (Jun 1998), http://doi.acm.org/10.1145/280277.280279

[20] Gagliardi, E.O., Taranilla, M.T., Berón, M., Hernández Peñalver, G.: La geometría computacional a nuestro alrededor. In: IV Workshop de Investigadores en Ciencias de la Computación (2002)

[21] Gutiérrez Retamal, G.A.: Métodos de acceso y procesamiento de consultas espacio-temporales. Ph.D. thesis, Universidad de Chile (2007)

[22] Guttman, A.: R-trees: A dynamic index structure for spatial searching. SIGMOD Rec. 14(2), 47–57 (Jun 1984), http://doi.acm.org/10.1145/971697.602266

[23] Houle, M.E.: Weak separation of sets. Ph.D. thesis, McGill University (1989)

[24] Houle, M.F.: Algorithms for weak and wide separation of sets. Discrete Applied Mathematics 45(2), 139–159 (1993)

[25] Hurtado, F., Mora, M., Ramos, P.A., Seara, C.: Separability by two lines and by nearly straight polygonal chains. Discrete Applied Mathematics 144(1), 110–122 (2004)

[26] Mahapatra, P.R.S., Goswami, P.P., Das, S.: Covering points by isothetic unit squares. In: Canadian Conference on Computational Geometry. pp. 169–172 (2007)

[27] Megiddo, N.: Linear programming in linear time when the dimension is fixed. J. ACM 31(1), 114–127 (1984), http://doi.acm.org/10.1145/2422.322418

[28] Nievergelt, J., Hinterberger, H., Sevcik, K.C.: The grid file: An adaptable, symmetric multikey file structure. ACM Transactions on Database Systems (TODS) 9(1), 38–71 (1984)

[29] OÑeil, D.: Nearest neighbors problem. In: Encyclopedia of GIS, pp. 783–787. Springer US (2008), http://dx.doi.org/10.1007/978-0-387-35973-1_869

[30] O'Rourke, J.: Computational geometry in C. Cambridge university press (1998)

[31] Pérez-Lantero, P.: Algoritmos eficientes para la separación bicromática con dos cajas. Master's thesis, Universidad de Sevilla (2009)

[32] Pérez-Lantero, P.: Geometric Optimization for Classification Problems. Ph.D. thesis, Universidad de Sevilla (2010)

[33] Rigaux, P., Scholl, M., Voisard, A.: Spatial databases: with application to GIS. Morgan Kaufmann (2001)

[34] Robinson, J.T.: The kdb-tree: a search structure for large multidimensional dynamic indexes. In: Proceedings of the 1981 ACM SIGMOD international conference on Management of data. pp. 10–18. ACM (1981)

[35] Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: ACM SIGMOD record. vol. 24, pp. 71–79. ACM (1995)

[36] Seara, C.: On geometric separability. Ph.D. thesis, Univ. Politecnica de Catalunya (2002)

[37] Sellis, T.K., Roussopoulos, N., Faloutsos, C.: The r+-tree: A dynamic index for multidimensional objects. In: Proceedings of the 13th International Conference on Very Large Data Bases. pp. 507–518. VLDB '87, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1987), http://dl.acm.org/citation.cfm?id=645914.671636

[38] Shamos, M.I.: Computational Geometry. Ph.D. thesis, New Haven, CT, USA (1978), aAI7819047

[39] Sheikhi, F., de Berg, M., Mohades, A., Monfared, M.D.: Finding monochromatic l-shapes in bichromatic point sets. In: Canadian Conference on Computational Geometry. pp. 269–272 (2010)

[40] Shekhar, S., Chawla, S.: Spatial databases - a tour. Prentice Hall (2003)

[41] Witzgall, C., Stoer, J.: Convexity and Optimization in Finite Dimensions-i. Springer (1970)

[42] Worboys, M.F., Duckham, M.: GIS: a computing perspective. CRC press (2004)