UNIVERSIDAD DEL BÍO-BÍO

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

Problemas de Separabilidad no Estricta con Restricciones en Bases de Datos Espaciales

Por
Marcelo Alejandro Araya Rivas

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

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

2018 - I

Agradecimientos

Resumen

Resumen Sea $S = R \cup A$, con A (conjunto de puntos azules) y R (conjunto de puntos rojos) en un espacio bidimensional. El problema de la separabilidad de conjuntos de puntos consiste en separar por medio de alguna figura geométrica los objetos de S de tal manera que los puntos azules queden separados de los rojos. Existen dos tipos de separabilidad: estricta y no estricta. En la estricta la figura geométrica separa completamente ambos conjuntos de puntos; en cambio en la no estricta pueden permanecer objetos de ambos conjuntos en un una misma zona. Para dicho problema existen en la literatura variados algoritmos que restringen el tamaño del problema a la cantidad de memoria principal. Sin embargo, debido a los grandes volúmenes de datos espaciales generados actualmente por dispositivos, tales como teléfonos móviles, GPS (Global Positioning System), etc. Se requieren algoritmos para procesarlos bajo el modelo de memoria secundaria. En esta tesis se plantean dos soluciones al problema de Separabilidad no estricta o débil con restricción (SDR) sobre grandes conjuntos de datos espaciales representados en estructuras de datos multidimensionales en memoria secundaria. Una de estas soluciones brinda un resultado exacto accediendo sólo a una parte de los nodos del índice multidimensional R-tree, el otro entrega un resultado aproximado con un 1,35 % de error.

Palabras Clave: Separabilidad Débil con Restricción, Geometría Computacional, Bases de Datos Espaciales.

Abstract

Abstract Let $S = R \cup A$, with A (set of blue points) and R (set of red points) in a two-dimensional space.The problem of the separability of sets of points consists in separating by means of some geometric figure the objects of S in such a way that the blue points are separated from the red ones. There are two types of separability: strict and not strict. In strict separability the geometric figure separates completely both sets of points; On the other hand, in the non strict one, objects of both sets can remain in a same zone. For this problem there are several algorithms in the literature that restrict the size of the problem to the amount of main memory. However, due to the large volumes of spatial data currently generated by devices, such as mobile phones, GPS (Global Positioning System), etc.Algorithms are required to process them under the secondary memory model. In this thesis, two solutions are proposed to the problem of non-strict or weak Separability with Restriction (SDR) on large spatial data sets represented in multidimensional data structures in secondary memory. One of these solutions provides an exact result by accessing only part of the nodes of the multidimensional index R-tree, the other one gives an approximate result with a 1.35 % error.

Keywords: Weak Separability with Restriction (SDR), Computational Geometry, Spatial Databases.

Índice

Parte I

Motivación

Capítulo 1

Introducción

1.1. Motivación

La importancia y utilidad del almacenamiento de grandes volúmenes de datos viene ligada a la capacidad de clasificarlos y analizarlos. Es decir, que almacenar datos solo con la finalidad de almacenarlos y sin la capacidad de procesarlos de manera eficiente no tiene sentido.

Las Bases de Datos Espaciales (BDEs) fueron la respuesta a la necesidad de almacenar y procesar grandes volúmenes de datos espaciales, es decir, estas se utilizan para almacenar objetos que representan espacios del mundo físico, o polígonos cuyas formas serían difíciles de manipular en una base de datos relacional. Las BDEs poseen tipos de datos propios para la representación de puntos, líneas, poli-líneas y polígonos. En general, la aplicación de éstas puede ser muy variada, y se caracterizan por almacenar grandes volúmenes de datos que al momento de consultar no requiere manipularlos todos a la vez gracias a la indexación espacial.

Los objetos espaciales están conformados por datos espaciales, los que representan puntos, líneas, rectángulos, regiones, superficies, volúmenes, e incluso los datos de una dimensión superior, la cual incluye al tiempo. Algunos de los ejemplos de estos datos espaciales consideran ciudades, ríos, caminos, estados, coberturas de cultivos, sierras, hasta representaciones de superficies en 2D y 3D, como los sistemas CAD (Diseño de computación asistida). Un ejemplo de esto último, es el mundialmente conocido programa autoCAD, etc.

A lo largo de la evolución de las BDEs estas se han ido dotando de algoritmos que resuelven problemas geométricos. Los problemas geométricos conforman el área específica, conocida como Geometría Computacional (GC), la que es una disciplina que brinda un marco teórico y formal para el diseño de estructuras de datos y algoritmos requeridos para dar soluciones a problemas de tipo geométrico que surgen en diversas áreas de la Informática.

Dentro de la Geometría Computacional, se han estudiado diversos tipos de problemas, algunos de estos son: la obtención del diagrama de Voronoi [12], la búsqueda de los k-vecinos más cercanos de un punto q [2], búsqueda de los k-pares de vecinos más cercanos[10], cálculo de la cerradura convexa[6] de un conjunto de puntos, entre otros. La clasificación de objetos espaciales en BDEs está estrechamente relacionada a los problemas de separabilidad de Objetos Geométricos.

Los criterios de separabilidad tienen aplicaciones interesantes, como por ejemplo el análisis

de imágenes, la clasificación de datos, etc. Siempre que sea necesario discriminar objetos en un espacio de trabajo, por algún atributo del mismo, los criterios de separabilidad juegan un papel importante.

Algunos de los problemas antes mencionados, pueden ser subclasificados como problemas de separabilidad. Lo que quiere decir que pueden ser resueltos mediante la separación de los objetos implicados.

Esta separación puede ser por medio de una figura geométrica como línea, cuña, doble cuña, banda o por figuras Geométricas Convexas.

La separabilidad puede ser clasificada como estricta o no estricta. Se habla de separabilidad estricta cuando la división por medio de las figuras antes mencionadas, no permite objetos desubicados, es decir cuando la división separa completamente objetos de distintas características. Recientemente se han diseñado algoritmos de separabilidad estricta mediante una línea recta [19] sobre conjuntos de objetos espaciales indexados por una estructura de datos multidimensional R-tree[11]. Por otro lado se habla de separabilidad no estricta o débil, cuando esta permite cierta cantidad de objetos desubicados.

En esta tesis se trata un caso especial del problema de separabilidad no estricta o débil, el cual considera que anticipadamente se decide los subespacios generados por la línea de separación en que cada tipo de puntos permanecerá. Denominaremos a este tipo de separabilidad como Separabilidad Débil con Restricción (SDR). En la *Figura* 1.1. se puede apreciar un ejemplo de un problema SDR.

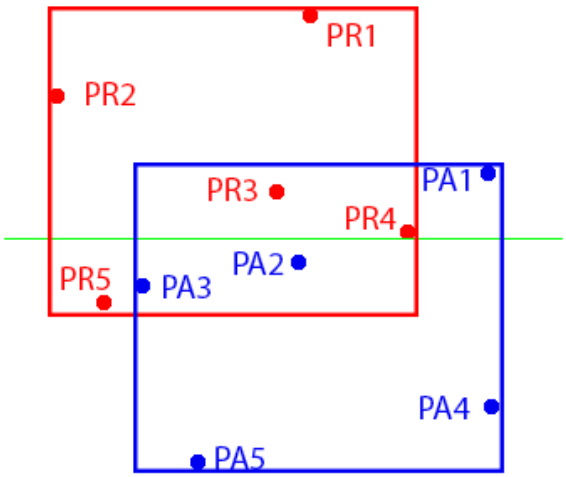


Figura 1.1: Representación de separabilidad no estricta.

En esta figura se observan dos sectores, un sector rojo y otro azul, en un espacio de dos dimensiones. Dentro de cada sector existen puntos rojos y azules respectivamente. Se busca trazar una línea horizontal (línea verde) que separe los puntos rojos de los azules, de tal manera que se encuentre la mayor cantidad de puntos rojos y menor cantidad de puntos azules desde la línea

hacia arriba. Bajo estas condiciones, alguién podría sentirse tentado en trazar la línea en el punto rojo $PR2$, obteniendo un área de solo puntos rojos. Esta solución sería correcta si estuviésemos buscando la separabilidad estricta, pero no para la SDR. Si bien esta línea cumpliría con la condición de obtener la menor cantidad de puntos azules (cero puntos azules), no cumple con obtener la mayor cantidad de puntos rojos. Otra solución incorrecta sería trazar la línea en el punto $PR5$, obteniendo la mayor cantidad de puntos rojos, pero no la menor cantidad de puntos azules. La solución óptima es trazar la línea verde en el punto rojo $PR4$, tal como se muestra en la *Figura* 1.1.

Un dato adicional que se puede concluir de la solución óptima, e incluso de las soluciones erróneas planteadas, es que la línea verde siempre se trazará en un punto rojo, porque ese es el color que se busca maximizar. En el caso de buscar maximizar los puntos azules, la línea será trazada sobre un punto azul. A continuación, mediante tres ejemplos, se muestran la utilidad de contar con una solución para el problema SDR.

Tala de árboles: Asuma un bosque en el que se entremezclan árboles nativos tales como Araucaria y Alerce (puntos azules en la Figura 1.2) que se encuentran protegidos por la ley Chilena 20283, que trata sobre *Recuperación del Bosque Nativo y Fomento Forestal*, con árboles introducidos (pino Oregón, puntos rojos en la *Figura* 1.2, donde las posiciones de los árboles se encuentran georeferenciadas. Una empresa de celulosa, desea comprar el bosque, sin embargo, necesita evaluar la rentabilidad de la inversión.

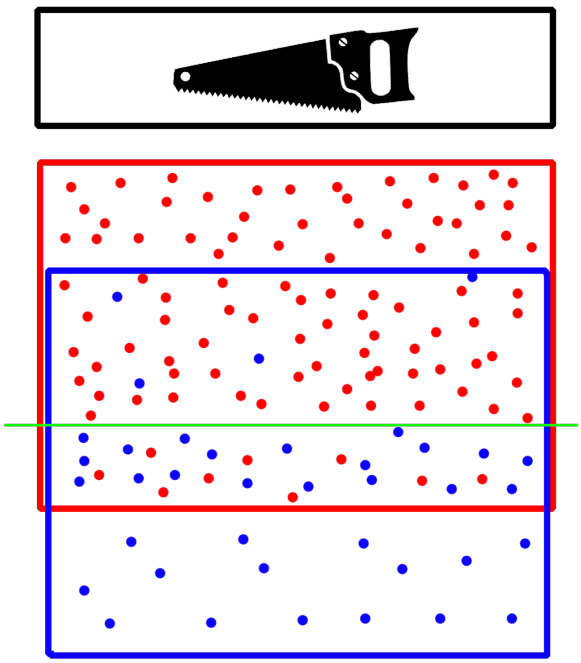


Figura 1.2: Representación Problema del Aserradero

La ley que se refiere a la recuperación del bosque nativo, indica que los árboles nativos no

pueden ser cortados, por lo tanto, si la empresa interesada decide comprar la propiedad, debe proceder a reubicarlos. El costo de reubicación de uno de estos árboles, equivale al doble de la ganancia generada por la tala de un árbol introducido. Por lo tanto, al trazar una línea utilizando el algoritmo de separabilidad, se podría determinar hasta qué punto (geográficamente hablando) es rentable comprar.

Búsqueda de minerales: Otro tipo de problema SDR puede ser el siguiente. Asumamos una mina a cielo abierto como las del norte de nuestro país. En estas existe la posibilidad de extraer el mineral en dirección vertical u horizontal, sin embargo existen rocas duras que encarecen esta tarea. Por medio de un Geo Radar que transmite una señal en el suelo, es posible obtener la ubicación de los minerales y de las rocas sólidas. Estas se representan mediante conjuntos de puntos. El conjunto de puntos rojos para los minerales y el conjunto de puntos azules para las rocas (ver *Figura* 1.3). El problema consiste en encontrar la forma óptima de extracción, ya sea por medio de la perforación vertical o excavación horizontal (de izquierda a derecha) y hasta qué punto es rentable la operación.

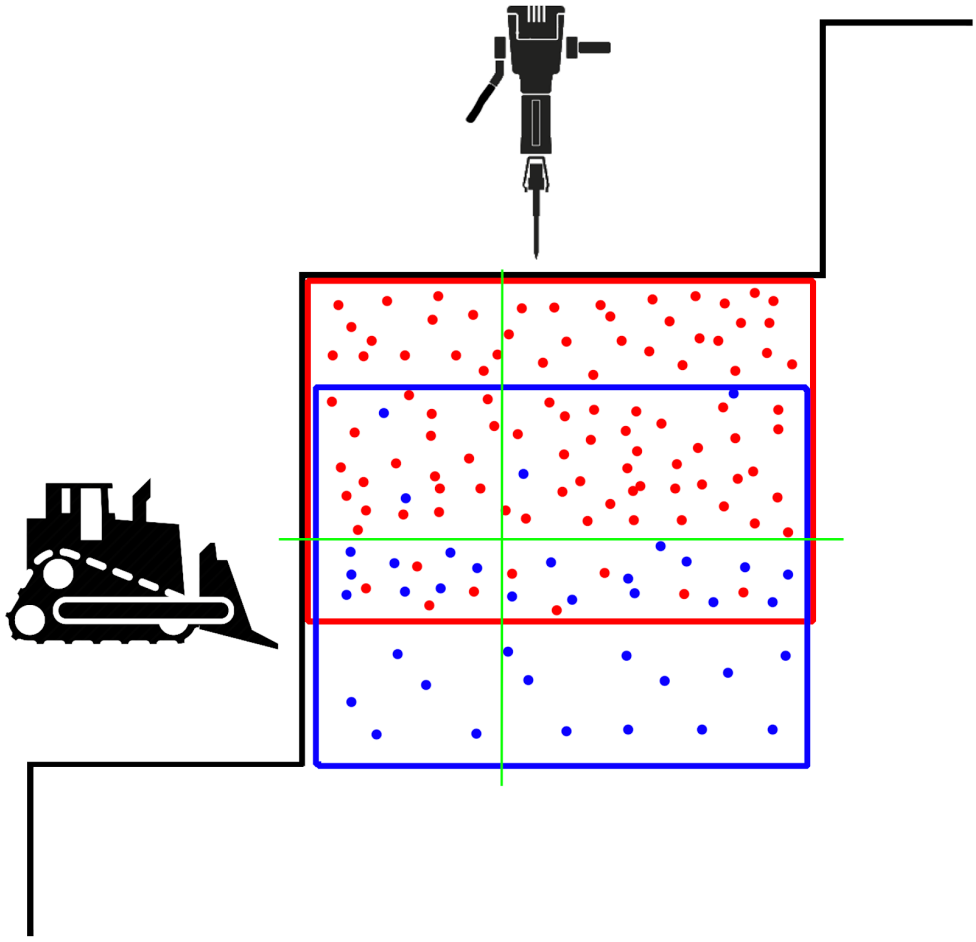


Figura 1.3: Representación Aplicación de Algoritmo en la Minería

Una mala elección del método de extracción podría verse reflejado en el tiempo que toma conseguir el mineral o en el peor de los casos, por daños en la maquinaria utilizada, producto del exceso de rocas. Por otro lado, la correcta elección del método de extracción, y la definición de la distancia hasta la que es óptimo avanzar, generarán mayor mineral en menor tiempo y permitirán que los operadores de la maquinaria puedan comenzar la extracción del mineral en otro punto de la mina. Lo que en resumidas cuentas, significa ganancia para la empresa.

Medicina: Otro ejemplo de un problema SDR se puede hallar en el área de la medicina.

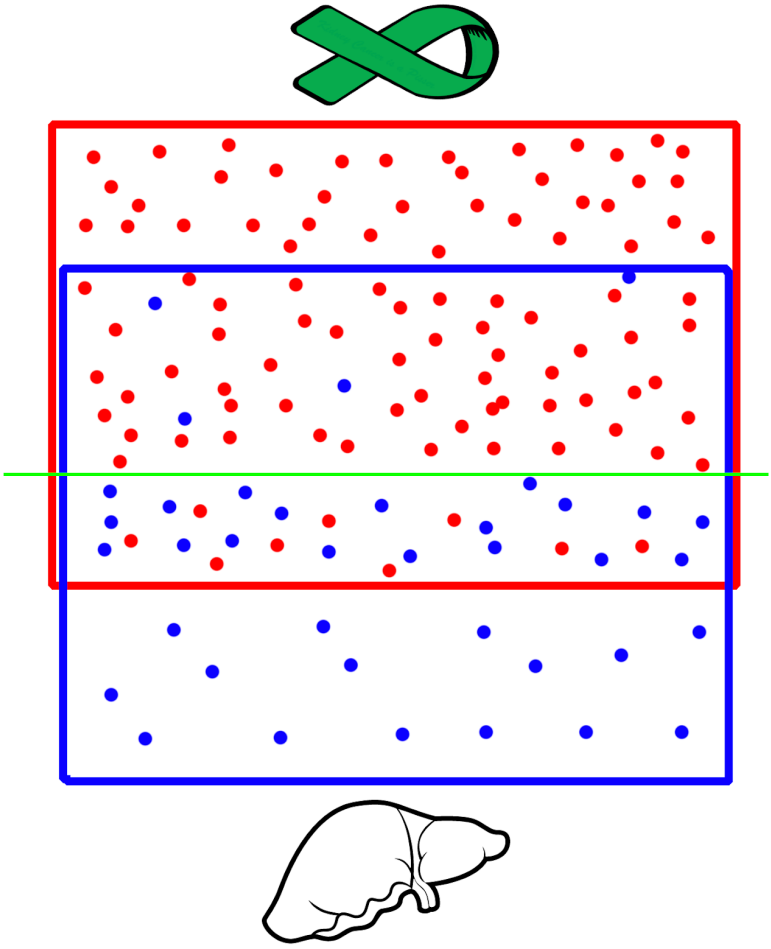


Figura 1.4: Representación Aplicación de Algoritmo en la Medicina

En algunos casos de tratamiento para cáncer de hígado irresecable en etapa I y II (específicamente algunos tumores de tamaño entre 2 a 5 cm que no hayan crecido hacia la vena hepática, que no se hayan extendido a los ganglios y no exista metástasis), se puede realizar un tratamiento de radicación, previo a la cirugía de extirpación del área afectada. Por ejemplo en el caso de un tratamiento de quimioterapia, donde después de la radiación el paciente aún presenta cierta cantidad de células cancerígenas y sea necesaria una operación en el órgano afectado, se debe

determinar hasta qué lugar se pueden eliminar la mayor cantidad de células cancerígenas representadas como puntos rojos en la *Figura* 1.4, afectando la menor cantidad de células sanas. Notar que en este ejemplo, la sección a eliminar se decide de manera anticipada. En la *Figura* 1.4 se ha decido eliminar la parte superior.

Tradicionalmente, tipos de problemas como la búsqueda del par más cercano[10], la búsqueda de los k-vecinos más cercanos de un puntoq[2], o problemas de separabilidad como los mencionados previamente, el diagrama de Voronoi[12] o el cálculo de la cerradura convexa[6], entre otros, han sido resueltos con algoritmos que llevan todos los datos a memoria principal, sin embargo, debido a que esta memoria es limitada y considerando el crecimiento explosivo de los datos espaciales, surge la necesidad de almacenarlos en memoria secundaria. Por esto último se han tenido que diseñar algoritmos para el caso en que los datos se encuentran en memoria secundaria.

En esta tesis se plantean algoritmos para resolver el problema SDR considerando que los puntos de los conjuntos se encuentran almacenados en R-trees, en memoria secundaria.

1.2. Hipótesis y objetivos de la investigación

Formalmente se define el problema de Separabilidad Débil con Restricción (SDR) de la siguiente forma: Sea $S = R \cup B \subseteq R^2$ con R un conjunto de puntos rojos y B un conjunto de puntos azules, o la orientación de la línea de separación l (horizontal o vertical), r la región con respecto a l (superior, inferior, izquierda o derecha) y t el tipo de punto (rojo o azul). El problema SDR consiste en encontrar la línea l de tal manera de maximizar la cantidad de puntos de tipo t menos los puntos del otro tipo (o color) en la región r, es decir, maximizar la expresión $(n - m)$ con n el número de puntos de tipo t y m el número de puntos del otro tipo o color.

Hipótesis

Es posible diseñar algoritmos para resolver el problema SDR considerando que los conjuntos de puntos (en un espacio 2-dimensional) se encuentran almacenados en un R-tree. Es posible conseguir que tales algoritmos accedan a una pequeña parte de los conjuntos siendo más eficientes, en tiempo y almacenamiento, que recuperar en memoria principal todos los puntos desde los R-trees y procesarlos mediante un algoritmo de separabilidad lineal.

Objetivo general

Diseñar, construir y evaluar algoritmos que tomen ventaja de las propiedades de las estructuras de datos espaciales para memoria secundaria y que permitan resolver el problema SDR.

Objetivos específicos

Definir los problemas de Separabilidad Débil con Restricción específicos a resolver (Tipo de figura de separación) y las estructuras de datos multidimensionales a utilizar. Diseñar e implementar algoritmos que resuelvan los problemas de Separabilidad Débil con Restricción ya definidos,

los cuales consideren que los datos o puntos están almacenados en estructuras de datos de memoria secundaria y que tomen ventaja de las propiedades de estas estructuras para resolver el problema de forma eficiente. Evaluar los algoritmos desarrollados utilizando conjuntos de datos sintéticos.

1.3. Alcance de la investigación

Esta tesis contempla el Problema de la Optimización Geométrica solamente para conjuntos de puntos, pertenecientes a un espacio bidimensional. La implementación de los algoritmos se realizará en el lenguaje de programación C++. Los algoritmos construidos se publicarán en el repositorio GitHub.

1.4. Metodología de trabajo

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

- Se realizará una revisión de la literatura, con el propósito de analizar los algoritmos y técnicas propuestos para resolver el problema de Separabilidad no Estricta con Restricción.
- Se realizará una revisión de la literatura con el objetivo de analizar las propiedades de algunas estructuras de datos espaciales que abordan la separabilidad en memoria secundaria.
- Se diseñarán algoritmos que tomen ventaja de las propiedades en las estructuras de datos de memoria secundaria, con el objetivo de disminuir la cantidad de datos a ser recuperados.
- Se implementarán algoritmos que permitan utilizar las técnicas diseñadas en la etapa anterior y que resuelvan el problema planteado.
- Se realizarán análisis experimentales, con datos sintéticos para observar el rendimiento de los algoritmos propuestos.

1.5. Contribuciónes de la Tesis

A continuación se enumeran las principales contribuciones de esta tesis.

- Se presentan cuatro algoritmos que resuelven el problema *SDR*. Dos de ellos, Algoritmo Solución Óptima horizontal y Algoritmo Solución Óptima vertical, entregan la solución de manera exacta, mientras que los otros dos, Algoritmo Aproximado horizontal y Algoritmo Aproximado vertical, tal como su nombre lo indica, entregan una solución aproximada.
- Al resolver el *SDR* con el Algoritmo de Solución Óptima se accede a una menor cantidad de bloques de disco en mayor o menor medida de acuerdo al solapamiento existente en las pruebas, además de disminuir la cantidad de memoria requerida para resolver instancias del problema.

- Con los Algoritmos Aproximados se resuelve el problema SDR accediendo como máximo a un 6 % de los nodos del R-tree y consumiendo un mínimo de memoria principal.

Los algoritmos desarrollados como solución al problema SDR que se proponen en esta tesis, fueron implementados en C++ y compartidos a la comunidad por medio de Github en:

- https://github.com/arayarivasmarcelo/ASO.git
- https://github.com/arayarivasmarcelo/AA.git

Los resultados de esta tesis fueron expuestos en los siguientes trabajos y foros:

- Marcelo Araya, Gilberto Gutiérrez. Problemas de Separabilidad No Estricta en Bases de Datos Espaciales. V Encuentro de Investigación de Estudiantes de Postgrado 2016. Universidad del Bío Bío Concepción.
- Marcelo Araya, Gilberto Gutiérrez. Problemas de Separabilidad No Estricta en Bases de Datos Espaciales. I Workshop de Tesistas Magíster en Ciencias de la Computación, Chillán 2017. Universidad del Bío Bío Chillán.

Finalmente, se encuentra en preparación un artículo para enviar a la revista IEEE Latin America Transactions

- Marcelo Araya, Gilberto Gutiérrez. Problemas de Separabilidad no Estricta con Restricciones en Bases de Datos Espaciales.

1.6. Estructura y contenido de la tesis

Esta tesis consta de 6 capítulos, distribuidos entre 4 partes.

La Parte I, titulada introducción, presenta la motivación, hipótesis y objetivos de la investigación, objetivos generales y específicos de la investigación. Definiendo también los Alcances y Metodología de trabajo utilizada. Y finalmente, se enumeran las Contribuciones realizadas por esta tesis.

La Parte II se titula, Trabajo Relacionado y busca entregar información relevante y conocimientos necesarios para comprender la Parte III. Está conformada por 2 capítulos; Bases de Datos Espaciales y Conocimientos Previos. El capítulo de Bases de Datos Espaciales, trata de introducir en las BDEs, indicando los tipos de datos espaciales, operaciones sobre objetos espaciales, métodos de indexación espacial, para luego tratar la Separabilidad bicromática, tanto, estricta como no estricta o débil y Problemas Geométricos en las Bases de Datos espaciales. El capítulo Conocimientos previos describe el R-tree, sus propiedades, consultas básicas, problemas y aplicaciones y algunas estructuras de datos básicas utilizadas en esta tesis tales como Heap y Colas de prioridad.

La Parte III, donde trata el tema central de esta tesis, se titula Algoritmos de separabilidad débil con Restricción. Esta consta de 2 capítulos; Algoritmos para el problema de separabilidad Débil con Restricción (SDR) y Experimentación. En el primero de estos, se explica el algoritmo de solución óptima (ASO) y Carga de puntos rojos, puntos azules sobre p y algoritmo vertical

y horizontal de solución óptima. Y el algoritmo aproximado (AA). Mientras que en el Capítulo Experimentación, se muestran la implementación, una descripción general de la experimentación, para luego analizar los datos por medio del porcentaje de filtrado, tiempo de ejecución y efecto de solapamiento, además de la conclusión del capítulo.

Finalmente la Parte IV, Conclusiones y trabajo futuro, tal como su nombre lo indica, expone las conclusiones obtenidas de esta tesis, además una propuesta para el trabajo futuro.

Parte II

Trabajo Relacionado

Capítulo 2

Bases de Datos Espaciales

2.1. Introducción

Las Bases de Datos Espaciales (BDEs) nacieron como consecuencia de la creciente necesidad de almacenar, representar y consultar grandes cantidades de objetos espaciales en forma natural [9]. Usualmente se encargan de almacenar datos relacionados con los espacios en el mundo físico, partes de organismos vivientes, diseños en ingeniería, objetos geométricos, entre otros espacios de interés.

Las BDEs fueron la respuesta a la necesidad de almacenar y procesar grandes volúmenes de datos espaciales[20], es decir, estas se utilizan para almacenar objetos que representan espacios del mundo físico, o polígonos cuyas formas serían difíciles de manipular en una base de datos relacional. Las BDEs poseen tipos de datos propios para la representación de puntos, líneas, polilíneas y polígonos. En general, la aplicación de éstas puede ser muy variada, y se caracterizan por almacenar grandes volúmenes de datos que al momento de consultar no requiere manipularlos todos a la vez gracias a la indexación espacial [17], [9]. Los datos espaciales consisten en objetos espaciales, los que se encuentran compuestos de puntos, líneas, rectángulos, regiones, superficies, volúmenes, e incluso los datos de una dimensión superior, la cual incluye al tiempo (datos espacio-temporales). Algunos de los ejemplos de estos datos espaciales incluyen ciudades, ríos, caminos, estados, coberturas de cultivos, sierras, piezas en un sistema CAD, etc.[20, 15].

Una BDE es capaz de modelar, almacenar y consultar tanto datos estándar no espaciales (o alfanuméricos) como datos espaciales. Estos últimos, según [9], se caracterizan por:

- Tener una estructura compleja. Un dato espacial puede estar compuesto de un sólo punto o de varios cientos de polígonos arbitrariamente distribuidos en el espacio y, por lo tanto, no se pueden almacenar en una sola tupla.
- No contar con un álgebra estándar definida sobre los datos espaciales y, por lo tanto, no existe un conjunto de operadores estandarizados. El conjunto de operadores es altamente dependiente del dominio de la aplicación, aunque algunos operadores son más comunes que otros.
- La mayoría de los operadores no son cerrados. La intersección de dos polígonos, por ejemplo,

no necesariamente entrega como resultado otro polígono. Esta propiedad es particularmente relevante cuando se quieren componer operadores tal como se hace con los operadores relacionales.

- El costo computacional de implementar los operadores espaciales es mucho mayor que los operadores relacionales.
- Las BDEs tienden a ser muy grandes. Por ejemplo, un mapa geográfico puede ocupar varios gigabytes de almacenamiento.

2.2. Tipos de datos espaciales

En el contexto de las BDEs, un objeto es modelado por un atributo espacial y diversos atributos no espaciales (números enteros, cadenas de caracteres, etc.). En el modelado de los atributos espaciales se utilizan tres abstracciones básicas: punto, línea y polígono o región. La En la *Figura* 2.1 a continuación, muestra 3 tipos de datos espaciales.

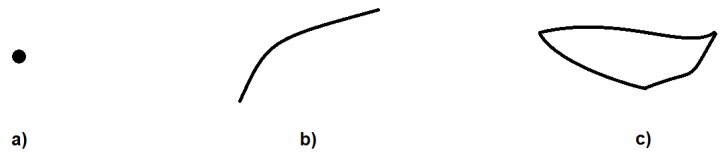


Figura 2.1: Tipos de datos espaciales. a) punto b) línea c) Polígono o región

El *punto* es una figura geométrica sin dimensiones, es decir adimensional, y no tiene longitud, área, volumen, ni otro ángulo dimensional. El punto se encarga de describir una posición determinada en el espacio, respecto de un sistema de coordenadas preestablecido. Algunos ejemplos de un punto en las BDE serían objetos de los cuales solo nos interesa su posición en el espacio como árboles, ambulancias, casas, edificios, antenas, etc.

Una *polilínea* se puede definir como un conjunto de puntos colocados unos detrás de otros por una distancia dada y comunican varios puntos o nodos esta nos permite modelar objetos tales como vías de tren, carreteras, líneas telefónicas, ríos, etc.

Un *polígono* o *región* son figuras planas conectadas por distintas líneas u objetos cerrados que cubren un área determinada, estos nos permiten modelar objetos como por ejemplo países, regiones o lagos.

2.3. Operaciones sobre objetos espaciales

Existen diversos tipos de operaciones sobre objetos espaciales. Las operaciones están determinadas por las relaciones que se establecen entre estos. En esta sección se abordarán operaciones y relaciones existentes entre objetos espaciales.

2.3.1. Relaciones topológicas

Las relaciones topológicas representan la manera en que los objetos interactúan entre sí, es posible distinguir 512 posibilidades de relaciones binarias entre dos objetos espaciales. En un espacio de dos dimensiones y para objetos de tipo polígono, sólo tienen sentido ocho de estas posibilidades [9], las que se encuentran representadas en la *Figura* 2.2 por imágenes y matrices binarias.

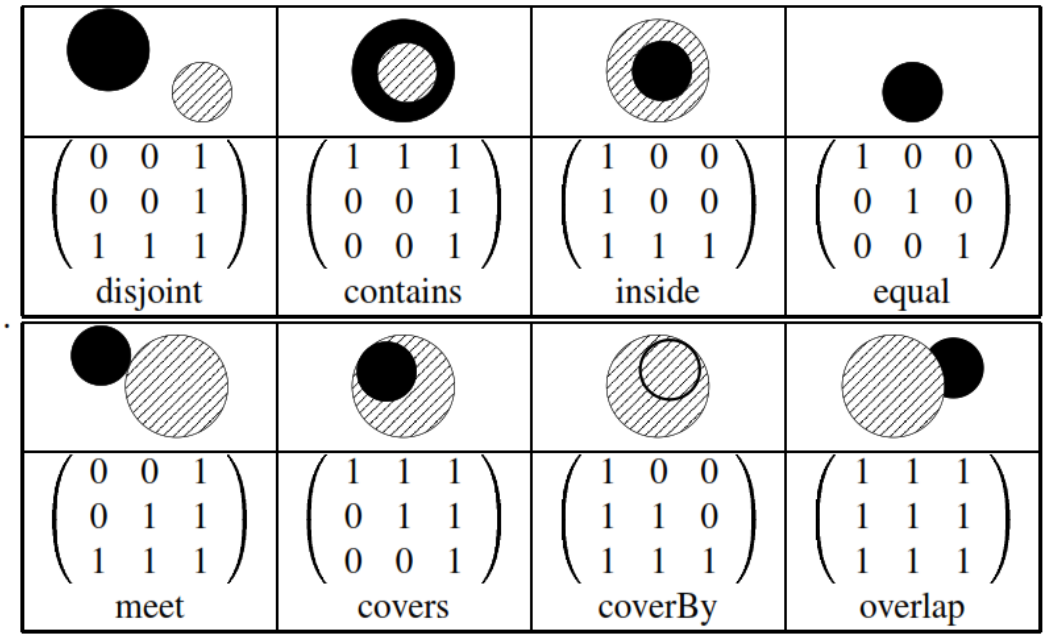


Figura 2.2: Relaciones topológicas binarias entre objetos de tipo región [9].

En la *Figura* 2.2 se representan dos objetos espaciales, el de color negro, será considerado como el objeto A y el que se encuentra achurado como objeto B. La matriz asociada a cada relación topológica representa la existencia de intersección (1 existe y 0 no existe intersección) entre los bordes, el exterior y el interior de cada objeto. Por ejemplo, para disjoint podemos ver que la fila tres tiene solo unos, lo que significa que el exterior de A se intersecta con el borde, el interior y exterior de B. A continuación brevemente se explican cada una de las relaciones topológicas.

- Disjoint: Cuando dos objetos espaciales A y B, se encuentran disjuntos o disjoint, es por que no poseen ninguna interacción.
- Contains: Se habla de contains cuando un objeto A contiene completamente a uno B, pero sin tocar sus bordes.
- Inside: Inside se trata del reverso de Contains, es decir el objeto B contiene al objeto A.

- Equal: Se conoce como iguales o equal a los objetos espaciales en los que sus intersecciones solo ocurren entre los interiores, bordes y exteriores de los objeto, en la matriz se representan como una diagonal de unos y los demás ceros.
- Meet: Se habla de meet o touch cuando dos objetos se tocan, pero no se solapan.
- Covers: Se habla de Covers cuando el objeto espacial A cubre la superficie del objeto B.
- CoverBy: Al contrario del término anterior, un objeto espacial es CoverBy cuando un objeto se encuentra cubierto por otro, y se hace referencia a este (objeto cubierto)
- Overlap: Un región cubre o solapa parte de la otra.

En la *Figura* 2.2 se puede apreciar ciertas similitudes entre las matrices binarias contains e inside. Estas matrices resultan ser matrices transpuestas, de la misma manera que covers con coverBy. También se destaca que para el caso de la matriz equal, esta posee una diagonal principal llena de unos y los demás ceros, es decir, esto ocurre cuando dos objetos espaciales de tipo región son iguales entre sí, de manera que su interior, borde y exterior se encuentren en intersección [9].

2.3.2. Relaciones de dirección u orientación

Las relaciones de orientación, pueden clasificarse en absoluta, relativa a un objeto o basada en el espectador [17]. Las relaciones del primer tipo se definen en el contexto de un sistema de referencia global, algunas de estas relaciones son: norte, sur, este, oeste, noreste, noroeste, sureste, etc. Las relaciones relativas a un objeto, se definen en base a un objeto dado, por ejemplo: arriba, abajo, derecha, izquierda, atrás, etc. Por último, las relaciones de dirección, se encuentran definidas con respecto a un objeto de referencia espacial, previamente designado por un espectador.

2.3.3. Relaciones Métricas

Las relaciones métricas corresponden a atributos numéricos asociados o que se pueden obtener a partir de la geometría o atributo espacial de un objeto. Por ejemplo, el área de un polígono, o la distancia desde un punto a una recta, por nombrar algunos.

2.4. Métodos de indexación espacial

Existe una amplia variedad de métodos de indexación espacial para memoria secundaria. Dichos métodos se pueden clasificar de acuerdo al tipo de objeto espacial que indexan. Por ejemplo KDB-tree [14], GridFile[13], entre otros son índices espaciales que asumen que los objetos son puntos. Un R-tree [11] por su parte, permite indexar principalmente objetos con extensiones (polígonos y polilíneas) aunque también es posible indexar puntos. A continuación describimos brevemente KDB-tree y R-tree. Dado que en esta tesis usamos R-tree, en el Capítulo 3 se entregan más detalles de la estructura (estructura de datos y algoritmos).

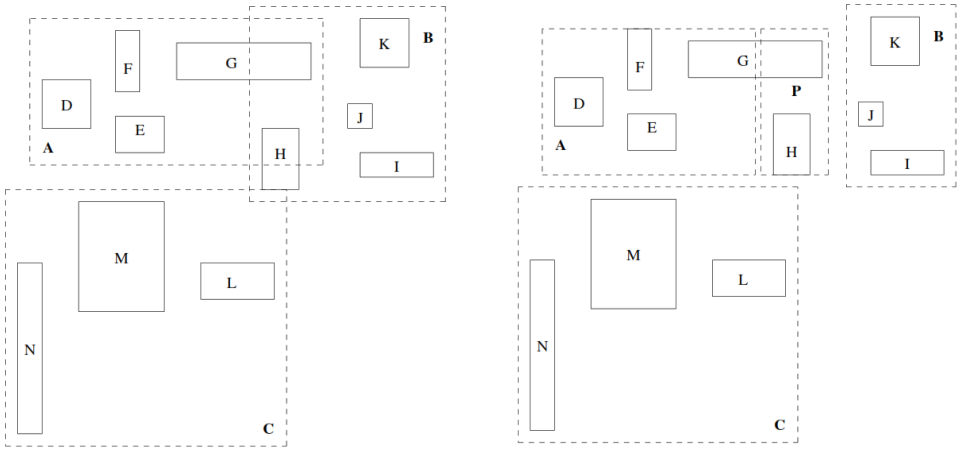
2.4.1. R-tree

Un R-tree corresponde a una extensión de un B-tree[3], y se encuentran diseñados para almacenar objetos espaciales. Este árbol es balanceado y representa una jerarquía de regiones rectangulares denominadas MBR (Minimum Bounding Rectangle), que se encuentran divididas en niveles. Cada nodo del R-tree representa un bloque de disco o página de datos. Todos los nodos hoja se encuentran en el último nivel.

Existen muchas variantes de R-tree. Por ejemplo, R+-tree [16] y R*-tree[4].

R+-tree

Corresponde a una modificación del R-tree, fué diseñada para tratar con uno de los principales problemas del R-tree. El problema se da específicamente al momento de realizar una búsqueda, el R-tree debe seguir más de un camino, por medio de los nodos internos hasta los nodos hoja, debido a que usualmente los MBR se intersectan. Este método se mejoró descomponiendo el espacio de los MBR intersectados, en celdas disjuntas que se mapean en segmentos. Este método basado en la disyunción, divide los MBR en sub MBRs disjuntos arbitrarios y luego agrupa estos sub MBRs en otra estructura. Esta estructura de datos se llama R+-tree.



(a) R-tree (b) R+-tree

Figura 2.3: [16].

La superposición de MBRs visible en la *Figura* 2.3(a) se evita dividiéndolos y creando rectángulos adicionales y más pequeños, como se puede observar en la *Figura* 2.3(b). Cada objeto está asociado con todos los MBRs que intersecta. Todos los MBRs en el árbol no se superponen (con la excepción de los MBRs de los objetos en los nodos hoja). El resultado es que puede haber varias rutas comenzando desde la raíz hasta el mismo objeto tal como se muestra en el árbol de la *Figura* 2.4. Esto puede conducir a un aumento en la altura del árbol, sin embargo, el tiempo de recuperación se acelera.

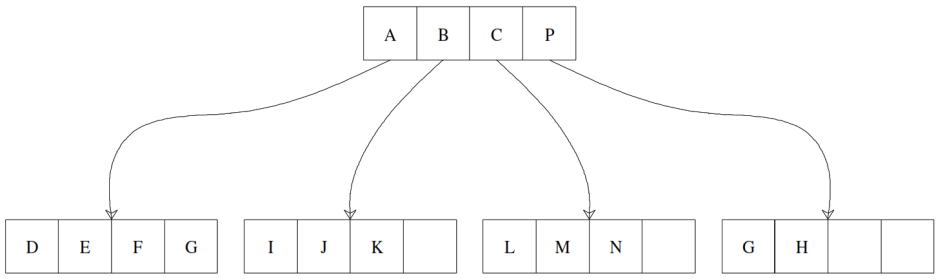


Figura 2.4: R+-tree [16].

R*-tree

Debido a que R-tree es una estructura completamente dinámica, se permite que exista más de un camino de búsqueda, por consecuencia de la superposición de los MBRs. Una de las operaciones afectadas por esta superposición, que puede generar múltiples caminos de búsqueda, es la búsqueda utilizada en la inserción de elementos. Según se explica en [4], la fase de inserción resulta ser crítica en el rendimiento de la etapa de búsqueda, ya que se ha comprobado que bajo diferentes distribuciones de datos, este último presenta distintos comportamientos.

R*-tree introduce una política de inserción, denominada *reinserción forzada*. El proceso de la *reinserción forzada* consiste en no dividir un nodo cuando este se encuentre lleno, mientras que propone eliminar p entradas del nodo y reinsertarlas en el árbol.

Los algoritmos propuestos para un R*-tree cuentan con la optimización de los siguientes parámetros [4]:

- El área de sobreposición entre regiones en el mismo nivel del árbol debería ser minimizada, para disminuir la probabilidad de seguir múltiples caminos.
- Los perímetros de los MBRs deberían ser minimizados. La forma rectangular preferida es el cuadrado, ya que es la representación rectangular más compacta.
- La utilización del almacenamiento debiera ser maximizada.

Gracias a los cambios enumerados, los resultados de rendimiento presentaron mejoras de hasta un 50 % en comparación con un R-tree básico, además de mostrar que la política de reinserción forzada puede optimizar la utilización de almacenamiento.

Por otra parte, las operación de búsqueda y eliminación no cambian de las utilizadas en R-tree.

2.4.2. KD-tree y KDB-tree

KD-tree [5] es una estructura de datos multidimensional que se utilizan para almacenar e indexar objetos de tipo punto en un espacio K-dimensional. Este es un árbol binario que representa una subdivisión recursiva del espacio utilizando hiperplanos de $n-1$ dimensiones. Siendo n el número de posibilidades de direcciones en las que los hiperplanos alternan.

Al igual que R-tree las hojas almacenan los puntos del conjunto de datos. Cada punto se almacena en una hoja, además cada hoja almacena al menos un punto.

En la *Figura* 2.5 se puede apreciar un KD-tree en un espacio bidimensional. A la izquierda se presenta la división del espacio generado por un KD-tree y a la derecha se presenta en forma de árbol.

Los nodos del árbol corresponden a las divisiones del espacio (estas se encuentran paralelas a los ejes X o Y). Las líneas que pasan por sobre los puntos dividen el espacio total en sub-espacios, por ejemplo p_1 divide en dos sub-espacios, p_2 y p_3, y a su vez p_3 se divide en dos sub espacios p_5 y p_6 y así sucesivamente.

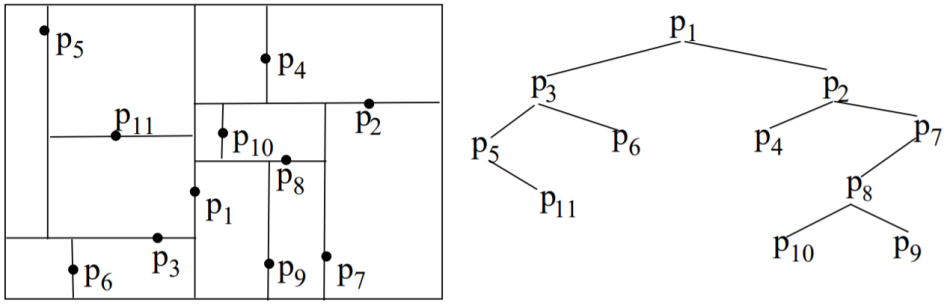


Figura 2.5: K-D-tree [9]

El KDB-tree [14] (*Figura* 2.6) es una estructura de búsqueda para grandes índices dinámicos multidimensionales. Corresponde a la combinación de KD-tree y B-tree. Este árbol contiene nodos internos y nodos hoja. Cada nodo se almacena como una página para que se pueda hacer un uso eficiente de la memoria secundaria con paginación. Es balanceada en el sentido de la cantidad de nodos a los que accede, ya que la ruta desde el nodo raíz a un nodo hoja es la misma para todos los nodos hoja. A diferencia de B-trees, no se puede garantizar el 50 % de utilización de las páginas, aunque se espera que en la práctica sea menos de la mitad, en comparación con el número total de páginas.

2.5. Separabilidad bicromática

La Geometría Computacional (GC), es una disciplina que brinda un marco teórico y formal para el diseño de estructuras de datos y algoritmos requeridos para dar soluciones a problemas de tipo geométrico que surgen en diversas áreas de la Informática. De manera informal el problema de la separabilidad bicromática consiste en lo siguiente: Asumiendo dos conjuntos de puntos en un espacio, uno que contiene puntos de color rojo y otro con puntos azules, la separabilidad lo que busca es lograr separar estos por medio de un objeto geométrico tal como línea recta, cuña, cono, u otros [19]. El problema de separabilidad posee muchas variantes y casos particulares, algunos ejemplos de estos son por medio de figuras geométricas tales como línea recta (*Figura* 2.8), banda (*Figura* 2.9), cuña (*Figura* 2.7 a), doble cuña (*Figura* 2.7 b), etc. Los problemas de separabilidad de objetos nacen de la necesidad en diversas áreas, por ejemplo estadísticas,

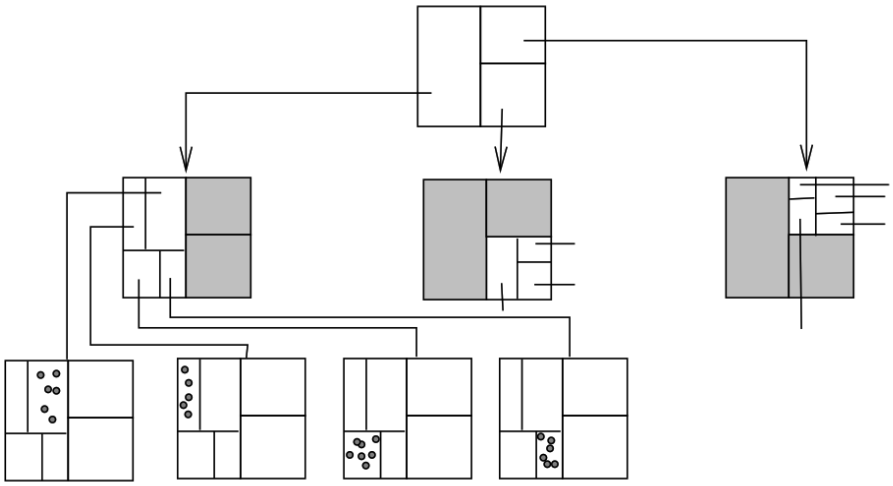


Figura 2.6: KDB-tree [14]

análisis de imágenes, localización, georreferencia, análisis de recursos naturales, medicina, etc. Dependiendo si los conjuntos resultantes son monocromáticos o bicromáticos se habla de separabilidad estricta y separabilidad débil (no estricta) respectivamente. La separabilidad débil o no estricta también se conoce como problema de optimización geométrica y se puede enunciar como Eliminar el menor número posible de puntos, de tal forma que los puntos restantes sean separables según la figura geométrica seleccionada [18]. Si la figura geométrica seleccionada es una línea recta, entonces se habla de separabilidad lineal. Considerando que es posible almacenar el conjunto S en la memoria principal, en la literatura se han propuesto varios algoritmos para resolver el problema de la separabilidad bicromática y sus variantes.

2.5.1. Separabilidad Estricta

La separabilidad puede ser clasificada como estricta cuando luego de separar por medio de un objeto geométrico, no existen puntos de ambos colores en una misma zona generada por el objeto. Por ejemplo, en la *Figura* 2.7 a) los puntos azules en el interior de la cuña resulta ser monocromáticos. A continuación se discuten los principales tipos de separabilidad geométrica estricta.

Separabilidad por cuña y por doble cuña:

Una cuña puede definirse como la unión de dos líneas con un origen en común, estas pueden formar un ángulo obtuso o convexo, sin embargo lo importante es que separen los puntos de distinto color. En la *Figura* 2.7 a) y b) se muestra escenarios de separabilidad por cuña y doble cuña respectivamente.

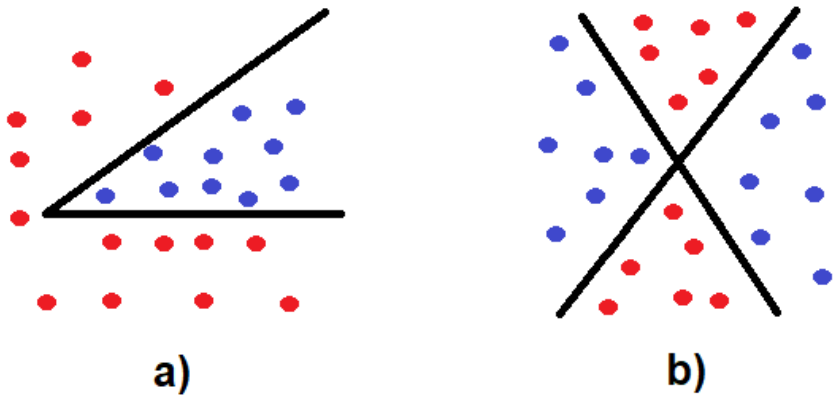


Figura 2.7: Separabilidad por a) cuña b) doble cuña

Separabilidad Lineal

Consiste en trazar una línea recta que separe dos conjuntos de puntos en 2 regiones, de tal manera que no existan regiones bicromáticas. La *Figura* 2.8 muestra un ejemplo de separabilidad lineal, por medio de una línea vertical.

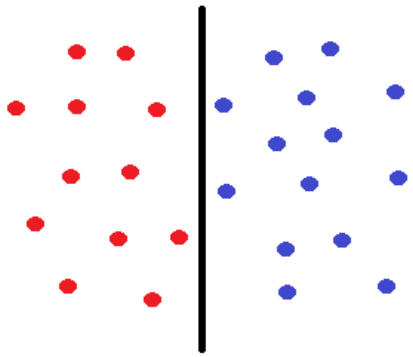


Figura 2.8: Separabilidad Lineal

Separabilidad por banda

Se habla de banda cuando se tienen dos líneas rectas paralelas, que se encuentran siempre a la misma distancia la una de la otra; no importa qué tanto se extiendan, estas nunca se tocarán. En la *Figura* 2.9 se observa que los punto azules están contenidos entre las líneas paralelas, mientras que fuera de ellas, solo existen puntos rojos.

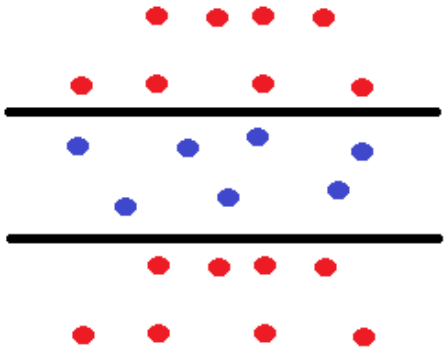


Figura 2.9: Separabilidad por banda

Separabilidad por Figuras Geométricas Convexas

Un polígono es convexo, sólo si todos sus ángulos internos son estrictamente menores o iguales a 180° grados. La separabilidad por figuras geométricas convexas consiste en dibujar un polígono en el plano R^2, con el objetivo de que este contenga el mayor número de puntos de un solo color. Como se aprecia en la *Figura* 2.10, puede ser realizada por un cuadrado o diversos polígonos convexos. Algunos ejemplos de estas serían separabilidad por caja, circular, etc.

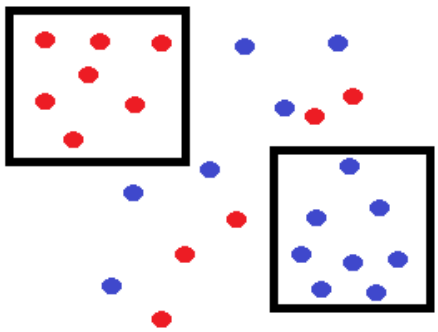


Figura 2.10: Separabilidad por caja.

Separabilidad no estricta o débil

La separabilidad no estricta o débil es, al contrario de la separabilidad estricta, no restrictiva al momento de separar los objetos, es decir permite que la figura geométrica no separe totalmente ambos conjuntos. Este tipo de separabilidad puede utilizar los mismos métodos anteriormente expuestos; línea, banda, cuña, doble cuña o figura geométrica convexa. En la *Figura* 2.11 se puede apreciar un ejemplo de separabilidad de manera no estricta, por medio del método de figura geométrica convexa (cajas). Aquí es posible notar que en las dos regiones de separación existen puntos desubicados, es decir, se mezclan puntos azules con puntos rojos. Pero en ambos casos existe un color predominante. En la caja de la derecha existe una mayoría de puntos azules, mientras que en la caja de la izquierda, predominan los puntos rojos.

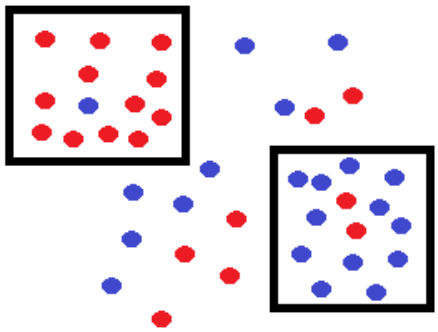


Figura 2.11: ejemplo de Separabilidad No Estricta.

La separabilidad débil con restricción será el tema central de esta tesis.

2.6. Problemas geométricos en el contexto de las BDEs

Existen muchos problemas geométricos que han sido resueltos utilizando estructuras de datos en memoria secundaria, principalmente R-tree. A continuación se describen algunos de estos y sus respectivas soluciones existentes.

Dentro de las BDEs resulta interesante el obtener información de dos conjuntos de datos distintos, por ejemplo el conjunto de las calles de una ciudad, y el conjunto de los puntos turísticos de la misma.

Uno de los problemas clásicos de abordar en geometría computacional, es los K pares de vecinos más cercanos de dos conjuntos distintos, del cual existen soluciones en el contexto de las BDE. Por ejemplo, lo presentado en Corral [7], propone una estrategia de ramificación y poda que usan propiedades del R-tree. Estos algoritmos utilizan métricas (*Figura* 2.12) que permiten establecer distancias entre los objetos analizados.

Con respecto a la estrategia mencionada, esta utiliza dos métricas, $MINMINDIST$ y

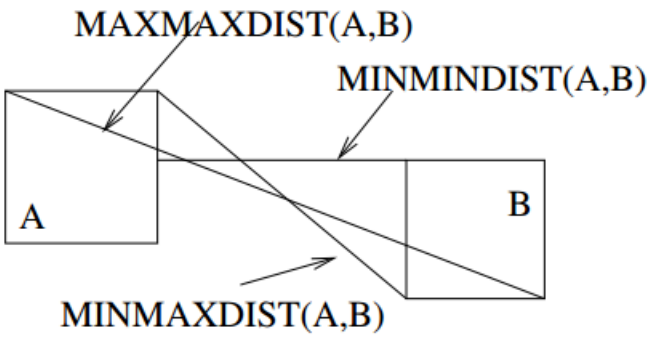


Figura 2.12: K pares de Vecinos más Cercanos.

$MINMAXDIST$, para realizar la poda del R-tree.

El algoritmo recorre el R-tree en profundidad, partiendo desde la raíz del árbol y cuando se detecta que una rama o subárbol en el recorrido del R-tree no llevará a una solución, esta se descarta.

Durante el recorrido del árbol y para cada nodo interno visitado, se calcula $MINMINDIST$ de cada uno de los MBRs del nodo. Las entradas del nodo se ordenan de acuerdo a $MINMINDIST$ en una lista denominada Active Branch List (ABL) y luego se aplican las estrategias de poda sobre ella. El algoritmo itera sobre la lista ABL hasta que esta se encuentre vacía. En cada iteración, el algoritmo selecciona el siguiente subárbol recorriendolo recursivamente [7].

Cerradura Convexa

La Cerradura Convexa o Convex Hull corresponde al polígono convexo más pequeño que encierra un conjunto de puntos, de modo que cada punto del conjunto se encuentre dentro del polígono o en su perímetro. En la *Figura* 2.13. se muestra una representación de la cerradura convexa capaz de encerrar un conjunto de puntos.

En la literatura es posible encontrar varios algoritmos para resolver la Cerradura Convexa de un conjunto de puntos, tales como Quick Hull, Jarvis'march, Graham's scan, Depth-First (FPO), entre otros.

Uno de los algoritmos que se ejecuta en memoria secundaria para calcular la cerradura convexa, es el propuesto por Böhm y kriegel en "Determining the Convex Hull in Large Multidimensional Databases" [6], en el proponen dos algoritmos que aprovechan las propiedades de las estructuras de índice multidimensional jerárquicos tales como R-trees. Uno de ellos se encarga de recorrer el índice en profundidad. El otro algoritmo asigna una prioridad a cada nodo activo (nodos que son conocidos por el sistema, y que no accede), que corresponde a la distancia máxima de la región del nodo al Convex Hull tentativo. Se muestra teórica y experimentalmente que sus algoritmos superan a las técnicas competitivas que no explotan los índices.

Existen variadas formas de aplicar el problema de la cerradura convexa a la resolución de otros problemas, como lo presentado en Torres [18], cuando determina si dos R-trees solapados, son separables mediante separabilidad lineal de manera estricta. Torres propone dos algoritmos

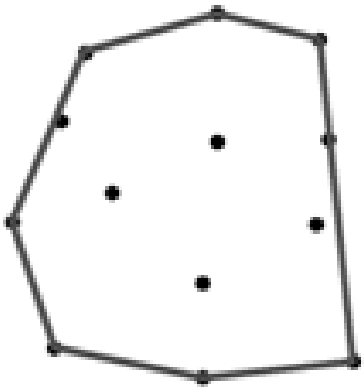


Figura 2.13: Representación Cerradura Convexa [8].

que resuelven el problema sobre conjuntos de puntos almacenados en R-trees distintos y uno de ellos utiliza cerraduras convexas como parte fundamental.

El algoritmo [19], calcula en cada nivel de los R-trees dos cerraduras convexas aproximadas, es decir, cerradura convexas basadas en los puntos de los MBRs de los nodos interiores de los R-trees. Una de las cerraduras se denomina optimista (*Figura* 2.14 a) y la otra pesimista (*Figura* 2.14 b), para luego determinar si las cerraduras son disjuntas, (lo que significa que son linealmente separables). Mediante la intersección de estas cerraduras convexas el algoritmo evita acceder a todas las páginas del R-tree, accediendo a una pequeña fracción de los bloques de los R-trees. Como resultado de esto, obtiene respuestas parciales con las cuales es posible decidir la separabilidad de los conjuntos sin haber alcanzado necesariamente las hojas de estos R-trees.

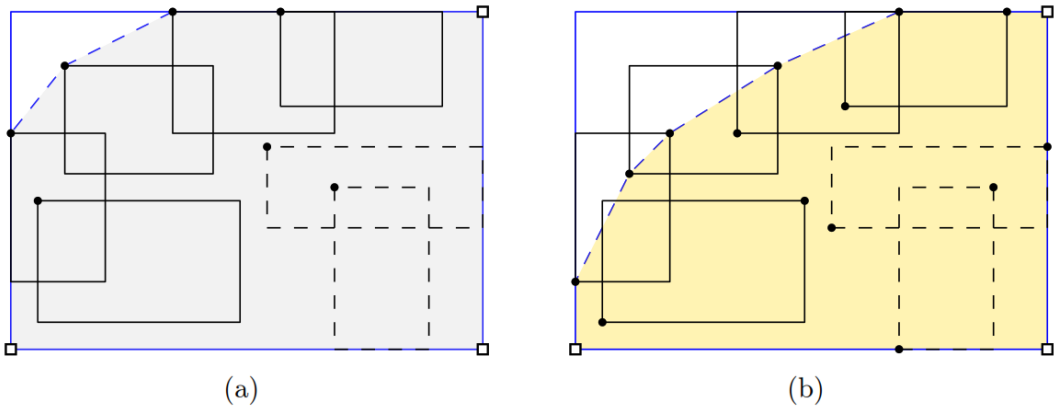


Figura 2.14: Cerradura Convexa Optimista y Pesimista [18].

Capítulo 3

Conocimientos Previos

3.1. Introducción

Este trabajo de tesis utiliza distintas estructuras de datos y consultas básicas asociadas y por lo tanto se requiere noción de las técnicas y estructuras de datos que emplearemos en la solución. Este capítulo describe estructuras de datos y explica conceptos necesarios para comprender el algoritmo propuesto en esta tesis, el que se expondrá en el Capítulo 4.

3.2. R-tree

El R-tree corresponde a una estructura de datos con forma de árbol balanceado utilizada para representar datos espaciales en memoria secundaria. Los datos en el R-tree se almacenan de manera jerárquica donde cada nodo posee un número finito de hijos que está limitado inferior y superiormente, excepto para la raíz del árbol. Cada nodo representa un área espacial y todos sus hijos están ubicados dentro de esta área [11].

En la *Figura* 3.1 se muestra la representación en forma de árbol de un R-tree, mientras que la *Figura* 3.2 da un ejemplo de la distribución espacial en distintos niveles, que presentan los datos del R-tree en un plano bidimensional (en el que se podrían representar, por ejemplo, coordenadas geográficas). Como se puede ver, la raíz corresponde a un rectángulo que encierra a todos sus hijos utilizando la menor superficie posible. Este rectángulo se conoce como MBR (por sus siglas en inglés Minimum Bounding Rectangle) y cada uno corresponde a un nodo del R-tree. En la *Figura* 3.1 se puede ver que los nodos hoja del árbol son los nodos $A1, A2, A3, B1, B2, B3, C1, C2, C3$ los cuales almacenan los puntos y se ubican en el mismo nivel del árbol. Todos los demás nodos del R-tree (que no son hojas) poseen como hijos más nodos.

3.2.1. Propiedades

Si se tiene a m como el número mínimo de entradas que se puede almacenar en un nodo, y sea M el número máximo de entradas almacenables por un nodo, un R-tree satisfacen las siguientes propiedades [9]:

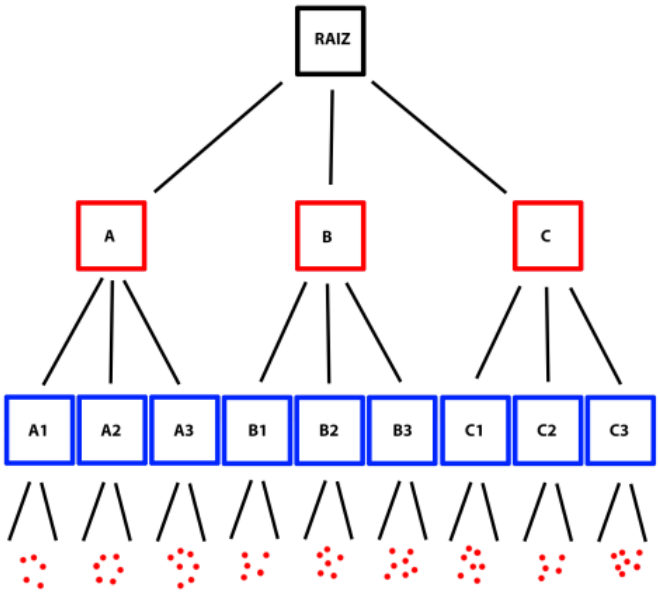


Figura 3.1: Representación R-tree en forma de árbol

- Cada nodo interno tiene entre m y M hijos, a menos que sea la raíz.
- Cada nodo contiene entre m y M entradas a menos que corresponda a la raíz.
- Para cada entrada $\langle MBR, oid \rangle$ en un nodo hoja, MBR es el mínimo rectángulo que contiene (espacialmente) al objeto.
- Para cada entrada de la forma $\langle MBR, ref \rangle$ de un nodo interno, MBR es el rectángulo más pequeño que contiene espacialmente los rectángulos definidos en el nodo hijo.
- El nodo raíz tiene al menos dos hijos, a menos que sea una hoja.
- Todas las hojas se encuentran al mismo nivel.
- El número mínimo de entradas es $m < (M/2)$

3.2.2. Operaciones básicas en un R-tree

Algunas operaciones básicas que pueden ser realizadas a un R-tree son búsqueda, eliminación e inserción, a continuación se presentan algunos algoritmos desarrollados en el trabajo de Gutiérrez [9].

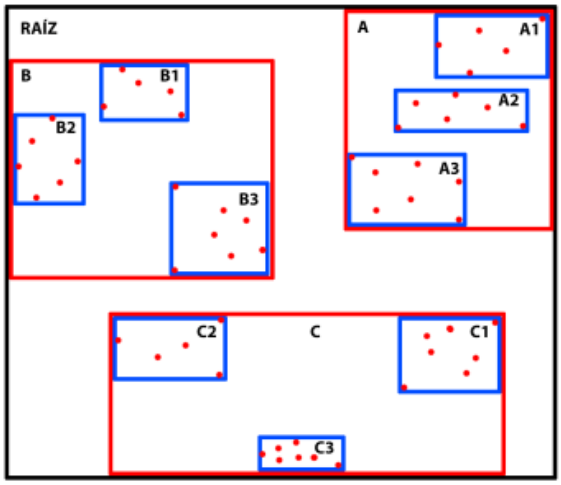


Figura 3.2: Representación R-tree

Búsqueda

La forma de recorrer un R-tree es desde la raíz del árbol R, descendiendo hasta alcanzar los nodos hoja. En los nodos internos el algoritmo debe decidir por cuál de sus hijos continuar la búsqueda, producto de que las particiones en los diferentes niveles no son disjuntas, puede ser necesario recorrer varios caminos desde un nodo interno a las hojas en una consulta, la decisión consiste en verificar si el MBR del nodo hijo interseca con el objeto de búsqueda Q, tal como se observa en el pseudocódigo del algoritmo en *Alg.* 3.1

Eliminación

Para eliminar una entrada u objeto desde un R-tree se comienza buscando como en el *Alg.* 3.1 el nodo hoja en el cual se encuentra el objeto como se muestra en el *Alg.* 3.2 Si dicho nodo queda con menos entradas que las permitidas, estas deberán ser reubicadas 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 el *Alg.* 3.2 se observa el pseudocódigo del algoritmo de eliminación del R-tree.

La función $EncontrarHoja()$, presente en el *Alg.* 3.2, consiste en realizar una búsqueda del objeto a eliminar dentro del R-tree. Mientras que la función $CondensarRtree()$ es utilizada luego de eliminar un nodo hoja, siempre cuando dicho nodo quede con menos entradas que las permitidas y sea necesario reubicar. Dicha función consiste en reajustar las áreas cubiertas por los rectángulos a partir de la raíz hacia la hoja.

Algoritmo BuscarEnRtree(T,Q)

{ L es un conjunto de punteros a objetos cuyos MBRs se intersecan con Q. E representa una entrada de un nodo, $E.MBR$ se usa para referirse al MBR de una entrada y $E.p$ para indicar, ya sea una referencia de un nodo interno del R-tree o una referencia donde se almacena el atributo espacial o geometría exacta del objeto}

```
1:   L = 0
2:   for cada entrada E ∈ T do
3:       if E.MBR ∩ Q ≠ 0 then
4:           if T es una hoja then
5:               L = L ∪ {E.p}
6:           end if
7:       else
8:           L = L ∪ BuscarEnRtree{E.p, Q}
9:       end if
10:  end for
11:  return L
```

Alg. 3.1: Búsqueda en R-tree [9].

Algoritmo EliminarEnRtree(T, E)

{ L apunta al nodo hoja del R-tree que contiene la entrada del objeto a eliminar}

```
1:   L = EncontrarHoja(T, E)
2:   if E no se encuentra then
3:       return "ERROR"
4:   else
5:       Remover E de L
6:       CondensarRtree(L)
7:       if después de condensar el R-tree, la raíz tiene un solo hijo then
8:           Dejar como nueva raíz del R-tree al hijo
9:       end if
10:  end if
```

Alg. 3.2: Eliminación en R-tree [9].

Inserción

La inserción de un objeto en un R-tree consiste en insertar el objeto en las hojas, además, si el nodo excede su capacidad máxima de almacenamiento, éste se divide, enviando un nuevo elemento

al padre, el que a su vez también se puede dividir y propagar un elemento a su correspondiente padre. Este proceso se repetirá recursivamente hasta llegar al nodo raíz, el que también se puede dividir y generar una nueva raíz. El *Alg.* 3.3 describe el procedimiento de inserción.

Algoritmo InsertarEnRtree(T,E)

{L contiene el nodo hoja donde debería ser insertada la entrada E}

```
1:  Sea L = SeleccionarHoja(T, E)
2:  if L tiene espacio suficiente para contener a E then
3:      Almacenar la entrada E en L y terminar
4:  else
5:      LL = DividirNodo(L) {Se obtienen dos nodos hojas L y LL que contienen todas
        las entradas de L y la entrada de E}
6:      AjustarArbol(L, LL)
7:      if la propagación provoca dividir la raíz then
8:          Crear una nueva raíz cuyos hijos son los nodos resultantes
9:      end if
10: end if
```

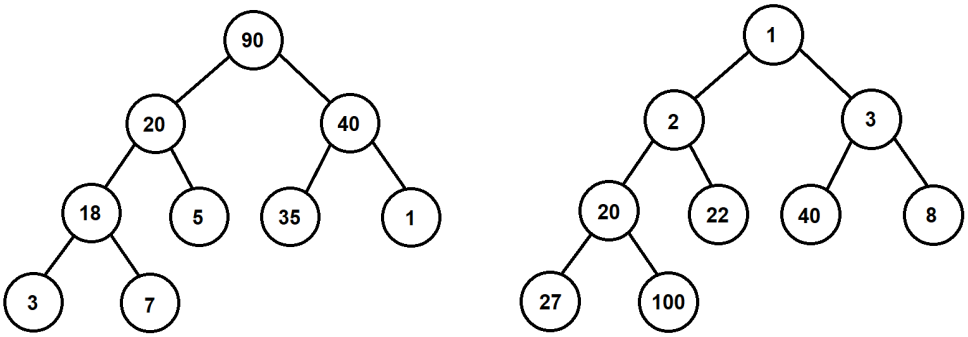
Alg. 3.3: Inserción en R-tree [9].

La función $SeleccionarHoja()$ presente en *Alg.* 3.3 consiste en seleccionar el nodo hoja del R-tree sobre el cual se desea hacer la inserción. Si la entrada en T no es un nodo hoja, busca las entradas T cuyos rectángulos necesitan crecer menos para contener $E.MBR$. Si hay varios, elige uno entre aquellos rectángulos que tengan la menor área. Realiza la función $seleccionarHoja()$ recursivamente hasta encontrar un nodo hoja donde insertar la entrada. La función $AjustarArbol()$ ajusta las áreas de los rectángulos de un R-tree, propagando hacia arriba una entrada cuando sea necesario. Si la entrada L no es raíz, ajusta los nodos dividiendo o agregando según corresponda. Agrega N si el nodo padre P tiene espacio, sino, divide P y lo ajusta nuevamente.

3.3. Heap

Un heap es una estructura de datos que almacena un árbol binario completo, en la que el elemento superior, también conocido como primer elemento, es el elemento más grande o más pequeño de la secuencia. Si el valor de cada nodo es mayor que o igual al valor de sus hijos, entonces la estructura heap es llamada $MaxHeap$. Si el valor cada nodo es menor que o igual al valor de sus hijos, entonces la estructura heap es llamada $MinHeap$. Esto se muestra gráficamente por medio de los árboles binarios de la *Figura* 3.3(a) y *Figura* 3.3(b). Como se puede observar de estas 2 Figuras, todo subárbol del Min Heap y Max Heap, es también un Min Heap y Max Heap, según corresponda.

También existe una tercera categoría, la cual mezcla ambas, Min y Max Heap y que se puede observar en la *Figura* 3.4.



(a) Representación Max Heap (b) Representación Min Heap

Figura 3.3

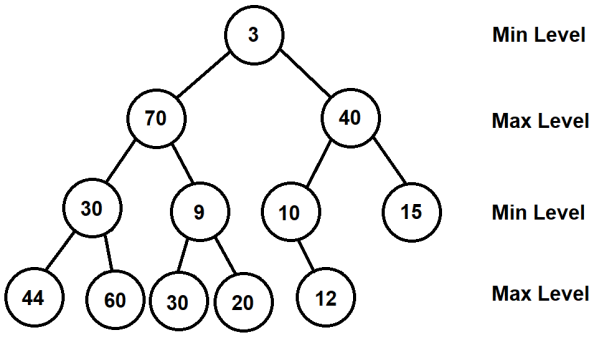


Figura 3.4: Representación Min Heap

En una estructura Min-Max Heap, un nivel satisface la propiedad Min Heap, y el siguiente nivel inferior satisface la propiedad Max Heap, alternadamente. Un Min-Max Heap es útil para colas de prioridad de doble fin.

3.3.1. Operaciones con Heap

La Standard Library(STL) o librería estándar es una colección de clases y funciones de C++, que proporciona cuatro algoritmos que dan soporte a operaciones con heaps. Las operaciones de inserción y eliminación en un heap tienen costo temporal logarítmico en función del tamaño de heap

- *make_heap*(): Es la función encargada de crear el heap. Se debe especificar el rango de elementos que habrá de crearse en un heap.
- *push_heap*(): Se encarga de insertar elementos al heap.
- *pop_heap*(): Elimina un elemento del heap.

- *sort_heap*(): Este ordena el heap.

3.4. Cola de Prioridad

Una cola de prioridad es una estructura de datos, similar a una cola en la que los elementos tienen adicionalmente, una prioridad asignada. Si más de un elemento tiene la misma prioridad, se atenderá de modo convencional según su orden de llegada.

Este tipo especial de colas tienen las mismas operaciones que las colas FIFO, pero con la condición de que los elementos se atienden en orden de prioridad [1].

Un ejemplo de la vida diaria sería la forma de atención que existe en la sala de urgencia de un hospital, ya que los enfermos se van atendiendo en función de la gravedad de su enfermedad, es decir de acuerdo a la gravedad de su dolencia, se le asigna una prioridad.

Entendiendo la prioridad como un valor numérico y asignando a altas prioridades valores pequeños, las colas de prioridad permiten añadir elementos en cualquier orden y recuperarlos de menor a mayor.

Los algoritmos de operaciones con la cola de prioridad, al igual que los algoritmos de las operaciones con Heap, también son proporcionadas por la Standard Library(STL).

A continuación se presentan algunas de estas operaciones:

- *push*() = Inserta un nuevo elemento en cola de prioridad.
- *pop*() = Quita el elemento encima de la cola de prioridad, reduciendo efectivamente su tamaño en uno. El elemento eliminado es el que tiene el valor más alto.
- *empty*() = Prueba si el contenedor está vacío, devuelve true si el la cola de prioridad está vacía, es decir, si su tamaño es cero .
- *size*() = Devuelve la cantidad de elementos en la cola de prioridad.
- *top*() = Devuelve el elemento superior de la cola.
- *swap*() = Intercambia los valores de dos variables o contenedores, entre sí.

  Esta colección de clases y funciones que se deja a nuestra disposición, fue imprescindible para la realización de los Algoritmos de Solución, que se tratarán en el capítulo 4.

Parte III

Algoritmos de Separabilidad Débil con Restricción

Capítulo 4

Algoritmos para el problema de Separabilidad Débil con Restricción (SDR)

4.1. Introducción

En esta sección se describen algoritmos que resuelven el problema SDR considerando que los conjuntos R y B se encuentran almacenados en memoria secundaria en R-trees diferentes, es decir, cada conjunto en su propio R-tree. Los algoritmos propuestos se clasifican como Algoritmo de solución óptima, y Algoritmo de solución Aproximada.

Antes de entrar a describir los algoritmos propiamente tal, en lo que sigue se dan una serie de definiciones utilizadas más adelante en la explicación de los algoritmos propuestos.

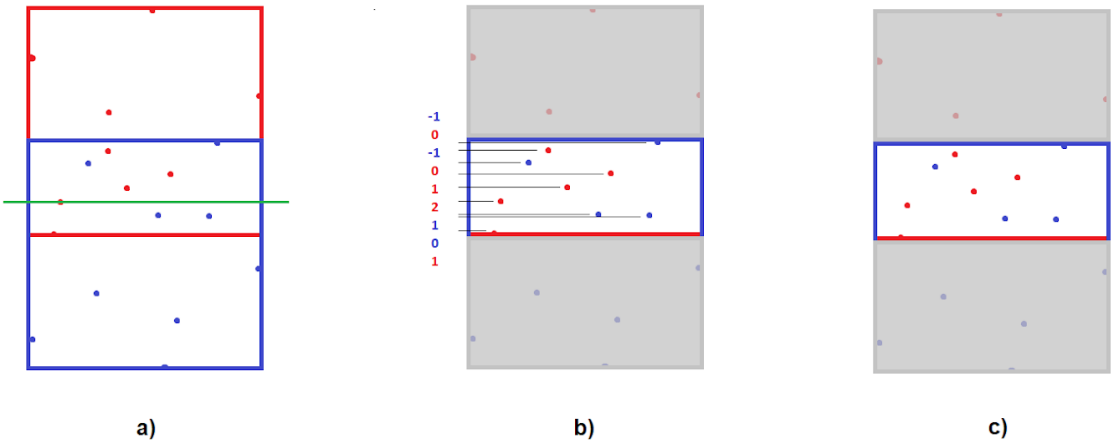


Figura 4.1: Ejemplo de solapamiento de dos R-tree.

Una línea de separación se define como la línea recta que permite separar S en dos conjuntos. En este trabajo una línea de separación (*Figura* 4.1(a)) es la que calculan los algoritmos propuestos y siempre es paralela a uno de los ejes del plano cartesiano.

La zona o área de interés corresponde al área no achurada de la *Figura* 4.2. Esta área resulta diferente de acuerdo a la orientación de la respuesta que se busque, es decir si en un mismo problema se busca una solución horizontal, el área de interés no será la misma de al buscar una respuesta vertical. Dentro de esta área será necesario analizar los nodos y explorar sus hijos. En cambio los nodos que no se intersectan con esta zona pueden ser descartados.

El puntaje se encuentra directamente asociado a la línea de separación, es más, el puntaje resulta ser la variable que se evalúa para determinar dónde se trazará esta línea de separación.

Los números a la izquierda de la *Figura* 4.1 (b) corresponden al resultado de una suma descendente acumulativa de puntos rojos. Cada vez que en el área de interés aparece un punto del color opuesto al que queremos maximizar, a este se le asigna el valor -1, es decir, se realizará una resta del puntaje total.

En la imagen antes mencionada se buscaba maximizar puntos rojos, por lo tanto, al encontrarnos con un punto azul como primer punto, el puntaje comienza con valor -1, y va aumentando o disminuyendo su valor a medida que continúa con la suma descendente hasta terminar de evaluar todo el area de interes. El objetivo del algoritmo es encontrar el mayor puntaje posible para determinar dónde ubicar la línea de separación.

Para el caso del algoritmo horizontal, es irrelevante la cantidad de puntos que existen fuera de la zona de interés, debido a que se conoce que ambas contienen únicamente puntos de un mismo color. Si nos fijamos en la *Figura* 4.1 (c) se puede ver que sobre la intersección de las áreas de ambos conjuntos (Rojo y Azul), solo existen puntos de color rojo, por lo que si trazamos una línea de separación ubicada inmediatamente sobre el MBR del conjunto azul, su puntaje es positivo, dado que no existen puntos azules. En la *Figura* 4.1 (c), se puede apreciar un área no achurada que corresponde a la zona de interés para el caso en el que se desea obtener es una línea de separación horizontal entre los conjuntos de puntos rojos y azules. Los MBRs de los nodos internos de los R-trees que intersectan la zona no achurada se deben explorar o expandir.

En corcondancia con los objetivos de esta tesis, los algoritmos que se explican a continuación resuelven el problema SDR. Esto quiere decir que se puede seleccionar si la separabilizar buscará maximizar puntos rojos o azules, de forma horizontal o vertical, hacia arriba o hacia abajo, o hacia izquierda o derecha, según corresponda.

Dada la simetría entre las diferentes opciones de separabilidad del problema SDR y a la similitud de los algoritmos para resolverlas es que en esta tesis se explican los algoritmos solo para algunos de los escenarios.

Concretamente se presentan los algoritmos (de solución óptima y aproximada) para el caso en que decide por una separación horizontal maximizando los puntos rojos por sobre la línea de separación. Sin embargo, para efectos de experimentación se implementaron todos los algoritmos para cada una los tipos de separabilidad establecidos.

4.2. Algoritmo de Solución Óptima (*ASO*)

En esta sección presentamos nuestro primer algoritmo de solución óptima (*ASO*). El algoritmo ha sido llamado de solución óptima debido a que en la mayoría de los casos recorre una pequeña porción de los R-trees que representan a los conjuntos. El algoritmo resuelve el caso particular de separabilidad horizontal maximizando la cantidad de puntos rojos por sobre la línea de separación.

El algoritmo realiza una exploración desde el nodo raíz hasta los nodos hoja de los R-trees, los que contienen los puntos, descartando aquellos elementos (nodos o puntos) que se encuentren fuera del área de interés. Los elementos que se encuentran fuera del área de interés se descartan porque no influyen en el resultado final y leerlos solo afectaría al algoritmo de manera negativa en términos de tiempo y espacio.

Si un MBR está totalmente fuera de la zona de interés, todos sus hijos también lo están y pueden ser descartados del procesamiento. En cambio si un MBR se encuentra total o parcialmente dentro de la zona de interés se deben explorar los hijos para determinar cuáles deben ser procesados. Una vez que se han obtenido todos los puntos dentro del área de interés, se procede con la búsqueda de la línea de separación.

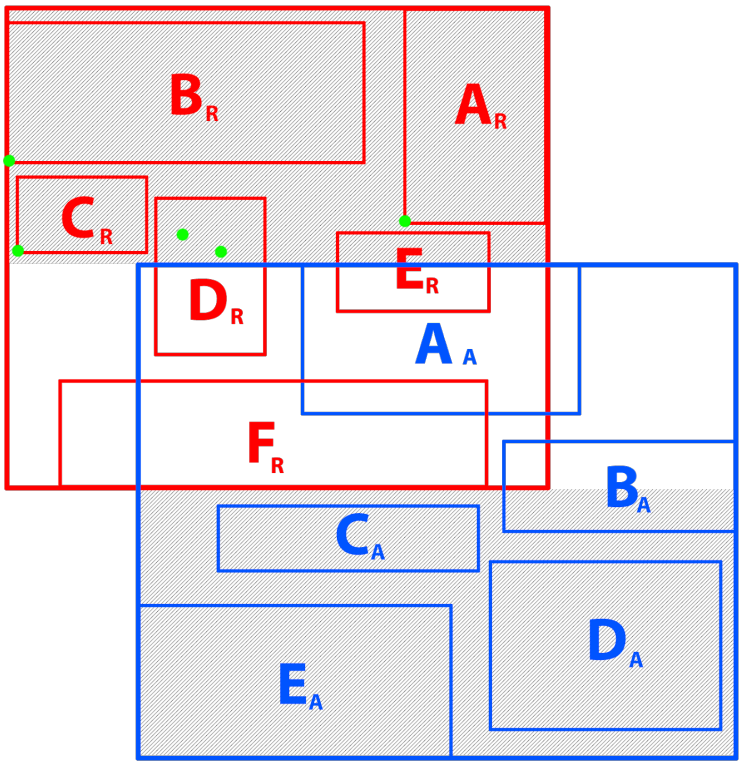


Figura 4.2: Ejemplo de solapamiento de dos R-tree.

Para hallar la línea de separación el algoritmo ordena los puntos que hay dentro del área de intersección. De esta manera puede recorrer los elementos ordenados y calcular en el proceso el puntaje asociado a cada línea de separación posible en la zona de interés para seleccionar aquella con mayor valoración.

En la *Figura* 4.2 se muestra un área achurada que sirve como referencia para identificar aquellos MBRs que son descartados (nodos $A_R, B_R, C_R, C_A, D_A, E_A$). El área no achurada corresponde a la zona de interés, es decir, la zona que determina los nodos que sí serán explorados para obtener la línea de separación en S.

Una vez identificados los nodos que están en el área de interés, éstos se deben explorar. Según lo anterior, durante la exploración del nodo B_A sólo se considera una parte de los puntos que contiene, así también en los nodos D_R y E_R. En cambio todos los puntos de los nodos A_A y F_R son evaluados durante la búsqueda de la línea de separación, por lo tanto ambos nodos deben ser explorados hasta alcanzar las hojas y extraer todos los puntos de ambos subárboles.

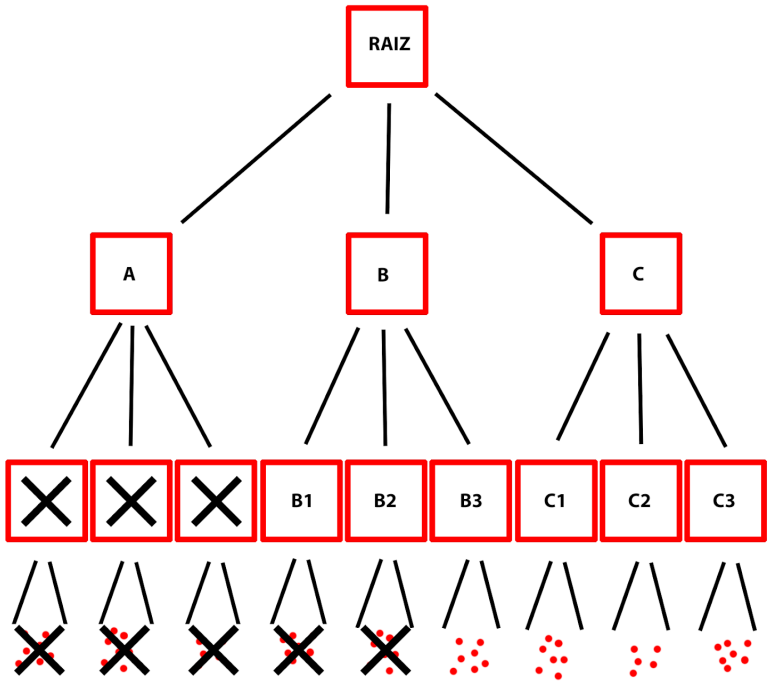


Figura 4.3: Nodos y puntos no leídos por el algoritmo.

Existen casos en los que no se encuentra una solución óptima dentro de la zona de interés, vale decir, que cualquier línea de separación dentro de dicha zona obtendría un puntaje negativo. Es por esto que durante la selección y descarte de elementos se almacena el elemento más próximo a

la zona de interés. Estos elementos podrían ser los límites de los nodos A_R, B_R y C_R. Así también se da que alguno de los puntos descartados desde los nodos D_R y E_R podrían ubicarse entre la intersección y los límites inferiores de los nodos anteriores. Estos elementos representados con color verde en la *Figura* 4.2 tienen como característica común que su puntaje asociado siempre será positivo.

Otra representación de la forma en que funciona el algoritmo, sería como se presenta en la *Figura* 4.3. En ella se aprecia el R-tree rojo, y en la parte superior la raíz; esta contiene los nodos A, B, C y estos a su vez $A_1, A_2, A_3, B_1, B_2, B_3, C_1, C_2, C_3$ y en la parte inferior, los puntos.

En la *Figura* 4.3 se puede apreciar los nodos que no son accesados por el algoritmo debido a que se encuentran fuera de la zona de interés. En este caso el algoritmo accede a los nodos A, B y C pero al clasificar los hijos de A fuera de la zona de interés estos no son leídos. Lo mismo ocurre en los nodos B_1 y B_2, al estar fuera de la zona de interés los puntos son descartados. En lo que sigue se explican en detalle los algoritmos que permite encontrar una línea de separación de tal manera de maximizar los puntos rojos sobre dicha línea. Los algoritmos asumen que los conjuntos involucrados de puntos Rojos y Azules no son vacíos.

4.2.1. Línea de Separación

El *Alg.* 4.1 *lineaSeparacion*, es el algoritmo principal, encargado de entregar el punto donde se debe trazar la línea de separación horizontal que entrega la respuesta. Este recibe como parámetros el MBR de cada conjunto (MBR_R y MBR_A) y dos listas que contienen los nodos internos de la raíz de cada conjunto ($ListaN_R$ y $ListaN_A$). Notar que si los MBRs de los conjuntos no se intersectan, entonces la solución es directa y corresponde a la coordenada Y del punto más bajo del MBR rojo (ver línea 5 del *Alg.* 4.1). Luego en las líneas 12 y 13 se obtienen los puntos rojos y azules que se encuentran en la zona de interés. Los puntos recuperados quedan almacenados en colas de prioridad $ColaP_R$ y $ColaP_A$ implementadas mediante heaps. Notar que para recuperar los puntos es necesario el MBR_A y a la inversa para cargar los puntos azules el MBR_R. Esto tiene el propósito de establecer el área de interés al momento de recuperar los puntos.

4.2.2. Carga Puntos Rojos

El *Alg.* 4.2 *cargaPuntosRojos*, tal como dice su nombre, es el responsable de cargar los puntos rojos que se encuentran dentro del área de interés. Recibe como parámetro $ListaN_R$ que contiene los nodos de la raíz del R-tree, además de MBR_A y MBR_R. Mientras $ListaN_R$ no esté vacía el algoritmo toma un nodo de la lista y analiza si corresponde a un nodo hoja. Si no es un nodo hoja, este analiza si el punto más bajo del nodo se encuentra fuera del área de interés (ver línea 5 de *Alg.* 4.2), y si de acuerdo a la coordenada Y se encuentra más bajo que $primerPR$, este es reemplazado con el fin de ir obteniendo el punto rojo que más se acerque al área de interés, pero que se encuentre por sobre esta. Si es el caso de que el nodo se encuentra dentro del área de interés (ver línea 8 de *Alg.* 4.2), el algoritmo se encarga de insertar en la lista sus hijos.

En el caso de que el nodo sea una hoja (ver línea 13 de *Alg.* 4.2), se consulta si se encuentra dentro del área de interés, si es así, inserta los puntos dentro de $ColaP_R$, sí no, actualiza el valor de $primerPR$ en caso de que se encuentre un punto más cercano al área de interés. El algoritmo *cargaPuntosAzules*, funciona de manera similar al algoritmo *cargaPuntosRojos*. Este se encarga

Algoritmo lineaSeparacion ($MBR_R, MBR_A, ListaN_R, ListaN_A$)

{MBR_R Corresponde al MBR del R-tree que contiene puntos rojos y MBR_A a uno que contiene puntos azules. max almacena el $puntaje$ máximo, restando la cantidad de puntos azules a los puntos rojos. $Pmax$ es una variable que almacena la coordenada Y del plano. R es la cantidad de puntos rojos, A cantidad de puntos azules. $ListaN_R$ Corresponden a los nodos de la Raíz del R-tree que contienen puntos rojos, al igual que $ListaN_A$ los nodos de la Raíz del R-tree de puntos azules, $ColaP_R$ almacena en una cola de prioridad los puntos de color rojo, $ColaP_A$ almacena en una cola de prioridad los puntos de color azul, $punto$ corresponde a la coordenada Y del punto del tope de la cola de prioridad, $primerPR$ es una variable global que es modificada en cargaPuntosRojos(), que se utiliza en caso que dentro del área de interés no exista una mayor cantidad de puntos rojos}

1: $max \leftarrow 0$
2: $Pmax \leftarrow 0$
3: $A \leftarrow 0$
4: $R \leftarrow 0$
5: **if** $CordY(MBR_R) \cap CordY(MBR_A) = \emptyset$ **then**
6: $Pmax \leftarrow CordY(MBR_R.Low)$
11: **else**
12: $ColaP_R \leftarrow$ cargaPuntosRojos($ListaN_R, MBR_A$)//carga en cola de prioridad puntos
13: $ColaP_A \leftarrow$ cargaPuntosAzules($ListaN_A, MBR_R$)//carga en cola de prioridad puntos
14: **while** Not Empty ($ColaP_R$) **do**
15: $punto \leftarrow Eliminar\ CordY(ColaP_R)$//devuelve el primer punto y lo elimina
16: $A \leftarrow A + puntosAzulessobreP(ColaP_A, punto)$ //cuenta puntos sobre punto
17: $R \leftarrow R + puntosRojossobreP(ColaP_R, punto)$ //cuenta puntos sobre punto
18: **if** $max < (R - A)$ **then**
19: $max \leftarrow (R - A)$
20: $Pmax \leftarrow punto$
21: **end if**
22: **end while**
23: **end if**
24: **if** $max < 1$ **then**
25: $Pmax \leftarrow primerPR$
26: **end if**
27: **return** $Pmax$

Alg. 4.1: *lineaSeparacion*.

de obtener la lista de puntos azules, considerando en la lista de nodos los que se encuentren dentro del área de interés.

Algoritmo cargaPuntosRojos($ListaN_R, MBR_A, MBR_R$)

{MBR_A Corresponde al MBR del R-tree que contiene puntos Azules, al igual que MBR_R al de puntos Rojos. $Mbr(Nodo)$ es una función que devuelve el MBR del nodo. $ListaN_R$ es una lista de nodos rojos, $ColaP_R$ es una cola de prioridad de nodos rojos. $primerPR$ es una variable global que almacena el primer punto rojo por sobre de la área de interés}

```
1:  primerPR ← CordY(MBR_R.high)
2:  while Not Empty (ListaN_R) do
3:      Nodo ← Eliminar(ListaN_R)
4:      if Not Leaf(Nodo) then
5:          if (CordY(Mbr(Nodo).Low) > CordY(MBR_A.High)) and
            (primerPR > CordY(Mbr(Nodo).Low) then
6:              primerPR ← CordY(Mbr(Nodo).Low)
7:          else
8:              for Cada hijo del nodo do
9:                  Inserta(ListaN_R, n)
10:             end for
11:         end if
12:     else
13:         for Cada punto p del Nodo do
14:             if CordY(p) ≤ CordY(MBR_A.High) then
15:                 Inserta(ColaP_R, p)
16:             else
17:                 primerPR ← Min(CordY(p), primerPR)
18:             end if
19:         end for
20:     end if
21: end while
22: return ColaP_R
```

Alg. 4.2: *cargaPuntosRojos*.

4.2.3. puntos azules sobre P

El *Alg.* 4.3 *puntosAzulessobreP* se encarga de contar la cantidad de puntos hasta una línea imaginaria paralela al eje Y. Esta obtiene la cantidad de puntos azules que se encuentran por sobre la coordenada Y (CY), eliminándolos de la cola de puntos. Este algoritmo tiene un similar llamado *puntosRojossobreP* que funciona de la misma manera utilizando $ColaP_R$ y que forma parte del *Alg.* 4.1 *lineaSeparacion* (ver línea 17).

En el caso que se necesite una solución vertical, el Algoritmo solución óptima, con pequeños ajustes puede calcular la línea de separación de manera vertical. Solo se deben cambiar las variables para que funcione en paralelo al eje X, sin embargo para propósitos de explicar el algoritmo,

Algoritmo *puntosAzulessobreP* $(ColaP_A, CY)$

{ CY corresponde al punto rojo analizado, $ColaP_A$ es la cola de puntos azules}

```
1:  puntosCant ← 0
2:  while Not Empty (ColaP_A) and CY ≤ CordY(tope(ColaP_A)) do
3:      puntosCant + +
4:      Eliminar(ColaP_A)
5:  end while
6:  return puntosCant
```

Alg. 4.3: $puntosAzulessobreP$.

solo se presenta en esta tesis el algoritmo de separabilidad horizontal.

4.3. Algoritmo Aproximado (AA)

El Algoritmo Aproximado entrega una solución aproximada horizontal. Este algoritmo evita acceder a todos los nodos y puntos, lo que consigue descendiendo por el R-tree hasta llegar al nivel 1, evitando leer el nivel 0 (nodos hoja), lugar donde se encuentran los puntos y que acceder tendría un costo mucho mayor en cuanto a tiempo y memoria.

Para este caso se considera que los nodos hojas se encuentran llenos de puntos en un 75 %, y poseen una similar cantidad de puntos.

Es importante mencionar que además de calcular la solución de manera horizontal, también es posible calcular la solución vertical con algunas modificaciones, principalmente cambiando el eje Y por el eje X. Para los siguientes algoritmos al igual que en el caso del ASO, se asume que los conjuntos involucrados no son vacíos.

4.3.1. Línea de Separación Aproximada

El algoritmo Aproximado se propone como una alternativa al algoritmo de Solución Óptima, con el objetivo principal de reducir aún más los tiempos de ejecución de su predecesor. Este algoritmo resulta interesante cuando se busca obtener una respuesta rápida que no requiera completa exactitud.

Tal como el Algoritmo de Solución Óptima, el algoritmo Aproximado inicia con la función principal que entrega el punto donde trazar la línea horizontal que se busca. Esta función corresponde al *Alg.* 4.4 $lineaSeparacionA$.

Recibe como parámetros el MBR de cada conjunto (MBR_A y MBR_R) y dos listas que contienen los nodos internos de la raíz de cada conjunto ($ListaN_R$ y $ListaN_A$). Este algoritmo comienza por determinar si los R-trees se encuentran solapados, si no es así, entonces la solución es directa y corresponde a la coordenada Y del punto más bajo del MBR rojo (ver línea 7 y 8

Algoritmo lineaSeparacionA ($MBR_R, MBR_A, ListaN_R, ListaN_A$)

{MBR_R Corresponde al MBR del R-tree que contiene puntos rojos y MBR_A a uno que contiene puntos azules. $Nmax$ es una variable que almacena el *puntaje* máximo, restando la cantidad de puntos azules a los puntos rojos. $Pmax$ es otra variable que almacena la coordenada Y del plano. $ListaN_R$ Corresponden a los nodos de la Raíz del R-tree que contienen puntos rojos, al igual que $ListaN_A$ de los puntos azules}

```
1:   Nmax ← 0
2:   Pmax ← 0
3:   ListaN_R ← Nodos Raíz R-tree Rojo
4:   ListaN_A ← Nodos Raíz R-tree Azul
5:   CantN_A ← 0
6:   CantN_R ← 0
7:   if MBR_R ∩ MBR_A = Ø then
8:        Pmax ← CordY(MBR_R.Low)
9:   else
10:       ColaN_R ← cargaNodosRojosA(ListaN_R, ListaN_A)
11:       ColaN_A ← cargaNodosAzulesA(ListaN_R, ListaN_A)
12:       while Not Empty (ColaN_R) do
13:            while Not Empty (ColaN_R) and Not Empty (ColaN_A) and
                    CordY(ColaN_A.Low) ≥ CordY(ColaN_R.Low) do
14:                 CantN_A ← CantN_A + ColaN_A(Children)
15:                 ColaN_A ← Eliminar()
16:            end while
17:            CantN_R ← CantN_R + ColaN_R(Children)
18:            if Not Empty (ColaN_R) and
                  CordY(Nmax) < CordY(CantN_R − CantN_A) then
19:                 Nmax ← (CantN_R − CantN_A)
20:                 Pmax ← CordY(ColaN_R.Low)
21:            end if
22:            ColaN_R ← Eliminar()
23:       end while
24:  end if
25:  return Pmax
```

Alg. 4.4: *lineaSeparacionA*.

de *Alg.* 4.4). Por el contrario, si los $MBRs$ de los conjuntos se intersecan, entonces se cargan los nodos rojos y azules en las colas de prioridad $ColaN_R$ y $ColaN_A$ respectivamente. Mientras la cola $ColaN_R$ no esté vacía, se recorre con el objetivo de encontrar la coordenada donde existe mayor cantidad de nodos de rojos que azules (Ver desde línea 12 en adelante).

4.3.2. Carga Nodos Rojos Aproximado

El *Alg.* 4.5 $cargaNodosRojosA$ se encarga de recorrer el árbol dentro de la zona de intersección. Este recorrido lo realiza por los nodos y sin leer ningún punto, cargando estos nodos en una cola de nodos ordenada. Los nodos que se encuentran sobre el área de interés son eliminados de la lista de nodos (ver línea 2 a línea 7). Posteriormente el algoritmo revisa los nodos en busca de los que se encuentran en el nivel 1 del árbol, para insertarlos en la cola de prioridad (ver línea 9), si no encuentra nodos de nivel 1, inserta a todos los hijos del nodo en la $ListaN_R$

Algoritmo cargaNodosRojosA($ListaN_R, ListaN_A, ColaN_R, MBR_R$**)**

{MBR_R Corresponde al MBR del R-tree que contiene puntos rojos y MBR_A a uno que contiene puntos azules. ColaN_R es una cola de prioridad de nodos rojos. $ListaN_R$ es una lista que contiene nodos rojos}

```
1:  while Not Empty (ListaN_R)do
2:      Nodo ← Eliminar(ListaN_R)
3:      while CordY(Mbr(Nodo).Low) > CordY(MBR_A.High)) do
4:          if Empty (ListaN_R) then
5:              return
6:          end if
7:          Nodo ← Eliminar(ListaN_R)
8:      end while
9:      if Nodo(Level) = 1 then
10:         ColaN_R.Insert()
11:     else
12:         for Cada hijo del Nodo do
13:             Inserta(ListaN_R)
14:         end for
15:     end if
16: end while
17: return ColaN_R
```

Alg. 4.5: $cargaNodosRojosA$.

Capítulo 5

Experimentación

En este capítulo se describe una serie de experimentos que tienen como propósito evaluar el rendimiento de los algoritmos propuestos. Los experimentos utilizan conjuntos de datos sintéticos con diferentes distribuciones.

5.1. Implementación

Los experimentos fueron ejecutados en un Notebook Lenovo ideapad $Y700$ bajo una máquina virtual Oracle VirtualBox con las siguientes características: 4 Gb de memoria RAM, Procesador Intel Core i7 dedicando 4 CPU 2.6 GHz. Los algoritmos fueron implementado en Lenguaje C++, utilizando la implementación del R-tree de la librería LibSpatialIndex (http://libspatialindex.org/).

5.2. Descripción general

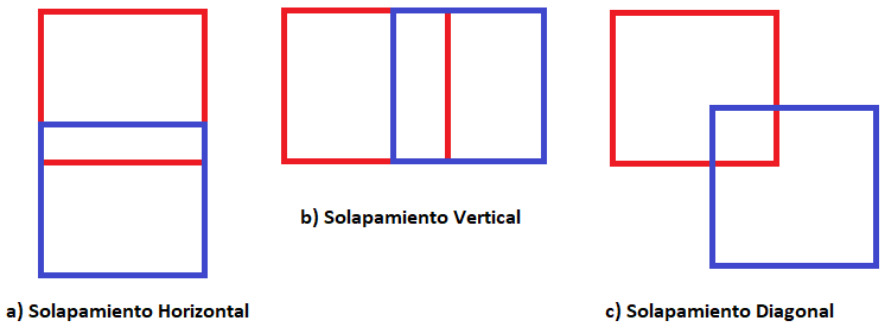


Figura 5.1: Tipos de solapamiento estudiados.

Para la experimentación se utilizaron cuatro conjuntos de puntos de tamaños diferentes, a saber de 1, 2, 5 y 10 millones por cada R-tree. Además, se consideraron conjuntos con solapamiento de 0 %, 25 %, 50 %, 75 % y 100 %. El solapamiento corresponde al área de intersección de las regiones o áreas definidas por los R-trees, llamadas MBR_R y MBR_A para el conjunto rojo y el conjunto azul respectivamente.

Se realizaron tres tipos de experimentos: (i) Porcentaje de filtrado, enfocado en medir el rendimiento de los algoritmos de acuerdo al porcentaje de nodos accesados de los R-trees, (ii) Tiempo de ejecución y (iii) Efecto del solapamiento, centrado en evaluar el efecto que tiene el porcentaje de intersección sobre el rendimiento de los algoritmos.

Debido a que la dirección en la que se realiza el solapamiento es importante, en los experimentos se consideraron tres tipos de solapamiento como se aprecia en la *Figura* 5.1: Solapamiento Horizontal, Vertical y Diagonal.

5.3. Porcentaje de Filtrado

El *Gráfico* 5.1 muestra el porcentaje de nodos de los R-trees, que el algoritmo *ASO* necesita acceder para resolver el problema SDR.

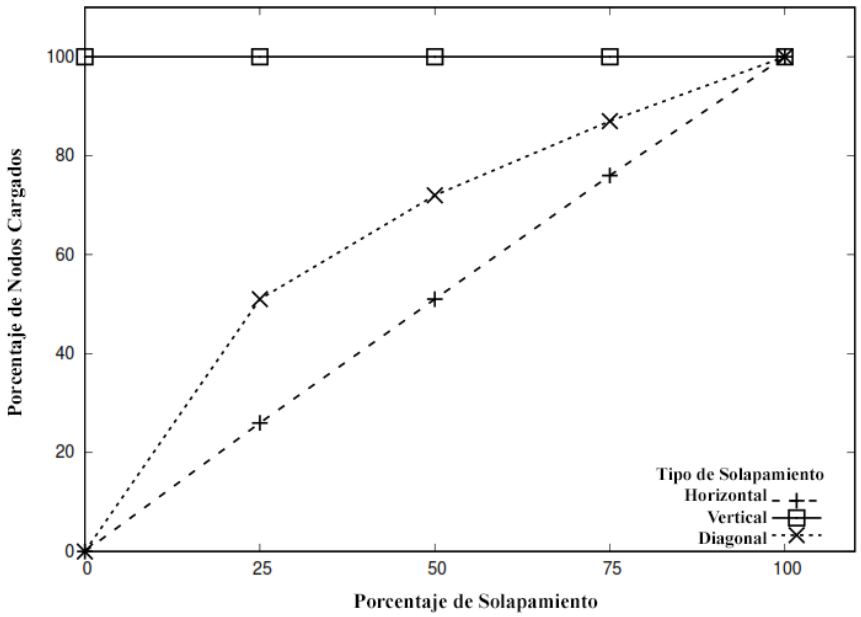


Gráfico 5.1: Nodos leídos por *ASO* para cada solapamiento.

Los datos de prueba utilizados corresponden a conjuntos de 1, 2, 5 y 10 millones de puntos por cada R-tree, realizadas bajo el mismo porcentaje de solapamiento. El análisis se realizó

considerando separación horizontal, maximizando los puntos rojos por sobre la línea de separación. Podemos ver que para conjuntos con porcentaje bajo de solapamiento el algoritmo evita acceder a una cantidad importante de bloques. Por ejemplo, para un 25 % de solapamiento horizontal (un escenario realista) se requiere acceder a menos de un 30 % de los bloques de los R-trees. Para el mismo 25 % de solapamiento diagonal el algoritmo requiere acceder a la mitad de los nodos, mientras que para el mismo porcentaje de solapamiento vertical debe acceder a la totalidad de los nodos.

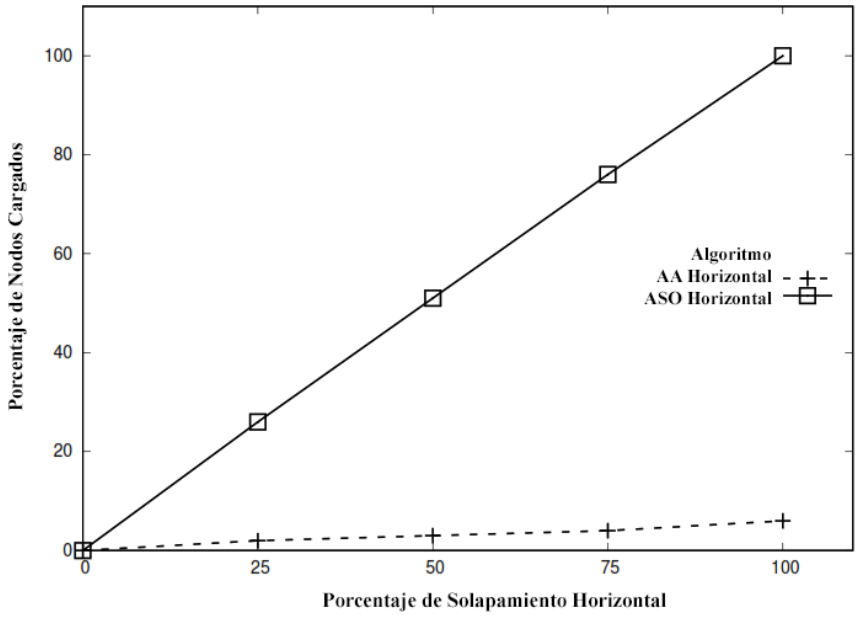


Gráfico 5.2: Nodos leídos vs solapamiento Horizontal.

Cómo es posible apreciar, el algoritmo presenta un mal rendimiento cuando el solapamiento es vertical, lo cual se debe a que la zona de interés corresponde a un porcentaje importante de la unión de las zonas de ambos conjuntos. En el caso del solapamiento vertical, al buscar una separación horizontal, es evidente que el algoritmo se verá obligado a leer todos los nodos. Sin embargo, en los casos de solapamiento diagonal y horizontal solo leerá lo que se encuentra dentro del área de interés. Notar que tanto el tipo como el porcentaje de solapamiento son fáciles de calcular ($O(1)$) y basta con analizar los MBRs de ambos R-trees.

En la *Figura* 5.1 a) se aprecian dos conjuntos de datos, uno con objetos de color rojo y el otro de color azul, separados por una línea de separación horizontal. Este es el tipo de solapamiento analizado en el *Gráfico* 5.2, en el cual se compara el desempeño del algoritmo solución óptima contra el algoritmo aproximado en cuanto a cantidad de nodos que debe leer para entregar una solución a medida que su solapamiento aumenta. Los datos utilizando en este caso corresponden

a conjuntos de prueba que tienen un tamaño de 1, 2, 5 y 10 millones de puntos por cada R-tree (rojo y azul).

Tamaño	Error por % Solapamiento					% Total de Error
	0%	25%	50%	75%	100%	
1	6,27	0,08	0,12	0,11	0,10	1,34
2	12,71	0,04	0,05	0,05	0,08	2,59
5	7,27	0,03	0,02	0,04	0,03	1,48
10	0,01	0,02	0,02	0,02	0,02	0,02
Promedio						1,35

Tabla 5.1: Porcentaje de error *AA*

En la *Tabla* 5.1 se presenta el porcentaje de error del Algoritmo *AA*. Esta tabla evalúa la calidad de las respuestas para conjuntos de 1, 2, 5 y 10 millones de puntos, en escenarios de solapamiento de 0, 25, 50, 75 y 100 %. Como en todos los experimentos que conforman este capítulo, se ejecutaron los algoritmos para la entrega de una respuesta horizontal considerando todos los tipos de solapamiento (horizontal, vertical y diagonal).

Distinto de como se puede creer, para calcular el porcentaje de error de las respuestas del algoritmo *AA*, no basta con comparar la posición de la línea de separación generada por *AA*, contra la respuesta entregada por *ASO*, porque la distancia entre una respuesta y otra no determina la calidad de esta.

Para determinar el porcentaje de error de las respuesta generadas por *AA* se creó un algoritmo evaluador. La forma en la que se obtiene el porcentaje de error de estas respuestas, es evaluando en función de los puntos que se quieren en una dirección, por lo que se utilizó la diferencia entre puntos rojos menos los puntos azules después de trazar la línea de separación, conseguido durante la ejecución del *AA* y el conseguido por el *ASO*.

Tras la ejecución de 120 pruebas se puede establecer que cuando se ejecuta el algoritmo *AA* la respuesta tiene un 1,35 % de error en promedio.

En el siguiente *Gráfico* 5.3 se presentan los resultados obtenidos de comparar el *ASO* contra el *AA*. Los datos utilizados corresponden a conjuntos de prueba de tamaño de 1, 2, 5 y 10 millones de puntos por cada R-tree(rojo y azul). Debido a que la solución requerida es contraria al solapamiento de las pruebas (solapamiento vertical *Figura* 5.1 b) , pero se busca una solución

horizontal), ambos algoritmos se enfrentan al peor caso y se ven obligados a leer todo, es por esto que el algoritmo ASO tiene una lectura del 100 % de los nodos, sin embargo el AA solo tiene un 6 % de nodos leídos debido a que en este no es necesario descender hasta los nodos hoja.

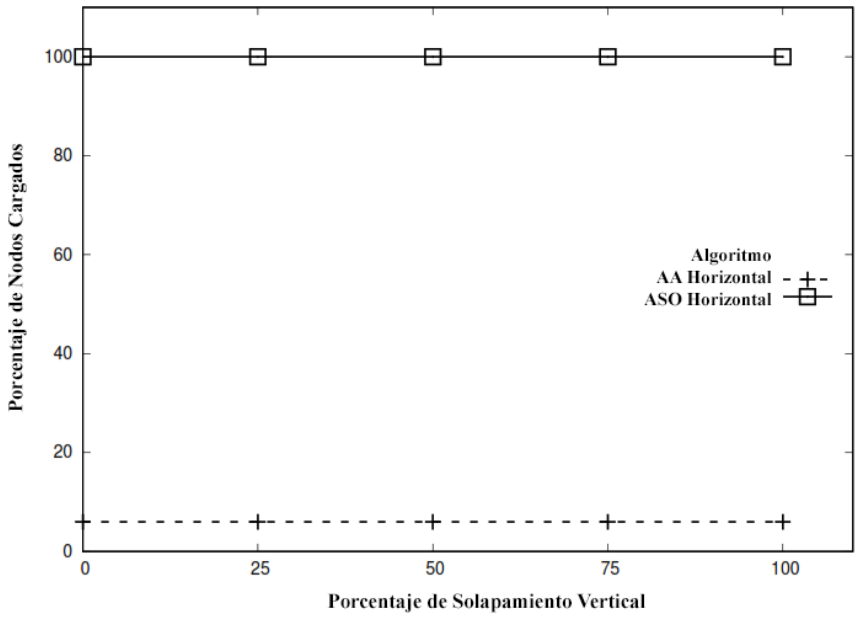


Gráfico 5.3: Nodos leídos vs solapamiento Vertical.

En el *Gráfico* 5.4 se aprecia el porcentaje de nodos leídos comparando el algoritmo *ASO* contra el algoritmo *AA*. En la *Figura* 5.1 c) podemos apreciar dos conjuntos de puntos con un solapamiento de forma diagonal, en este se aprecia una línea de separación horizontal que representa el sentido en el cual se busca la solución. De la misma manera que el algoritmo anterior, los datos de prueba están conformados por conjuntos de 1, 2, 5 y 10 millones de puntos por cada R-tree (rojo y azul). Si observamos el *Gráfico* 5.4, podemos ver que medida que el algoritmo aproximado se ve enfrentado a mayores porcentajes de solapamiento, aumenta el porcentaje de nodos leídos, teniendo su peor caso cuando evalúa MBRs que se encuentran solapados en un 100 %. Aun en su peor caso el algoritmo *AA* solo llega a leer un 6 % del total de nodos. En el caso del algoritmo ASO, el porcentaje de nodos leídos también aumenta de acuerdo al solapamiento, pero de forma más evidente, llegado a leer en su peor caso, el 100 % de los nodos.

En el *Gráfico* 5.5 se presenta el porcentaje de nodos leídos por *AA* para cada tipo de solapamiento, mientras busca una solución de manera Horizontal. Los datos utilizando corresponden a pruebas de 1, 2, 5 y 10 millones de puntos por cada R-tree. Se aprecia que al igual que el Algoritmo de Solución Óptima, este al enfrentar un solapamiento de manera horizontal, se forma una línea creciente a medida que se va aumentando el solapamiento. En el caso del solapamiento

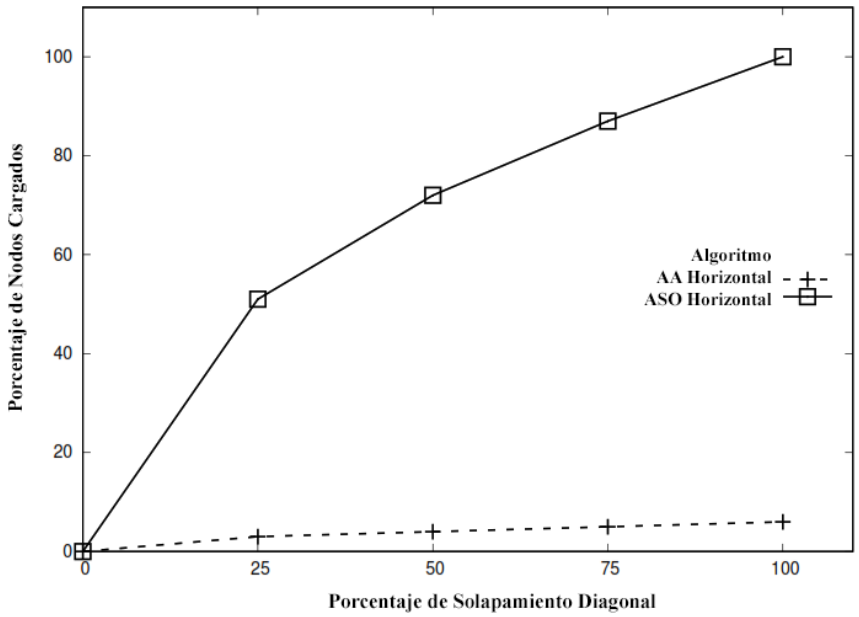


Gráfico 5.4: Nodos leídos vs solapamiento Diagonal.

diagonal, se forma una curva debido a que se ve obligado a leer una mayor cantidad de nodos que el caso anterior.

Cuando el algoritmo se enfrenta a un solapamiento vertical, este debe leer todos los nodos de los niveles superiores, sin embargo a pesar de esto no leerá más de un 6 % de los nodos, para poder entregar una solución con un porcentaje de error de un 1,35 %. Se considera que este porcentaje de error es aceptable en comparación al tiempo, y memoria utilizada por el ASO. En promedio el Algoritmo Aproximado se ve obligado a leer solo un 3,97 % del total de nodos.

5.4. Tiempo de Ejecución

El tiempo de ejecución de los algoritmos tiende a ser una variable determinante al momento de concluir si un algoritmo es o no eficiente y resulta decisivo cuando se barajan opciones para utilizar en algún proyecto. Para tener una referencia sobre cuánto tiempo toma resolver el problema SDR recorriendo todos los nodos y accediendo a todos los puntos, se creó un algoritmo básico que funciona de esta forma y se bautizó como *Algoritmo Ingenuo*. Este algoritmo no forma parte de las soluciones propuestas y se utiliza solo en este capítulo por necesidades de la experimentación. Ya que en este proyecto de tesis se generaron 2 algoritmos, se consideró pertinente contraponer ambos en función del tiempo que toman para entregar una respuesta y compararlos también con el *Algoritmo Ingenuo*. Para abordar esta temática, los resultados de la

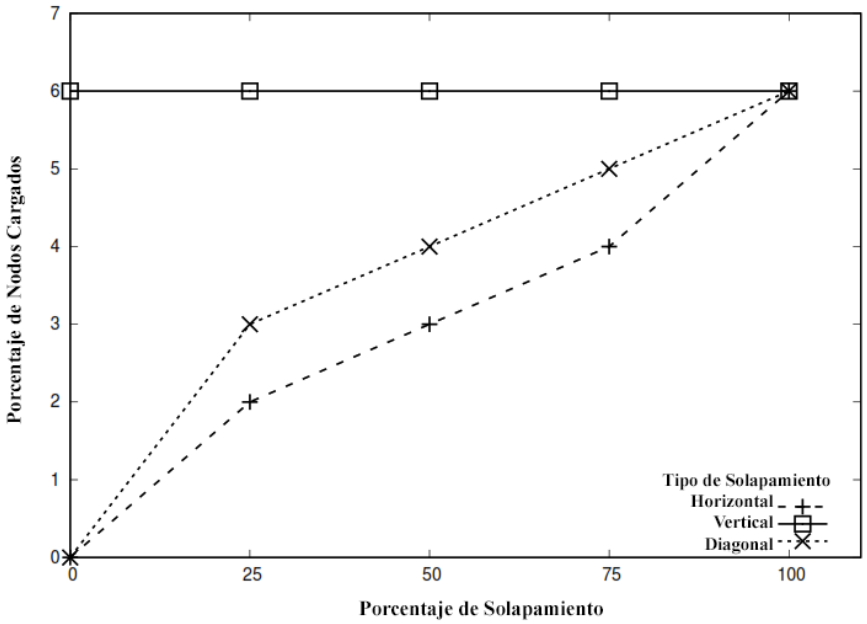


Gráfico 5.5: Nodos leídos por AA para cada solapamiento.

experimentación se trabajaron comparativamente, evaluando el algoritmo ASO contra AA, para buscar una solución horizontal. El conjunto de datos sintéticos utilizado en esta experimentación se encuentra formado por conjuntos de datos de 1000, 20.000, 30.000, 50.000, 60.000, 100.000, 1.000.000 y 10.000.000 de puntos por cada R-tree (rojo y azul) con una distribución uniforme y un porcentaje de solapamiento del 50 % horizontal en todos los casos. Esto nos permitió tener una idea de los tiempos que considera el algoritmo para resolver el problema con distintas cantidades de puntos pero con el mismo porcentaje de solapamiento. En el caso de los dos algoritmos propuestos, el tiempo de ejecución se encuentra directamente relacionado con la cantidad de bloques de los R-trees que es necesario acceder, ya que una vez recuperados los puntos desde los R-trees el procesamiento en memoria se realiza mediante algoritmos de tiempo $O(s \log_2 s)$, con s el tamaño máximo de las listas que contienen los punto en memoria (rojos y azules).

Como se puede apreciar, en la Tabla 5.2 que muestra los resultados de la experimentación del *Algoritmo Ingenuo*, ASO y AA, existe una superioridad en términos de tiempo de ejecución de este último algoritmo, la que se hace evidente desde el conjunto n^o 3 en adelante. Esta superioridad se debe a su característica principal, la de entregar resultados aproximados sin tener la necesidad de leer los puntos de los conjuntos. Si comparamos en la tabla al algoritmo ASO con su par izquierdo, vemos que presenta una mejora temporal considerable. A pesar de que ambos trabajan para entregar respuestas exactas, el algoritmo ASO debe su rapidez a que no lee todos los puntos, sino solo los puntos ubicados en el área de interés.

Conjunto	Número de Puntos en cada conjunto	Tiempo Promedio de Ejecución Algoritmo		
		Ingenuo	ASO	AA
1	1.000	0 Segundos	0 Segundos	0 Segundos
2	20.000	1 Segundos	0 Segundos	0 Segundos
3	30.000	1 Segundos	1 Segundos	0 Segundos
4	50.000	2 Segundos	1 Segundos	0 Segundos
5	60.000	2 Segundos	1 Segundos	0 Segundos
6	100.000	2 Segundos	2 Segundos	0 Segundos
7	1.000.000	29 Segundos	21 Segundos	2 Segundos
8	10.000.000	9 Minutos 12 Segundos	2 Minutos 47 Segundos	17 Segundos

Tabla 5.2: Tiempo de ejecución *Algoritmo Ingenuo* vs ASO.

5.5. Efecto de Solapamiento

Con el propósito de apreciar el efecto del solapamiento en el desempeño de los algoritmos presentados, se evalúan los tiempos promedios de ejecución y la calidad de las respuestas generadas por los algoritmos ASO y *AA*. En la Tabla 5.3 se presenta el Algoritmo ASO y *AA* bajo distintos porcentajes de intersección (1 %, 5 %, 10 %, 50 %), y con la misma cantidad de puntos en cada R-tree.

Solapamiento	Cantidad De Puntos	Tiempo Promedio Algoritmo	
		ASO	AA
1%	10.000.000	10 Segundos	0 Segundos
5%	10.000.000	1 Minuto 19 Segundos	2 Segundos
10%	10.000.000	2 Minutos 26 Segundos	5 Segundos
50%	10.000.000	2 Minutos 51 Segundos	18 Segundos

Tabla 5.3: Tiempo de Ejecución vs Porcentaje de Solapamiento

5.6. Conclusión

En la presente tesis se plantea una solución para resolver un problema de Optimización Geométrica en el contexto de las Bases Datos Espaciales, específicamente, el problema SDR y considerando que los puntos se encuentran almacenados en R-trees, una estructura de memoria secundaria.

El algoritmo propuesto ASO y AA, dan una solución al problema SDR otorgando un resultado exacto y otro aproximado, además mejoran el tiempo del *algoritmo Ingenuo*, debido a que se logra tomar una ventaja de las propiedades que ofrece el R-tree, con el fin de evitar leer la estructura completa. Sin embargo, existen casos en los que ambos R-trees se encuentren completamente solapados, en los que se deben leer todos los nodos. En el peor de los casos el algoritmo ASO y AA se comportan como el *algoritmo Ingenuo*, leyendo todos los nodos.

Luego de analizar los datos, se puede apreciar una tendencia clara a disminuir el tiempo cuando la intersección es menor, sin embargo esto es sólo perceptible cuando el porcentaje de intersección es vertical debido a que este algoritmo analiza de manera horizontal la maximización de puntos rojos hacia arriba.

Parte IV

Conclusiones y Trabajo Futuro

Capítulo 6

Conclusiones y Trabajo Futuro

Las Bases de Datos Espaciales (BDEs) aparecen como una solución al gran volumen de datos espaciales que actualmente se genera y que muchas aplicaciones informáticas necesitan almacenar y procesar. Con el propósito de apoyar estas aplicaciones, las BDEs han debido desarrollar algoritmos para resolver problemas geométricos clásicos para los cuales las soluciones suponían que los datos residen en memoria principal. En este capítulo se detallan los aportes y conclusiones de esta tesis, como también trabajos futuros interesantes de abordar.

6.1. Conclusiones

- Los resultados de este trabajo complementan los reportados en [5], donde se presenta una solución a la separabilidad estricta de dos conjuntos de puntos en R-trees solapados, a diferencia de esta tesis donde se presentan soluciones de separabilidad débil con restricción. Nuestros resultados son útiles en el escenario en que los conjuntos no son separables linealmente. Nuestros algoritmos permiten encontrar una línea paralela a uno de los ejes que separe los conjuntos de puntos minimizando los puntos de ambos conjuntos que se encuentren en una misma zona.

- En esta tesis se evaluó el comportamiento de los algoritmos propuestos, *ASO* y *AA*, en función del tamaño de los conjuntos de puntos y el porcentaje de solapamiento. Para ambos algoritmos, el porcentaje de nodos que son leídos no está determinado por el tamaño de los conjuntos (número de puntos), si no por el tamaño del área de interés. Área que varía de acuerdo al tipo y porcentaje de solapamiento.

- Cuando se observan pruebas de diferentes tamaños e igual porcentaje de solapamiento, realizadas a ASO y *AA*, se puede inferir que el desempeño de ambos algoritmos en términos de tiempo, está directamente relacionado con el tamaño de los conjuntos. Mientras que en casos de conjuntos de tamaño constante y solapamiento variable, lo que determina el tiempo es el tamaño del área de interés. Ahora bien, al comparar los resultados de la experimentación, se puede concluir que aunque los tiempos de ambos algoritmos aumenten por la razones mencionadas, el algoritmo *AA* entrega resultados en menor o igual tiempo

de ejecución que ASO.

- El algoritmo ASO, entrega una solución exacta para todos los casos, sin acceder a todos los nodos de los R-trees. Este algoritmo se enfoca en analizar sólo las áreas superpuestas o solapadas, siempre que se busque una solución en la misma dirección del solapamiento. Por ejemplo cuando el problema presenta un 25 % de solapamiento horizontal y se busca una solución horizontal, el algoritmo visita solo un 25 % de los nodos, lo que significa mejoras en cuanto a tiempo y uso de memoria en comparación con algoritmos que leen todos los puntos de los R-trees para entregar una solución (*Algoritmos Ingenuos*). Es importante destacar que en las situaciones en las que se busca una solución perpendicular a la dirección del solapamiento, el algoritmo se ve obligado a leer todos los puntos, y no presenta ventajas.

- Con respecto al algoritmo AA, este representa una mejora significativa en comparación a los algoritmos que necesitan leer puntos para lograr entregar un resultado. Este algoritmo puede entregar una solución aproximada leyendo solo un 3,97 % del total de nodos (en promedio de los casos de 1, 2, 5 y 10 millones de puntos por R-tree con solapamientos de 0 %, 25 %, 50 %, 75 % y 100 %, en direcciones horizontal, vertical y diagonal), con un margen de sólo 1,35 % de error. Este porcentaje se considera aceptable considerando el ahorro de tiempo y reducción en espacio de memoria que genera.

6.2. Trabajo futuro

En esta sección se han incluido una serie de problemas interesantes de abordar y que pueden complementar los algoritmos propuestos en esta tesis.

- Los algoritmos propuestos en esta tesis dan solución al problema de SDR entregando una respuesta lineal (horizontal y/o vertical) de forma exacta (ASO) o aproximada (AA). Sin embargo, existen escenarios en los que aplicar una respuesta lineal resulta complejo, como por ejemplo en el área de la salud. Para estos casos se propone resolver el problema SDR de forma exacta y aproximada, considerando otras figuras geométricas de separación tales como, caja, cuña, doble cuña, etc.

- Resolver la misma problemática tratada en esta tesis, utilizando estructuras de datos que permitan una indexación más rápida, para reducir aún más los tiempos de ejecución.

- Estudiar otras estructuras de datos multidimensionales cuyas propiedades sean favorables para resolver el problema SDR.

- Extender nuestros algoritmos de tal manera de resolver el problema de separabilidad no estricta mediante una línea recta y considerando los conjuntos de puntos almacenados en una estructura de datos multidimensional en memoria secundaria, .

Bibliografía

[1] Colas de prioridad. http://estructuradedatosyalgoritmos.blogspot.com/2009/04/cola-de-prioridad.html, accessed: 2018-3-12

[2] Nearest neighbor queries. In: Roussopoulos, N., Kelley, S., Vincent, F. (eds.) Series in Computer Science, pp. 25–35. ACM SIGMOD, San Jose, California, USA (1995)

[3] Bayer, R., McCreight, E.M.: Organization and maintenance of large ordered indexes. In: Pioneers and Their Contributions to Software Engineering, pp. 41–59 (1972)

[4] Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The r*-tree: an efficient and robust access method for points and rectangles. ACM SIGMOD Record 19(2), 322–331 (1990)

[5] Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM 18(9), 509–517 (Sep 1975)

[6] Böhm, C., Kriegel, H.P.: Determining the convex hull in large multidimensional databases. In: Lecture Notes in Computer Science, pp. 294–306 (2001)

[7] 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 (2004)

[8] Graham, R.L.: An efficient algorithm for determining the convex hull of a finite planar set. Inf. Process. Lett. 1(4), 132–133 (1972)

[9] Gutiérrez, G.: Métodos de Acceso y Procesamiento de Consultas Espacio-Temporales. Ph.D. thesis, Universidad de Chile (2007)

[10] Gutiérrez, G., Sáez, P.: The k closest pairs in spatial databases. Geoinformatica 17(4), 543–565 (2012)

[11] Guttman, A., Stonebraker, M., California Univ Berkeley Electronics: R-trees: A Dynamic Index Structure for Spatial Searching (1983)

[12] Kalantari, B.: Voronoi diagrams and polynomial Root-Finding. In: 2009 Sixth International Symposium on Voronoi Diagrams (2009)

[13] Nievergelt, J., Hinterberger, H., Sevcik, K.C.: The grid file: An adaptable, symmetric multikey file structure. In: Lecture Notes in Computer Science, pp. 236–251 (1981)

[14] Robinson, J.T.: The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes (1981)

[15] Samet, H.: Applications of Spatial Data Structures: Computer Graphics, Image Processing, and Other Areas. Addison-Wesley / Helix Books (1990)

[16] Sellis, T.K., Roussopoulos, N., Faloutsos, C.: The R+-Tree: A dynamic index for Multi-Dimensional objects. In: Proceedings of the 13th International Conference on Very Large Data Bases. pp. 507–518. Morgan Kaufmann Publishers Inc. (Sep 1987)

[17] Shekhar, S., Chawla, S.: Spatial Databases: A Tour. Pearson (2003)

[18] Torres, C.: Algoritmos de Separabilidad de Objetos en grandes conjuntos de datos espaciales. Master's thesis, Universidad del Bío Bío, Chile (2016)

[19] Torres, C., Pérez-Lantero, P., Gutiérrez, G.: Linear separability in spatial databases. Knowl. Inf. Syst. (2017)

[20] Wiegand, N.: Review of spatial databases with application to GIS by philippe rigaux, michel scholl, and agnes voisard. morgan kaufmann 2002. ACM SIGMOD Record 32(4), 111 (2003)