Universidad del Bío-Bío
Facultad de Ciencias Empresariales
Departamento de Ciencias de la Computación y
Tecnologías de la Información

Búsqueda del par de puntos más cercanos sobre conjuntos no indexados

Por

Miguel Rodrigo Pincheira Caro

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

Profesor guía : **Gilberto Gutiérrez**
Profesor co-guía : **Luis Gajardo**

Chillán - Chile
Enero de 2012

...a mimor Tamy

Agradecimientos

Al terminar esta nueva etapa de aprendizaje, hay muchas personas a las que quisiera agradecer, ya que con su apoyo he logrado terminar este proyecto, que significa una nueva etapa en mi vida.

Empezé a estudiar este Magister con mi polola Tamy, y tal como la investigación y el desarrollo del tema fue creciendo, también lo hizo nuestro amor. Por eso, en primer lugar quiero agradecer a la que hoy es mi mujer, y con quien he empezado nuestra familia, el mayor de los proyecto que pudiese imaginar. Gracias Amor, por apoyarme en todo momento, de muchas más formas de las que yo podía imaginar. Gracias por cuidarme y por, desinteresadamente, dar de nuestro tiempo juntos, para que pudiese terminar esta etapa. Gracias Amor, por la ayuda, paciencia y dedicación con la que me ayudaste, leyendo y corrigendo esta Tesis, que debes sentirla tanto tuya, como mía.

También quiero agradecer a mis padres que, con su ejemplo, me demostraron que trabajando duro, se puede lograr cualquier cosa y, que es nuestra responsabilidad aprovechar al máximo nuestros talentos. Gracias por su paciencia y dedicación desde que nací, y por ser los mejores padres que pude haber tenido. Gracias también por su apoyo desde pequeño, por comprar el primer computador, dejarme usarlo y perdonar todas las veces que perdí sus documentos, por probar cosas nuevas en él. Sé que ustedes están contentos desde incluso antes que obtuviera mi título, pero espero que este nuevo paso, lo podamos celebrar más aún. Y mi familia no estaría completa, sin agradecer a mi tía Beña, por todo el cariño y cuidado que me dío desde pequeño, y me sigue dando hasta hoy.

Mi gratitud también va para mi amigo, colega Luis Gajardo, que además fue el co-guía de esta Tesis. Gracias por ser un excelente profesor, y además un muy buen amigo. Su ayuda me permitió ser Ingeniero y su apoyo, hoy también me ayuda a terminar este trabajo.

Finalmente, quiero agradecer a mi guía en esta tesis, el profesor Gilberto Guitierréz. Gracias por su paciencia y siempre excelente disposición para guiarme, enseñarme y sobre todo, corregirme, para lograr siempre los mejores resultados. Gracias también por presentarme y guiarme en este nuevo mundo de la investigación y, específicamente, de la investigación en las Base de Datos. En pre-grado, el primer ramo de informática que me costó entender, fue también el primer ramo que usted me dictó. Sus temas siempre fueron los más complicados, y también los más interesantes. Por eso, hacer esta Tesis en uno de sus temas, y trabajar en conjunto con usted, fue un desafío doblemente gratificante.

RESUMEN

El problema de encontrar “El par de vecinos más cercanos” entre dos conjuntos, ha sido estudiado desde mucho tiempo. La primeras soluciones eficientes surgieron desde el enfoque de la geometría computacional, pero con la restricción que todos los elementos debían estar en memoria. Sin embargo, desde el ámbito de las Bases de Datos Espaciales (BDE), donde el volumen de datos es mucho mayor, estas soluciones no son aplicables. En este ámbito también existen soluciones eficientes las que consideran que los datos disponen o se encuentran almacenados en algún tipo de índice multidimensional, siendo uno de los más utilizado el R-Tree.

Sin embargo, existen situaciones o escenarios en los cuales ninguno de los conjuntos se encuentra indexado, como por ejemplo, cuando ambos conjuntos corresponden al resultado de consultas sobre las BDE. En este caso, no es posible aplicar directamente los algoritmos existentes, ya que las operaciones para construir los índices multidimensionales son costosas. En esta tesis se presentan dos propuestas para dar solución a la “Búsqueda del par de puntos más cercanos sobre conjuntos no indexados” problema para el que, hasta la fecha y de acuerdo a lo investigado en la literatura, no existen soluciones eficientes disponibles.

La primera propuesta Bucket Closest Pair (BCP), es un algoritmo de dos etapas que primero divide el conjunto en una serie de Buckets, para luego procesarlos buscando el par más cercano. La estructura de datos Bucket, utiliza el concepto de Minimum Bounding Rectangle (MBR) para representar los elementos que contiene. Además, la existencia de una serie de métricas entre los MBR, permite a BCP filtrar los elementos y evitar comparaciones innecesarias.

Experimentos realizados demostraron que BCP encuentra la solución en un 1.8% del tiempo requerido solamente para crear los R-Tree. Por otra parte, al utilizar el algoritmo Block Nested Loop (BNL) como testigo para comparar nuestra propuesta, los resultados mostraron que BCP encuentra la solución usando, en promedio, un 1% del tiempo con solamente el 10% de los accesos a disco requeridos por BNL, cuando los conjuntos no se intersectan. Sin embargo, el rendimiento de nuestra propuesta se ve afectado desfavorablemente por algunos parámetros, como la cantidad de memoria utilizada, y la intersección de las áreas cubiertas por los conjuntos.

Para solucionar este problema, se presentó un segundo algoritmo, llamado Bucket Closest Pair - Vector Approximation (BCP-VA), que también utiliza la estructura Bucket, pero con un enfoque distinto para particionar el conjunto antes de procesarlo.

Al someter a BCP-VA a los mismos experimentos que BCP, se demostró que, si bien son necesarios una cantidad levemente mayor de accesos a disco en la primera etapa, existe una mejora considerable en el tiempo total del algoritmo, sobre todo cuando existe más memoria utilizable. Además, este nuevo esquema de particionamiento, logra una división más uniforme del conjunto, lo que permite mejorar el rendimiento cuando los conjuntos presentan intersección. Por ejemplo, para el caso en que los conjuntos presentan un 50% de intersección, BCP-VA requiere, en promedio, de sólo un 20% del tiempo utilizado por BCP.

De acuerdo a nuestros resultados experimentales, podemos concluir que nuestra propuesta BCP-VA es una solución eficiente para el problema de encontrar el par de vecinos más cercanos, sobre conjuntos no indexados, ya que encuentra la solución en menos tiempo que el requerido para crear estructuras del tipo R-Tree, y además ofrece notables mejoras al compararlo con el algoritmo BNL.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Algoritmos

Capítulo 1

Introducción

1.1 Introducción

Hoy en día el uso de Sistemas de Administración de Base de Datos (SABD) es algo cotidiano. Es un hecho que cualquier empresa, sin importar su tamaño, dispone de grandes almacenes de datos comúnmente numéricos y relacionados con sus finanzas. Y lo más importante, quienes administran estos almacenes son capaces de gestionar y consultar los datos, en busca de información, de una manera fácil y eficiente. Esta simplicidad presente en los SABD, normalmente orientados a los tipos de datos estándar (numéricos, caracteres, string, entre otros) ha dado pie a que muchas otras aplicaciones quieran sacar provecho de estos sistemas. No obstante, estas aplicaciones no trabajan solamente con números o palabras, sino que con nuevos tipos de datos, cada vez más complejos y en un mayor volumen.

Un ejemplo de estas nuevas aplicaciones son los Sistemas de Información Geográfica (SIG), en cuyas Bases de Datos los datos numéricos son ahora acompañados de datos espaciales. Producto de lo anterior, muchos de los fabricantes de Sistemas de Administración de Bases de Datos, como por ejemplo Oracle y Postgres, han incorporado en sus productos, la capacidad para administrar estos nuevos datos. Es decir, han implementado la definición, métodos de acceso y un lenguaje de consulta eficiente para manipular estos datos, con el propósito de facilitar la implementación de aplicaciones espaciales.

Pero no sólo los SIG, se benefician de los datos espaciales. Las aplicaciones de diseño por computador, o el procesamiento de imágenes satelitales, también requieren de los SABD con capacidades espaciales, dando origen a los llamados Sistemas Administradores de Base de Datos Espaciales (SABDE).

Una de las principales características de estos SABDE es su capacidad para responder consultas sobre Bases de Datos Espaciales (BDEs). Es decir, los SABDE deben disponer de tipos de datos, métodos de acceso, y lenguajes de consulta eficientes, sobre los datos que almacenan y gestionan. Si bien existen varios tipos de consultas espaciales, las más conocidas son :

- Consulta Exacta [Sam90], que busca un objeto espacial determinado en un conjunto de objetos espaciales.
- Consulta de Rango [Sam90], que busca todos los objetos espaciales que están dentro de una región definida.

- Consulta del vecino más próximo [RKV95], que busca, entre un conjunto de objetos espaciales, el objeto más próximo a otro previamente dado.

- Join Espacial [BKS93b]. Es un tipo de consulta que tiene como entrada dos o más conjuntos de objetos espaciales, y determina pares (o tuplas de objetos), que verifican determinado predicado espacial, siendo el más común el solape de ambos conjuntos.

Otra consulta espacial muy importante, es la que determina el par (o los *k*-pares) de vecinos más cercanos de dos conjuntos de objetos espaciales, $k-CPQ(P,Q)$, donde P y Q representan los dos conjuntos. Esta consulta, se puede ver como un tipo de Join Espacial, considerando que todos los pares de objetos del conjunto son candidatos al resultado final, pero también es posible verla como una consulta del tipo vecino más próximo, en el sentido que se define una métrica para comparar los candidatos.

Desde un punto de vista práctico, un ejemplo de esta consulta es tratar de responder una pregunta como: *"¿Cuales son los hoteles de 3 estrellas que se encuentran a menos de 100 metros de un restaurant?"*. Tal como un SABD no debe revisar todos los elementos del conjunto, sino que se apoya en índices para responder la consulta, el mismo comportamiento se replica en los SABDE. En [HS98a],[CMTV04],[Cor02], se proponen varios algoritmos que permiten evaluar una consulta $k-CPQ(P,Q)$ en ambientes de BDEs, basándose en la estructura de datos espacial R-Tree [Gut84].

Sin embargo, el descomunal aumento en el volumen de datos que se debe gestionar, a veces presenta casos donde alguno de los conjuntos, o incluso ambos, no se encuentran indexados. En el ejemplo de la consulta anterior, uno de los conjuntos donde se busca el par más cercano corresponde a los resultados de otra consulta (hoteles 3 estrellas), donde evidentemente no existen índices, y el costo de crearlos sería muy alto, considerado que ese conjunto no se volverá a utilizar. Si además aplicáramos otro filtro, por ejemplo, *"y que el restaurant sea del tipo tenedor libre"*, no contamos con índices en ninguno de los conjuntos.

El problema de encontrar el par de vecinos más cercanos, cuando no se dispone de índices ha sido abordado desde el punto de vista de la geometría computacional, obteniendo resultados bastante eficientes. Sin embargo, todos los algoritmos necesitan cargar los datos en la memoria principal, lo que no siempre es posible en ambientes de Bases de Datos Espaciales, donde la cantidad de información es enorme.

Desde el punto de vista de las Bases de Datos Espaciales, la solución al problema se ha basado en la creación de estructuras multidimensionales, que son usadas como índices, y que permiten una búsqueda eficiente sobre estos conjuntos. En [HS98a] y [Cor02] se presentan soluciones utilizando este enfoque.

Pero la creación y mantención de esos índices es costosa, y se justifica solamente cuando tales índices pueden ser reutilizados. Sin embargo, han sido las mismas aplicaciones y el volumen de información que manejan, lo que ha hecho aparecer nuevos escenarios para este problema, como por ejemplo, conjuntos que son el resultado de una consulta. Estos conjuntos no disponen de índices, y su creación no se justifica considerando que no se volverán a utilizar. Actualmente, existen algoritmos que pueden resolver el problema, cuando solamente uno de los conjuntos se encuentra indexado. Estos algoritmos construyen un índice para el conjunto que no lo tiene a partir del que ya existe, y luego, aplican uno de los algoritmos propuestos para el caso en que ambos conjuntos disponen de índices espaciales, como los propuestos en [GG09].

En este trabajo se pretende dar solución al problema de la “Búsqueda del par de puntos más cercanos sobre conjuntos no indexados” para el que, hasta la fecha, y de acuerdo a lo investigado en la literatura, aún no hay soluciones disponibles. Para esto se presentan y analizan dos soluciones que, mediante una estructura de datos especialmente diseñada en este trabajo, responden la consulta de una manera eficiente, sin la necesidad de crear índices espaciales.

1.2 Objetivos de la Tesis y Alcances de la Investigación

El objetivo de la tesis es diseñar y evaluar algoritmos y estructuras de datos para resolver el problema del par de vecinos más cercanos sobre Bases de Datos Espaciales, sin utilizar índices. El alcance de la investigación limita el trabajo a conjuntos no indexados compuesto por puntos en dos dimensiones. Sin embargo, la extensión hacia otras dimensiones no debería presentar complicaciones, así como tampoco lo sería adaptar los algoritmos para encontrar los k-vecinos más cercanos, con $k > 1$, dentro de los conjuntos antes mencionados.

1.3 Contribuciones de la Tesis

Las principales contribuciones de esta tesis son las siguientes:

1. Se creó la estructura de datos Bucket, basada en el concepto de *MBR* (Minimun Bounding Rectangle) usado en la mayoría de los métodos de acceso espacial, como por ejemplo el R-Tree. Al igual que en dichos métodos, esta estructura es utilizada para ordenar los datos y, posteriormente, buscar candidatos que cumplan la consulta dada sin tener que analizarlos uno a uno. Además, gracias a la utilización de un buffer, esta estructura permite procesar grandes volúmenes de datos, con una menor cantidad de accesos a disco y un menor tiempo que los métodos convencionales.

2. Se presentó el algoritmo “Bucket Closest Pair” (BCP) que, mediante el uso de la estructura Bucket y un esquema de trabajo de dos etapas (particionamiento y procesamiento), logra encontrar el par de vecinos más cercanos, sin necesidad de crear índices espaciales. Dicho algoritmo se comparó con las soluciones basadas en métodos de acceso espacial, superándolos ampliamente en rendimiento. Esta propuesta también se evaluó en una serie de escenarios, donde la consulta fue ejecutada sobre conjuntos de distinto tamaño, distribución y porcentajes de intersección, y utilizando distintas cantidades de memoria. En todas las pruebas, BCP logró excelentes resultados, superando al algoritmo “Block Nested Loop” (BNL), usado como testigo para comparar nuestra propuesta.

3. Se presentó el algoritmo “Bucket Closest Pair -Vector Approximation” (BCP-VA), una mejora de BCP, que utiliza la misma estructura de datos Bucket y la misma etapa de procesamiento, pero con un algoritmo distinto en la etapa de particionamiento. Este nuevo enfoque, basado en particionar el espacio que utilizan los datos y no los datos como tal, permitó mejorar aquellos casos más desfavorables para BCP. El nuevo algoritmo se sometió a los mismos experimentos de la propuesta anterior, logrando mejorar el rendimiento, sobre todo en aquellos escenarios desfavorables para BCP, como lo fueron los porcentaje de intersección distintos a 0, y una cantidad de memoria más grande para trabajar.

En base a estas contribuciones, se han generado las siguientes presentaciones y publicaciones:

- Miguel Pincheira, Gilberto Gutiérrez. "Closest pair query on Spatial data sets without index". Publicado en "XXIX International Conference of the Chilean Computer Science Society (SCCC), 2010", IEEE, páginas 178-182, año 2010.
- Miguel Pincheira, Gilberto Gutiérrez. "Closest pair query on Spatial data sets without index". XXIX Conferencia Internacional de la Sociedad Chilena de Ciencia de la Computación. Antofagasta(Chile). 18 de Noviembre de 2010.
- Miguel Pincheira."Búsqueda del par de vecinos más cercano sobre conjuntos no indexados". 1er Encuentro de Tesistas. Antofagasta(Chile), 19 de Noviembre de 2010.
- Miguel Pincheira."Bucket Closest Pair: Un algoritmo para la búsqueda del par de vecinos más cercano sobre conjuntos no indexados". Día de la Facultad de Ingeniería y Negocios, Universidad Adventista de Chile, Chillán (Chile). 20 Octubre de 2010.
- Miguel Pincheira."Búsqueda del par de vecinos más cercano sobre conjuntos no indexados".II Seminario Herramientas Informáticas de la Actualidad. Chillán (Chile), 24 de Noviembre de 2010.

1.4 Descripción del documento

Una vez realizada la introducción al problema, y explicado los objetivos de la tesis, en la siguiente sección detallamos el contenido del resto del documento. En el Capítulo 2 se presentan las Bases de Datos Espaciales, entregando una definición de las mismas. También se especifican los tipos de datos y operadores, que servirán a su vez, para describir las consultas espaciales y definir formalmente, aquellas más usadas. Luego se explica como un SABDE da respuesta a estas consultas, utilizando métodos de acceso espaciales. Finalmente el capítulo termina con una descripción de la estructura de datos R-Tree, como uno de los métodos de acceso espacial más utilizado por las BDEs. Con este capítulo esperamos dar un marco de referencia al ambiente de las Bases de Datos Espaciales, que es el escenario donde se desarrolla el problema que se plantea en esta tesis.

El Capítulo 3 presenta la consulta base para esta tesis, "El par de vecinos más cercanos", primero definiéndola de una manera formal, para luego presentar un estado del arte sobre los trabajos relacionados con la misma. Los dos enfoques más importantes desde los cuales ha sido abordada esta consulta son la geometría computacional y los sistemas de Bases de Datos Espaciales. Veremos que, si bien desde el enfoque de la geometría computacional existen soluciones eficientes para resolver el problema, estas soluciones consideran trabajar con todos los datos en la memoria principal. Por otra parte, desde el punto de vista de las Bases de Datos Espaciales, el volumen de información hace imposible contener todos los datos en memoria principal, por lo que se hace necesario trabajar utilizando el almacenamiento secundario. En este capítulo también se presentan una serie de propuestas que, basadas en métodos de accesos espaciales, solucionan el problema de una manera eficiente para el enfoque de las Bases de Datos Espaciales. Sin embargo, veremos que estas propuestas se basan en la existencia de índices en los conjuntos, lo que nos lleva finalmente a plantear el caso que se pretende resolver en esta tesis, que

es el solucionar el problema del par de vecinos más cercanos, sobre Bases de Datos Espaciales, cuando ninguno de los conjuntos se encuentra indexado.

En el Capítulo 4 se presenta la primera solución al problema, el algoritmo BCP (Bucket Closest Pair), que mediante el uso de la estructura de datos Bucket, diseñada para el algoritmo, y mediante un sencillo particionamiento de los conjuntos, logra dar respuesta a la consulta antes mencionada, sin crear índices. Una serie de experimentos fueron diseñados y ejecutados para evaluar el rendimiento del algoritmo comparándolo, por ejemplo, con el enfoque tradicional basado en los índices espaciales, como los presentados en el Capítulo 3. Además, se probó el algoritmo en una serie de escenarios distintos, para evaluar su rendimiento al ejecutar la consulta. Cierran el capítulo los resultados de estos experimentos, junto con las conclusiones sobre este algoritmo propuesto, comentando cuales fueron los casos favorables y los desfavorables.

Luego, el Capítulo 5 presenta la segunda solución al problema, el algoritmo BCP-VA (Bucket Closest Pair - Vector Approximation), que es una mejora a BCP. Esta nueva propuesta utilizando la misma estructura de datos, pero con un algoritmo más complejo al momento de particionar los conjuntos. Este nuevo esquema de particionamiento tiene como objetivo mejorar aquellos casos que fueron desfavorables para la primera solución. Los mismos experimentos hechos sobre BCP, fueron realizados con esta nueva propuesta y demostraron que, efectivamente BCP-VA mejora el rendimiento de BCP, especialmente en los escenarios desfavorables.

El documento finaliza en el Capítulo 6, donde se resumen las conclusiones más importantes obtenidas de este trabajo, en base a los dos algoritmos propuestos. También se propone una serie de trabajos futuros que sería posible desarrollar, en torno a este tema de investigación.

Capítulo 2

Bases de Datos Espaciales

2.1 Introducción

En la era de las tecnologías de la información en la que nos encontramos, son cada vez más los datos con los que trabajamos día a día y, sin darnos cuenta, es cada vez más variada la naturaleza de los mismos. En este ámbito, los Sistemas Administradores de Base de Datos (SABD) son el silencioso testigo de la dependencia que hemos creado con nuestra información. Si bien su origen fue ligado a los negocios y la contabilidad, hoy en día son mucho más que un gran y eficiente almacén de datos numéricos. La capacidad de los SABD de manejar los datos, y responder consultas sobre los mismos, de una manera consistente y eficiente, ha superado su diseño original, que era generar un simple informe de estados financieros, dando paso a fuentes valiosas de información para la toma de decisiones.

Pero ya no es sólo números lo que necesitamos guardar. La inclusión de las tecnologías de la información en otras actividades como el diseño, la fotografía, la geografía o la cartografía han planteado nuevos desafíos a los SABD, presentando nuevos tipos de datos, y en un volumen cada vez mayor. Sin ir más lejos, ya desde el año 2003 el Sistema de Observación de la Tierra (EOS) de la Nasa generaba diariamente un terabyte de datos para ser procesados [SC03]. Si bien las imágenes satelitales son un ejemplo de datos espaciales, no son el único. Por ejemplo, los pixeles en una fotografía, las carreteras en un país, las casas en el mapa de una ciudad, o bien los distintos componentes electrónicos ubicados en la superficie de un microchip diseñado a gran escala, es nueva información que queremos almacenar, gestionar y, sobretodo, consultar de una manera rápida y eficiente.

Los SABD tradicionales se han debido extender con el propósito de representar y consultar objetos espaciales, lo que ha llevado al diseño de nuevas estructuras de datos para almacenar dichos objetos y, por lo mismo, ha impulsado también la creación de métodos de acceso espaciales, junto con lenguajes de consulta que respondan de una manera eficiente sobre esta nueva información, dando origen a los Sistemas de Administración de Base de Datos Espaciales (SABDE). Una buena definición para los SABDE es la presentada en [Gut84]: “Un Sistema Administrador de Bases de Datos Espaciales es un Sistema Administrador de Bases de Datos, capaz de manejar datos de naturaleza espacial, y ofrecer un lenguaje de consulta para manipular dichos datos”.

En el siguiente capítulo definiremos las Bases de Datos Espaciales mediante sus tipos de datos y operadores, que además servirán para definir las consultas espaciales más comunes. Luego analizaremos cómo se procesan estas consultas, describiendo los métodos de acceso espaciales y

detallando un poco más el R-Tree, uno de los métodos más utilizados. Estos conceptos nos darán un marco de referencia del ambiente de las Bases de Datos Espaciales, que es donde se desarrolla esta tesis.

2.2 Tipos de datos y operadores espaciales

Debido a la amplia gama de los datos espaciales es muy difícil definir un tipo estándar de datos y un conjunto de operadores espaciales lo suficientemente general, para que abarquen todas las aplicaciones. Sin embargo, según [GG98] las siguientes propiedades son comúnes para todos los tipos de datos, y nos dan un marco de referencia para su definición, independiente de la naturaleza de los mismos:

- Los datos espaciales tienen una estructura compleja. Un dato espacial puede estar compuesto de un solo punto (un píxel en una imagen), o de varios cientos de polígonos arbitrariamente distribuidos en el espacio (una imagen satelital) y, por lo tanto, no pueden almacenarse en una sola tupla.
- Las Bases de Datos Espaciales tienden a ser muy grandes: Por ejemplo, un mapa geográfico puede ocupar varios gigabytes de almacenamiento.
- No existe un álgebra estandar 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 quiere componer operadores tal como los operadores relacionales.
- El costo computacional de implementar los operadores espaciales es mucho mayor que los operadores relacionales.

Normalmente, un objeto en una Base de Datos Espacial se define mediante varios atributos no espaciales (el color de un píxel, el nombre de una calle, etc.) y un atributo espacial de algún tipo (la posición del píxel, la ubicación de la calle). Para modelar estos atributos se utilizan tres abstracciones fundamentales que son: punto, línea y región (o polígono) [GS05], que se pueden apreciar en la Figura 2.1 Un punto representa un objeto para el cual solamente nos interesa su posición en el espacio, como un píxel en una fotografía, una casa en un mapa, etc. Una línea es una abstracción que podemos ver como una serie de puntos contiguos como, por ejemplo, un río en una fotografía satelital, una calle en un mapa, etc. Finalmente, una región permite modelar objetos con cobertura espacial, como un país, una región, etc.

2.3 Consultas espaciales

Una de las características que convierten a un SABDE en una potente herramienta es la capacidad de responder consultas relacionadas con los atributos espaciales de los elementos [Cor02]. Como

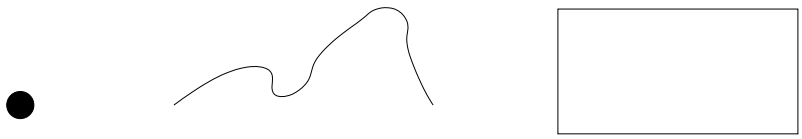


Figura 2.1: Objetos Espaciales: Punto, línea y superficie según [GS05]

definición podemos decir que, dado un conjunto de objetos espaciales, el resultado de una consulta espacial es el conjunto de todos aquellos elementos espaciales o, cuyo atributo espacial $o.G$ verifica un determinado predicado espacial. A continuación se definen las consultas espaciales según [Gut07] y que, de acuerdo a [GG98], son las más comúnes.

2.3.1 Consulta Exacta: Exact Match Query (EMQ)

Dado un objeto espacial o', encontrar todo objeto o cuyo atributo espacial es igual al de o':

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

2.3.2 Consulta para la localización de un punto: Point Query (PQ)

Dado un punto $p \in \mathbb{R}$, encontrar todos los objetos o que cubren a p:

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

2.3.3 Consulta de Rango o Ventana: Range/Windows Query (WQ)

Dado un rango $q \subseteq \mathbb{R}$, encontrar todos los objetos o que tienen al menos un punto en común con q:

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

2.3.4 Consulta de Solape: Intersection Query (IQ)

Dado un objeto o' con atributo espacial $o'.G \subseteq \mathbb{R}$, encontrar todos los objetos o que tienen al menos un punto en común con o':

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

2.3.5 Consulta de Encubrimiento: Enclosure Query (EQ)

Dado un objeto o' con atributo espacial $o'.G \subseteq \mathbb{R}$, encontrar todos los objetos o que lo encierran o encubren completamente:

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

2.3.6 Consulta de Inclusión: Contaiment Query (CQ)

Dado un objeto o' con atributo espacial $o'.G \subseteq \mathbb{R}$, encontrar todos los objetos o encerrados por o':

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

2.3.7 Consulta de Adyacencia: Adjancency Query (AQ)

Dado un objeto o' con atributo espacial $o'.G \subseteq \mathbb{R}$, encontrar todos los objetos o adyacentes a o':

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

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

2.3.8 Consulta del vecino más próximo: (Nearest-Neightbor Query) : NNQ

Dado un objeto o' con atributo espacial $o'.G \subseteq \mathbb{R}$, encontrar todos los objetos o que tienen la mínima distancia o':

$NNQ(o') = \{o | \forall o'' : dist(o'.G, o.G) \leq dist(o'.G, o''.G)\}$ En este caso se define $dist()$ entre atributos espaciales, como la distancia entre sus puntos más cercanos.

2.3.9 Join Espacial: Spatial Join (SP)

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

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

Existe una amplia variedad de predicados θ, no obstante, de acuerdo a [GG98] el predicado de intersección tiene un rol muy importante, pues con él es posible obtener casi todos los restantes predicados que puedan existir.

2.3.10 Consulta del par de vecinos más cercanos: Closest Pair Query: (CQP)

Dadas dos colecciones de objetos espaciales R y S, encontrar el par de elementos $(v, v') \in RxS$ tal que su distancia sea la menor entre todos los pares de objetos $(o, o') \in RxS$. Al ser ésta la consulta central en la que se enfoca nuestra investigación, una definición más completa se encuentra en la la Sección 4.

2.4 Procesamiento de consultas basado en métodos de acceso espacial

Para responder a estos tipos de consulta, se supone la existencia de un índice espacial (concepto que se definirá más adelante) y se sigue el procedimiento que se ilustra en la Figura 2.2.

En resumen, podemos decir que el procedimiento para responder a la consulta se hace en dos etapas: la etapa de filtrado y la etapa de refinamiento [BKSS94]. En la etapa de filtrado, se utiliza el índice espacial para seleccionar un conjunto de objetos candidatos que pueden cumplir con el predicado de la consulta (utilizando propiedades del índice, sin tener que procesar el predicado espacial), para luego en la etapa de refinamiento, procesar los objetos obtenidos de la primera etapa, comparando el predicado de la consulta con los objetos candidatos. Si bien se puede pensar que la primera etapa puede consumir mayor tiempo, al tener que trabajar con todo el conjunto, la utilización de los índices ayuda a disminuir el costo de esta etapa y debido a que los objetos espaciales tienen una estructura compleja, la etapa de refinamiento puede llegar

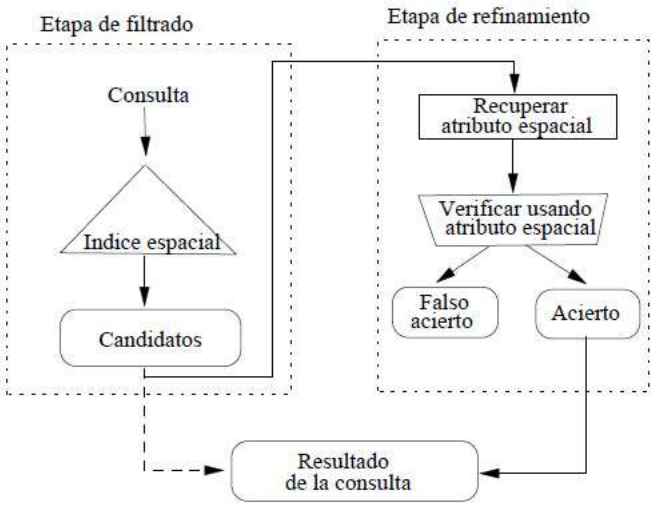


Figura 2.2: Secuencia de una consulta espacial

a incurrir en un costo muy alto de CPU (Central Processing Unit), siendo una componente no despreciable del costo total del consulta. Es por esta razón que la existencia de los métodos de acceso múltidimensionales son una parte importante dentro de la eficiencia que pueda tener un SABDE para responder las consultas espaciales.

2.5 Métodos de acceso espaciales/multidimensionales

Uno de los principales objetivos de un SABDE es el de proporcionar métodos de acceso y algoritmos eficientes para el procesamiento de consultas y manipulación de datos espaciales [Güt94], es decir, la recuperación y actualización de los datos no solamente por medio de sus atributos descriptivos, sino utilizando sus características espaciales. De esta forma, el procesamiento de las consultas espaciales antes mencionadas requieren el acceso rápido a los objetos, sin la necesidad de recorrer todos los objetos de la Base de Datos. Esto obliga a contar con métodos de acceso espaciales (multidimensionales), que permitan mantener un índice organizado por el atributo espacial de los objetos [Güt94].

Los métodos clásicos de acceso unidimensionales, como el B-Tree [BM72], son insuficientes para dar un respuesta eficiente, lo que ha hecho surgir la necesidad de nuevos métodos de acceso que sean capaces de responder consultas por múltiples atributos. Las primeras estructuras multidimensionales que surgieron estaban pensadas para almacenarse en memoria principal. Una de las más representativas es el k-d-Tree [Ben75], que es una estructura utilizada para gestionar puntos de n dimensiones. El k-d-Tree es un árbol binario que representa una subsivisión recursiva del espacio, en subespacios cada vez más pequeños, utilizando hiperplanos de $(n-1)$ dimesiones de forma muy similar a como los árboles binarios dividen el espacio unidimensional. En la Figura

2.3 se muestra un conjunto de puntos en el espacio de dos dimensiones, y su representación en un k-d-Tree. Cada línea corresponde a un nodo en el árbol, con capacidad igual a 1. Cada valor del nodo del k-d-Tree divide el espacio en dos, según una determinada dimensión. La división se realiza fijando una dimensión en el nodo raíz (Y), siguiendo con otra dimensión (X) en los nodos del siguiente nivel, y así sucesivamente, intercambiando las dimensiones de manera rotativa.

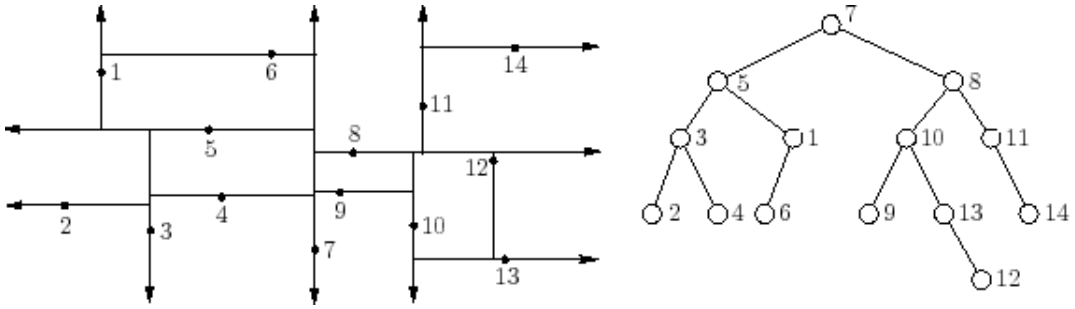


Figura 2.3: Ejemplo de un k-d-Tree

Sin embargo, el hecho que estas estructuras deban almacenarse totalmente en la memoria principal, las hacen inoperantes cuando el volumen de datos es muy grande, como es el caso de las Bases de Datos Espaciales actuales. Esta restricción hace obligatorio el uso de almacenamiento secundario para poder trabajar con todos los datos.

En este nuevo escenario, se han propuesto varias estructuras tipo índice multidimensional, que además de trabajar en memoria principal, utilizan almacenamiento secundario para guardar información. Algunas han sido diseñadas inicialmente para indexar conjuntos de puntos, y luego han sido extendidas para otros tipos de objetos espaciales. Ejemplos de estas estructuras son Grid-File [NHS84], K-D-B-Tree [Rob81]. Por otro lado, para representar los objetos más complejos, una de las pioneras es el R-Tree [Gut84], que es también una de las más populares, y que ha dado pie a una serie de variantes como el R^+-Tree [SRF87] y R^*-Tree [BKSS90]. La característica común que presentan estos métodos de acceso, es que agrupan los objetos en regiones minimales, es decir, regiones ajustadas a los objetos que contienen [Cor02] y luego trabajan en base a esta abstracción de los objetos. Una revisión más detallada de los métodos de acceso se pueden encontrar en [GG98] y [BBK01]. En este documento solamente se profundizará en el R-Tree, por ser una de las estructuras más usadas, y porque además, ha sido tomada como referencia para comparar y evaluar los algoritmos propuestos en esta tesis.

2.6 R-Tree como método de acceso multidimensional

El R-Tree [Gut84] es uno de los métodos de acceso multidimensional, más utilizados y estudiados, que ha sido adoptado por SABD comerciales como Oracle y Postgres, y que corresponde a una extensión natural del índice unidimensional B-Tree [BM72]. El R-Tree es una estructura arbórea, balanceada en altura, que representa una jerarquía de regiones rectangulares minimales, llamados *MBR* (Minimun Bounding Rectangle) y que se definen más adelante en la Sección 2.6.1. A diferencia de los B-Tree, que asocian rango de valores en sus nodos, los R-Trees asocian *MBR*.

En un R-Tree, como el que se aprecia en la Figura 2.5, cada nodo corresponde a una página o bloque de disco. Los nodos hojas de un R-Tree contienen entradas de la forma *(MBR,oid)*, donde *oid* es el identificador del objeto espacial en la Base de Datos, *MBR* (Minimun Bounding Rectangle) es un réctángulo multidimensional que corresponde al mínimo rectángulo que encierra al objeto espacial. Los nodos internos (nodos no-hojas), contienen entradas de la forma *(MBR, ref)*, donde *ref* es la dirección del correspondiente nodo hijo en R-Tree y *MBR* es el rectángulo mínimo que contiene a todos los rectángulos definidos en las entradas del nodo hijo [Gut07]. La representación de los *MBR* definidos en un R-Tree, se pueden apreciar en la Figura 2.4.

Sea M el número máximo de entradas que se puede almacenar en un nodo, y sea $m \leq \frac{M}{2}$ un parámetro especificando el número mínimo de entradas en un nodo, entonces, un R-Tree satisface las siguiente propiedades [Gut84].

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

Además, según [Gut84], en un R-Tree también se cumple que :

- La altura de un R-Tree que almacena N claves es, a lo más $(\log_m N) - 1$.
- El número máximo de nodos es $\frac{N}{m} + \frac{N}{m^2} + ... + 1$.
- La utilización del almacenamiento (en el peor de los casos), para todos los nodos, excepto la raíz, es $\frac{N}{m}$.
- Los nodos tienden a tener más de m entradas, lo cual permite que la altura del árbol disminuya y mejore la utilización del almacenamiento.

De todas las operaciones necesarias para mantener y trabajar con una estructura del tipo R-Tree, en el marco de esta tesis nos parece importante explicar el algoritmo de búsqueda de un R-Tree, que se detalla en Alg. 2.1. La definición del resto de las operaciones sobre esta estructura, así como los algoritmos para la inserción y eliminación de elementos, pueden ser encontrados con más detalle en [Gut07] y [Gut84]. Sin embargo, antes de detallar el algoritmo de búsqueda, debemos explicar con más detalle el concepto de *MBR* dentro de un R-Tree, que también es usado por una serie de algoritmos que resuelven el problema del par de vecinos más cercanos, y que son comentados en la Sección 3.4. Además el concepto de *MBR*, junto con una serie de propiedades que se explicarán más adelante en este documento, son la base para los algoritmos presentados en esta tesis.

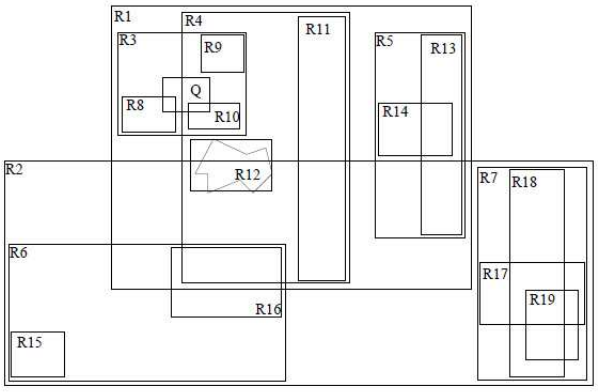


Figura 2.4: *MBR* de un R-Tree

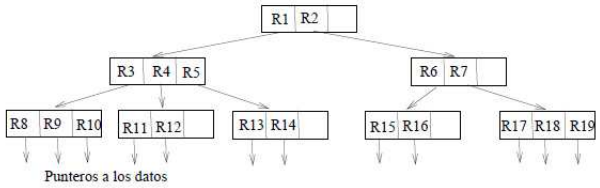


Figura 2.5: R-Tree formado por los *MBR* de la Figura 2.4

2.6.1 MBR: Minimun Bounding Rectangle

Es común que los métodos de acceso espacial almacenen aproximaciones de los objetos con los que trabajan, y luego usen estas aproximaciones para indexar los datos, y así obtener de una forma rápida y eficiente aquellos elementos que cumplan una consulta dada. Los tipos de objeto utilizados para esta aproximación dependen de la aplicación, y pueden ser rectángulos, cerraduras convexas, o figuras de varios lados. En [BKS93a] se analizan y comparan varias de estas aproximaciones.

En el caso de los R-Trees (y las estructuras derivadas de él), el objeto utilizado para esta aproximación es el Mínimo Rectángulo Contenedor, o Minimun Bounding Rectange (*MBR*), como el que se aprecia en la Figura 2.6. De acuerdo a [PSTE95], una de las principales ventajas de usar este objeto es el hecho que solamente se requiere de dos puntos para su representación. Particularmente, cualquier objeto Q es representado por el par ordenado (Q_i, Q_s) que corresponde a la esquina inferior izquierda (Q_i) y a la esquina superior derecha (Q_s) del rectángulo más pequeño que cubre completamente al objeto Q. Cuando ubicamos dos *MBR* en un plano, se pueden definir 169 relaciones espaciales entre estos *MBR*, basadas en relaciones topológicas como “disjuntos” o “contiene”. Un detalle de estas relaciones se puede encontrar en [PSTE95] y [PT97].

La importancia de estas relaciones entre los *MBR*, es que ofrecen información de las relaciones

que tienen los objetos agrupados por estos *MBR*. Si por ejemplo, dos *MBR* están “disjuntos”, entonces se puede concluir que los objetos de estos *MBR* también están disjuntos unos de otros. Esto es de vital importancia al ejecutar consultas, como las que veremos más adelante en este documento. Por ejemplo, la consulta “Encontrar todos los objetos cubiertos por el objeto Q”, consistirá de todos los objetos cuyos *MBR*’s satisfagan la relación de “estar cubiertos por Q”.

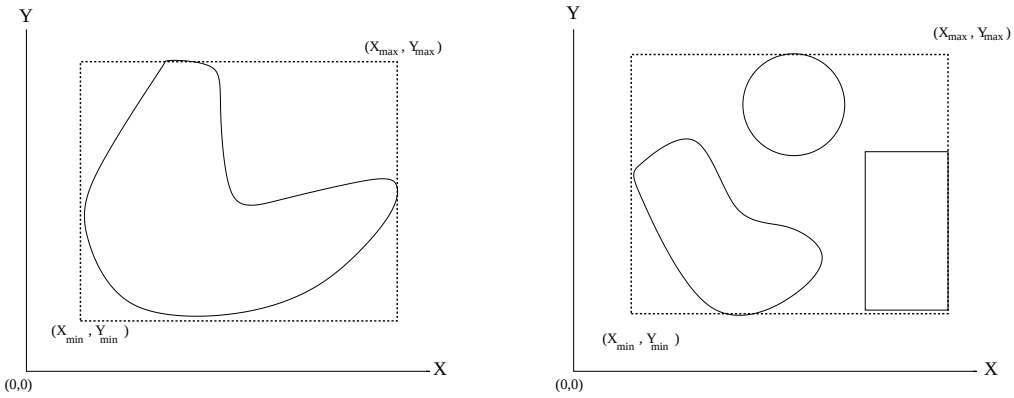


Figura 2.6: Ejemplos de MBR sobre figuras en el espacio

2.6.2 Búsqueda en un R-Tree

El algoritmo de búsqueda utilizado por el R-Tree es similar al de un B-Tree. Por ejemplo, para una consulta de tipo WQ (definida previamente en la Sección 2.3.3) la búsqueda se inicia en el nodo raíz del árbol y desciende por él hasta alcanzar un nodo hoja. En un nodo interno, la decisión sobre cuáles hijos continuar la búsqueda se hace verificando si el *MBR* (de la correspondiente entrada del nodo) y Q (el espacio de la consulta), se intersectan. 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. El algoritmo de búsqueda se muestra en el Alg. 2.1. Este sencillo algoritmo y su bajo costo hace de R-Tree uno de los métodos más utilizados para gestionar información espacial. Sin embargo, las operaciones para la creación de estos índices, así como aquellas necesarias para la mantención de los mismos, tienen un costo mucho mayor que las operaciones de búsqueda. Una explicación detallada de dichos algoritmos pueden ser encontrada en [Gut07].

Un punto importante a notar acerca de este algoritmo, es que utiliza los *MBR* para tomar las decisiones y escoger elementos candidatos, y no los elementos de manera unitaria. Este enfoque, es también utilizado por algoritmos que resuelven la consulta del par de vecinos más cercanos, sobre conjuntos indexados, y además servirá como base para la propuesta de esta tesis, que no necesita dichos índices.

```
1: BuscarEnRTree(T, Q) { T es el nodo donde se inicia la búsqueda. L es un conjunto de
   punteros a objetos cuyos MBR se intersectan con Q. } {E representa una entrada en un nodo.
   E.MBR se usa para referise al MBR de la entrada, y E.p para indicar, ya sea una referencia
   a un nodo interno del R-Tree, o una referencia donde se almacena el atributo espacial del
   objeto}
2: L = ∅
3: for cada entrada E ∈ T do
4:    if E.MBR ∩ Q ≠ ∅ then
5:       if T es una hoja then
6:          L = L ∪ {E.p}
7:       end if
8:    else
9:       L = L ∪ BuscarEnRTree(E.p, Q)
10:   end if
11: end for
12: return L
```

Alg. 2.1: Algoritmo de búsqueda en un R-Tree [Gut07]

Capítulo 3

El par de vecinos más cercanos

3.1 Introducción

Muchos trabajos de investigación dedicados a las consultas espaciales basadas en métodos de acceso multidimensionales, se centran en optimizar la consulta del vecino más cercano o su extensión para los *k*-vecinos más cercanos. En el siguiente capítulo presentaremos de manera formal dicha consulta, primero definiendo la métrica de distancia, concepto clave para entender y resolver la consulta del par de vecinos más cercanos.

Primero veremos cómo este problema ha sido resuelto desde el punto de vista de la geometría computacional, donde presentaremos y explicaremos brevemente algunos de los algoritmos más importantes que resuelven el problema de manera eficiente, cuando es posible trabajar con todos los datos en memoria principal.

Finalmente, en este capítulo veremos el problema desde el enfoque de las Bases de Datos Espaciales, donde el volumen de datos, hace inaplicable los algoritmos propuestos en la geometría computacional. Presentaremos algunas de las soluciones más importantes y veremos que estas soluciones se basan en la existencia de índices como los presentados en el capítulo anterior. Esto nos llevará a analizar el escenario de resolver la consulta del par de vecinos más cercanos, en las Bases de Datos Espaciales, cuando los conjuntos no disponen de dichos índices, que es el tema central de esta tesis.

3.1.1 Definición de la métrica distancia y espacio euclídeo

Para la mayoría de los problemas geométricos, relativos a puntos y su ubicación en el espacio, interviene el cálculo implícito o explícito de la distancia. Por eso, antes de definir la consulta del par de vecinos más cercanos, debemos tener claro el concepto de distancia. La noción de distancia se utilizará para definir el concepto de espacio Euclídeo *k*-dimensional, que es un caso especial del espacio métrico. Las siguientes definiciones son tomadas de [Cor02] y explican claramente este concepto.

Definición 1 (Métrica y espacio métrico:) *Sea P un conjunto de puntos. Una métrica sobre P es una función* ***dist****:* $P \times P \rightarrow \Re$ *(siendo* $\Re$ *el conjunto de los números reales) que satisface los siguientes axiomas para todo* $p,q,s \in P$*:*

1. $dist(p,q) \geq 0$
2. $dist(p,q) = 0 \Leftrightarrow p = q$
3. $dist(p,q) = dist(q,p)$
4. $dist(p,q) \leq dist(p,s) + dist(s,q)$

La métrica ***dist*** *se denomina distancia entre dos puntos e induce una topología sobre P, denominada topología métrica definida por* ***dist****. Una topología definida sobre un conjunto es una familia de subconjuntos que satisface ciertos axiomas. Al par (P,****dist****) se le denomina espacio métrico. Si a* $\subset$ *P, al par* $(A, \textbf{dist}|A_xA)$ *se le llama subespacio métrico de P.*

Como ejemplo de métrica distancia sobre puntos definidos por coordenadas reales (elementos de P), tenemos la L_t-métrica, L_t-distancia o métrica de Minkowski, $d_t(p,q)$ que pasamos a definir a continuación:

Definición 2 (L_t-distancia o Métrica de Minkowski) *Definimos la* L_t*-distancia,* $d_t(p,q)$*, entre los puntos* $p = (p_1, p_2, ...p_k); p_i \in \Re \;\; y \;\; q = (q_1, q_2, ...q_k); q_i \in \Re$ *en el espacio k-dimensional como:*

$$d_t(p,q) = (\sum_{i=0}^{k} |p_i - q_i|^t)^{\frac{1}{t}}, \;\; si \;\; 1 \leq t \leq \infty$$

$$d_\infty(p,q) = max_{1 \leq i \leq k} |p_i - q_i|, \;\; si \;\; t = \infty$$

Definición 3 (Espacio Euclídeo k-dimensional, E^k:) *El* ***espacio Euclídeo*** k***-dimesional*** *(donde k es un número entero positivo) es el espacio métrico definido por* $E^k = \{p = (p_1, p_2 ... p_k); p_i \in \Re, 1 \leq i \leq k\}$ *con la topología determinada por la métrica Euclídea (distancia Euclídea).*

$$d(p,q) = \sqrt{\sum_{i=0}^{k} |p_i - q_i|^2}$$

Por ejemplo, si tomamos dos puntos p y q, en el espacio Euclídeo de dos dimensiones E^2, con coordenadas $p = (p_1, p_2) \;\; y \;\; q = (q_1, q_2)$, obtenemos la siguiente expresión para la distancia Euclídea:

$$d(p,q) = \sqrt{|p_1 - q_1|^2 + |p_2 - q_2|^2}$$

3.2 Definición de la consulta “El par de vecinos más cercanos”

Encontrar el “par de vecinos más cercanos” consiste en encontrar los puntos más cercanos entre sí, entre dos conjuntos de puntos. En otras palabras, el par de vecinos más cercanos de P y Q es el par que tiene la menor distancia entre todos los pares de objetos que pueden ser formados eligiendo un objeto de P y otro de Q. El problema se puede extender a encontrar los k-pares de vecinos más cercanos con $k > 1$, es decir, $k-CPQ(P,Q)$. Según [CMTV04] y [Cor02], formalmente el problema de encontrar el par de objetos más cercanos se puede definir de la siguiente manera:

Definición 4 *Sea $P = \{p_1, p_2, \ldots, p_n\}$ y $Q = \{q_1, q_2, \ldots, q_m\}$ dos conjuntos de puntos, el par de vecinos más cercano de los dos conjuntos ($1-CPQ(P,Q)$) corresponde al par*

$$(p_z, q_l), p_z \in P \wedge q_l \in Q$$

tal que

$$dist(p_i, q_j) \geq dist(p_z, q_l), \forall p_i \in P \wedge \forall q_j \in Q$$

3.3 Solución desde el punto de vista de la geometría computacional

La búsqueda del “par de vecinos más cercanos”, es un problema que ha sido resuelto desde el punto de vista de la geometría computacional, con algoritmos que trabajan con todos los datos en memoria, y que resuelven el problema de manera eficiente [DHKP97][Smi95].

Por ejemplo, en [SH75] se presenta un algoritmo recursivo con tiempo de ejecución $O(N * \log N)$ para encontrar el par de vecinos más cercanos en un conjunto de puntos P en un espacio E^2. Dicho algoritmo está basado en una estrategia de divide y vencerás, donde primero se ordenan los puntos según un eje para luego, utilizando esa ordenación, dividir el conjunto en dos mitades P_i y P_d de acuerdo a una línea imaginaria x_m en el eje X. En este punto, el par más cercano del conjunto está en una de las dos mitades o bien, está formado por un elemento de cada mitad. Primero, se busca en cada mitad el par más cercanos Par_i y Par_d, procesando de manera recursiva P_i y P_d respectivamente. Luego se establece una distancia d_{min} como la menor entre la distancia que separa al Par_i o la distancia que separa al par Par_d. Usando esta d_{min} se procede a buscar el par más cercano Par_{id} tomando un elemento de P_i y otro P_d, procesando solamente aquellos elementos que se encuentran dentro de la distancia d_{min} de la línea divisoria en x_m. Finalmente, el par más cercano es aquel con la menor distancia entre los pares Par_i, Par_d o Par_{id}. Dicho algoritmo se muestra en Alg. 3.1.

En [BS76] se presenta una extensión para el algoritmo anterior de manera que puede resolver el problema en espacios con $d \geq 2$ dimensiones. Sin embargo, desde el punto de vista de geometría computacional, el cálculo del par más cercano es considerado estático, debido a que una vez encontrado el dicho par, se asume que no se realizan actualizaciones de los conjuntos, es decir, no se agregan o eliminan elementos. Además, la geometría computacional también limita el tamaño de los conjuntos, de tal manera que puedan ser contenidos en memoria principal. Estas restricciones no permiten que dichos algoritmos sean aplicables en el ámbito de las BDEs, donde el volumen de datos es mucho mayor, haciendo necesario un nuevo enfoque para trabajar.

1: **ParMasCercano(*P*)** {*Sea P un conjunto de puntos*}
2: Ordenar los puntos según la coordenada X
3: Definir una línea vertical X_m en la mitad de X
4: Dividir P en dos conjuntos de igual tamaño P_D y P_D, de acuerdo a X_m
5: Sea P_I una de las mitades de P
6: Sea P_D la otra mitad de P
7: Par_i=**ParMasCercano(P_I)**
8: Par_d=**ParMasCercano(P_D)**
9: Sea d_i la distancia del par más cercano Par_i
10: Sea d_d la distancia del par más cercano Par_d
11: d_{min} = **MINIMO(d_i,d_d)**
12: C_i= Elementos de P_I cuya distancia a $x_m < d_{min}$
13: C_d= Elementos de P_D cuya distancia a $x_m < d_{min}$
14: Par_{id} = par con la menor distancia, tomando un elemento de C_i y otro de C_d
15: **return** El Par con la menor distancia entre Par_i,Par_d,Par_{id}

Alg. 3.1: Algoritmo Divide y Vencerás para buscar el par más cercano [RKV95].

3.4 Solución desde el punto de vista de las Bases de Datos Espaciales

Si bien los algoritmos presentados en la sección anterior resuelven el problema de manera eficiente, cuando los conjuntos son muy grandes, no es posible mantener estructuras con todos los datos en la memoria principal y es necesario almacenarlas en disco, como es el caso de las BDEs. Esta consulta se puede ver como un tipo especial de join espacial (Sección 2.3.9), considerando que todos los pares de objetos del conjunto son candidatos al resultado final, pero también es posible verla como una consulta del tipo vecino más próximo (Sección 2.3.8), en el sentido de que las métricas y heurísticas de poda para algoritmos branch-and-bound son similares para reducir el espacio de búsqueda y generar el conjunto resultado [Cor02]. Al intentar resolver la consulta del par de vecinos más cercanos, en las BDEs podemos encontrarnos con dos escenarios claros: que los conjuntos dispongan de un índice espacial, o bien, que dichos conjuntos no se encuentren indexados. Primero analizaremos el escenario sobre conjuntos indexados, veremos que ha sido estudiado y que además existen propuestas que resuelven el problema de una manera eficiente. Luego analizaremos el segundo escenario, sobre conjuntos sin indexar. Veremos que este caso aún no ha sido estudiado en profundidad y que, además, no existen soluciones que resuelvan el problema completamente. Es en este escenario donde se enfocará el trabajo de esta tesis.

3.4.1 Sobre conjuntos indexados

Los trabajos que han abordado este problema obteniendo los mejores resultados, están basados en índices multidimensionales, particularmente R-Tree o alguna de sus variaciones (R*-Tree, R+-Tree).

Uno de los primeros trabajos en esta área es el presentado en [HS98a]. En éste se aborda el procesamiento de dos consultas basadas en el "Spatial Join": el "Distance Join" donde, una vez calculado el producto cartesiano de dos conjuntos (S_1, S_2), el resultado se ordena mediante la distancia entre los elementos obtenidos de cada conjunto, y el "Distance Semi-Join" donde, para cada elemento de S_1, se encuentra el objeto más cercano en S_2. La solución planteada es un algoritmo incremental, sobre índices del tipo R-Tree (R_1, R_2) y que utiliza una cola de prioridad. Cada item de la cola contiene una tupla compuesta por un elemento de $R1$ y otro de $R2$, ordenados según la distancia entre ambos elementos. Como los elementos pueden ser objetos o nodos, las tuplas de la cola pueden pueden ser nodo/nodo, nodo/objeto, objeto/nodo y objeto/objeto. Además, si en las hojas se pueden almacenar MBRs, entonces los pares pueden ser nueve tipos, de los cuales solamente se utilizan cinco: nodo/nodo, nodo/mbr, mbr/nodo, mbr/mbr y objeto/objeto.

En cada paso del algoritmo se obtiene el elemento en la cabeza de la cola de prioridad, es decir, la tupla que tiene la menor distancia. Si la tupla de elementos corresponde a un par de objetos, entonces esta tupla es el par más cercano, y no es necesario seguir procesando la cola puesto que, de los elementos que aún quedan en la estructura , ninguno tendrá una distancia menor que la tupla obtenida.

La versión básica de este algoritmo se muestra en Alg. 3.2 y 3.3. En [HS98b] se presentan más detalles del algoritmo, así como optimizaciones hechas para obtener un mejor rendimiento.

```
1: IncDistJoin(R1, R2) {Sean R1yR2 dos índices del tipo R-Tree}
2: Sea Q una cola de prioridad
3: Encolar(Q, 0, R1.Raiz, R2.Raiz)
4: while Mientras hay elementos en Q do
5:    Sea T una tupla obtenida de la cola Q
6:    Sean T1 y T2 los elementos de la tupla T
7:    if T1 y T2 son Objetos then
8:       return T
9:    else if T1 y T2 son MBR then
10:      Sea D = Distancia(T1, T2)
11:      if La cola esta vacía o D ≤ Distancia del tope de la cola then
12:         return T
13:      else
14:         Encolar(Q, D, T1, T2)
15:      end if
16:   else if T1 es nodo then
17:      ProcesarNodo(Q,T1)
18:   else
19:      ProcesarNodo(Q,T2)
20:   end if
21: end while
```

Alg. 3.2: Algoritmo básico para Join Incremental de distancia [HS98a].

En [SML00] se presentan mejoras al algoritmo presentado en [HS98a]. Utilizando la técnica de Barrido del Plano ("Plane Sweep"), junto con un esquema de varias etapas basado en Heap, se obtiene el conjunto de solución al Join Basado en Distancia de una forma iterativa incremental. Los resultados de este trabajo muestran una mejora notable respecto al número de accesos a disco, cálculos de distancia y tiempo de respuesta.

Otro trabajo importante, es el propuesto en [Cor02] y [CMTV04], donde se presentan dos algoritmos del tipo branch-and-bound, uno recursivo y otro iterativo para responder a la consulta del par de vecinos más cercanos. El algoritmo recursivo, es un enfoque básico para resolver la consulta sobre dos conjuntos indexados con R-Tree de igual altura. Dicho algoritmo se explica en Alg 3.4. Por otra parte, el algoritmo no recursivo utiliza un heap, y una serie de métricas entre los *MBR* para encontrar la solución. Este algoritmo se muestra en Alg. 3.5. Si bien ambas soluciones trabajan sobre árboles de la misma altura, el autor propone un método para poder aplicar el algoritmo en cualquier tipo de árbol R-Tree. Además también propone el uso de estas métricas sobre los *MBR*, para optimizar el rendimiento del algoritmo recursivo.

En los algoritmos propuestos en [Cor02], podemos encontrar una serie de métricas sobre los *MBR*, que son usadas por el algoritmo no recursivo, y que definen una serie de propiedades sobre las distancias entre estos objetos. Particularmente tres de estas métricas son de vital importancia para los algoritmos presentados en esta tesis y, por lo tanto, serán explicadas con más detalle en la Sección 3.4.1.1.

```
1:  ProcesarNodo(Q,E) {Sea Q una cola y E un nodo.}
2:  Sea Nodo el primer item del E
3:  Sea Item el otro elemento de E
4:  if Nodo es una hoja then
5:     for Para cada entrada O en el Nodo do
6:        Encolar(Q, Distancia(O, Item), (O, Item))
7:     end for
8:  else
9:     for Para cada Hijo H en el Nodo do
10:       Encolar(Q, Distancia(H, Item), (H, Item))
11:    end for
12: end if
```

Alg. 3.3: Algoritmo para procesar nodo en Join Incremental de distancia [HS98a].

```
1:  AlgoritmoRecursivo(R1, R2)  {Sean R1yR2 dos índices del tipo R-Tree}
2:  Empezar desde la raíz de ambos arboles (De igual altura)
3:  Sea T la Mínima distancia encontrada. Para iniciar T = ∞
4:  if Se accede a nodos internos then
5:     Propagarse hacia abajo de manera recursiva, para todos los posibles pares de MBR
6:  else if Se accede a dos hojas then
7:     Calcular la distancia D entre todos los posibles pares de puntos
8:     if D < T then
9:        T = D
10:    end if
11: end if{El par de vecinos más cercanos, corresponde al valor final de T.}
```

Alg. 3.4: Algoritmo recursivo para buscar el par de vecinos más cercanos, usando R-Trees [Cor02].

```
1:  AlgoritmoIterativo(R_1, R_2) {Sean R_1 y R_2 dos índices del tipo R-Tree}
2:  Empezar desde la raíz de ambos arboles (De igual altura)
3:  Sea T la Mínima distancia encontrada. Iniciar T = ∞
4:  Sea Heap un Heap ordenado por MINMINDIST. Iniciar H.
5:  if Si se accede a un par de nodos internos then
6:     Calcular el mínimo valor de MINMAXDIST M para todos lo pares de MBR
7:     if M < T then
8:        T = D
9:     end if
10:    Calcular MINMINDIST M para todos lo pares de MBR
11:    Insertar en el Heap los pares cuyos valor de MINMINDIST ≤ T
12: else if Se acceden a dos hojas then
13:    Calcular la distancia d para cada posible par de puntos
14:    if (d < T) then
15:       T = d
16:    end if
17: end if
18: if el Heap esta vacío then
19:    Terminar
20: else
21:    Obtener el objeto del tope del Heap
22:    if Si este objeto tiene MINMINDIST > T then
23:       Terminar
24:    else
25:       Repetir el algoritmo desde el paso 5.
26:    end if
27: end if
```

Alg. 3.5: Algoritmo iterativo para buscar el par de vecinos más cercanos usando R-Trees [Cor02].

3.4.1.1 Métricas sobre *MBR*

Los algoritmos presentados en [Cor02] y [CMTV04] hacen uso de ciertas propiedades sobre los *MBR*, que representan los objetos indexados en los R-Trees. Estas propiedades, o métricas, fueron definidas en [Cor02] y establecen relaciones de distancia entre objetos en el espacio, pudiendo ser estos objetos dos *MBR*, o un *MBR* y un punto. Tres de estas métricas (MAXMAXDIST, MINMINDIST y MINMAXDIST) se muestran en la Figura 3.2 y se definen en esta sección.

No obstante, antes de explicar MAXMAXDIST, MINMINDIST y MINMAXDIST, es necesario definir los conceptos básicos de MINDIST y MAXDIST. Para explicar estos dos conceptos, debemos tener clara la definición de distancia mínima.

De acuerdo a [RKV95], la distancia mínima de un punto p a un objeto espacial R, representada por $||(p,R)||$, para el espacio k-dimensional se define por la fórmula:

$$||(p,R)|| = min\left\{\sqrt{\sum_{i=1}^{k}|p_i - x_i|^2}\right\}, \forall x = (x_1, x_2, ...x_k) \in R \tag{3.1}$$

De forma equivalente, podemos definir esta distancia en función de las caras de un objeto en el espacio k-dimensional. De esta manera, la distancia mínima de un punto p a un objeto espacial R, caracterizado por sus conjuntos de caras, se define como :

$$||(p,R)|| = min_{f \in F(R)}\left\{min_{p_f \in f}\left\{d(p,p_g)\right\}\right\} \tag{3.2}$$

donde $F(R)$ es el conjunto de caras que forman el objeto R en el espacio de k-dimensiones. Además, f es una instancia de esas caras en $F(R)$ y d es la función que calcula la distancia entre dos puntos p y p_f, donde p_f es un punto en alguna de las caras de R.

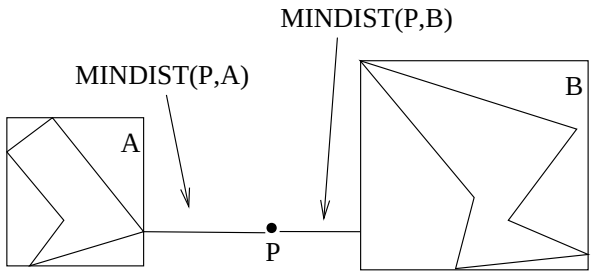


Figura 3.1: Definición de MINDIST

Con el concepto de la distancia mínima, podemos definir la métrica MINDIST(p,R) entre un punto p y un *MBR* R. Esta métrica representa el mínimo valor de distancia desde un punto a un *MBR*, definido por dos puntos.

Sea p un punto y O un objeto espacial, representado por el *MBR* R que lo contiene. Supongamos R en un espacio k-dimensional definido por dos puntos $s(s_1, s_2, \ldots, s_k)$ y $t(t_1, t_2, \ldots, t_k)$. Si p está dentro de R, entonces MINDIST es cero. Si p se encuentra fuera de R, se usa la distancia euclidiana entre p y el lado más cercano del rectángulo R.

La Ecuación 3.3 define MINDIST.

$$MINDIST(p,R) = \sum_{i=1}^{n} |p_i - r_i|^2 \tag{3.3}$$

donde

$$r_i = \begin{cases} s_i \text{ si } p_i < s_i \\ t_i \text{ si } p_i > t_i \\ p_i \text{ en otro caso} \end{cases}$$

Teorema 1 *Dado un punto p y un MBR R el cual contiene a un conjunto de objetos* $o = \{o_i, 1 \leq i \leq m\}$*, la siguiente desigualdad es siempre verdadera.*

$$\forall o \in O, MINDIST(P,R) \leq \|(P,o)\|$$

La demostración de este teorema se encuentra en [RKV95].

MINDIST se usa para determinar el objeto más cercano a p de todos los contenidos en R (ver Figura 3.1). La igualdad en el teorema anterior se produce cuando un objeto de R toca el círculo con centro en p y radio la raíz cuadrada de MINDIST.

De igual manera, es posible definir una extensión de MINDIST, para calcular la mínima distancia entre los puntos de dos lados de un objeto espacial. Sean N_P y N_Q dos conjuntos finitos de puntos y M_P y M_Q los MBRs de N_P y N_Q respectivamente. Sea r_1, r_2, r_3 y r_4 los cuatro lados de M_P y s_1, s_2, s_3 y s_4 los cuatros lados de M_Q. Se define MINDIST(r_i, s_i) como la distancia mínima entre dos puntos que caen sobre r_i y s_i. De la misma manera se define MAXDIST(r_i, s_i) como la máxima distancia entre dos puntos que caen sobre r_i y s_i.

Basándose en MINDIST y MAXDIST, las métricas entre dos objetos espaciales MINMINDIST, MINMAXDIST y MAXMAXDIST, se definen de la siguiente manera:

$$MINMINDIST(M_P, M_Q) = \min_{i,j}\{MINDIST(r_i, s_j)\}$$

En el caso de que los MBR's se intersecten, MINMINDIST(M_P, M_q) es cero. Las siguientes dos métricas, sin embargo, se pueden definir tanto si los MBR's se intersectan o no.

$$MINMAXDIST(M_P, M_Q) = \min_{i,j}\{MAXDIST(r_i, s_j)\}$$

y

$$MAXMAXDIST(M_P, M_Q) = \max_{i,j}\{MAXDIST(r_i, s_j)\}$$

Para cada par de puntos (p_i, q_j), $p_i \in M_P$ y $q_j \in M_Q$ se cumple la siguiente relación:

$$MINMINDIST(M_P, M_Q) \leq dist(p_i, q_j) \leq MAXMAXDIST(M_P, M_Q) \tag{3.4}$$

Además existe al menos un par de puntos (p_i, q_j), con $p_i \in M_P$ y $q_j \in M_Q$ tal que

$$dist(p_i, q_j) \leq MINMAXDIST(M_P, M_Q) \tag{3.5}$$

Estas métricas se aprecian en la Figura 3.2. Más detalles acerca de su definición, propiedades y así como las demostraciones de los teoremas, se pueden encontrar en [RKV95] y [Cor02].

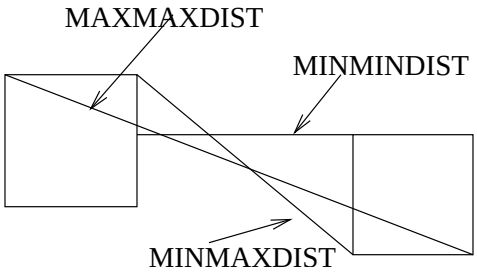


Figura 3.2: Dos MBR's y las métricas definidas entre ellos

3.4.2 Sobre conjuntos sin indexar

El uso cada vez más masificado de datos espaciales ha presentado el problema de la búsqueda del par más cercano para conjuntos que no son indexados, como pueden ser, el resultado de una consulta de selección sobre la BDE, donde uno de los dos conjuntos, o incluso ambos, pueden no tener índice. La solución para este caso, es construir un índice y aplicar los algoritmos antes mencionados. Si bien los algoritmos de búsqueda sobre estas estructuras son eficientes, las operaciones necesarias para crear y mantener dichos índices son costosas, como se comentó en la definición de los R-Tree (Sección 2.5). Además, este costo aumenta en la medida que también lo hace el tamaño de los conjuntos, como es el caso de las Bases de Datos Espaciales. Dicho costo, no se justifica cuando los índices no serán reutilizados.

Un caso particular de este escenario es cuando sólo uno de los conjuntos se encuentra indexado. Este caso fue tratado en [GG09], donde se presenta un algoritmo para encontrar el par de vecinos más cercanos considerando que sólo uno de los conjuntos de puntos se encuentra indexado por un R-Tree (R_p). Para ello se utiliza un pequeño índice en memoria, el cual particiona el espacio total de manera recursiva con una profundidad igual a la altura del R-Tree del primer conjunto (R_p). En dicha estructura se van almacenando objetos del conjunto sin índice (Q). Una vez completada su capacidad, la estructura se utiliza en conjunto con el R-Tree de R_p, para seleccionar un par candidato de vecinos más cercanos utilizando un algoritmo similar al descrito en [Cor02], explicado anteriormente en la Sección 3.4.1. La distancia del par candidato se utiliza para filtrar los objetos del conjunto Q por medio de una estrategia basada en la métrica MINDIST (entre un punto y un *MBR*) definida en la Sección métricas (3.4.1.1).

Sin embargo, para el caso en que ninguno de los conjuntos se encuentra indexado, en [QTP+08] se plantea una solución para manejar las consultas del tipo k-CPQ, en un rango determinado. Este trabajo propone una búsqueda progresiva, para luego usar R*-Trees para almacenar los pares más cercanos. Esta solución esta basada en un estudio similar presentado por [SZS03], donde se analiza el caso de la consulta CPQ, sobre el mismo set de datos, e incluso se plantea una extensión de dicha propuesta, para incluir un rango de consultas. No obstante, ambos trabajos realizan las búsqueda del par más cercano sobre un rango de los conjuntos no indexados, y además se apoyan en la creación de índices en los conjuntos antes de realizar la búsqueda.

Una aproximación básica para responder esta consulta, sobre todo el conjunto y sin la utilización de índices, es utilizar una adaptación del algoritmo BNL (Block Nested Loop), definido para gestionar el join en las bases de datos relacionales, y que se detalla en el Alg. 3.6. Este algoritmo tiene como entrada dos conjuntos S y R y un predicado que se pretende evaluar entre

todos los elementos de ambos conjunto. Mientras que un algoritmo de fuerza bruta, toma un elemento de *S* y lo compará con todos los elemento de *R*, BNL primero toma un subconjunto de elementos de *S*, y los elementos de ese subconjunto son evaluados contra todos los elementos del *R*, pero teniendo la precaución de tomar subconjuntos de *R*. De esta forma, al cargar subconjuntos de datos, BNL logra minimizar los accesos a disco. Además, la mayoría de las veces BNL se implementa de tal manera de que cada subconjunto de elementos de *R* utilice la mayor cantidad de memoria principal, antes que tener que cargar un segundo grupo desde la memoria secundaria.

Las complejidad del algoritmo BNL es $O(\frac{p_r * p_s}{M})$ donde M es el número de páginas disponibles en la memoria y p_r y p_s son los tamaños de R y S en páginas de memoria.

```
1: BNL(R, S)
2: Sea M el máximo de elementos que se pueden cargar en memoria
3: while Existen elementos en R, poner en M_r los M/2 elementos de R do
4:     while Existen elementos en S, poner en M_s los M/2 elementos de S do
5:         Procesar los elementos de M_s y M_r de acuerdo al predicado
6:     end while
7: end while
```

Alg. 3.6: Versión básica del Algoritmo BNL.

Si bien este algoritmo no es eficiente para las Bases de Datos Espaciales, es una mejor alternativa a realizar una comparación por fuerza bruta de todos los elementos del conjunto y, por lo tanto, es una buena base para comparar nuevas soluciones, como la propuesta en el siguiente capítulo.

Capítulo 4

Propuesta BCP

4.1 Introducción

En esta sección describimos una propuesta, compuesta de una estructura de datos llamada Bucket, y un algoritmo Bucket Closest Pair (BCP), que apunta a resolver el problema del par de vecinos más cercanos, sobre conjuntos que no tienen disponible un índice espacial.

A diferencia de las soluciones existentes desde el enfoque de la geometría computacional (vistas en la Sección 3.3), nuestra propuesta no necesita cargar todos los datos en la memoria principal, lo que la hace una opción viable para trabajar en el ámbito de las Bases de Datos Espaciales, donde el volumen de datos es mayor.

Por otro lado, y en contraste a aquellas vistas para las Bases de Datos Espaciales, donde los datos son indexados mediante estructuras, nuestro algoritmo BCP no necesita crear dichas estructuras para procesar la consulta.

Nuestra propuesta utiliza un esquema de dos pasos (particionamiento y procesamiento), basado en el algoritmo *Spatial Hash-Join* propuesto en [LR96] para procesar la operación de reunión espacial de dos conjuntos de objetos no indexados. Además, BCP utiliza una abstracción de los elementos basada en *MBR*, tal como lo hacen los métodos de acceso espaciales, vistos en la Sección 2.5. Asimismo BCP aprovecha una serie de métricas existentes entre los *MBR*, (descritas en las Sección 3.4.1.1) para filtrar los elementos y evitar comparaciones innecesarias. Finalmente, la inclusión de un buffer en la estructura Bucket, permite a nuestro algoritmo reducir el número de accesos a disco. Todas estas características permiten a BCP encontrar la solución al problema en un menor tiempo y con un menor consumo de recursos.

4.2 Descripción Bucket Closest Pairs

Al igual que los métodos de acceso espaciales (vistos en la Sección 2.5), que guardan aproximaciones de los objetos con los que trabaja para realizar la indexación y búsqueda, el algoritmo propuesto utiliza el concepto de Minimun Bounding Rectangle, (definido en la Sección 2.6.1) para representar los elementos del conjunto.

En base a esta abstracción, se diseñó una estructura de datos llamada “Bucket” (ver Figura 4.2), que además de agrupar los elementos y representarlos mediante un *MBR*, busca minimizar el acceso al disco utilizando un buffer de memoria, para realizar la lectura y escritura de los elementos

durante todo el algoritmo. Esta estructura se explica en detalle en la Sección 4.2.1.

El funcionamiento del algoritmo, para un par de conjuntos sin indexar, es el siguiente: en la primera etapa, llamada **"Particionamiento"**, cada conjunto se divide en un Set de Buckets (ver Figura 4.1), de manera de agrupar los elementos en estas estructuras, que se mantienen en memoria, y que luego, en la segunda etapa, llamada **"Procesamiento"**, son procesadas. En esta segunda etapa, los Buckets son filtrados utilizando las métricas MINMINDIST y MINMAXDIST definidas entre los *MBR* de cada Bucket, lo que permite disminuir la cantidad de consultas que es necesario hacer para encontrar el par de vecinos más cercanos. El algoritmo se muestra en Alg. 4.1. Las dos etapas, de particionamiento y procesamiento son detalladas en las Secciones 4.2.2 y 4.2.3 respectivamente.

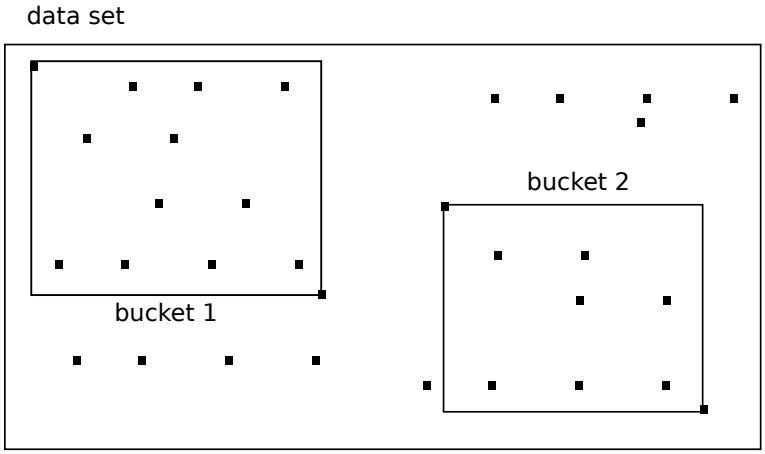


Figura 4.1: Partición de los datos en Buckets

4.2.1 Estructura de datos Bucket

Es común en los métodos de acceso espacial, guardar aproximaciones de los objetos con los que se trabaja, y usar estas aproximaciones tanto para indexar como para procesar los datos, de manera de obtener de forma rápida y eficiente aquellos elementos que cumplan una consulta dada. En el caso de los R-Trees (y las estructuras derivadas de él), el objeto utilizado para esta aproximación es el MBR (definido en 2.6.1), que requiere solamente de dos puntos para su representación.

Tomando esta abstracción como base, se creó la estructura de datos Bucket, que se mantiene en memoria principal, y que además se encarga de agrupar los datos. Esta estructura está definida por la tupla $\langle MBR, pBuffer, pFile \rangle$, donde *MBR* es la abstracción que representa a los puntos que contenidos en el Bucket, $pBuffer$ indica un buffer donde se mantienen en memoria una parte de los objetos, y $pFile$ es un puntero al almacenamiento secundario donde están almacenados todos los elementos contenidos en esa estructura. El objetivo de $pBuffer$ es guardar temporalmente los objetos que se van insertando en el Bucket al hacer el particionamiento del conjunto, de manera que, solamente una vez que $pBuffer$ se ha llenado, los objetos se transfieren al almacenamiento secundario indicado por $pFile$. De esta forma, se evita realizar un acceso por cada punto que se debe almacenar en disco. Este $pBuffer$ es también usado en la etapa de procesamiento, de manera

similar, para ir leyendo objetos desde el almacenamiento secundario y procesarlos en memoria, sin tener que acceder a los datos uno por uno.

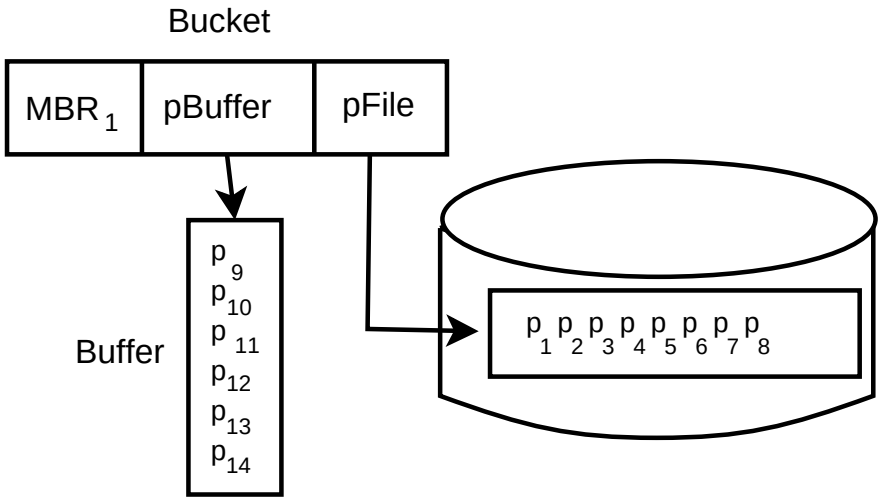


Figura 4.2: La estructura de datos Bucket

1: **BCP(**S_1**,** nb_1**,** S_2**,** nb_2**)**
2: *Sean* S_1 *y* S_2 *dos conjuntos de datos espaciales sin indexar*
3: *Sean* nb_1 *y* nb_2 *el número de Buckets en que se particionará* S_1 *y* S_2 *respectivamente*
4: ***Etapa de Particionamiento***
5: $B1$ = **Particionar**(S_1, nb_1)
6: $B2$ = **Particionar**(S_2, nb_2)
7: $B1$ *y* $B2$ *representan los sets de Buckets obtenidos de particionar* S_1 *y* S_2 *respectivamente*
8: ***Etapa de Procesamiento***
9: P = **Procesar**($B1$,nb_2,$B2$,nb_2)
10: P es el par de puntos más cercanos. {Con un elemento de S_1 y otro de S_2}
11: **return** P.

Alg. 4.1: El algoritmo BCP para encontrar el par de vecinos más cercanos sobre conjuntos no indexados

4.2.2 Etapa 1: Particionamiento

La primera etapa del algoritmo, tiene como objetivo dividir cada conjunto ($S1$ y $S2$), en un Set de Buckets ($B1$ y $B2$), de manera de realizar el resto del procesamiento sobre esta estructura y no sobre cada uno de los elementos del conjunto. Para esto, primero se toma uno de los conjuntos de entrada y se crea el primer Set de *nb* Buckets. Esta cantidad *nb* de Buckets es uno de los parámetros del algoritmo y depende de cuánta memoria se quiere asignar para que el algoritmo trabaje.

Para crear estos *nb* Buckets, se implementó un sistema de particionamiento muy básico, que toma los *nb* primeros puntos del conjunto y crea los *nb* Buckets. El *MBR* de cada Bucket se define

como el punto tomado para crearlo, y luego se guarda dicho punto en el $pBuffer$ del Bucket.

Después de creados los *nb* Buckets, por cada punto restante en el conjunto se debe decidir en cuál de los *nb* Buckets creados se almacenerá. El criterio utilizado para decidir dónde incluir este nuevo punto, es determinar cuál Bucket aumentará en menor cantidad el área de su *MBR* al agregar este nuevo punto. Si existen varios Buckets candidatos en base a este criterio, entonces se discrimina en base a la cantidad de puntos que ya contiene dicho Bucket. Cuando se ha escogido el Bucket, este punto se almacena en el $pBuffer$ de este Bucket y se actualiza el *MBR* de ese Bucket si es necesario. Una vez que el $pBuffer$ se encuentra completo, entonces todos estos puntos son llevados al almacenamiento secundario, apuntado por $pFile$. El tamaño de $pBuffer$ se define considerando el tamaño de las páginas de memoria, para así evitar accesos adicionales al disco.

Cuando ya no hay más puntos en el conjunto, los elementos que eventualmente se encuentran en los $pBuffer$ de los *nb* Buckets residentes en memoria son llevados al almacenamiento secundario. De esta forma en memoria solamente se mantiene el Set de los *nb* Buckets. El *MBR* de cada Bucket es la abstracción de todos los elementos contenidos en ese Bucket, mientras que $pFile$ apunta al lugar donde físicamente están almacenados dichos elementos. Una vez terminado el proceso con el primer conjunto, la operación se vuelve a realizar sobre el segundo conjunto, de manera de obtener el segundo Set de Buckets. El algoritmo para esta primera etapa de particionamiento se muestra en Alg. 4.2.

```
1:  Particionar(S, nb)
2:  Sea S es el que conjunto que se va a particionar y sea B el conjunto de los nb Buckets
3:  for cada punto p_i de S, 1 ≤ i ≤ nb do
4:      B_i = CrearBucket(p_i)
5:  end for
6:  for cada punto p restante en S do
7:      Sea I = {B_i, 1 ≤ i ≤ nb | p ∩ B_i.MBR ≠ ∅}
8:      if I ≠ ∅ then
9:          Sea i = min{Area(B_i.MBR), 1 ≤ i ≤ |I|}
10:     else
11:         Sea i el Bucket que tiene que aumentar menos su área tal que contenga a p.
12:     end if
13:     AgregarPuntoBucket(B_i, p)
14: end for
15: for cada Bucket B_i, 1 ≤ i ≤ nb do
16:     Guardar los objetos de B_i.pBuffer en el disco usando B_i.pFile
17: end for
18: return B
```

Alg. 4.2: Etapa 1 del algoritmo BCP: Particionamiento del conjunto en sets de Buckets

En base al algoritmo utilizado en esta etapa de particionamiento, podemos determinar que la cantidad de accesos a disco requeridos *AP* para particionar cada conjunto en los sets de Buckets, no depende de estos, sino que del tamaño de los datos, y de la página con la que se está trabajando. Dicha cantidad de accesos está acotada por la fórmula $AP = \frac{TC * TO}{TP}$ donde TC es el tamaño del conjunto (número de puntos), TO es el tamaño de cada objeto (en bytes) y TP es el tamaño de la página en el sistema operativo (en bytes).

Por ejemplo, para un conjunto de 200.000 objetos, donde cada punto utiliza 12 bytes, y considerando una página de tamaño 1024 bytes, el algoritmo hará aproximadamente 4700 accesos para particionar ambos conjuntos y crear los dos sets de Buckets.

1: **CrearBucket**(p)
2: *Sea p un punto y sea B un Bucket*
3: Reservar memoria para $B.pBuffer$ del tamaño de una página.
4: Apuntar $B.pFile$ a un archivo para almacenar los elementos. {Si un archivo es usado para cada Bucket, entonces $B.pfile$ es un descriptor de archivo.} {Si un solo archivo es usado para todos los Buckets, entonces $B.pfile$ apunta al último bloque de disco donde se guardan los objetos.}
5: Crear un $B.MBR$ con ambos límites iguales a p
6: Guardar p en $B.pBuffer$.
7: **return** B

Alg. 4.3: Algoritmo utilizado para crear un Bucket.

1: **AgregarPuntoBucket**(B,p)
2: **if** $B.pBuffer$ está lleno **then**
3: Guardar los objetos de $B.pBuffer$ en el disco, al espacio apuntado por $B.pFile$. { Si un archivo es usado para cada Bucket, entonces $B.pfile$ es un descriptor de archivo.} {Si un solo archivo es usado para todos los Buckets, entonces $B.pfile$ apunta al último bloque de disco donde se guardan los objetos.}
4: **end if**
5: Guardar p en $B.pBuffer$.
6: Ajustar los límites de $B.MBR$ para que contenga a p.

Alg. 4.4: Algoritmo utilizado para agregar elemento a un Bucket.

4.2.3 Etapa 2: Procesamiento

La segunda etapa del algoritmo, considera como entrada los dos Sets de Buckets $B1$ y $B2$ generados en la etapa de particionamiento. Luego, se crea un heap donde se irán insertando todos los pares de Buckets que se pueden obtener, tomando un elemento de $B1$ y otro de $B2$, ordenados por la métrica MINMINDIST entre los MBR's del par de Buckets.

Al insertar el primer par de Buckets, el algoritmo establece una distancia d cuyo valor es fijado con el valor de MINMAXDIST entre los MBR de este par de Buckets. Posteriormente, por cada uno de los pares de Buckets restantes en los conjuntos, antes de ser insertados, se calcula la métrica MINMAXDIST y es utilizada para ajustar el valor de d. Usando este valor, el algoritmo descarta de ser insertados en el heap a todos aquellos pares de Buckets cuya métrica MINMINDIST sea superior a d, pues en ese par de Buckets, no existe un par de elementos cuya distancia sea inferior a d, de acuerdo a las propiedades de las métricas MINMAXDIST Y MINMINDIST, explicadas en la Sección 3.4.1.1.

Cuando ya se han procesado todos los pares de Buckets de $B1$ y $B2$, entre aquellos que no han sido descartados, y que han sido insertados en el heap, se encuentra el par de vecinos más cercanos.

Por lo tanto, el último paso de esta etapa, consiste en extraer desde el heap cada par de Buckets, y buscar entre todos los elementos contenidos en cada Bucket el par de puntos más cercanos. Además, si definimos *m* como el MINMINDIST entre los *MBR* del primer par de Buckets tal que $m > d$, entonces notar que a partir de este punto no es necesario continuar examinando los elementos del Heap, pues todos los MINMINDIST de los pares restantes serán mayores que *m* y el par de vecinos más cercanos se encontrará en el par de Buckets con la menor distancia encontrada hasta el momento. Es importante notar que en esta segunda etapa también es usado $pBuffer$ para cargar datos desde el disco a la memoria, y de esta forma, minimizar los accesos a disco. Este etapa del algoritmo se detalla en Alg. 4.5.

```
1:  Procesamiento(B1,nb1,B2,nb2)
2:  Sea H un Heap ordenado por MINMINDIST
3:  Sea d = MINMAXDIST(B1_1.MBR, B2_1.MBR)
4:  for cada posible par de Buckets (B1_i, B2_j) do
5:     d = min{d, MINMAXDIST(B1_i.MBR, B2_j.MBR)} {Se usa la metrica MINMAXDIST para filtrar
       los Buckets que se insertan en el Heap.}
6:     if MINMINDIST(B1_i.MBR, B2_j.MBR) < d then
7:        Insertar en h el par de Buckets (B1_i, B2_j)
8:     end if
9:  end for
10: (B^p, B^q)= Eliminar(H) {Sean B^p y B^q un par de Buckets.}
11: (p,q)=ParMasCercanoBucket(B^p, nb1, B^q, nb2)
12: d = distancia(p,q)
13: while quedan elementos en H do
14:    (B^i, B^j)= Eliminar(H)
15:    if MINMINDIST(B^i.MBR, B^j.MBR) > d then
16:       return (p,q)
17:    end if
18:    (p_i, q_j)=ParMasCercanoBucket(B^i, nb1, B^j, nb2)
19:    d_i = distancia(p_i, q_j)
20:    if d_i ≤ d then
21:       (p,q) = (p_i, q_j)
22:       d = d_i
23:    end if
24: end while
```

Alg. 4.5: Etapa 2 del algoritmo BCP: Procesamiento de los sets de Buckets

4.3 Experimentación con BCP

En la siguiente sección, se describen una serie de experimentos realizados con el objeto de probar nuestro algoritmo BCP, para responder la consulta del par de vecinos más cercanos sobre conjuntos no indexados en distintos escenarios. Los objetivos de los experimentos fueron:

- Comparar BCP con las soluciones que utilizan índices espaciales.

```
1: ParMasCercanoBucket(B1,nb1,B2,nb2)
2: Aumentar el tamaño de B1.pBuffer.
3: Aumentar el tamaño de B2.pBuffer. {Como en ésta etapa solamente se analizan los
   elementos de dos Buckets a la vez, se aumenta el tamaño del Buffer, de manera de utilizar
   la memoria reservada por los otros Buckets.}
4: while Quedan elementos en B1.pFile para cargar en B1.pBuffer do
5:    Cargar B1.pBuffer con puntos desde B1.pFile
6:    while Quedan elementos en B2.pFile para cargar en B2.pBuffer do
7:       Cargar B2.pBuffer con puntos desde B2.pFile
8:       for cada punto p_i en B1.pBuffer do
9:          for cada punto q_j en B2.pBuffer do
10:            Buscar el par (p,q) con la menor distancia.
11:         end for
12:      end for
13:   end while
14: end while
15: Reducir el tamaño de B1.pBuffer y B2.pBuffer a su tamaño original.
16: return (p,q)
```

Alg. 4.6: Algoritmo utilizado para buscar el par más cercano dentro de dos Bucket.

- Evaluar el rendimiento de BCP utilizando distintas cantidades de memoria.
- Evaluar el rendimiento de BCP sobre conjuntos de distinto tamaño.
- Evaluar el rendimiento de BCP en conjuntos con distintos porcentajes de intersección.
- Evaluar el rendimiento de BCP en conjuntos de distintas distribuciones.
- Medir el rendimiento de BCP sobre conjuntos de datos reales.

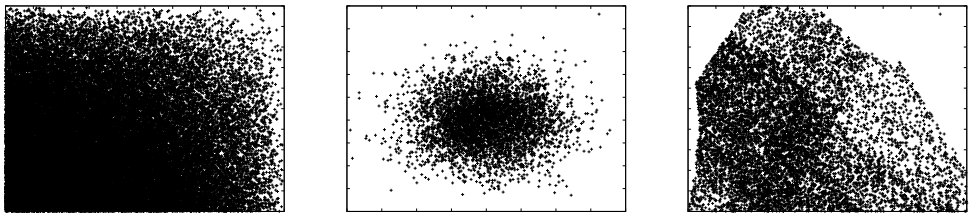
Como medida de rendimiento, y de acuerdo a lo propuesto en [CMTV04] y [CMTV00], se consideró el tiempo total del algoritmo y la cantidad de accesos a disco requeridos para obtener la solución.

En la primera prueba, BCP se comparó con una de las soluciones basada en índices espaciales R-Tree (vistas en la Sección 3.4.1), mientras que en el resto de las pruebas, se utilizó el algoritmo BNL (definido en la Sección 3.6) como testigo para comparar nuestra propuesta.

Los algoritmos fueron implementados en lenguaje JAVA, para poder comparárlos con librerías que implementan el R-Tree. Se utilizó la versión 1.6, y las pruebas fueron realizadas en un computador DELL INSPIRON N4030, con la siguiente configuración:

- Procesador Intel Core i3 de 2.58 Ghz.
- Memoria Ram de 4 Gigabytes, DDR3 de 800 Mhz.
- Disco duro de 200 Gigabytes, SATA de 7200 rpm.
- Sistema operativo Linux Fedora 15 de 64 bits.

Otros parámetros más específicos de los experimentos se detallan a continuación.



(a) Dist. Uniforme (b) Dist. Gaussiana (c) Datos Reales

Figura 4.3: Tipos de Datos

4.3.1 Condiciones y parámetros de los experimentos

Para la implementación de los algoritmos se consideraron objetos del tipo punto, en dos dimensiones. En la implementación con Java, se utilizó el tipo de dato $float$. Sabiendo que el tamaño de página del sistema operativo utilizado es de 4096 bytes, en los experimentos se consideró un tamaño de página de 1024 bytes, de manera de minimizar los accesos a disco adicionales realizados por otras tareas ejecutadas en el sistema operativo al momento de realizar las pruebas.

Respecto a los tamaños de los conjuntos usados como datos de prueba, estos fueron tres: 200K, 400K y 600K, donde K representa 1000 puntos. Si bien, los valores utilizados normalmente en las pruebas de algoritmos con métodos de acceso espaciales, como los presentados en [CMTV04],[CMTV00] y [QTP$^+$08], es de alrededor de 10K a 100K, quisimos aumentar significativamente el tamaño de los conjuntos de manera de evaluar los resultados sobre parámetros más allá de los comunes, considerando que el volumen de información en los sistemas de Bases de Datos Espaciales aumenta cada día. Cabe especificar también que en cada prueba se utilizó la misma cantidad de puntos para ambos conjuntos.

Además, los conjuntos de entrada para el algoritmo fueron clasificados en tres grupos, de acuerdo a la naturaleza de sus elementos, a saber:

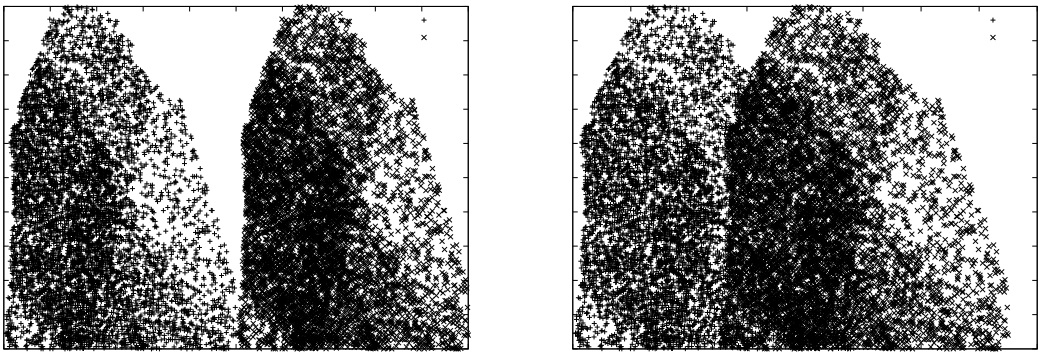
1. Datos generados de manera sintética, con una distrubución uniforme sobre el espacio, designado como U (ver Figura 4.3-a)

2. Datos generados de manera sintética, con una distrubución Gaussiana, designado como G (ver Figura 4.3-b)

3. Datos reales, obtenidos de sistemas de información geográfica, normalizados en coordenadas entre 0 y 1, designado como R (ver Figura 4.3-c)

Asimismo, para las pruebas donde se evaluaba el grado de intersección de los conjuntos, se consideraron cinco porcentajes de overlap (intersección) entre las áreas de los conjuntos: 0%, 25%, 50%, 75% y 100%. A modo de ejemplo, en la Figura 4.4 se muestran dos conjuntos de datos reales, con overlap de 0% y 50%.

Finalmente, es importante mencionar que cada experimento se realizó cuatro veces, sobre diferentes conjuntos (con las mismas características de tamaño, distribución y overlap según correspondía al experimento) y los resultados acá presentados indican el promedio de esos 4



(a) 0% Overlap (b) 50% Overlap

Figura 4.4: Ejemplos de Overlap entre conjuntos

experimentos. La discusión de estos resultados, así como las conclusiones obtenidas de los mismos, se realiza en la Sección 4.4.

4.3.2 Experimento 1: BCP y R-Tree

El primer experimento tuvo como objetivo comparar BCP con algoritmos que utilizan índices espaciales para procesar la consulta del par de vecinos más cercanos, como los vistos en la Sección 3.4.1. Puesto que la mayoría de estos algoritmos utilizan variantes de la estructura R-Tree, lo primero fue crear dichos índices sobre los conjuntos de prueba. Para este experimento, los conjuntos de prueba fueron datos sintéticos con distribución uniforme, de tamaño 200K,400K y 600K, sin intersección (0% overlap). Además se evaluó el rendimiento de BCP con cuatro cantidades de memoria distinta, es decir con la memoria necesaria para generar sets de 64, 128, 256 y 512 Buckets. En la Tabla 4.1 se muestran los resultados obtenidos al crear el índice R-Tree, utilizando la librería spatialindex [Had11], que también está implementada en Java. En esta primera prueba se utiliza el tiempo como medida de rendimiento.

Tiempo utilizado por el algoritmo (en segundos)

Tamaño Conjunto	R-Tree	BCP (64)	BCP (128)	BCP (256)	BCP (512)
200.000	236	2	3	4	8
400.000	475	5	7	11	17
600.000	893	9	11	16	25
Mejora Promedio		99.03%	98.67%	98.07%	96.74%

Tabla 4.1: Resultados Experimento 1: Comparación entre R-Tree y BCP

Los resultados de esta prueba indican que en promedio para todos los escenarios, BCP encuentra la solución en solamente un 1.8% del tiempo necesario solamente para construir los índices de la estructura R-Tree. Podemos apreciar que el mejor resultado obtenido necesita solo un 0,97% del tiempo total, y el peor un 3.4%. Es importante notar que el tiempo de R-Tree, no

incluye el procesamiento de la consulta que encuentra el par de puntos más cercano. Si bien con la herramienta utilizada para construir los R-Trees no se conocen los accesos a discos, el indicador de tiempo ofrece una muy buena referencia para mostrar la ventaja del algoritmo BCP para encontrar la solución sin tener que crear los índices antes de procesar la consulta. Además, conviene hacer notar que la cantidad de memoria asignada (número de Buckets) no influye significativamente en el tiempo utilizado para encontrar la consulta cuando se compara con la creación de R-Trees. De todas formas los siguientes experimentos aclararán de una mejor manera la influencia de este parámetro sobre el comportamiento del algoritmo.

4.3.3 Experimento 2: BNL y BCP en conjuntos uniformes sin intersección

En el segundo experimento comparamos BCP contra el algoritmo Block Nested Loop (BNL) el cual ha sido definido para realizar la operación de reunión en ambientes de bases de datos relacionales [ME92] y que se muestra en el Alg. 3.6. En la literatura no encontramos un algoritmo ad-hoc para resolver el problema planteado. Por esta razón adaptamos BNL y lo implementamos utilizando también el lenguaje JAVA, con el objeto de contar con un testigo contra el cual comparar BCP.

Nuestra propuesta se comparó con BNL teniendo cuidado de que ambos algoritmos utilicen la misma cantidad de memoria, esto es si BCP considera 128 Buckets, la misma cantidad de memoria ocupada por esos 128 Buckets se utiliza por BNL. Para el experimento se utilizaron cinco cantidades de memoria distinta (32, 64, 128, 256 y 512 Buckets), para determinar además, como este parámetro influye en el desempeño de los algoritmos.

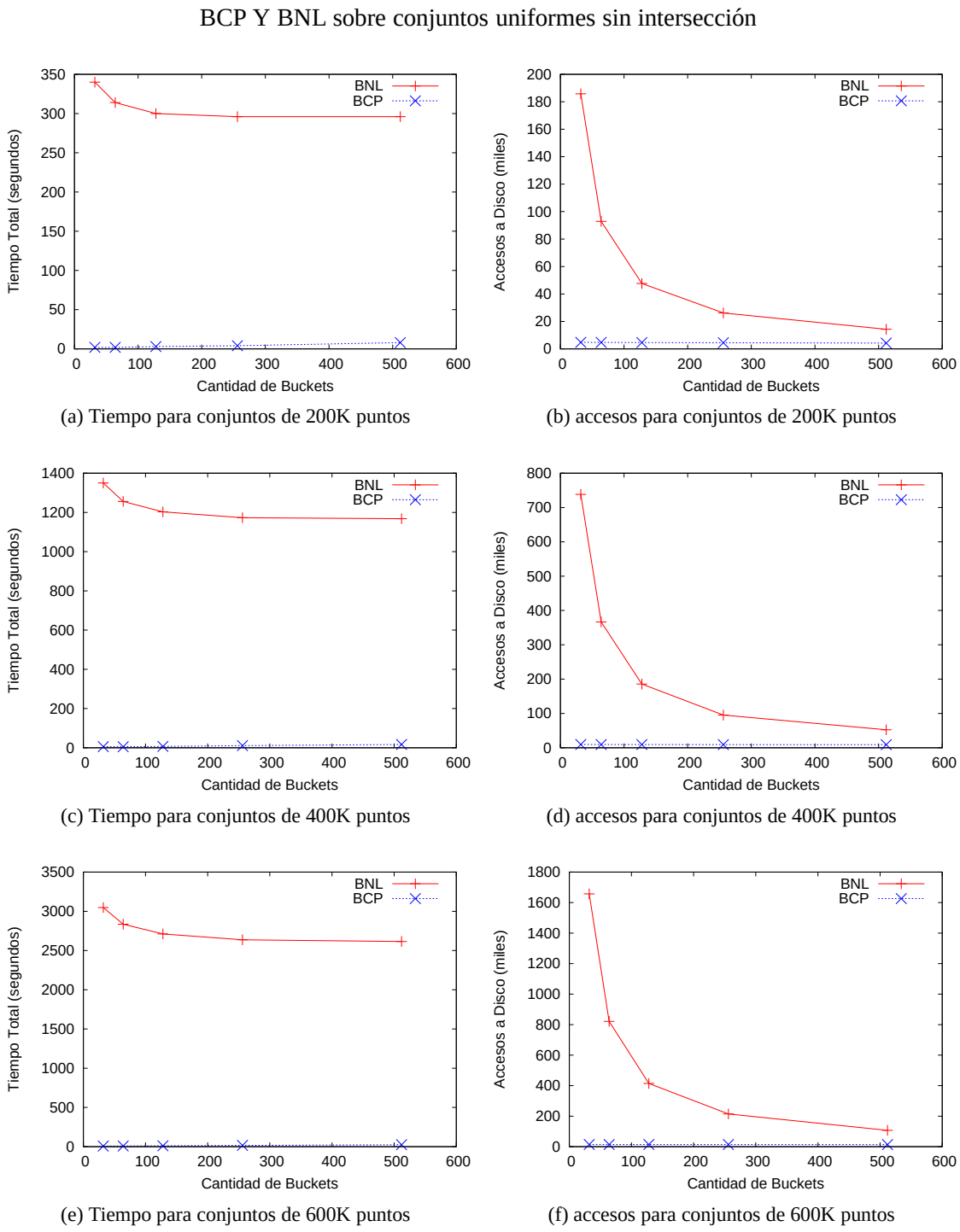
Como otro de los objetivos es medir cómo influye el tamaño del conjunto sobre el rendimiento de BCP, la prueba se realizó sobre datos con distribución uniforme, sin intersección entre ellos (0% overlap), con tres tamaños distintos, 200K,400K y 600K, donde K representa a mil puntos.

En cada prueba, se midió el tiempo total y la cantidad de accesos al disco que necesitó el algoritmo para encontrar el par de puntos más cercano entre los dos conjuntos. Los resultados de esta prueba se muestran en la tabla 4.2, ordenados de acuerdo al tamaño de los conjuntos.

En estos resultados, podemos ver que BCP requiere de un menor tiempo que BNL para encontrar el par de vecinos más cercanos, sobre todo cuando la cantidad de memoria asignada es menor. En la medida que aumentamos el número de Buckets (la cantidad de memoria disponible) el desempeño de BNL mejora, sobre todo en lo que respecta a accesos a disco, ya que puede cargar más puntos en cada lectura realizada. Si embargo, al ser BNL un algoritmo de fuerza bruta, el tiempo total se mantiene relativamente constante, puesto que comparará todos los elementos de ambos conjuntos, disminuyendo el desempeño total del algoritmo, independiente de esta mejora en los accesos.

Por otra parte, BCP muestra una superioridad en todos los casos, sobre todo en el tiempo total. Si bien, en la medida que la cantidad de Buckets aumenta, también lo hacen los accesos a disco de BCP, acercándose a BNL, el tiempo total sigue siendo significativamente menor, ya que BCP utiliza un proceso de filtrado, de manera de minimizar la cantidad de comparaciones necesarias para encontrar el par de vecinos más cercanos. Estos se puede apreciar en la Figura 4.5, donde se graficaron el tiempo total y los accesos a disco de BCP y BNL, separados de acuerdo al tamaño de los conjuntos.

De acuerdo a estos resultados, podemos concluir que cuando utilizamos poca memoria BCP es mucho mejor que BNL, y si bien cuando la cantidad de memoria aumenta lo hace también el desempeño de BNL, BCP sigue siendo superior independiente del tamaño del conjunto. En el peor de los casos (512 Buckets para conjuntos de 200K), BCP necesita un 30% de los accesos utilizados por BNL en aproximadamente un 3% del tiempo, mientras que en el mejor de los casos (32 Buckets para conjuntos de 600K), BCP supera a BNL necesitando solamente un 0.85% de los accesos en un 0.26% del tiempo.



(a) Tiempo para conjuntos de 200K puntos

(b) accesos para conjuntos de 200K puntos

(c) Tiempo para conjuntos de 400K puntos

(d) accesos para conjuntos de 400K puntos

(e) Tiempo para conjuntos de 600K puntos

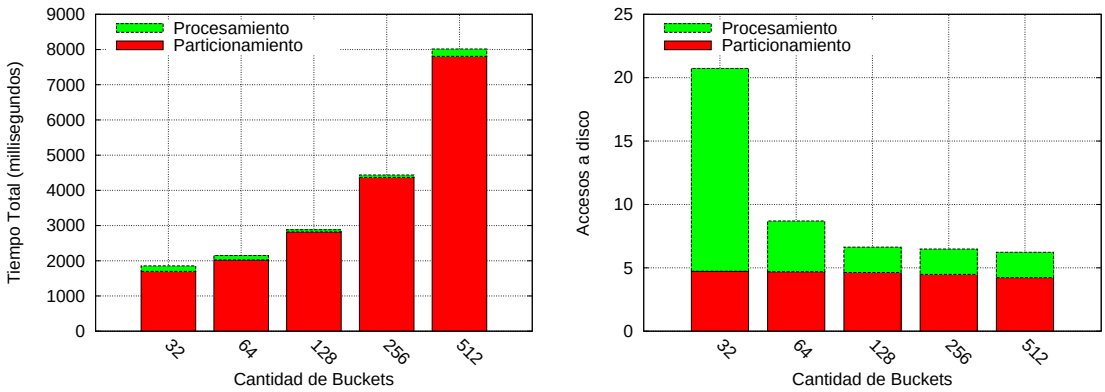
(f) accesos para conjuntos de 600K puntos

Figura 4.5: Rendimiento de BCP con distintas cantidades de Memoria (Buckets), sobre conjuntos sin intersección

Una de las cosas que llama la atención en los resultados de los experimentos es el hecho que, al aumentar la cantidad de memoria disponible, el algoritmo BCP disminuya su rendimiento, al contrario de lo sucedido con BNL. Esto se debe al algoritmo de particionamiento en la primera etapa, cuyo tiempo está determinado por $O(n * nb)$, donde n es la cantidad de elementos, y nb es el número de Buckets. Dado que una mayor cantidad de memoria, implica una mayor cantidad de Buckets, al momento de asignar un punto, es necesario realizar más comparaciones para determinar en cuál Bucket se almacenará dicho punto. Además, como la cantidad de accesos está determinada por la cantidad de elementos del conjunto y el tamaño de la página con que se trabaja, este valor se mantiene independiente de la cantidad de Buckets. Por otra parte, en un escenario donde no hay intersección entre los conjuntos, las métricas de filtrado en el procesamiento (segunda etapa) disminuyen considerablemente el número de Buckets que es necesario procesar, lo que significa que el mayor costo del algoritmo se encuentra en la primera etapa. Para entender mejor esto, en la Figura 4.6 se grafican los tiempos y accesos de cada etapa del algoritmo (Particionamiento y Procesamiento), sobre conjuntos de 200K sin intersección, utilizando 5 cantidades de memoria distinta (32, 64, 128, 256 y 512 Buckets).

En la Figura 4.6 podemos apreciar que con 32 Buckets se necesitan 4.745 accesos en 2 segundos, mientras que con 512 Buckets, la cantidad de accesos es un poco menor pero el tiempo es 8 segundos, es decir, casi 4 veces más que con menos memoria. Si miramos la Figura 4.6 a) se ve claramente que el tiempo de la etapa de Particionamiento, aumenta con la cantidad de Buckets, mientras que el tiempo de Procesamiento, es casi el mismo.

Por otro lado, en la Figura 4.6 b), se puede apreciar que la cantidad de accesos en la primera etapa, se mantiene relativamente constante, mientras que la segunda etapa los accesos disminuyen en la medida que existen más Buckets. Esto se explica dado que al existir menos Buckets, habrá más elementos asignados a cada uno y, por lo tanto, al procesar un Bucket, serán necesarios más accesos a disco para poder obtener todos los elementos de éste. Sin embargo, este aumento en los accesos no se ve reflejado en el tiempo total de la etapa, dado que la cantidad de elementos que es necesario procesar, es prácticamente la misma, y la cantidad de Buckets solo influye en los accesos a disco necesarios para recuperar dichos elementos.



(a) Tiempo por etapas

(b) accesos por Etapa

Figura 4.6: Costos de las etapas del algoritmo BCP con 5 cantidades de memoria, sobre conjuntos de 200K sin intersección

Respecto al tamaño del conjunto, que era otro de los objetivos a evaluar, en los experimentos podemos apreciar que BCP supera a BNL para todos los tamaños de conjuntos, tanto en tiempo total como en cantidad de accesos a disco. Además, BCP aumenta su ventaja sobre BNL en la medida que aumenta el tamaño de los conjuntos y disminuye la memoria disponible para trabajar.

		BNL		**BCP**	
Tamaño	Buckets	Tiempo (s)	accesos a disco	Tiempo (s)	accesos a disco
200K	32	340	185.794	2	4.745
200K	64	314	92.896	2	4.698
200K	128	300	47.638	3	4.635
200K	256	296	26.200	4	4.486
200K	512	296	14.290	8	4.229
PROMEDIO		**309**	**73.364**	**4**	**4.559**
400K	32	1.350	738.263	6	9.671
400K	64	1.256	366.749	5	9.467
400K	128	1.203	185.755	7	9.387
400K	256	1.173	95.258	11	9.265
400K	512	1.168	52.391	17	9.021
PROMEDIO		**1.230**	**287.683**	**9**	**9.362**
600K	32	3.048	1.657.406	8	14.349
600K	64	2.835	821.558	9	14.256
600K	128	2.712	414.350	11	14.172
600K	256	2.637	214.318	16	14.010
600K	512	2.615	107.158	25	13.790
PROMEDIO		**2.769**	**642.958**	**14**	**14.115**

Tabla 4.2: Resultados Experimento 2: Compración entre BNL y BCP sobre conjuntos con distribución uniforme sin intersección y con distintas cantidades de memoria

4.3.4 Experimento 3: BNL Y BCP sobre conjuntos con distinto porcentaje de intersección

Como los primeros experimentos se realizaron sobre conjuntos sin intersección (0% overlap), en la siguiente prueba se evaluó el rendimiento de BCP y BNL, sobre datos cuyas áreas se intersectan, como se aprecia en la Figura 4.4. Para esto, se realizó la consulta del par de vecinos más cercanos sobre conjuntos de datos sintéticos, con una distribución uniforme, y de tamaño 200K, 400K y 600K, pero cuyas áreas presentaban 5 porcentajes de intersección distintos, a saber 0% sin intersección, 25%, 50% y 75% y 100% de overlap. Además, en base a la prueba anterior, se decidió omitir la cantidad de memoria de 32 Buckets, puesto que es una cantidad demasiado pequeña y además provoca tiempos de respuesta no tan eficientes tanto para BCP como para BNL. Los resultados de esas pruebas se detallan en la Tabla 4.3 a) para el tiempo total utilizado por el algoritmo y en la Tabla 4.3 b) para los accesos a disco. Puesto que el rendimiento de BNL fue el mismo para todos los porcentajes de interesección de los conjuntos, los resultados de este algoritmo se detallan una sola vez en cada tabla, a diferencia de los resultados de BCP, que se muestran por cada porcentaje de overlap.

Este experimento demostró que el rendimiento de BCP sí se ve afectado por la intersección de las áreas que abarcan los conjuntos, a diferencia de BNL que no ve disminuido su rendimiento por este parámetro. Incluso podemos apreciar que BCP realiza más accesos a disco que BNL cuando el tamaño de los conjuntos es pequeño (200K), la cantidad de memoria asignada es la mayor (512 Buckets) y el porcentaje de intersección es mayor que 0. No obstante, el tiempo total necesitado por BCP sigue siendo menor que BNL para todos los casos evaluados.

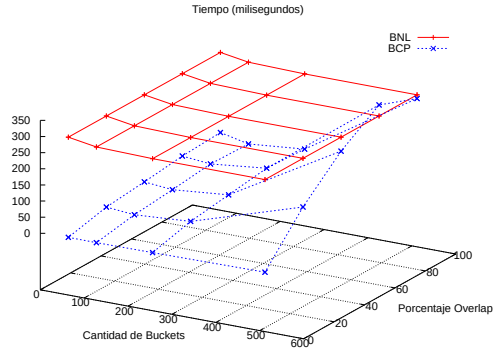
Esta diferencia entre los accesos y tiempo, se explica por la naturaleza de BNL, que lee y procesa todos los datos de ambos conjuntos, mientras que BCP utiliza un esquema de partición y procesamiento para evitar hacer comparaciones innecesarias. Además, gracias al uso de la estructura Bucket, que incluye un buffer, cuando BCP realiza un acceso a disco, carga en memoria puntos de la estructura que no necesariamente son procesados, ya que algunos elementos son descartados gracias a las métricas definidas entre los *MBR* de los Buckets. Es decir, BNL procesa todos los elementos cargados en memoria después de un acceso a disco, mientras que BCP no.

Podemos apreciar que el peor caso está dado en el conjunto de 200K cuando se trabaja con 512 Buckets y los conjuntos tienen un 100% de intersección. En este escenario BCP necesita casi 3 veces más accesos que BNL, y encuentra la solución con una mejora de apenas un 4% en el tiempo total. Mientras que en el mismo escenario de intersección, cuando el tamaño del conjunto es de 600K, BCP logra encontrar la solución en solo un 12% del tiempo requerido por BNL, realizando solo un 3% de los accesos a disco, cuando se utilizan 256 Buckets.

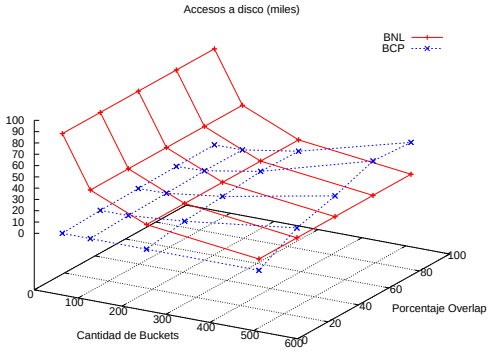
Estos resultados se aprecian mejor en la Figura 4.7 donde se comparan gráficamente BCP y BNL, tanto desde el punto de vista del tiempo, como los accesos a disco, ordenados por los distintos tamaños de los conjuntos. En dicha Figura, se aprecia claramente cómo BNL no altera su rendimiento por los distintos porcentajes de overlap, pero sí por la cantidad de memoria disponible. Por el contrario, podemos ver que el rendimiento de BCP sí varía en la medida que lo hace el porcentaje de overlap entre los conjuntos,

Podemos concluir que, si bien el porcentaje de intersección afecta significativamente la cantidad de accesos de BCP sobre todo en los conjuntos de menor tamaño, de todas formas nuestra propuesta presenta una mejora sobre BNL, especialmente cuando se utiliza poca memoria. Este resultado es consistente con los obtenidos en el primer experimento, donde podemos apreciar que

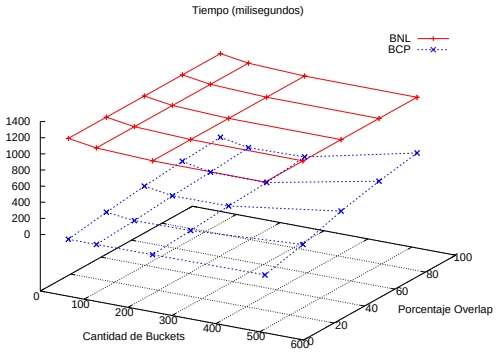
BCP Y BNL sobre conjuntos uniformes con intersección



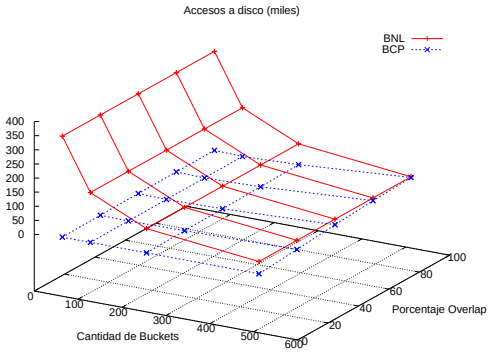
(a) Tiempo para conjuntos de 200K puntos



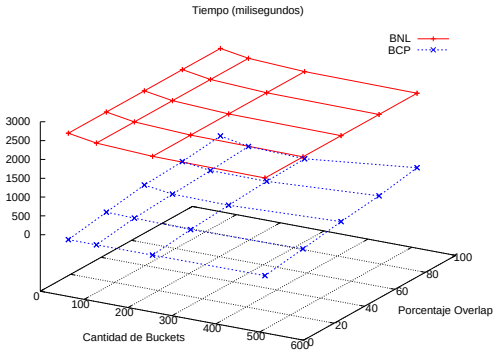
(b) accesos para conjuntos de 200K puntos



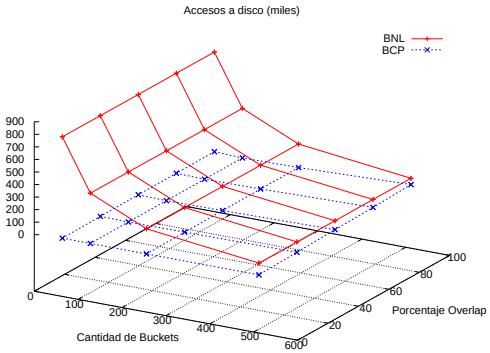
(c) Tiempo para conjuntos de 400K puntos



(d) accesos para conjuntos de 400K puntos



(e) Tiempo para conjuntos de 600K puntos



(f) accesos para conjuntos de 600K puntos

Figura 4.7: Rendimiento de BCP con distintas cantidades de memoria (Buckets) sobre conjuntos con distinto porcentaje de intersección

con menos memoria BCP encuentra la solución más rápidamente que BNL.

Para entender mejor el comportamiento del algoritmo frente a la memoria asignada y los porcentajes de intersección, tal como se hizo en el experimento anterior, en la Tabla 4.4 se detallaron los tiempos de cada etapa del algoritmo, para el conjunto de 200K de acuerdo a los distintos porcentajes de intersección. Dichos resultados demuestran que, independiente de la intersección entre los conjuntos, en la primera etapa del algoritmo la cantidad de accesos al disco se mantiene constante, mientras que el tiempo aumenta en la medida que hay más Buckets. Además, cuando la intersección es 0%, el mayor costo del algoritmo está en la primera etapa. Por otra parte, en la segunda etapa, el tiempo y la cantidad de accesos aumenta en la medida que hay más interesección. De hecho, para cualquier valor de intersección mayor que 0%, la parte del algoritmo más costosa es la segunda etapa, puesto que al haber intersección entre las áreas de los conjuntos, habrá también intersección entre las áreas de los Buckets y cuando esto sucede, las reglas de filtrado no son tan eficientes descartando elementos.

4.3.5 Experimento 4: BNL Y BCP sobre conjuntos con distinta distribución

Los experimentos realizados anteriormente tuvieron como entrada, conjuntos de datos sintéticos, distribuidos uniformemente en el espacio. Sin embargo, uno de los objetivos de los experimentos es comprobar si el comportamiento del algoritmo varía con la distribución de los puntos en el espacio. Para evaluar esto, se probó BCP y BNL sobre conjuntos sin intersección, pero donde la distribución de los datos no es uniforme. En el experimento se consideraron 5 escenarios distintos: UxG, UxR, GxG, RxR, y GxR, donde U son los datos sintéticos uniformemente distribuidos, G son datos sintéticos con una distrución Gaussiana sobre el espacio y R corresponde a conjuntos de datos reales, que es otro de los objetivos a evaluar en las pruebas. Cabe mencionar que, utilizando esta nomenclatura, todos los experimentos anteriores fueron realizados sobre conjuntos del tipo UxU.

En la tabla 4.5 se muestran los resultados de tiempo total (en segundos) y cantidad de accesos a disco necesitados por BCP para encontrar el par de vecinos más cercanos sobre los escenarios descritos anteriormente. Además, en la tabla se incluye el resultado de BNL (que es independiente de la distribución del conjunto) y los resultados de BCP sobre conjuntos del tipo UxU, obtenidos anteriormente en el primer experimento. Para cada prueba se muestra el promedio del tiempo total y de la cantidad de accesos, utilizando cuatro cantidades de memoria (64, 128, 256 y 512 Buckets), sobre conjuntos de distinto tamaño (200K,400K y 600K), sin intersección.

En base a estos resultados, podemos considerar que nuestro algoritmo tiene el mismo comportamiento sin importar la distribución de los elementos de cada conjunto. De hecho, el mejor caso y el peor caso, son exactamente los encontrados en el experimento 2, además coinciden para todas las distribuciones evaluadas, lo que significaría que la distribución no es un parámetro que pueda afectar significativamente el rendimiento del algoritmo. También podemos apreciar que, si bien BNL es totalmente independiente de la distribución de los conjuntos y se obtuvo el mismo resultado en cada prueba, la ventaja de BCP sigue siendo mayor, al igual que en los resultados del experimento 2. En la Figura 4.8, se muestra BNL y BCP con todas las distribuciones probadas (incluido el escenario UxU), ordenados por el tamaño de los conjuntos, y considerado el número de Buckets como parámetro variable.

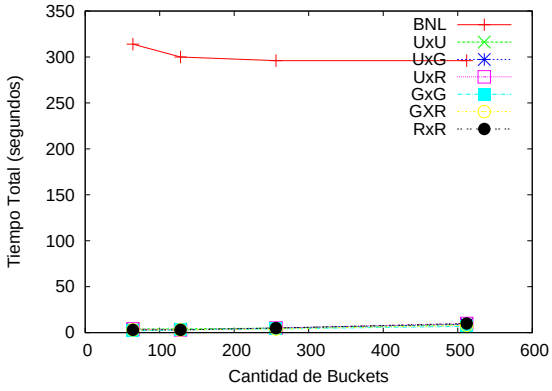
En los gráficos de la Figura 4.8, se aprecia claramente que no hay diferencias significativas en el rendimiento de BCP si se cambia la distribución de los conjuntos, o incluso si se trata de datos reales. Esto es de suma importancia para evaluar el algoritmo, puesto que el esquema de partición definido es bastante sencillo y podría haberse visto afectado por la distribución del conjunto. No obstante, las pruebas demostraron que, al menos para los escenarios evaluados, la distribución total de los elementos no tienen mayor influencia sobre los resultados. Considerando esto, no se estimó necesario repetir la prueba de la distribución con distintos porcentajes de intersección de los conjuntos.

4.4 Análisis de resultados y conclusiones

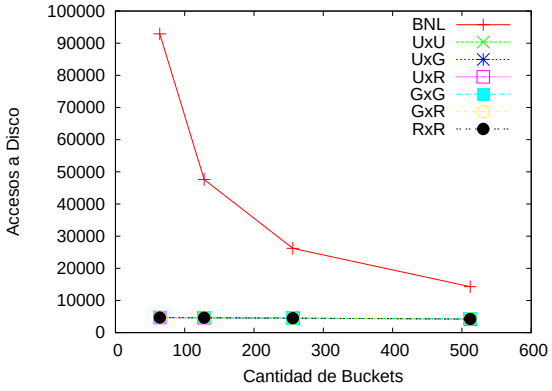
De acuerdo a los experimentos realizados, BCP es una alternativa muy eficiente para resolver la consulta del par de vecinos más cercanos sobre conjuntos sin indexar, cuando no es posible cargar todos los datos en memoria, y cuando no se quiere crear dichos indices para apoyar la búsqueda.

Al comparar nuestra propuesta con las soluciones que usan índices espaciales, como el R-Tree (presentadas en la Sección 3.4.1), BCP encuentra la solución en un 1.8% del tiempo requerido

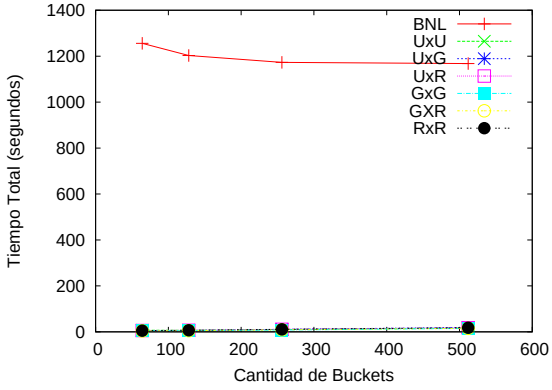
BCP Y BNL sobre conjuntos con distintas distribuciones, sin intersección



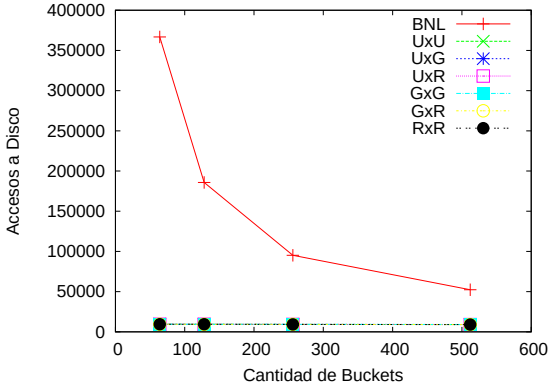
(a) Tiempo para conjuntos de 200K puntos



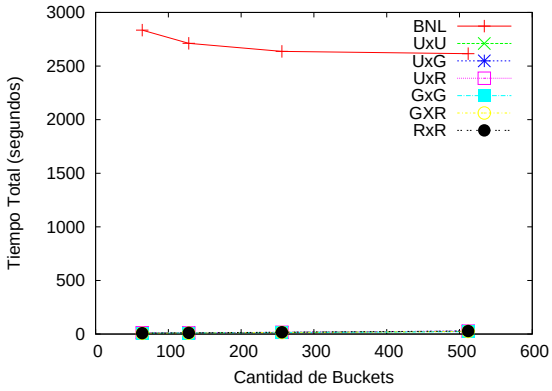
(b) accesos para conjuntos de 200K puntos



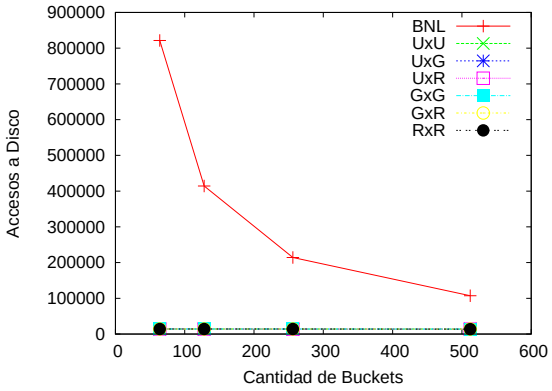
(c) Tiempo para conjuntos de 400K puntos



(d) accesos para conjuntos de 400K puntos



(e) Tiempo para conjuntos de 600K puntos



(f) accesos para conjuntos de 600K puntos

Figura 4.8: Rendimiento de BCP con distintas cantidades de memoria, sobre conjuntos con distinta distribución

solamente para crear los R-Tree, es decir, sin considerar el tiempo adicional requerido para realizar la consulta sobre los conjuntos indexados.

Por otra parte, si comparamos BCP con BNL, nuestra propuesta encuentra la solución a la consulta en mucho menor tiempo y utilizando, en la mayoría de los casos, menos accesos a disco. Además nuestro algoritmo demostró ser más eficiente en recursos, puesto que cuando ambos algoritmos utilizan la misma cantidad de memoria, BCP siempre tiene un mejor desempeño con menos memoria disponible. Incluso, en la medida que el tamaño del conjunto es mayor, el rendimiento de BCP aumenta también, en comparación a BNL, independiente de la distribución de los elementos de cada conjunto en el espacio.

Sin embargo, el rendimiento de BCP se ve afectado por la intersección de las áreas cubiertas por los datos. Dada la naturaleza de BNL, este no se ve afectado por el porcentaje de overlap entre los conjuntos, mientras que BCP aumenta el tiempo que necesita para encontrar la solución, en la medida que también aumenta el porcentaje de intersección.

A continuación se hace un breve análisis del costo del algoritmo, y luego se explica más detalladamente cómo los diferentes parámetros de los experimentos afectan a BCP en encontrar la solución al problema del par de vecinos más cercanos. Finalmente, se establece el peor caso esperado y las mejoras que deben hacerse al algoritmo para obtener un mejor rendimiento.

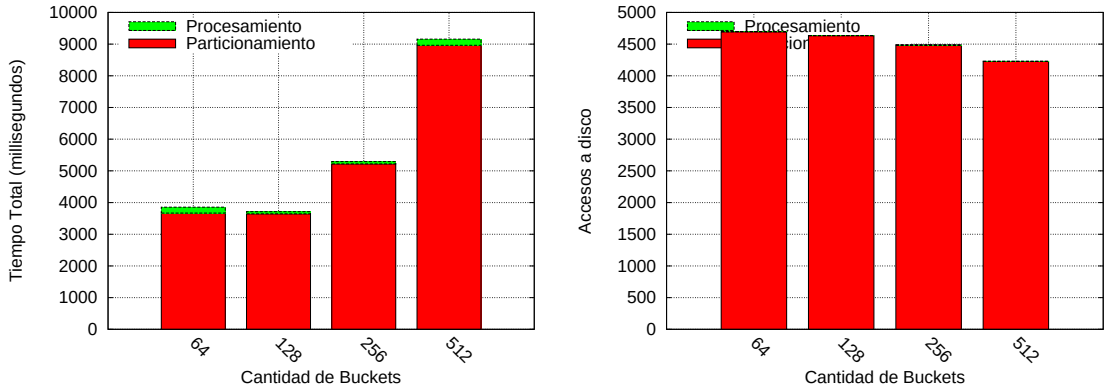
4.4.1 Análisis de costo del algoritmo

En la implementación utilizada para las pruebas, se consideró un tamaño de la página 1024 Bytes. Cada Bucket incluye una página definida por $pBuffer$, un *MBR* definido por dos pares de puntos, y un puntero a un archivo $pFile$, lo que significa que cada Bucket utiliza aproximadamente 1044 Bytes. Cuando asignamos 64 Buckets, se utilizan aproximadamente 66 KB de memoria para procesar cada conjunto. Por otra parte, cuando utilizamos 512 Buckets, se utilizan en promedio 534 KB de memoria, para ambos conjuntos. Considerando que la misma memoria utilizada por los Buckets en la primera etapa para el particionamiento, es utilizada también en la segunda etapa para la búsqueda, se demuestra que BCP es un algoritmo eficiente en recursos, y que puede obtener buenos resultados sin necesidad de consumir demasiada memoria, ni tiempo de procesador.

4.4.2 Sobre la cantidad de memoria utilizada

El primer parámetro evaluado fue el número de Buckets en que se divide cada conjunto, que a su vez corresponde a la cantidad de memoria que utiliza el algoritmo.

Los resultados de los experimentos confirman que la cantidad de accesos a disco para la primera etapa (particionamiento) es casi constante, y no depende de la cantidad de Buckets, sino del tamaño del conjunto. También se pudo apreciar que, cuando los conjuntos no tienen intersección entre sus áreas, una cantidad de memoria mayor, es decir, un número mayor de Buckets, significa una disminución en la cantidad de accesos a disco, pero un aumento en el tiempo total del algoritmo. Esto se puede ver en la Figura 4.9, que grafica los distintos tiempos y accesos a discos de las dos etapas del algoritmo. En esta figura es posible apreciar también que el tiempo de procesamiento se mantiene relativamente constante en la medida que la cantidad de memoria aumenta, mientras que el tiempo de particionamiento aumenta considerablemente en la medida que se utilizan más Bucket, y en general representa la mayor cantidad del tiempo consumido por el algoritmo, para el caso de conjuntos sin intersección.



(a) Tiempo para conjuntos de 200K puntos

(b) accesos para conjuntos de 200K puntos

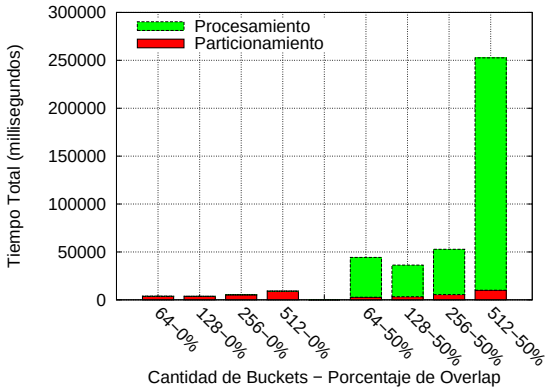
Figura 4.9: Costos de las etapas del algoritmo BCP en conjuntos sin intersección, usando cuatro cantidades de memoria.

Como se explicó anteriormente, esto se debe al esquema de particionamiento del algoritmo. En esta etapa, cada vez que un punto debe ser asignado a un Bucket, mientras más memoria hay disponible, es necesario realizar más comparaciones entre los Buckets para decidir en cuál dejar el punto. Como no existe ningún ordenamiento en los Buckets, y simplemente se crean tomando los *nb* primeros puntos del conjunto de *n* elementos, en el peor de los casos habrá que realizar O($O(nb*n)$ comparaciones para determinar escoger el Bucket. Si bien utilizar poca memoria hace al algoritmo más eficiente, este debería ser una de las mejoras a realizar, de modo que un aumento de memoria, no haga decrecer el rendimiento del algoritmo, sino que pueda particionar el conjunto de una manera más eficiente. Por ejemplo, junto con la creación de los Buckets, podría utilizarse algún tipo de índice espacial para estos, o bien, ordenarlos según algún criterio. De esta forma, cuando se está particionando el conjunto, y se debe escoger el Bucket de destino para un punto, este proceso se haga utilizando algún algoritmo eficiente, como los existentes en los índices espaciales, o bien en las listas ordenadas.

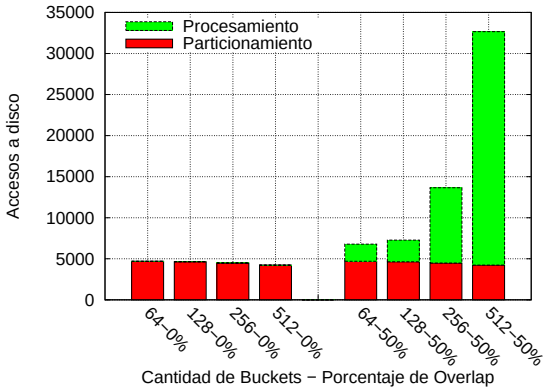
Si bien para todas las cantidades de memoria evaluadas, los resultados fueron favorables para BCP, la variación de los mismos producida por este parámetro, implica la existencia de un valor que optimice los tiempos y accesos del algoritmo. Es probable que este valor óptimo pueda ser calculado en base al resto de los parámetros, como el tamaño de los conjuntos y la intersección de los mismos. Experimentalmente, se puede apreciar que, al menos para el caso de conjuntos sin intersección, los menores tiempos se pueden logran con valores entre los 64 y 128 Buckets. Sin embargo, y debido al alcance de esta tesis, no se hará un análisis más detallado de este fenómeno y solamente se propone como un trabajo futuro.

4.4.3 Sobre la interesección entre los Conjuntos

Uno de los parámetros que más afecta el rendimiento del algoritmo es la intersección de las áreas de los conjuntos. Cuando este parámetro es distinto de 0%, BCP ve afectado su rendimiento aumentando los accesos a disco y el tiempo de ejecución, sobre todo en la segunda etapa del algoritmo.



(a) Tiempo para conjuntos de 200K puntos



(b) accesos para conjuntos de 200K puntos

Figura 4.10: Costos de las etapas del algoritmo BCP en conjuntos con intersección.

A modo de ejemplo, en la Figura 4.10 se grafican los tiempos y accesos de las dos etapas de BCP, en conjuntos con 0% y 50% de intersección, sobre conjuntos de 200K. En dicha Figura podemos apreciar que, si bien la primera etapa del algoritmo (el particionamiento) tiene una cantidad casi constante de accesos a disco y no varía de acuerdo a la intersección, el tiempo utilizado por esta etapa sí se ve afectado por la cantidad de Buckets. Por otra parte, cuando la intersección de los conjuntos es 0%, el tiempo y los accesos de la segunda etapa son marginales en comparación a la primera etapa de particionamiento.

Sin embargo, cuando la intersección de los conjuntos aumenta, también el costo de la segunda etapa. Dentro de este escenario, podemos apreciar que también al aumentar la memoria, aumenta el tiempo y los accesos de nuestra propuesta. Este aumento de los tiempos en la segunda etapa, se explica en parte por un fenómeno similar al ocurrido en el particionamiento. Al exister más Buckets, es necesario realizar más comparaciones entre ellos para tomar ciertas decisiones. Además, como este procesamiento se apoya en una serie de métricas para filtrar elementos y consultas innecesarias, estas métricas no resultan ser tan efectivas cuando hay interesección entre los Buckets.

4.4.4 Sobre la distribución de los Conjuntos

De acuerdo a los experimentos realizados sobre conjuntos con distintas distribuciones y además con conjuntos de datos reales, el algoritmo BCP no se ve afectado por este parámetro, y su comportamiento es idéntico para cualquier distribución de los datos en el conjunto, aún con el sencillo esquema de particionamiento definido.

4.4.5 Peores casos

Los peores rendimientos se dieron para los conjuntos más pequeños, utilizando mayor cantidad de memoria (mayor número de Buckets). Tal como se explicó anteriormente, esto se debe al esquema de particionamiento utilizado en la primera etapa del algoritmo. Si bien la cantidad de accesos a disco es casi constante en esta etapa, y depende del tamaño del conjunto, la

cantidad de comparaciones que es necesario realizar para determinar en qué Bucket se debe guardar un punto varía con este parámetro. Como el esquema de particionamiento no ordena los Buckets, la búsqueda del Bucket destino es lineal, y en el peor de los casos, necesitará $O(nb+n)$ comparaciones para encontrarlo, donde *nb* representa la cantidad de Buckets y *n* la cantidad de puntos del conjunto.

Cuando no existe intersección entre los conjuntos, y en la medida que el tamaño del conjunto es menor, el tiempo de la etapa de particionamiento es mucho mayor que el tiempo utilizado por la segunda etapa de procesamiento y, por lo tanto, el algoritmo ve mermado su rendimiento, en la medida que la primera etapa aumenta su tiempo de ejecución, como ocurre con una cantidad de memoria demasiado grande en comparación a la cantidad de elementos del conjunto,

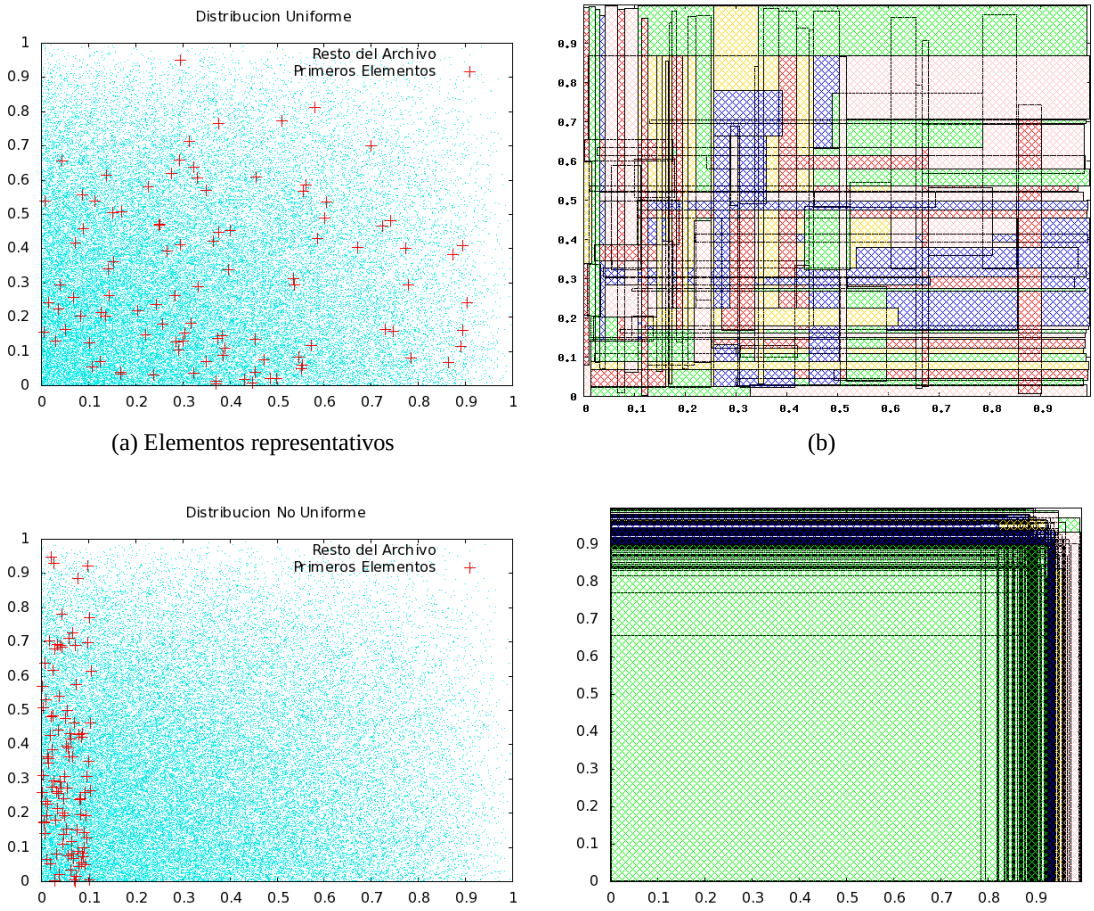
El segundo parámetro que perjudica el rendimiento de BCP es la intersección de los conjuntos. En la medida que se aumenta la intersección de las áreas de los conjuntos, existirán más pares de Buckets donde las métricas entre sus *MBR* no servirán de filtro para descartar elementos que deben ser procesados.

Si bien el problema de la intersección de los Buckets se presenta cuando el porcentaje de overlap es mayor que 0, un análisis más detallado del algoritmo, nos lleva a esperar que una situación similar ocurra en cualquier otro escenario, producto del particionamiento realizado por el algoritmo. Pese a que las pruebas demostraron que la distribución de los elementos del conjunto en el espacio no influye el rendimiento del algoritmo, el actual esquema de particionamiento, que consiste en tomar los *nb* primeros elementos del conjunto, podría generar una mala partición de dicho conjunto, cuando estos *nb* primeros elementos no sean representativos del resto de los elementos. En la Figura 4.11 se ilustra este ejemplo, considerando que la Figura 4.11 a) existe un conjunto cuyos primeros elementos se encuentran distribuidos de manera relativamente uniforme al resto de aquellos restantes en el conjunto, lo que da como resultado una partición similar a la mostrada en la Figura 4.11 b). Por otro lado, en la Figura 4.11 c), se muestra un conjunto cuyos elementos no son representativos en relación al resto, lo que genera una partición con mucha intersección entre los Buckets de dicho conjunto, lo que a su vez, disminuye la efectividad de las métrica y aumenta el tiempo total del algoritmo.

4.4.6 Mejoras sobre BCP

Si bien con esta primera versión del algoritmo se obtuvieron buenos resultados para encontrar el par de vecinos más cercanos sobre conjuntos no indexados, sin necesidad de crear índices espaciales, el análisis del peor caso nos indica que el esquema de particionamiento es uno de los principales problemas del del algoritmo. De hecho, en la etapa de particionamiento no hay ningún algoritmo, simplemente se toman los *nb* primeros elementos. Esto no permite sacar provecho de la cantidad de memoria asignada, ni garantiza crear un Set de Buckets que contengan los puntos distribuidos de la manera más uniformemente posible, disminuyendo la intersección dentro del mismo Set.

Para mejorar estos escenarios desfavorables, en el siguiente capítulo se presenta una nueva propuesta, explicando las mejoras realizadas en la etapa de particionamiento.



(a) Elementos representativos (b)

(c) Elementos no representativos (d)

Figura 4.11: Ejemplo de particiones realizadas por BCP

		BNL	BCP				
Tamaño	Buckets		0%	25%	50%	75%	100%
200K	64	314	4	32	44	59	66
200K	128	300	4	24	36	50	47
200K	256	296	5	36	53	70	63
200K	512	296	9	145	253	270	284
PROMEDIO		**302**	**6**	**59**	**97**	**112**	**115**
400K	64	1.256	7	79	139	185	221
400K	128	1.203	7	40	84	115	157
400K	256	1.173	10	46	88	116	171
400K	512	1.168	17	132	283	391	479
PROMEDIO		**1.200**	**10**	**74**	**149**	**202**	**257**
600K	64	2.835	10	175	334	394	511
600K	128	2.712	11	157	232	294	370
600K	256	2.637	15	129	213	286	321
600K	512	2.615	25	172	333	454	639
PROMEDIO		**2.700**	**15**	**158**	**278**	**357**	**460**

a) Tiempo (segundos)

		BNL	BCP				
Tamaño	Buckets		0%	25%	50%	75%	100%
200K	64	92.896	4.698	6.228	6.787	7.534	7.966
200K	128	47.638	4.635	6.255	7.271	8.425	8.120
200K	256	26.200	4.486	10.494	13.648	17.060	16.137
200K	512	14.290	4.229	22.989	32.665	41.763	42.436
PROMEDIO		**45.256**	**4.512**	**11.492**	**15.093**	**18.696**	**18.665**
400K	64	366.749	9.467	11.912	13.821	15.519	16.562
400K	128	185.755	9.387	10.320	11.343	12.428	13.447
400K	256	95.258	9.265	12.453	15.214	18.114	21.706
400K	512	52.391	9.021	20.041	32.865	42.619	49.811
PROMEDIO		**175.038**	**9.285**	**13.682**	**18.311**	**22.170**	**25.382**
600K	64	821.558	14.256	19.133	23.182	25.152	28.383
600K	128	414.350	14.172	16.491	17.575	18.946	20.203
600K	256	214.318	14.010	18.677	21.847	24.938	26.515
600K	512	107.158	13.790	24.608	36.070	44.801	57.325
PROMEDIO		**389.346**	**14.057**	**19.727**	**24.669**	**28.459**	**33.107**

b) accesos a disco

Tabla 4.3: Resultados de BNL y BCP sobre conjuntos con distinto porcentaje de intersección

		Particionamiento		**Búsqueda**	
Overlap	Buckets	Tiempo (ms)	accesos a disco	Tiempo (ms)	accesos a disco
0%	64	3668	4694	186	4
0%	128	3646	4633	66	2
0%	256	5221	4484	71	2
0%	512	8963	4227	195	2
25%	64	2211	4694	29834	1534
25%	128	3123	4633	21214	1622
25%	256	4982	4480	31298	6014
25%	512	9139	4227	136188	18762
50%	64	2686	4694	41601	2093
50%	128	3320	4633	32975	2638
50%	256	5540	4480	47211	9168
50%	512	10093	4229	242531	28436
75%	64	2407	4694	56326	2840
75%	128	3242	4633	47112	3792
75%	256	5234	4484	64276	12576
75%	512	9042	4227	321233	40536
100%	64	2530	4700	63518	3266
100%	128	3613	4626	43219	3494
100%	256	5515	4489	57727	11648
100%	512	9348	4234	274956	38202

Tabla 4.4: Detalle de tiempo y accesos para las etapas de BCP sobre conjuntos con distinta intersección

		BNL	BCP					
Tamaño	Buckets		UxU	UxG	UxR	GxG	GxR	RxR
200	64	314	4	3	4	2	3	3
200	128	300	4	3	3	3	3	3
200	256	296	5	5	5	4	4	5
200	512	296	9	9	10	7	8	10
PROMEDIO		302	6	5	6	4	5	5
400	64	1.256	7	6	6	5	4	5
400	128	1.203	7	6	7	5	5	7
400	256	1.173	10	11	11	8	9	11
400	512	1.168	17	19	18	15	15	18
PROMEDIO		1.200	10	11	11	8	8	10
600	64	2.835	10	10	12	7	7	8
600	128	2.712	11	10	10	8	8	11
600	256	2.637	15	16	16	13	15	17
600	512	2.615	25	28	27	22	24	28
PROMEDIO		2.700	15	16	16	13	14	16

a) Tiempo (segundos)

		BNL	BCP					
Tamaño	Buckets		UxU	UxG	UxR	GxG	GxR	RxR
200	64	92.896	4.698	4.694	4.704	4.682	4.690	4.703
200	128	47.638	4.635	4.628	4.631	4.630	4.635	4.633
200	256	26.200	4.486	4.457	4.503	4.416	4.462	4.502
200	512	14.290	4.229	4.292	4.249	4.346	4.305	4.254
PROMEDIO		45.256	4.512	4.518	4.522	4.519	4.523	4.523
400	64	366.749	9.467	9.465	9.470	9.478	9.460	9.471
400	128	185.755	9.387	9.378	9.388	9.374	9.383	9.393
400	256	95.258	9.265	9.291	9.269	9.310	9.288	9.269
400	512	52.391	9.021	9.097	9.009	9.160	9.072	9.006
PROMEDIO		175.038	9.285	9.308	9.284	9.331	9.301	9.285
600	64	821.558	14.256	14.264	14.308	14.246	14.243	14.244
600	128	414.350	14.172	14.158	14.172	14.140	14.151	14.180
600	256	214.318	14.010	14.033	14.032	14.056	14.055	14.047
600	512	107.158	13.790	13.827	13.790	13.866	13.833	13.788
PROMEDIO		389.346	14.057	14.071	14.076	14.077	14.071	14.065

b) accesos a disco

Tabla 4.5: Resultados de BNL y BCP sobre conjuntos con distinta distribución sin intersección.

Capítulo 5

Propuesta BCP-VA

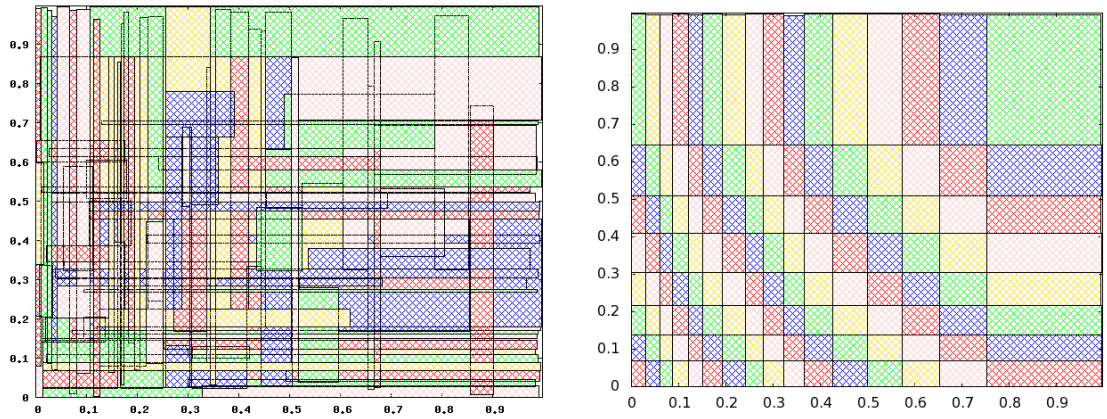
5.1 Introducción

Las pruebas a las que se sometió BCP demostraron que el algoritmo supera a BNL en todos los escenarios evaluados. No obstante, los resultados hicieron notar que mientras BNL mejora su rendimiento en la medida que hay más memoria disponible, BCP empeora su desempeño cuando hay más Buckets. Los experimentos también mostraron que, en la medida que la intersección entre los conjuntos aumenta, también lo hace el tiempo que necesita el algoritmo para encontrar la solución, y si bien siempre es menor que BNL, la cantidad de accesos a disco de la propuesta supera a BNL en algunos casos.

Basados en estos resultados, y luego de analizar en detalle el algoritmo BCP, concluimos que los casos desfavorables son causados por el sencillo esquema de particionamiento implementado en BCP. Idealmente todos los objetos del conjunto deberían quedar equitativamente distribuidos en los Buckets, con la menor intersección posible entre ellos. Sin embargo, con el algoritmo implementado en la etapa de particionamiento de BCP, esto no se logra en todos los escenarios.

Para solucionar el problema del particionamiento, y como esto afecta el rendimiento de BCP, se implementó una nueva versión, aplicando un algoritmo más eficiente al momento de particionar el conjunto. Este nuevo algoritmo, llamado BCP-VA (Bucket Closest Pair -Vector Approximation), también usa la estructura Bucket, y al igual que BCP tiene un esquema de dos partes, particionamiento y procesamiento. Mientras que el procesamiento se mantuvo igual que en BCP, para el particionamiento se implementó un algoritmo basado en el esquema del “Vector-Approximation File ” [BW97], utilizado para la búsqueda en espacios multimedimensionales, y que se explica con más detalle en la Sección 5.2.

Esta nueva propuesta BCP-VA fue sometido a las mismas pruebas que BCP de manera de comprobar el rendimiento de este nuevo algoritmo y demostrar de manera experimental, si efectivamente el nuevo esquema de particionamiento ofrece mejoras al momento de realizar la consulta del par más cercano sobre conjuntos no indexados. Los resultados de estos experimentos son presentados en la Sección 5.3 de este capítulo, para luego terminar con las conclusiones sobre esta nueva propuesta.



(a) Partición realizada por BCP

(b) Partición realizada por BCP-VA

Figura 5.1: Comparación entre el esquema de partición de BCP y BCP-VA

5.2 Algoritmo de Particionamiento de BCP-VA

La idea central de BCP-VA es dividir el espacio que abarca el conjunto, y no los objetos del mismo. El espacio se particiona en una serie de vectores ordenados, que a su vez, representan los Buckets en que se divide el conjunto. De esta forma, se logra una distribución más uniforme de los objetos en los distintos Buckets, tal como se aprecia en la Figura 5.1, donde se compara el particionamiento hecho por BCP con el de BCP-VA, sobre un conjunto de 400K con distribución uniforme y utilizando 128 Buckets. Este enfoque de particionamiento del espacio en vectores, es ampliamente utilizado en las búsquedas sobre espacios métricos, como se presenta en [BW97]. En dicho trabajo, los autores demuestran la efectividad de VA sobre conjuntos de alta dimensionalidad. Además explican cómo el uso de este tipo de particionamiento evita la "maldición de la dimensionalidad", fenónemo adverso para los algoritmos, presente cuando la cantidad de dimensiones es demasiado grande.

Nuestro algortimo BCP-VA necesita, a diferencia de BCP, dos parámetros adicionales para trabajar por cada conjunto S. El primer parámetro es el *MBR* que contiene a todos los puntos de S es decir, el *MBR* que define los límites del conjunto. El otro parámetro, *samples*, indica qué cantidad de puntos se tomarán como muestra del conjunto S, para realizar el particionamiento del espacio. De esta forma, en vez de tomar los *nb* primeros objetos para crear los *nb* Buckets, BCP-VA utiliza los *samples* primeros puntos a modo de muestras (o samples).

La idea central del "Vector-Approximation" es particionar el espacio d-dimensional en 2^b particiones, de modo que cada partición tenga asociado un identificador de b bits. Los b bits se reparten en $b_1 + b_2 + ... + b_d$ $(= b)$ bits, donde b_i bits se asocian a la i-ésima dimensión, para $i = 1...d$. El número de bits b_i para establecer las particiones en cada dimensión i se determina en función del número total de bits (b) y del número de dimensiones d como sigue:

$$b_i = \left\lfloor \frac{b}{d} \right\rfloor + \begin{cases} 1 & i \leq b \mod d \\ 0 & \text{en caso contrario} \end{cases}$$

Además, el número de bits en cada dimensión se utiliza para determinar los puntos de partición y consecuentemente las regiones dentro de cada dimensión. En particular existen 2^{b_i} regiones

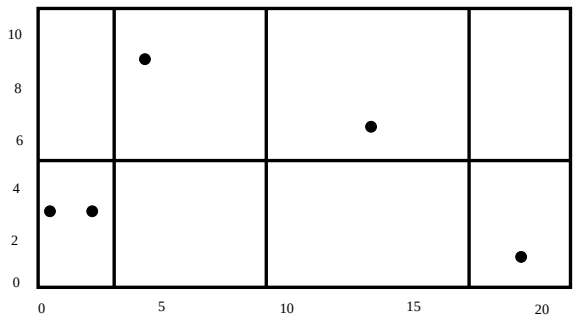


Figura 5.2: División de un espacio de dos dimensiones, de acuerdo al algoritmo de BCP-VA

dentro de la dimensión *i*, agrupadas en un matriz V y definidas por $2^{b_i}+1$ puntos de partición. De esta forma, para obtener *nb* Buckets, debemos hacer $b = \log_2 nb$.

En cada dimensión *i*, los 2^{b_i} puntos de partición $V_i[0], V_i[1], \ldots, V_i[2^{b_i}]$ se establecen de tal manera que cada una de las regiones tenga aproximadamente la misma cantidad de puntos. Para esto, en nuestro algoritmo consideraremos una muestra del conjunto igual a *samples* puntos. En la Figura 5.2 se muestra un ejemplo de este algortimo, para un conjunto en dos dimensiones (d=2), con 5 puntos de *samples*, 8 Buckets (que implica b=3), donde b_1=1 y b_2=2.

Una vez que el espacio que contiene a los puntos ha sido dividido, se procede a repartir el resto de los puntos del conjunto dentro de la matriz V, que a su vez representa a los *nb* Buckets.

A diferencia del algoritmo anterior, en esta etapa de particionamiento, no se busca el Bucket que debe aumentar en menor cantidad su área para contener al punto, sino que simplemente se busca dentro de cuál de los vectores, el elemento está contenido. Como la lista de vectores se encuentra ordenada según cada eje, y además, cada punto queda contenido solamente dentro de uno los Buckets, la búsqueda es mucho más eficiente. Este nuevo esquema de partición se muestra en el Alg. 5.1.

```
 1: Particionar(S, nb,MBR,samples)
 2: Sea M el conjunto formado por los samples primeros puntos de S.
 3: Calcular b = Entero superior de log2 nb
 4: V=ParticionarEspacio(MBR,b,M)
 5: Sea B el conjunto de Buckets, representado por V
 6: for cada punto p restante en S do
 7:     Sea i el indice del Vector que contiene al punto p
 8:     i = VectorContienePunto(V,p)
 9:     AgregarPuntoBucket(B_i,p)
10: end for
11: for cada Bucket B_i, 1 ≤ i ≤ nb do
12:     Guardar los objetos de B_i.pBuffer en el disco usando B_i.pFile
13: end for
14: return B
```

Alg. 5.1: Etapa 1 del algoritmo BCP-VA: Particionamiento del conjunto en Sets de Buckets utilizando vectores

1: **ParticionarEspacio**(MBR,b,M)
2: Sea d las dimensiones del conjunto M
3: *El valor de b se divide en d partes.*
4: **for** $1 \leq i \leq d$ **do**
5: $b_i = \lfloor \frac{b}{d} \rfloor + \begin{cases} 1 & i \leq b \mod d \\ 0 & \text{en caso contrario} \end{cases}$
6: $n_i = 2^{b_i}$
7: n_i *es la cantidad de puntos de partición en la dimensión* i*, aparte del límite inferior.*
8: **end for**
9: Sea V una matriz con d filas en donde cada fila, de tamaño $n_i, 1 \leq i \leq d$, almacena los puntos de partición de la dimensión i
10: **for** i = 1 to d **do**
11: Sort(M,i) {Ordena los puntos del arreglo M según la dimensión i}
12: $V[i][0] =$ límite inferior del MBR en la dimensión i
13: $V[i][n_i] =$ límite superior del MBR en la dimensión i
14: Sea $np = \left\lfloor \frac{size\ of\ M}{n_i} \right\rfloor$
15: $k = 1$ {Subíndice de los sucesivos puntos de partición}
16: $j = np$ {Subíndices de los puntos de M que se van tomando para obtener los puntos de partición}
17: **while** $j \leq$ (size of M) $- np$ **do**
18: $V[i][k] = M_j.coordenada[i]$ {El k-ésimo punto de partición según la coordenada i se toma como el valor de la coordenada i del j-ésimo punto de M}
19: $k = k+1$
20: $j = j+np$
21: **end while**
22: **end for**
23: **return** V

Alg. 5.2: Algoritmo para particionar el espacio

1: **VectorContienePunto**(V,p)
2: Sea x la dimensión del conjunto
3: **for** cada dimensión i, $1 \leq i \leq x$ **do**
4: Buscar elemento E_i de V_i que contiene a p según p_i.
5: **end for**
6: Sea i el Bucket asociado al elemento $V(E)$.
7: **return** i

Alg. 5.3: Algoritmo para encontrar el vector que contiene un punto.

Como se puede apreciar en Alg. 5.1, el procedimiento de particiomiento no es complejo, y si bien es necesario trabajar con una mayor cantidad de puntos en comparación al algoritmo de particionamiento implementado en BCP, este aumento en los accesos traerá mejoras en la siguiente etapa del algoritmo. Para determinar el parámetro *samples*, es decir, cuántos son los puntos con los que se trabajará, en [BW97] plantean que tomando una muestra no mayor a un 5% del conjunto es posible obtener muy buenos resultados para el particionamiento del espacio.

5.3 Experimentación de BCP-VA

En la siguiente sección, se describen una serie de experimentos realizados con la nueva propuesta BCP-VA, para poder comparar su rendimiento frente a la propuesta anterior BCP. Los objetivos planteados para estos experimentos son exactamente los mismos evaluados al realizar las pruebas con BCP, definidos en la Sección 4.3. No obstante, se agregaron dos experimentos adicionales, primero para evaluar las diferencias de rendimiento entre los algoritmos de particionamiento, y luego para determinar el valor de la muestra que es necesario tomar en cada conjunto.

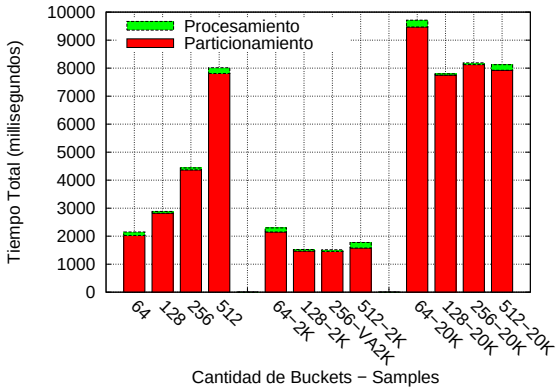
- Comparar el tiempo para particionar de BCP y BCP-VA.
- Evaluar el rendimiento de BCP-VA de acuerdo al tamaño de la muestra.
- Evaluar el rendimiento de BCP-VA utilizando distintas cantidades de memoria.
- Evaluar el rendimiento de BCP-VA sobre conjuntos de distinto tamaño.
- Evaluar el rendimiento de BCP-VA en conjuntos de distintas distribuciones.
- Evaluar el rendimiento de BCP-VA en conjuntos con distintos porcentajes de intersección.
- Medir el rendimiento de BCP-VA sobre conjuntos de datos reales.

5.3.1 Experimento 1: Tiempo de Particionamiento de BCP y BCP-VA

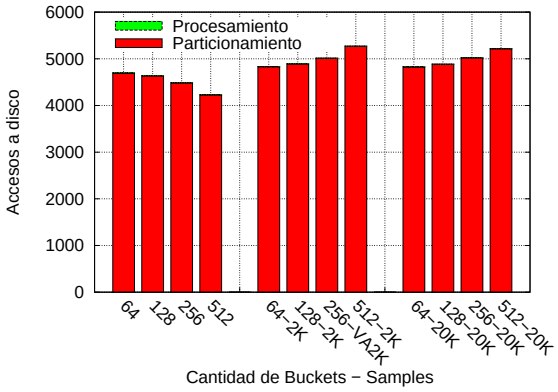
Como la principal diferencia entre BCP y BCP-VA se encuentra en el algoritmo de particionamiento, la primera prueba fue comparar el rendimiento de ambos algoritmos por etapas. Para esto, se ejecutó la consulta del par de vecinos más cercanos sobre conjuntos uniformes sin intersección, de tamaños 200K, 400K y 600K, y con cuatro cantidades de memoria distinta (64, 128, 256 y 512 Buckets). Además, como otro de los objetivos era evaluar el tamaño de la muestra, BCP-VA se ejecutó considerando muestras de 2.000 y 20.000 samples, equivalentes a un 1% y10% del conjunto de 200K, a un 0.5% y 5% del conjunto de 400K y a un 0.3% y 3% del conjunto de 600K.

En la Tabla 5.1 se detallan los accesos por etapa, mientras que la Tabla 5.2 se muestran los tiempos de ejecución. En la Figura 5.3, se grafican el tiempo total y los accesos de las distintas etapas del algortimo, donde los primeros cuatro bloques de cada figura, (que representan las cuatro cantidades de memoria utilizada) corresponden a los resultados del algoritmo BCP, los siguientes cuatro al algoritmo BCP-VA tomando 2.000 puntos de muestra, y los últimos cuatro al algoritmo BCP-VA tomando 20.000 puntos de muestra, ordenados de acuerdo al tamaño de los conjuntos.

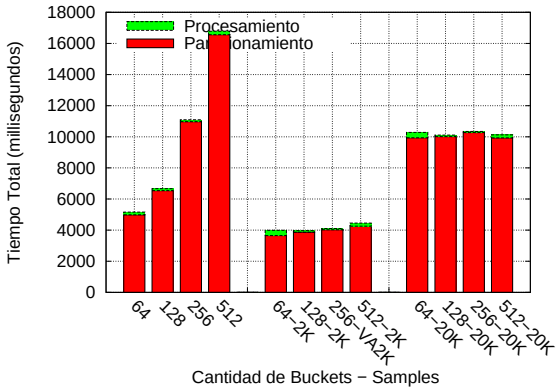
Lo primero que observamos es que, la cantidad de accesos a disco aumenta levemente en la etapa de particionamiento de BCP-VA en comparación con BCP. Si bien el peor caso de este



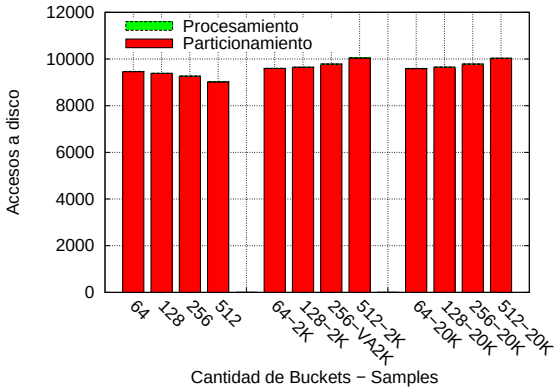
(a) Tiempo para conjuntos de 200K puntos



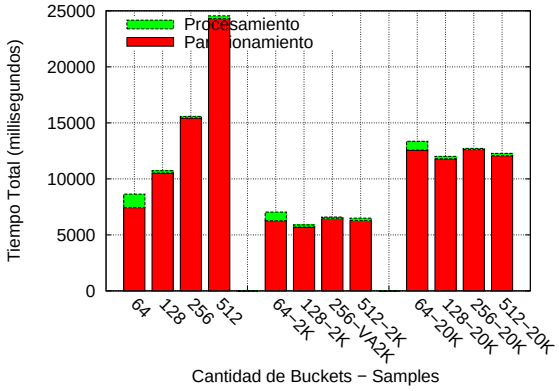
(b) accesos para conjuntos de 200K puntos



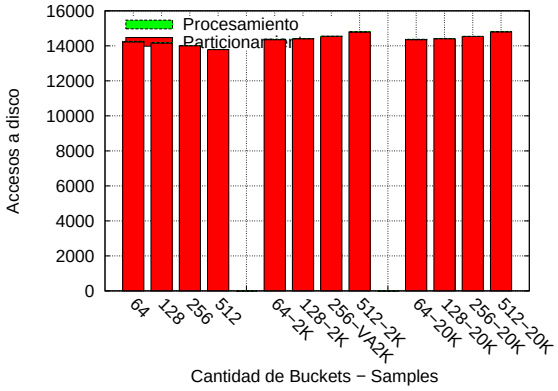
(c) Tiempo para conjuntos de 400K puntos



(d) accesos para conjuntos de 400K puntos



(e) Tiempo para conjuntos de 600K puntos



(f) accesos para conjuntos de 600K puntos

Figura 5.3: Costos de las etapas del algoritmo BCP y BCP-VA con distintos samples

aumento es un 25% más de accesos a disco, en promedio, el nuevo algoritmo de particionamiento requiere solamente de un 7% más de accesos. Además, podemos apreciar que esta variación de los accesos no se ve afectado por el tamaño de la muestra. Por otro lado, al igual que en el experimento realizado sobre BCP con conjuntos sin intersección, la cantidad de accesos requeridos por la segunda etapa, es muy inferior a las realizadas en la etapa de particionamiento.

Respecto al tiempo del algoritmo, podemos apreciar que con el tamaño de muestra más grande, equivalentes a un 10% del conjunto (20.000 samples para 200K) BCP-VA no logra mejorar el tiempo de BCP, obteniendo como peor caso, una diferencia de 125 milisegundos más que BCP, equivalentes a una merma del 6% del tiempo. Sin embargo, con el resto de los tamaño de muestra (5%,3%,1%,0.5% y 0.3%) BCP-VA mejora considerablemente el desempeño sobre BCP. Esta mejora en el tiempo del algoritmo, también se ve incrementada en la medida que aumenta la memoria disponible. De hecho, es en el escenario con 512 Buckets, donde BCP-VA logra su mejor tiempo, necesitando 6231 milisegundos menos que BCP, es decir, una mejora del 80%. Este problema, que se presenta con la cantidad de samples más grande, se explica considerando que un aumento en la muestra, implica que hay más elementos para ordenar, al momento de crear los vectores en la etapa de particionamiento y, por lo tanto, el tiempo requerido en esta etapa es mayor.

Si analizamos en conjunto ambas medidas de rendimiento (tiempo y accesos a disco), podemos apreciar que, si bien el nuevo algoritmo requiere una cantidad mayor de accesos a disco en la etapa de particionamiento (alrededor de un 25% para el peor caso), en promedio solo requiere de un 7% adicional. Este aumento en el acceso a disco es esperado, puesto que el nuevo algoritmo de particionamiento de los conjuntos, es más complejo que el implementado en BCP, y por lo tanto necesita accesos adicionales. Aún así, este aumento es marginal en comparación a la mejora de tiempo que ofrece BCP-VA sobre BCP, puesto que para tamaños de muestra menores al 10% del conjunto, BCP-VA supera notablemente a BCP.

Esta mejora en el tiempo, a expensas de accesos adicionales, se debe a la mejora en el tiempo requerido por el nuevo algoritmo de particionamiento, para ubicar los puntos del conjunto en los distintos Buckets. Con el algoritmo implementado en BCP, para cada punto que era necesario ubicar, se debía revisar todos los Buckets existentes, para luego determinar cuál de los Buckets aumentaba en menor cantidad su *MBR* para contener al punto. Sin embargo, en este nuevo algoritmo el conjunto está dividido en una serie de vectores ordenados, sin intersección entre ellos, lo que significa que un punto estará contenido solamente dentro de un Bucket, limitado por los vectores que están asociados a dicho Bucket. Por lo tanto, la búsqueda del Bucket de destino puede ser tan buena como cualquier algoritmo de búsqueda sobre conjuntos ordenados.

En conclusión, podemos decir que para conjuntos de distribución uniforme sin intersección, tomando una muestra no mayor a un 5% del conjunto, la nueva propuesta BCP-VA ofrece una mejora considerable sobre BCP. Si bien la cantidad de accesos a disco durante la primera etapa aumenta levemente (alrededor de un 7%), este aumento produce una mejora en el tiempo total del algoritmo cercana al 47%. Además, el nuevo algoritmo de particionamiento es capaz de disminuir el tiempo total de particionamiento en la medida que dispone de más memoria para trabajar, lo que significa otra mejora sobre BCP.

		BCP		**BCP-VA** (2.000 Samp)		**BCP-VA** (20.000 Samp)	
Tamaño	Buckets	Part.	Proces.	Part.	Proces.	Part.	Proces.
200K	64	4.694	4	4.828	6	4.824	6
200K	128	4.633	2	4.890	2	4.883	2
200K	256	4.484	2	5.012	2	5.020	2
200K	512	4.227	2	5.269	2	5.214	2
PROMEDIO		**4.510**	**3**	**5.000**	**3**	**4.985**	**3**
400K	64	9.459	8	9.593	12	9.586	12
400K	128	9.383	4	9.644	2	9.647	2
400K	256	9.263	2	9.779	2	9.778	2
400K	512	9.019	2	10.040	2	10.029	2
PROMEDIO		**9.281**	**4**	**9.764**	**5**	**9.760**	**5**
600K	64	14.226	30	14.353	20	14.353	20
600K	128	14.168	4	14.408	2	14.410	2
600K	256	14.008	2	14.544	2	14.536	2
600K	512	13.788	2	14.793	2	14.804	2
PROMEDIO		**14.048**	**10**	**14.525**	**7**	**14.526**	**7**

Tabla 5.1: accesos a disco por etapas de BCP y BCP-VA

		BCP		**BCP-VA** (2.000 Samp)		**BCP-VA** (20.000 Samp)	
Tamaño	Buckets	Part.	Proces.	Part.	Proces.	Part.	Proces.
200K	64	2025	124	2150	147	9469	244
200K	128	2821	61	1470	52	7748	50
200K	256	4368	69	1456	58	8127	61
200K	512	7807	207	1576	198	7924	200
PROMEDIO		**4.255**	**115**	**1.663**	**114**	**8.317**	**139**
400K	64	4984	173	3651	339	9938	344
400K	128	6542	127	3870	97	10013	95
400K	256	10974	119	4028	77	10271	73
400K	512	16564	223	4254	199	9931	192
PROMEDIO		**9.766**	**161**	**3.951**	**178**	**10.038**	**176**
600K	64	7436	1202	6266	774	12566	788
600K	128	10522	205	5699	192	11793	215
600K	256	15427	150	6487	105	12615	110
600K	512	24308	246	6290	203	12065	211
PROMEDIO		**14.423**	**451**	**6.186**	**319**	**12.260**	**331**

Tabla 5.2: Tiempo de ejecución (milisegundos) por etapas de BCP y BCP-VA

5.3.2 Experimento 2: BCP-VA y BCP en conjuntos uniformes sin intersección

Considerando que el experimento anterior demostró que un tamaño de muestra pequeño (alrededor de 1% del total de elementos) dió los mejores resultados para conjuntos con distribución uniforme sin intersección, en esta prueba se analizó el rendimiento de BCP-VA utilizando 2000 samples, comparándolo con BCP sobre conjuntos de cardinalidad 200K, 400K y 600K, sin intersección y utilizando cuatro cantidades de memoria. Estos resultados se muestran en la Tabla 5.3, que es un resumen de la tabla 5.1 y de la tabla 5.2, de manera de tener una mirada global a los resultados, y no según las etapas.

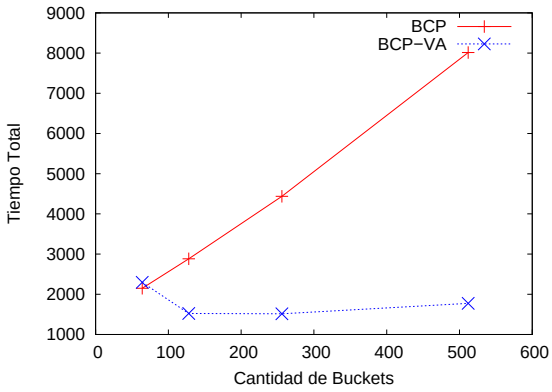
		BCP		**BCP-VA**	
Tamaño	Buckets	Tiempo (ms)	accesos	Tiempo (ms)	accesos
200	64	2.149	4.698	2.297	4.834
200	128	2.882	4.635	1.522	4.892
200	256	4.437	4.486	1.514	5.014
200	512	8.014	4.229	1.774	5.271
PROMEDIO		4.371	4.512	1.777	5.003
400	64	5.157	9.467	3.990	9.605
400	128	6.669	9.387	3.967	9.646
400	256	11.093	9.265	4.105	9.781
400	512	16.787	9.021	4.453	10.042
PROMEDIO		9.927	9.285	4.129	9.769
600	64	8.638	14.256	7.040	14.373
600	128	10.727	14.172	5.891	14.410
600	256	15.577	14.010	6.592	14.546
600	512	24.554	13.790	6.493	14.795
PROMEDIO		14.874	14.057	6.504	14.531

Tabla 5.3: Rendimiento de BCP y BCP-VA sobre conjuntos con distribución uniforme, sin intersección

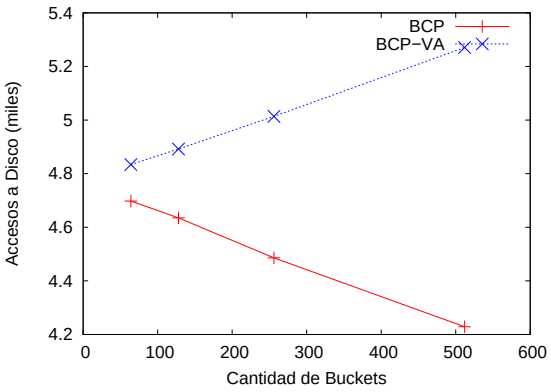
De los resultados podemos apreciar que, si bien el número de accesos totales del algoritmo aumenta en promedio un 7% (siendo el peor caso un aumento del 25% y el mejor tan solo de un 1%), el tiempo total del algoritmo mejoró cerca de un 50% en promedio.

Además, tal como lo hace BCP, esta segunda propuesta mejora su desempeño en la medida que aumenta el tamaño de los conjuntos, sin embargo, BCP-VA mejora su tiempo de respuesta en la medida que tiene más memoria para trabajar, a diferencia de lo ocurrido con BCP. Esto se aprecia más claramente en las Figura 5.4 donde se grafican el tiempo y los accesos de BCP y BCP-VA, con distintas cantidades de memoria, ordenados de acuerdo al tamaño de los conjuntos.

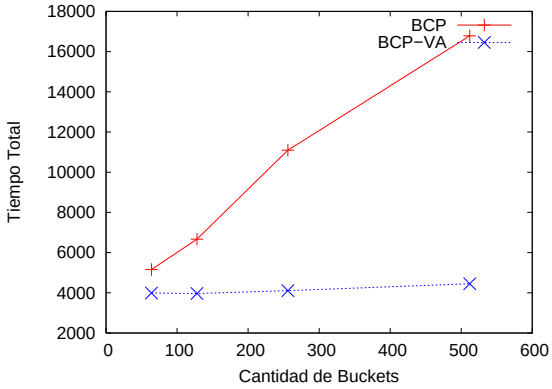
Si bien en las gráficas vemos que hay una diferencia desfavorable de BCP-VA sobre BCP en lo que respecta a accesos a disco, esta diferencia es compensada con una mejora considerable en el tiempo de ejecución, que era uno los resultados esperados al aplicar un esquema de particionamiento más complejo que el propuesto en BCP. De todas formas, la cantidad de accesos realizados por BCP-VA sigue siendo muy inferior a los realizados por BNL, el testigo utilizado en nuestra primera propuesta.



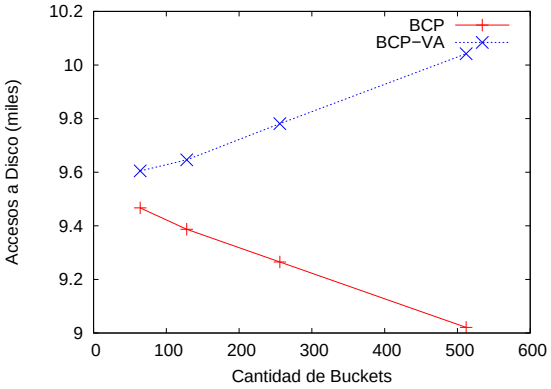
(a) Tiempo (ms) para conjuntos de 200K puntos



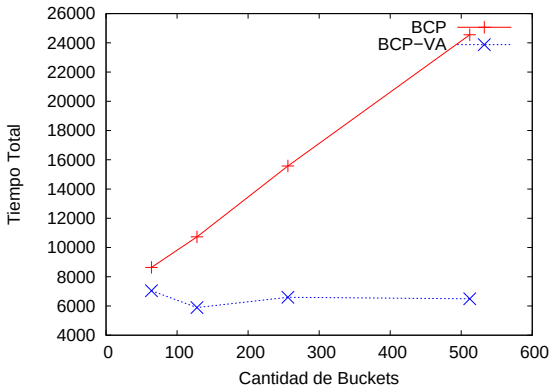
(b) accesos para conjuntos de 200K puntos



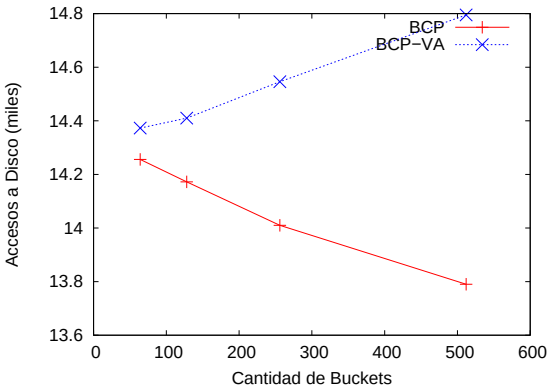
(c) Tiempo (ms) para conjuntos de 400K puntos



(d) accesos para conjuntos de 400K puntos



(e) Tiempo (ms) para conjuntos de 600K puntos



(f) accesos para conjuntos de 600K puntos

Figura 5.4: Tiempo y accesos a disco de BCP y BCP-VA sobre conjuntos con distribución uniforme, sin intersección

5.3.3 Experimento 3: BCP-VA sobre conjuntos con distinta área de interesección

Uno de los objetivos de BCP-VA fue mejorar el particionamiento de BCP, esperando aumentar el rendimiento del algoritmo cuando los conjuntos presentaban intersección entre sus áreas. Para evaluar esto, en el siguiente experimento se probó BCP y BCP-VA sobre conjuntos que presentaban cinco porcentajes de intersección entre sus áreas, a saber 0%, 25%, 50%, 75% y 100% de overlap. Se consideraron tres tamaños de conjuntos (200K, 400K, 600K), con una distribución uniforme y utilizando cuatro cantidades de memoria (64, 128, 256 y 512 Buckets). Los resultados se muestran en la Tabla 5.4 para el conjunto de 200K, en la Tabla 5.5 para los conjuntos de 400K y en la Tabla 5.6 para los conjuntos de 600K.

			Tiempo (ms)			**accesos**		
Tamaño	Overlap	Buckets	BCP	BCP-VA	% Mejora	BCP	BCP-VA	% Mejora
200K	0%	64	3.854	2.238	42%	4.698	4.834	-3%
200K	0%	128	3.712	1.546	58%	4.635	4.892	-6%
200K	0%	256	5.292	1.571	70%	4.486	5.014	-12%
200K	0%	512	9.158	1.788	80%	4.229	5.271	-25%
PROMEDIO			5.504	1.786	63%	4.512	5.003	-11%
200K	25%	64	32.045	9.908	69%	6.228	5.356	14%
200K	25%	128	24.337	6.079	75%	6.255	5.250	16%
200K	25%	256	36.280	5.589	85%	10.494	5.772	45%
200K	25%	512	145.327	13.183	91%	22.989	6.788	70%
PROMEDIO			59.497	8.690	80%	11.492	5.792	36%
200K	50%	64	44.287	15.940	6%4	6.787	5.722	16%
200K	50%	128	36.295	9.491	74%	7.271	5.490	24%
200K	50%	256	52.751	8.387	84%	13.648	6.252	54%
200K	50%	512	252.624	23.051	91%	32.665	7.748	76%
PROMEDIO			96.489	14.217	78%	15.093	6.303	43%
200K	75%	64	58.733	19.607	67%	7.534	5.977	21%
200K	75%	128	50.354	12.076	76%	8.425	5.700	32%
200K	75%	256	69.510	10.701	85%	17.060	6.686	61%
200K	75%	512	330.275	25.169	92%	44.763	8.617	81%
PROMEDIO			127.218	16.888	80	19.446	6.745	49%
200K	100%	64	66.048	22.921	65%	7.966	6.124	23%
200K	100%	128	46.832	13.978	70%	8.120	5.815	28%
200K	100%	256	63.242	11.755	81%	16.137	6.868	57%
200K	100%	512	284.304	32.547	89%	42.436	9.037	79%
PROMEDIO			115.107	20.300	76%	18.665	6.961	47%

Tabla 5.4: Rendimiento de BCP-VA y BCP sobre conjuntos de 200K con distintos porcentaje de intersección

De estos resultados podemos concluir que, aunque existe un leve aumento en la cantidad de accesos a disco (como se observó en los experimentos anteriores), este nuevo esquema de particionamiento significa una mejora considerable en el tiempo total del algoritmo. Si bien,

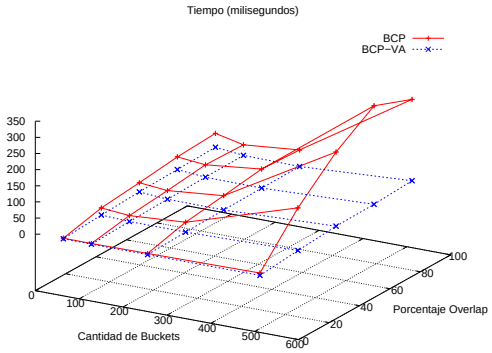
			Tiempo (ms)			accesos		
Tamaño	Overlap	Buckets	BCP	BCP-VA	% Mejora	BCP	BCP-VA	% Mejora
400K	0%	64	7.019	4.912	30%	9.467	9.605	-1%
400K	0%	128	6.652	4.126	38%	9.387	9.646	-3%
400K	0%	256	9.822	4.146	58%	9.265	9.781	-6%
400K	0%	512	16.886	4.189	75%	9.021	10.042	-11%
PROMEDIO			10.095	4.343	50	9.285%	9.769	-5 %
400K	25%	64	78.564	25.973	67%	11.912	10.446	12%
400K	25%	128	40.179	13.706	66%	10.320	9.917	4%
400K	25%	256	46.147	10.288	78%	12.453	10.322	17%
400K	25%	512	132.453	14.238	89%	20.041	11.096	45%
PROMEDIO			74.336	16.051	75%	13.682	10.445	19 %
400K	50%	64	139.473	45.181	68%	13.821	11.032	20%
400K	50%	128	84.282	24.392	71%	11.343	10.128	11%
400K	50%	256	87.664	17.537	80%	15.214	10.766	29%
400K	50%	512	282.952	25.924	91%	32.865	11.975	64%
PROMEDIO			148.593	28.259	77%	18.311	10.975	31 %
400K	75%	64	185.493	61.001	67%	15.519	11.566	25%
400K	75%	128	115.256	33.883	71%	12.428	10.344	17%
400K	75%	256	116.056	24.248	79%	18.114	11.203	38%
400K	75%	512	391.437	36.530	91%	42.619	12.938	70%
PROMEDIO			202.061	38.916	77%	22.170	11.513	38%
400K	100%	64	220.648	80.509	64%	16.562	12.276	26%
400K	100%	128	156.566	42.478	73%	13.447	10.542	22%
400K	100%	256	170.612	30.447	82%	21.706	11.662	46%
400K	100%	512	478.549	46.533	90%	49.811	13.834	72%
PROMEDIO			256.594	49.992	77%	25.382	12.079	41%

Tabla 5.5: Rendimiento de BCP-VA y BCP sobre conjuntos de 400K con distintos porcentaje de intersección

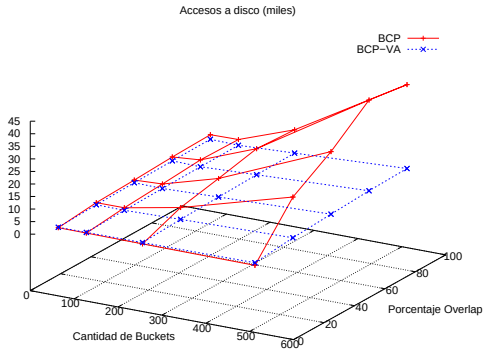
cuando no hay intersección entre los conjuntos, los accesos a disco de BCP-VA comparados con BCP aumentan en promedio un 10%, en la medida que la intersección aumenta los accesos totales disminuyen entre un 20% y un 40% aproximadamente.

Por otro lado, el tiempo total del algoritmo, para todos los escenarios disminuye considerablemente, siendo los peores casos una mejora de tan solo 30% del tiempo, para los conjuntos con 0% de intersección y 64 Buckets, mientras que en los otros porcentajes de intersección, y con 512 Buckets se logró mejorar el tiempo total del algoritmo en un 90%.

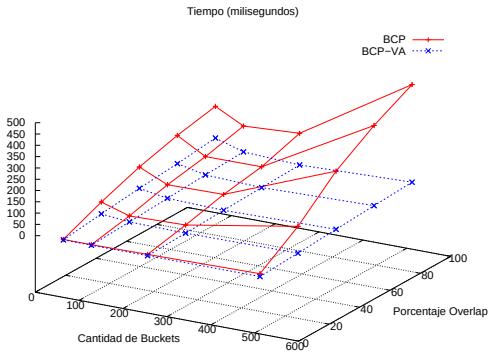
Esto nos demuestra que BCP-VA se comporta mucho mejor frente a los distintos porcentajes de intersección entre los conjuntos. Si bien el tiempo y los accesos del algoritmo aumentan en la medida que lo hace también la intersección, este parámetro no influye tanto como lo hace con la propuesta anterior. En la Figura 5.5 podemos ver que BCP-VA se mantiene más plana, en comparación a BCP, que varía bastante en la medida que cambia la intersección de los conjuntos.



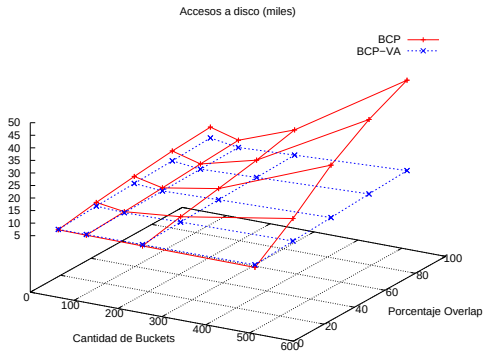
(a) Tiempo para conjuntos de 200K puntos



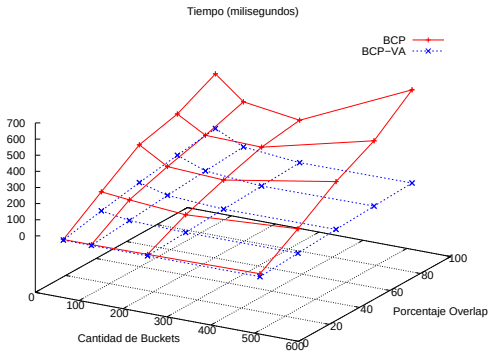
(b) accesos para conjuntos de 200K puntos



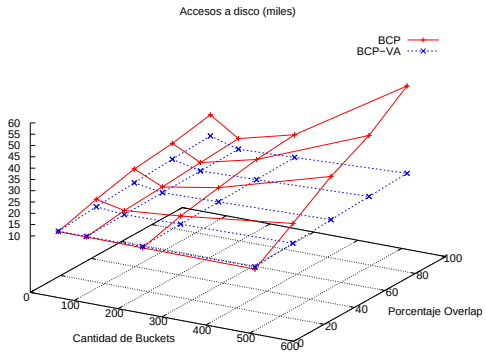
(c) Tiempo para conjuntos de 400K puntos



(d) accesos para conjuntos de 400K puntos



(e) Tiempo para conjuntos de 600K puntos



(f) accesos para conjuntos de 600K puntos

Figura 5.5: Comparación entre BCP y BCP-VA sobre conjuntos con distinto porcentaje de intersección

			Tiempo (ms)			accesos		
Tamaño	Overlap	Buckets	BCP	BCP-VA	% Mejora	BCP	BCP-VA	% Mejora
600K	0%	64	10.204	7.087	31%	14.256	14.373	-1%
600K	0%	128	11.045	5.712	48%	14.172	14.410	-2%
600K	0%	256	15.261	6.226	59%	14.010	14.546	-4%
600K	0%	512	24.695	6.303	74%	13.790	14.795	-7%
PROMEDIO			15.301	6.332	53%	14.057	14.531	-3 %
600K	25%	64	174.625	56.752	68%	19.133	15.869	17%
600K	25%	128	157.146	28.947	82%	16.491	14.729	11%
600K	25%	256	128.994	19.758	85%	18.677	15.084	19%
600K	25%	512	172.129	20.505	88%	24.608	15.854	36%
PROMEDIO			158.224	31.491	80%	19.727	15.384	21 %
600K	50%	64	334.283	100.374	70%	23.182	17.086	26%
600K	50%	128	231.920	53.387	77%	17.575	15.013	15%
600K	50%	256	213.108	32.807	85%	21.847	15.596	29%
600K	50%	512	332.633	36.877	89%	36.070	16.917	53%
PROMEDIO			277.986	55.861	80%	24.669	16.153	31 %
600K	75%	64	393.750	137.960	65%	25.152	18.107	28%
600K	75%	128	294.305	73.467	75%	18.946	15.253	19%
600K	75%	256	285.636	44.996	84%	24.938	16.025	36%
600K	75%	512	454.265	49.772	89%	44.801	17.819	60%
PROMEDIO			356.989	76.549	78%	28.459	16.801	36 %
600K	100%	64	511.097	172.535	66%	28.383	19.047	33%
600K	100%	128	370.345	91.156	75%	20.203	15.491	23%
600K	100%	256	321.084	58.109	82%	26.515	16.467	38%
600K	100%	512	638.657	61.400	90%	57.325	18.696	67%
PROMEDIO			460.296	95.800	78%	33.107	17.425	40 %

Tabla 5.6: Rendimiento de BCP-VA y BCP sobre conjuntos de 600K con distintos porcentaje de intersección

5.3.4 Experimento 4: BCP-VA sobre conjuntos con distinta distribución en el espacio.

El último experimento al que fue sometido BCP-VA tuvo como objetivo evaluar su rendimiento cuando la distribución de los elementos del conjunto no es uniforme. Para esto se buscó el par de vecinos más cercanos sobre datos de prueba que tenían una distribución Uniforme (U), una distribución Gaussiana (G) y un conjunto de datos Reales (R), evaluando los mismos 5 escenarios detallados en la Sección 4.3.5 y que corresponden a : UxG, UxR, GxG, RxR, y GxR.

Las pruebas se realizaron sobre conjuntos de tamaño 200K, 400K y 600K, sin intersección entre ellos, y utilizando cuatro cantidades de memoria distinta (64,128, 256 y 512 Buckets). Los resultados de estos experimentos se detallan en la Tabla 5.7 a) para el tiempo, y b) para los accesos a disco.

En la tabla 5.7 podemos apreciar que la distribución de los conjuntos no presenta una variación

significativa en los resultados obtenidos. El tiempo total y los accesos a disco de BCP-VA no experimentan una variación que pueda ser atribuida a la distribución del conjunto, ya que las diferencias entre los valores, en comparación al escenario UxU utilizado en los experimentos anteriores, van desde un -7% a un +7%, siendo el promedio 0% de diferencia para los accesos, mientras que para el tiempo total, las variaciones van desde un -30% a +30%, con un promedio 10% para todos los casos.

5.4 Análisis de resultados y conclusiones

La primera conclusión que podemos obtener de los experimentos es que, el nuevo esquema de particionamiento ofrece una gran mejora para BCP-VA sobre BCP. Este nuevo algoritmo es capaz de mantener el rendimiento de la primera propuesta con una cantidad pequeña de memoria, pero BCP-VA además es capaz de mejorar su desempeño en la medida que hay más memoria disponible. Esta nueva propuesta también ofrece mejores resultados que BCP, cuando el porcentaje de intersección entre los conjuntos es mayor que 0%.

La primera prueba demostró que, si bien con el nuevo algoritmo de particionamiento son necesarios una cantidad mayor de accesos en la primera etapa, este leve costo en accesos implica una mejora considerable en el tiempo total del algoritmo. El crear una serie de vectores ordenados permite que el tiempo necesario para distribuir todos los elementos del conjunto disminuya en la medida que hay más memoria disponible. Si los conjuntos no presentan intersección, entonces al ser esta primera etapa la más costosa, el algoritmo obtiene una mejora considerable en su tiempo total de ejecución.

Por otra parte, el particionar el conjunto de una manera más eficiente y sin intersección, garantiza que el algoritmo tendrá un mucho mejor desempeño en la segunda etapa. Esta mejora en la etapa de procesamiento, logra también disminuir significativamente el tiempo total del algoritmo en los escenarios donde sí existe intersección entre los conjuntos.

A continuación se comenta cómo el resto de los parámetros influyeron el desempeño de la nueva propuesta.

5.4.1 Sobre el número de samples

Los experimentos mostraron que una cantidad mayor de samples produce un deterioro del rendimiento del algoritmo. Esto se explica dado que un aumento en la cantidad de samples implica más elementos que deben ser ordenados. Sin embargo, este costo adicional en tiempo también implica una partición más efectiva para las siguientes etapas. Los experimentos demostraron que basta una muestra de entre 0.3% (2000 samples para 600.000 puntos) y 5% (2000 samples para 400.000 puntos) para obtener buenos resultados, tanto en tiempo de particionamiento como en tiempo de búsqueda para todos los escenarios evaluados.

5.4.2 Sobre la cantidad de memoria

Una de las mayores ventajas que ofrece BCP-VA sobre BCP, es que puede aprovechar de una manera aún más eficiente los recursos existentes. Si bien con cantidades pequeñas de memoria (pocos Buckets), la nueva propuesta logra los mismos resultados que BCP, en la medida que existen más Buckets, BCP-VA puede mejorar su rendimiento.

Por ejemplo, para los conjuntos de 600K sin intersección, BCP logra su mejor tiempo con 64 Buckets (10.204 ms.), mientras que en el mismo escenario, BCP-VA logra su mejor tiempo con 512 Buckets (6.303 ms) . Esta diferencia significa una mejora de casi un 38% en el tiempo a favor de BCP-VA. De manera similar, si analizamos los resultados sobre conjuntos de 600K con 50% de intersección, con 64 Buckets logra un tiempo de 334.283 mientras que BCP-VA, utilizando 512 Buckets, logra un tiempo de 36.877 ms, que es una diferencia cercana al 89% a favor de BCP-VA.

Al igual que BCP, Los resultados muestran que el comportamiento de BCP-VA varía según la cantidad de Buckets. Si bien en la medida que el parámetro aumenta, también lo hace el

rendimiento de BCP-VA, es probable que también exista un valor que optimice el rendimiento del algoritmo en relación a otros parámetros del algoritmo. Sin embargo, y dado el alcance de esta tesis, ese valor no será estudiado y se propone como un trabajo futuro.

5.4.3 Sobre la intersección de los conjuntos

Uno de los escenarios más desfavorables para BCP, y que se esperaba mejorar con BCP-VA, es aquel donde los conjuntos presentan un porcentaje de intersección distinto a 0%. Esto se debe principalmente a que las métricas usadas en la segunda etapa disminuyen su efectividad en la medida que existe intersección entre los Buckets, lo que produce que esta etapa sea más costosa que la primera, aumentando el tiempo total del algoritmo.

Los experimentos demostraron que BCP-VA logró mejorar su rendimiento en la medida que aumenta la intersección de los conjuntos. Si bien la segunda etapa sigue siendo más costosa, y dicho costo aumenta en la medida que hay más Buckets para trabajar, esta propuesta mejora notoriamente el desempeño sobre BCP.

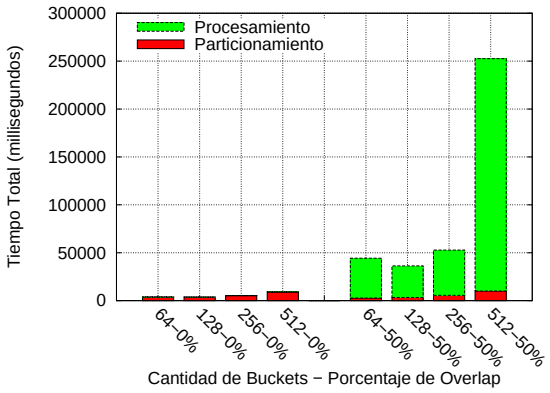
En la Figura 5.6 se grafican los tiempos y accesos de las dos etapas de cada uno de los algoritmos, para conjuntos de 200K, con distintos porcentajes de intersección. En cada imagen, los cuatro primeros bloques corresponden a los resultados con distintas cantidades de memoria (64, 128, 256 y 512 Buckets) con 0% y los siguientes cuatro bloques a los resultados con distintas cantidades de memoria, pero con 50% de intersección.

Como podemos apreciar en la Figura 5.6 a), BCP aumenta considerablemente la cantidad de tiempo en la segunda etapa del algoritmo (procesamiento) en la medida que aumenta la intersección de los conjuntos. Como se comentó anteriormente, el aumento en la intersección de los Buckets implica que las métricas utilizadas para filtrar estos elementos son menos eficientes y, por lo tanto, es necesario realizar más accesos a disco y también más comparaciones entre elementos contenidos en los Buckets. Adicionalmente, debido al esquema de particionamiento implementado en BCP, puede existir intersección entre los Buckets del mismo conjunto, lo que empeora aún más la ineficiencia de las métricas y produce un aumento mayor en la cantidad de accesos a disco. Esto produce que, en el peor de los casos (con 512 Buckets) y cuando existe un 50% de intersección entre los conjuntos, BCP necesite 125 veces mś tiempo que cuando la intersección es 0%.

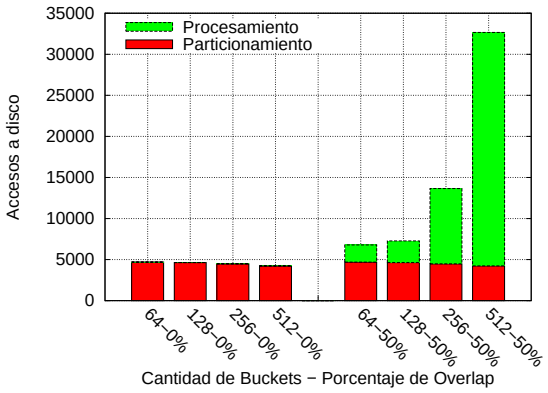
Por el contrario, si bien BCP-VA mantiene una cantidad de accesos a disco similar que BCP (Figura 5.6 b) y d)), la cantidad de tiempo requerido para procesar disminuye considerablemente, para los escenarios con porcentaje de interesección distinto de 0%. Esto se logra gracias al nuevo esquema de particionamiento que crea una Set de Buckets mucho mejor distribuido en el espacio y eliminado la intersección entre los Buckets de un mismo conjunto. De esta manera, aunque cuando las métricas no logran filtrar tantos elementos como cuando no existe intersección, sí se logra disminuir la cantidad de accesos causados por el mal particionamiento del conjunto. Gracias a este algoritmo, en el peor de los casos (con 512 Buckets), cuando existe un 50% de intersección entre los conjuntos, BCP-VA necesita solamente 1.5 veces más tiempo que cuando no hay intersección.

5.4.4 Sobre la distribución de los Conjuntos

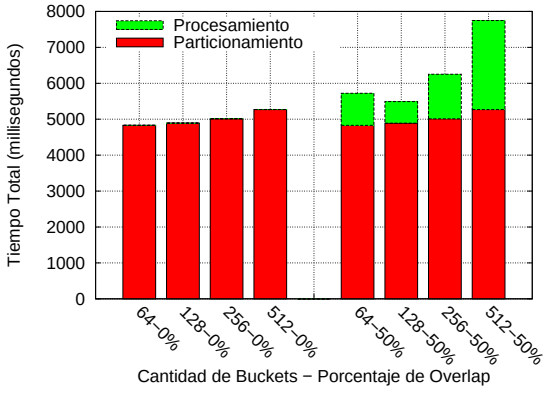
En base a los experimentos que buscaban evaluar el rendimiento de BCP-VA sobre conjuntos con distinta distribución, podemos concluir que este parámetro no afecta mayormente el rendimiento del algoritmo. Incluso los experimentos demostraron que para los datos reales se necesitan menos accesos y menos tiempo para encontrar la solución, en comparación al resto de los escenarios



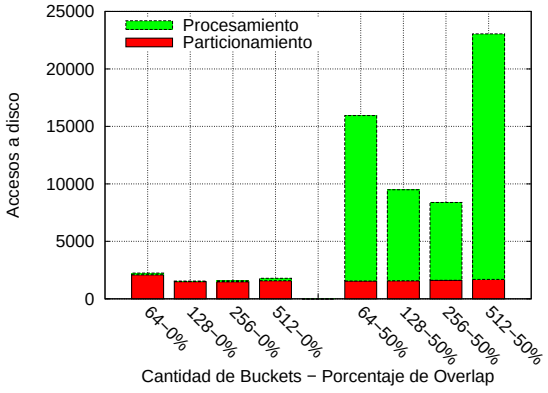
(a) Tiempos BCP



(b) accesos a disco BCP



(c) Tiempo BCP-VA



(d) accesos a disco BCP-VA

Figura 5.6: Comparación entre las etapas BCP y BCP-VA sobre conjuntos de 200K con distinto porcentaje de intersección

evaluados. Este comportamiento es consecuente con el nuevo esquema de particionamiento, puesto que el algoritmo divide el espacio en un conjunto de regiones que se ajustan mejor a la distribución de los objetos de cada conjunto.

		BCP-VA					
Tamaño	Buckets	UxU	UxG	UxR	GxG	GxR	RxR
200K	64	2.297	2.336	1.777	1.879	2.336	2.070
200K	128	1.522	1.768	1.624	1.239	1.566	1.765
200K	256	1.514	1.656	1.704	1.396	1.656	2.082
200K	512	1.774	1.566	1.962	1.411	1.768	2.060
PROMEDIO		**1.777**	**1.994**	**1.767**	**1.481**	**1.832**	**1.994**
400K	64	3.990	3.525	3.406	3.316	3.525	4.220
400K	128	3.967	3.155	3.143	2.631	3.094	3.567
400K	256	4.105	3.025	3.331	2.519	3.025	3.608
400K	512	4.453	3.094	4.162	2.520	3.155	3.787
PROMEDIO		**4.129**	**3.796**	**3.511**	**2.747**	**3.200**	**3.796**
600K	64	7.040	5.460	5.413	7.269	5.460	6.732
600K	128	5.891	4.751	5.389	4.304	4.483	6.435
600K	256	6.592	4.943	6.539	4.335	4.943	6.020
600K	512	6.493	4.483	5.490	4.208	4.751	6.432
PROMEDIO		**6.504**	**6.405**	**5.708**	**5.029**	**4.909**	**6.405**

a) Tiempo (milisegundos)

		BCP-VA					
Tamaño	Buckets	UxU	UxG	UxR	GxG	GxR	RxR
200K	64	4.834	4.829	4.829	4.830	4.829	4.838
200K	128	4.892	5.267	4.889	4.880	4.883	4.894
200K	256	5.014	5.010	5.021	5.004	5.010	5.026
200K	512	5.271	4.883	5.280	5.254	5.267	5.280
PROMEDIO		**5.003**	**5.010**	**5.005**	**4.992**	**4.997**	**5.010**
400K	64	9.605	9.612	9.597	9.624	9.612	9.615
400K	128	9.646	10.025	9.652	9.658	9.651	9.664
400K	256	9.781	9.772	9.773	9.770	9.772	9.772
400K	512	10.042	9.651	10.047	10.022	10.025	10.066
PROMEDIO		**9.769**	**9.779**	**9.767**	**9.769**	**9.765**	**9.779**
600K	64	14.373	14.397	14.373	14.458	14.397	14.406
600K	128	14.410	14.813	14.419	14.434	14.420	14.426
600K	256	14.546	14.545	14.546	14.546	14.545	14.548
600K	512	14.795	14.420	14.797	14.828	14.813	14.796
PROMEDIO		**14.531**	**14.544**	**14.534**	**14.567**	**14.544**	**14.544**

b) accesos a disco

Tabla 5.7: Resultados de BCP y BCP-VA sobre conjuntos con distinta distribución sin intersección.

Capítulo 6

Conclusiones

6.1 Conclusiones Finales

Encontrar el par de vecinos más cercanos es una consulta muy utilizada y que tiene variadas aplicaciones prácticas. Si bien este problema ha sido investigado y resuelto desde el campo de la geometría computacional, es en el ámbito de las Bases de Datos Espaciales donde ha recibido mayor atención en el último tiempo.

En este campo, las principales estrategias para resolver el problema consisten en el uso de estructuras y métodos de accesos espaciales, siendo uno de los más usados el R-Tree y las variantes que han surgido de él. Si bien resolver las consultas con la ayuda de estos métodos es muy eficiente, las operaciones para crear y mantener dichos índices son bastantes costosas y se justifican considerando que dichos índices puedan ser reutilizados posteriormente. Sin embargo, cuando estos índices no están disponibles, es un caso que aún no ha sido resuelto de una manera eficiente.

El objetivo de esta tesis fue resolver el problema de encontrar el par de vecinos más cercano, en los ambientes de las Bases de Datos Espaciales, sobre conjuntos sin indexar. Este escenario, se puede encontrar, por ejemplo, cuando los conjuntos donde se desea buscar corresponden a los resultados de otras consultas.

El primer algoritmo propuesto, Bucket Closest Pair (BCP) apunta a resolver el problema utilizando un enfoque de dos etapas: el particionamiento y el procesamiento. En la primera etapa, el particionamiento, cada conjunto se divide en un Set de Buckets, que es una estructura creada especialmente para el algoritmo. Dicha estructura utiliza el concepto del MBR para representar en memoria todos los elementos que contiene, mientras que estos son almacenados en el disco. Además, mediante el uso de un buffer en la estructura, se disminuyen los accesos a disco durante ambas etapas del algoritmo. En la segunda etapa, nuestra propuesta utiliza una serie de métricas definidas entre los MBRs que permiten el filtrado de los Buckets, sin necesidad de rescatar los datos desde el disco. Es solamente una vez terminado el filtrado, que se realiza la búsqueda sobre los elementos contenidos en los Buckets que no han sido filtrados.

Al comparar nuestra propuesta, con las soluciones basadas en índices espaciales del tipo R-Tree, los resultados mostraron que BCP fue capaz de procesar los conjuntos y encontrar la solución en tan solo un 1.8% del tiempo requerido por R-Tree para crear los índices, sin llegar a procesar la consulta.

Nosotros comparamos nuestros algoritmos contra una adaptación de BNL, un algoritmo definido para resolver el join en las bases de datos relacionales. Una serie de experimentos

demostraron que BCP ofrece una mejora sustancial sobre BNL y que, si bien en algunos escenarios BCP requiere más accesos a disco que BNL, nuestro algoritmo encuentra la respuesta en un tiempo menor sin importar el tamaño de los conjuntos, ni la distribución de los mismos.

Sin embargo, estos experimentos también demostraron que, si bien BNL mejora su desempeño en la medida que dispone de más memoria para trabajar, BCP sufre un deterioro en el rendimiento. Además, en la medida que la intersección entre los conjuntos aumenta, el tiempo total de BCP aumenta también, mientras que BNL no es influenciado por este parámetro.

Un análisis de estos escenarios desfavorables, llevaron a la conclusión que estos problemas con BCP se deben al esquema de particionamiento utilizado. Cuando los Buckets de un Set intersectan sus áreas, las métricas utilizadas para el filtrado disminuyen su efectividad, obligando al algoritmo a procesar más elementos para encontrar la respuesta. Este problema aumenta, en la medida que también lo hace la intersección entre ambos conjuntos. Con esto se concluyó que, si bien la propuesta superó a BNL, se debe realizar una mejora al algoritmo que particiona el conjunto, con el objeto de lograr una mejor distribución de los elementos,y así poder garantizar un mejor desempeño en cualquier escenario.

La segunda propuesta, Bucket Closes Pair - Vector Approximation (BCP-VA), utiliza también la estructura Bucket y además el mismo enfoque de dos esquemas que BCP. Sin embargo, en BCP-VA se implementó un algoritmo de particionamiento que, mediante el uso de una muestra de los elementos, es capaz de dividir el espacio (no los elementos) de una manera mucho más uniforme, garantizando que no hay intersección entre los Buckets de un mismo Set, sin importar la naturaleza de los conjuntos.

Los experimentos sobre este nuevo algoritmo demostraron que, si bien el esquema de particionamiento requiere de una cantidad adicional de accesos a disco, estos accesos adicionales son compensados por una mejora en el tiempo total del algoritmo. Además este nuevo algoritmo, permite aprovechar de una manera mucho más eficiente la memoria disponible, logrando mejorar su desempeño en la medida que aumenta la cantidad de Buckets, contrario a lo que ocurría con BCP.

Por otra parte, este nuevo esquema de particionamiento también logra minimizar el impacto de la intersección de los conjuntos, disminuyendo considerablemente la variación obtenida en el tiempo total del algoritmo cuando la intersección entre los conjuntos es mayor que 0%.

Finalmente, podemos concluir que nuestra propuesta BCP-VA es una solución eficiente para el problema de encontrar el par de vecinos más cercanos sobre conjuntos no indexados, ya que encuentra la solución en menos tiempo que el requerido para crear estructuras del tipo R-Tree, y además ofrece notables mejoras sobre una adaptación del algoritmo BNL.

6.2 Trabajos Futuros

Durante el desarrollo de este trabajo se han identificado una serie de problemas y escenarios similares a los acá presentados, que no fueron analizados y que se encuentran directamente relacioandos con la línea de investigación de esta tesis.

- Determinar si existe una forma de estimar el valor óptimo para la cantidad de memoria (número de Buckets) que debe utilizar el algoritmo, para un escenario dado. Es decir, determinar si existe una relación entre los distintos parámetros sobre los que trabaja BCP-VA (tamaños de los conjuntos, intersección de los mismos, cantidad de memoria disponible,

velocidad del procesador, velocidad del disco, etc.), que permita determinar la cantidad de Buckets que se deben utilizar para obtener el mejor resultado.

- Evaluar el algoritmo sobre escenarios donde uno de los conjuntos dispone de un índice, para determinar si el tiempo necesitado para construir solo un índice es menor que el que utiliza nuestro algoritmo para encontrar la respuesta. Además podría determinarse si algunas de las propiedades de dichos índices pueden ayudar a mejorar el algoritmo.

- Extender el algoritmo BCP-VA para encontrar los k-vecinos más cercanos, puesto que actualmente se resuelve el problema para $k = 1$. Experimentos preliminares demostraron que con el uso de un heap, es posible encontrar más de un par de elementos, sin aumentar demasiado el costo del algoritmo. Sin embargo, aún hay muchos detalles que evaluar sobre este tema.

- Extender la implementación del algoritmo para más de dos dimensiones, lo que de acuerdo a la implementación realizada, no debería presentar mayores inconvenientes. Además, el uso del "Vector Approximation" garantiza, a juicio de sus autores, no tener problemas con la "Maldicion de la dimensionalidad".

- Desarrollar un modelo de costo para BCP-VA de manera de estimar, si es posible, la cantidad de accesos totales necesarios por el algoritmo de acuerdo a los distintos parámetros dados.

Referencias

[BBK01] C. Böhm, S. Berchtold, and D.A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys (CSUR)*, 33(3):322–373, 2001.

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

[BKS93a] T. Brinkhoff, H.P. Kriegel, and R. Schneider. Comparison of approximations of complex objects used for approximation-based query processing in spatial database systems. In *Data Engineering, 1993. Proceedings. Ninth International Conference on*, pages 40–49. IEEE, 1993.

[BKS93b] T. Brinkhoff, H.P. Kriegel, and B. Seeger. Efficient processing of spatial joins using R-trees. *ACM SIGMOD Record*, 22(2):246, 1993.

[BKSS90] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. *The R*-tree: an efficient and robust access method for points and rectangles*, volume 19. ACM, 1990.

[BKSS94] T. Brinkhoff, H.P. Kriegel, R. Schneider, and B. Seeger. *Multi-step processing of spatial joins*, volume 23. ACM, 1994.

[BM72] R. Bayer and E.M. McCreight. Organization and maintenance of large ordered indexes. *Acta informatica*, 1(3):173–189, 1972.

[BS76] J.L. Bentley and M.I. Shamos. Divide-and-conquer in multidimensional space. In *Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 220–230. ACM, 1976.

[BW97] S. Blott and R. Weber. A simple vector-approximation file for similarity search in high-dimensional vector spaces. *ESPRIT Technical Report TR19, ca*, 1997.

[CMTV00] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Closest pair queries in spatial databases. *ACM SIGMOD Record*, 29(2):200, 2000.

[CMTV04] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Algorithms for processing K-closest-pair queries in spatial databases. *Data & Knowledge Engineering*, 49(1):67–104, 2004.

[Cor02] Antonio Corral. *Algoritmos para el Procesamiento de Consultas espaciales utilizando R-Trees. La consulta de los Pares más cercanos y su aplicación en las Bases de Datos Espaciales*. PhD thesis, Universidad de Almería, 2002.

[DHKP97] M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. A Reliable Randomized Algorithm for the Closest-Pair Problem* 1. *Journal of Algorithms*, 25(1):19–51, 1997.

[GG98] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys (CSUR)*, 30(2):170–231, 1998.

[GG09] Luis Gajardo and Gilberto Gutierrez. El par de vecinos más cercanos: hacia propuestas de algoritmos en escenarios en que uno de los conjuntos no se encuentra indexado. In *XXI Encuentro Chileno de Computación (ECC)*. 2009.

[GS05] R.H. Güting and M. Schneider. *Moving objects databases*. Morgan Kaufmann Pub, 2005.

[Gut84] A. Guttman. *R-trees: a dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[Güt94] R.H. Güting. An introduction to spatial database systems. *The VLDB Journal*, 3(4):357–399, 1994.

[Gut07] Gilberto Gutiérrez. *Métodos de Acceso y Procesamiento de Consultas Espacio-Temporales*. PhD thesis, Universidad de Chile, 2007.

[Had11] Marios Hadjieleftheriou. Libspatialindex website. http://libspatialindex.github.com/, Última visita: 26 Enero de 2011.

[HS98a] G.R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. *ACM SIGMOD Record*, 27(2):248, 1998.

[HS98b] G.R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *ACM SIGMOD Record*, volume 27, pages 237–248. ACM, 1998.

[LR96] M.L. Lo and C.V. Ravishankar. Spatial hash-joins. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, page 258. ACM, 1996.

[ME92] P. Mishra and M.H. Eich. Join processing in relational databases. *ACM Computing Surveys (CSUR)*, 24(1):63–113, 1992.

[NHS84] J. Nievergelt, H. Hinterberger, and K.C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems (TODS)*, 9(1):38–71, 1984.

[PSTE95] D. Papadias, T. Sellis, Y. Theodoridis, and M.J. Egenhofer. *Topological relations in the world of minimum bounding rectangles: a study with R-trees*, volume 24. ACM, 1995.

[PT97] D. Papadias and Y. Theodoridis. Spatial relations, minimum bounding rectangles, and spatial data structures. *International Journal of Geographical Information Science*, 11(2):111–138, 1997.

[QTP+08] S. Qiao, C. Tang, J. Peng, H. Li, and S. Ni. Efficient k-Closest-Pair Range-Queries in Spatial Databases. In *Web-Age Information Management, 2008. WAIM'08. The Ninth International Conference on*, pages 99–104, 2008.

[RKV95] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Acm Sigmod Record*, volume 24, pages 71–79. ACM, 1995.

[Rob81] J.T. Robinson. The kdb-tree: a search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM SIGMOD international conference on Management of data*, pages 10–18. ACM, 1981.

[Sam90] H. Samet. *The design and analysis of spatial data structures*, volume 85. Addison-Wesley Reading MA, 1990.

[SC03] S. Shekhar and S. Chawla. *Spatial databases: a tour*, volume 1. Prentice Hall Upper Saddle River, NJ, 2003.

[SH75] M.I. Shamos and D. Hoey. Closest-point problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 151–162. IEEE, 1975.

[Smi95] M. Smid. Closest point problems in computational geometry. *MAX PLANCK INSTITUT FUR INFORMATIK-REPORT-MPI I*, 1995.

[SML00] H. Shin, B. Moon, and S. Lee. Adaptive multi-stage distance join processing. In *ACM SIGMOD Record*, volume 29, pages 343–354. ACM, 2000.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. 1987.

[SZS03] J. Shan, D. Zhang, and B. Salzberg. On spatial-range closest-pair query. *Advances in Spatial and Temporal Databases*, pages 252–269, 2003.