UNIVERSIDAD DEL BÍO-BÍO
FACULTAD DE CIENCIAS EMPRESARIALES
DEPARTAMENTO DE SISTEMAS DE INFORMACIÓN

ALGORITMO DE BÚSQUEDA BI-OBJETIVO PARA LA OPTIMIZACIÓN DEL CÁLCULO DE LA CONSULTA SKYLINE SOBRE UNA ESTRUCTURA DE DATOS COMPACTA K²-TREE.

PROYECTO DE TÍTULO PRESENTADO POR BAYRON DOUGLAS REYES CID
PARA OBTENER EL TÍTULO DE INGENIERO CIVIL EN INFORMÁTICA
DIRIGIDO POR DR. RODRIGO TORRES AVILÉS Y DRA. MÓNICA CANIUPÁN MARILEO

2023.2

A mi madre, mi familia, mis amigos y al Leo Cid (QEPD).

Agradecimientos

Quisiera partir por agradecer a mis profesores guía que me orientaron en este proceso de Proyecto de Título, quienes pacientemente supervisaron mi trabajo, tanto en las etapas de investigación como de desarrollo, también a todos los profesores y profesoras de la Universidad del Bío-Bío que fueron parte de mi formación profesional.

Mi familia, en su conjunto, la cual ha sido un pilar sólido en mi vida académica. Agradezco las horas de paciencia, las conversaciones alentadoras y el amor constante que me han brindado en cada fase de mi educación. Su respaldo ha sido esencial y ha contribuido enormemente a mi formación profesional. A mis leales amigos les dedico un agradecimiento especial. Sus risas, palabras alentadoras y comprensión han iluminado mis días y han sido un apoyo invaluable durante esta travesía académica. Cada momento compartido ha hecho que este camino sea más memorable y significativo. Al Leo Cid (QEPD), mi tío, quien siempre me alentó a estudiar y nunca rendirme pese a todas las adversidades, le dedico este proyecto, sea donde sea que estés, lo vamos a lograr.

A todos aquellos que han sido parte de mi experiencia Universitaria, desde profesores y mentores, hasta compañeros de clase, les agradezco por cada interacción que ha enriquecido mi aprendizaje. Cada consejo compartido y cada palabra de aliento han dejado una huella duradera en mi desarrollo académico y personal.

Este viaje de aprendizaje ha sido posible gracias al apoyo de cada uno de ustedes. Con gratitud sincera, dedico esto a todos ustedes y espero seguir contando con su apoyo en las futuras etapas de mi vida. Finalmente, este camino llega a su punto de conclusión, y espero con ansias todo lo que depara el futuro, tanto en el ámbito profesional como personal.

Resumen

En la actualidad, la compleja tarea de toma de decisiones donde los usuarios se enfrentan a numerosas opciones e información, el rápido aumento de los sistemas de apoyo para la toma de decisiones y el constante crecimiento de los datos multidimensionales plantean diversos desafíos, tales como la contrariedad de sus criterios, el hecho de que no siempre exista una única respuesta optima, o que una misma consulta con distintos usuarios y preferencias, pueda dar como resultado variadas respuestas. La consulta Skyline destaca entre estos sistemas, al extraer objetos de interes desde conjuntos de datos multidimensionales, devolviendo solo objetos "óptimos".

Diversos algoritmos, desde estrategias de pre-ordenación hasta el uso de Estructuras de Datos Compactas, abordan la consulta Skyline. Estas estructuras eficientes buscan almacenar información en espacios mínimos de memoria sin sacrificar la eficiencia de sus operaciones. La Estructura de Datos Compacta k^2-tree, diseñada originalmente para representar Grafos Web, destaca en esta consulta al reducir comparaciones de dominancia y optimizar el procesamiento de los datos, siendo especialmente útil en aplicaciones móviles, las que cuentan con memoria y acceso a internet limitado.

La aproximación de subconjuntos aplicada a la búsqueda Bi-objetivo, nos permite identificar un subconjunto representativo de datos en lugar del análisis del conjunto completo, esta estrategia es valiosa para abordar problemas computacionales complejos, mejorando la eficiencia y manejo de grandes conjuntos de datos. La integración de la aproximación de subconjuntos y la búsqueda bi-objetivo implica encontrar un subconjunto representativo de datos Skyline óptimo o no dominado en relación con dos objetivos definidos por el usuario. La importancia de estos elementos radica en su capacidad para abordar la complejidad de las decisiones multidimensionales y ofrecer soluciones equilibradas y representativas. Estos enfoques no solo ahorran tiempo, al centrarse en opciones que mejor se ajustan a las preferencias individuales del usuario, sino que también se centran en mejorar la eficiencia computacional y el manejo de grandes conjuntos de datos.

Una estrategia de solución para potenciar la investigación y desarrollo en este campo implica la continua exploración de algoritmos más eficientes y estructuras de datos innovadoras. Además, la colaboración interdisciplinaria entre expertos en bases de datos, inteligencia artificial y diseño de algoritmos puede impulsar el desarrollo de soluciones más avanzadas y adaptativas para la toma de decisiones basada en múltiples criterios.

De forma general, éste proyecto propone, analiza e implementa un algoritmo que permite responder la consulta Skyline utilizando los principios de la aproximación de Subconjuntos relacionada a una busqueda Bi-objetivo, todo esto sobre datos almacenados en una estructura compacta k^2-tree. El objetivo de éste algoritmo, es reducir de forma considerable los tiempos de respuesta en comparación a otros algoritmos de consulta Skyline, junto con la obtención de subconjuntos de datos mucho más acordes a las preferencias y criterios del usuario.

Palabras Clave: Skyline, Región de Pareto, k^2-tree, Bitmap, Dominancia, Búsqueda Bi-Objetivo, Región de Dominancia, Branch and Bound Skyline k^2-tree (BBSk), Función de Costos, Optimización de operaciones, Heap, Memoria principal.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Partiendo desde la premisa y el problema en el cual una persona decida viajar a una ciudad que se encuentra cerca de un centro esquí durante un fin de semana. Para ello, al momento de buscar hospedaje en un hotel, se encuentra con una lista de opciones disponibles. Para escoger el mejor hospedaje de dicha lista, deberá de elegir los criterios con los que decidirá cuál es el más conveniente. Para este caso suponga que la persona prefiere el hotel que se encuentre más cerca del centro de esquí, y al mismo tiempo sea el más asequible o más barato. Éstas dos preferencias suelen ser contradictorias debido a que comúnmente el hospedaje suele tener un costo monetario mayor a medida en que se encuentre más cerca de los centros de esquí. Ademas, la persona les da distinta importancia a ambos criterios, puede que no le importe tanto el costo monetario en relación a la proximidad con el centro de esquí, o puede que no le importe tanto la distancia al centro de esquí en relacion al costo monetario del hospedaje. Por lo cual, la persona deberá de conseguir lograr satisfacer ambas preferencias en relación a la importancia que les da a los costos de hospedaje y la distancia que separa el hotel del centro de esquí, de tal forma que pueda encontrar el mejor hotel en relación con sus preferencias y criterios.

En el mundo existen una infinidad de situaciones y contextos en las cuales los usuarios buscan tomar una decisión frente a una enorme cantidad de opciones e información, teniendo en cuenta una considerable cantidad de criterios, los cuales, muchas veces pueden ser contradictorios. Por ejemplo, escoger la mejor vivienda a comprar considerando el precio y la distancia a un colegio, o un hospital, o un lugar de trabajo, o una estación del transporte público; escoger la mejor ubicación para una sucursal considerando aspectos estratégicos económicos o de marketing; entre otros casos. En todos estos casos, el usuario debe encontrar la opción que mejor se ajusta a sus preferencias dentro de una enorme cantidad de informacion existente, haciendo uso de múltiples criterios multidimensionales que deben ser agrupados para encontrar la mejor opción.

El rápido aumento de los sistemas de soporte para la toma de decisiones basadas en múltiples criterios y el constante crecimiento en el tamaño de los datos multidimensionales, han motivado a los investigadores a explorar nuevos y eficaces enfoques para el procesamiento de datos con el objetivo de obtener resultados útiles [13]. Muchos de estos sistemas se caracterizan por múltiples aspectos, tales como la contrariedad de sus criterios, el hecho de que no siempre exista una única respuesta optima, o que para una misma consulta con distintos usuarios e influenciados por sus preferencias personales, pueda dar como resultado variadas respuestas.

La consulta Skyline ha adquirido una relevancia significativa en el área de las investigaciones de bases de datos, especialmente para extraer objetos destacados desde diversos conjuntos de datos multidimensionales [15]. En el contexto de ésta consulta, dado un conjunto de puntos, la consulta Skyline devuelve solo aquellos puntos que no son dominados por otros. Un punto domina otro punto

si es tan bueno o mejor en todas las dimensiones y mejor en al menos una dimensión [10]. La popularidad de la consulta Skyline se debe a la simplicidad de su enfoque y a su aplicabilidad en el apoyo para la toma de decisiones basadas en criterios múltiples, considerando las preferencias del usuario [13]. La consulta Skyline tiene múltiples aplicaciones, tanto en las áreas de negocios, economía, entornos distribuidos como arquitecturas en la nube, redes P2P, redes inalámbricas, etc.

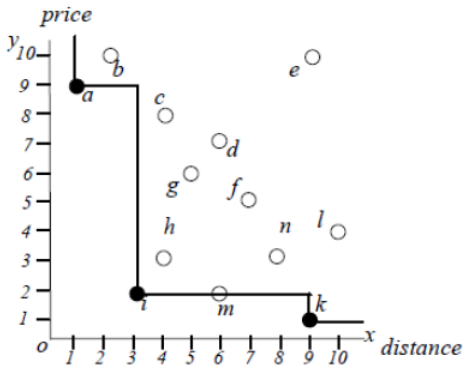

Figura 1.1: Consulta Skyline aplicada al problema inicial.

Por ejemplo, dado el problema de la habitación de hotel y el centro de esquí, y de acuerdo a la Figura 1.1. El eje Y corresponde a los precios de los hospedajes y el eje X a la distancia del hospedaje al centro de esquí, los puntos negros corresponden a todos aquellos hoteles más convenientes respecto a su relacion precio/distancia del centro de esquí, y por contraparte los puntos blancos corresponden a las opciones descartadas según ésta misma relación.

En la literatura, se han propuesto diversos algoritmos para abordar esta consulta. Entre las estrategias destacadas se incluye la pre-ordenación del conjunto de puntos o su pre-procesamiento mediante alguna estructura o índice que almacene dichos puntos. Estas estrategias buscan disminuir la cantidad de comparaciones de dominancia entre los puntos de un conjunto analizado, con el objetivo de reducir el costo de acceso a memoria secundaria y el costo de procesamiento [16].

En contraste, una Estructura de Datos Compacta se define como un diseño eficiente que busca utilizar una cantidad mínima de espacio de memoria para almacenar información, a la vez que mantiene la capacidad de realizar operaciones de manera eficaz [3]. La clave radica en que estas estructuras puedan gestionar consultas relevantes sin necesidad de descomprimir la información almacenada. Las Estructuras de Datos Compactas poseen múltiples utilidades, tales como la representación de imágenes, mosaicos de datos, georreferencias, relaciones binarias, diccionarios, Grafos Web, secuencias de ADN, arboles binarios de decisión, entre otras [4].

Las Estructuras de Datos Compactas pueden ser útiles en aplicaciones móviles, especialmente cuando la memoria es limitada y el acceso a internet puede ser escaso. Supongamos que la persona que buscaba inicialmente una habitación de hotel cercana a un centro de esquí no tiene acceso a internet, pero cuenta con una aplicación móvil en su Smartphone que almacena toda la información turística relevante de la ciudad, incluyendo hoteles, restaurantes y lugares de interés. Utilizar Estructuras de Datos Compactas se convierte en una alternativa eficiente para almacenar esta información en el espacio reducido de la memoria del Smartphone. En este escenario, la consulta Skyline puede ayudar a la persona a encontrar las opciones más interesantes para su estadía.

Por ejemplo, podría consultar el hotel más recomendado y asequible, el restaurante mejor valorado y cercano, el cine más próximo, entre otras consultas relacionadas con su búsqueda de una experiencia que satisfaga sus preferencias.

En particular, la aplicación de la Estructura de Datos Compacta k²-tree se presenta como una propuesta interesante para abordar esta consulta. Esta estructura, introducida en [3], fue inicialmente concebida con el objetivo de representar Grafos Web, pero ha demostrado ser igualmente útil para la representación de relaciones binarias.

El k²-tree es una estructura útil para abordar la consulta Skyline, ya que aprovecha sus propiedades para reducir el número de comparaciones de dominancia, disminuyendo así la carga de procesamiento. Además, al tratarse de una Estructura de Datos Compacta, posibilita almacenar más información con un uso más eficiente de la memoria. Esto facilita el procesamiento de datos en la memoria principal, reduciendo la necesidad de acceder a la memoria secundaria. Por ejemplo, consideremos la situación de la persona que buscaba un alojamiento cercano a un centro de esquí. La información sobre distintos hoteles, incluyendo tarifas, servicios y calificaciones de usuarios, podría organizarse de manera eficiente en un k²-tree, permitiendo así responder de manera rápida y eficaz a consultas Skyline.

Como se puede observar en la Figura 1.3, a la izquierda está la representación gráfica de un k²-tree como un plano cartesiano que almacena datos, por ejemplo y teniendo en cuenta el problema inicial, el eje Y representa precio, el eje X representa el precio del hospedaje, y los puntos representan la relacion precio/distancia de estos. Al lado derecho está la representación gráfica de la estructura del k²-tree.

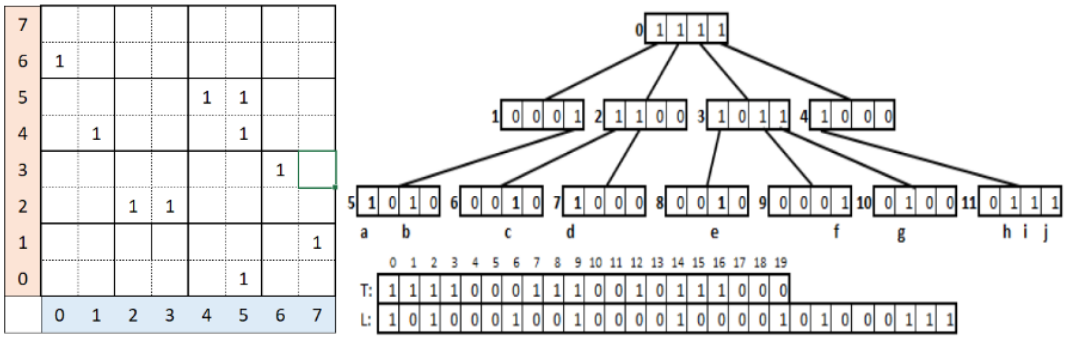

Figura 1.2: Representación gráfica de un k²-tree.

Por otro lado, la aproximación de subconjuntos aplicados a una búsqueda Bi- objetivo [2], hace referencia a un enfoque práctico que combina ambos conceptos.

La aproximación de subconjuntos Skyline [17], consiste en identificar y seleccionar un subconjunto representativo de datos Skyline, en lugar de encontrar el conjunto completo. Esta estrategia se utiliza a menudo para abordar problemas computacionales complejos donde el análisis completo del conjunto de datos puede ser costoso en términos de tiempo o recursos computacionales. Al seleccionar un subconjunto representativo, se puede lograr una solución más rápida o eficiente, aunque a veces con una pérdida aceptable de precisión o exhaustividad. En el contexto de la búsqueda Skyline o la optimización multicriterio [18], la aproximación de subconjuntos puede aplicarse para reducir la complejidad del procesamiento, seleccionando solo un subconjunto representativo de soluciones Skyline en lugar de analizar todo el conjunto. Esto es útil para mejorar la eficiencia computacional y manejar conjuntos de datos grandes.

Mientras que la búsqueda Bi-objetivo [19] consiste en un tipo de búsqueda en la que se intenta encontrar soluciones optimas o no dominadas en relación con dos objetivos simultáneamente.
En términos más simples, implica la exploración de soluciones que buscan optimizar dos criterios o metas al mismo tiempo, sin que una solución sea mejor que otra en ambos objetivos. En contextos como la optimización multicriterio o la toma de decisiones basada en múltiples criterios, la búsqueda Bi-objetivo es esencial para identificar soluciones que equilibren dos aspectos o metas diferentes. Por ejemplo, en problemas de planificación, podrías buscar una solución que minimice el tiempo de ejecución y maximice la utilización de recursos al mismo tiempo. En la búsqueda Bi-objetivo, las soluciones no se clasifican como simplemente "mejores" o "peores", sino que se evalúan en términos de su rendimiento relativo en ambos objetivos, lo que da lugar a conjuntos de soluciones no dominadas. Estas soluciones representan compromisos entre los objetivos y ofrecen opciones diversas para el usuario o el tomador de decisiones.

En general, la combinación de estos dos conceptos implica encontrar un subconjunto representativo de datos Skyline que sea óptimo o no dominado en relación a dos objetivos definidos, con el fin de ser una aplicación útil para la toma de decisiones donde se busca unificar dos criterios diferentes [20]. Por ejemplo, dado el problema inicial de la persona que busca un hospedaje cercano a un centro de esquí. Para hacer esta búsqueda de manera eficiente, se emplea una estrategia de aproximación de subconjuntos con búsqueda Bi-objetivo. En lugar de evaluar todos los hoteles disponibles, se selecciona un subconjunto inicial basado en criterios como ubicación geográfica o categoría de precios, utilizando así la aproximación de subconjuntos. En este conjunto inicial, se realiza una búsqueda Bi-objetivo para evaluar cada hotel en dos criterios fundamentales: la proximidad al centro de esquí y el costo de la estadía. La búsqueda Bi-objetivo permite identificar hoteles que forman parte del conjunto Skyline, es decir, opciones no superadas por otras en ambas dimensiones. La persona recibe como resultado una lista de hoteles que constituyen el conjunto Skyline Bi-objetivo. Estos hoteles representan opciones óptimas en cuanto a la relación precio/distancia, ofreciendo así soluciones equilibradas y representativas. Esta estrategia eficiente de búsqueda no solo facilita la toma de decisiones, sino que también ahorra tiempo al enfocarse en opciones que mejor se ajustan a las preferencias individuales en cuanto a ubicación y presupuesto.

En resumen, en éste proyecto se propone, analiza e implementa un algoritmo que permite responder la consulta Skyline utilizando los principios de la aproximación de Subconjuntos relacionada a una busqueda Bi-objetivo, todo esto sobre datos almacenados en una estructura compacta k^2-tree. El objetivo de éste algoritmo, es reducir los tiempos de respuesta de forma considerable en comparación a otros algoritmos de consulta Skyline, junto con obtener un subconjunto de datos mucho más acorde a las preferencias del usuario.

1.1 Contenidos del documento

El presente documento está distribuido en 7 capítulos

- El capítulo 1. Introducción, presenta como dice su nombre la introducción al proyecto y breves definiciones de los conceptos claves es éste, junto con la estructura de organización del presente documento.
- El capítulo 2, Definicion de éste proyecto, presenta los objetivos generales y específicos de éste proyecto, los alcances y límites de este, junto con la metodología de trabajo de él.
- El capítulo 3, Conceptos preliminares, presenta las estructuras de datos compactas Bitmap y k^2-tree, la definición de la consulta Skyline, el algoritmo Branch and Bound Skyline k^2-tree (BBSk) que opera sobre las estructuras mencionadas, junto con el análisis de los componentes teóricos de la aproximación de subconjuntos Skyline aplicada a la búsqueda Bi-objetivo.
- El capítulo 4, Funciones de Costos aplicadas como condición, presenta la investigación realizada para este proyecto, la cual consiste en el análisis de las regiones de dominancia, el análisis de las funciones de costo de una búsqueda Bi-objetivo y la implementación de estas funciones como condiciones de comparación y descarte.
- El capítulo 5, Branch and Bound Skyline k^2-tree cost, presenta la propuesta de esté proyecto de título para responder la consulta Skyline sobre datos almacenados en un k^2-tree, utilizando los principios de la búsqueda Bi-objetivo como condición de comparación y de descarte.
- El capítulo 6, Implementación y experimentación, presenta la implementación del algoritmo, los bancos de datos utilizados en la experimentación, el entorno de pruebas, los resultados de la experimentación y una conclusión respectos a estos.
- El capítulo 7, Conclusiones y trabajo futuro, presenta las conclusiones del proyecto junto con las sugerencias para un trabajo futuro relacionado a ése proyecto.

Cada uno de los capítulos del presente documento se subdivide en secciones, cada una de las secciones se subdivide en ítems y cada ítem se subdivide en sub-ítems.

Capítulo 2

Definición de éste proyecto

En el siguiente capítulo se presentan los objetivos generales y específicos de éste proyecto, los alcances y límites de este, junto con la metodología de trabajo de él.

2.1 Objetivo general

Diseñar e implementar un algoritmo de búsqueda Bi-objetivo sobre la consulta Branch and Bound Skyline en datos indexados en un k^2-tree (BBSk), con el fin de optimizar las operaciones en términos de tiempo de ejecución y respuestas acotadas a una cantidad finita de parámetros de entrada definidos.

2.2 Objetivos específicos

- Investigar la estructura compacta de datos k^2-tree y analizar sus características.
- Estudiar la implementación de la consulta Skyline y comprender su funcionamiento,
- Revisar la implementación del algoritmo BBSk^2-tree, junto con sus componentes.
- Analizar la implementación de la búsqueda Bi-objetivo aplicable a la consulta BBSk^2-tree.
- Desarrollar un algoritmo de búsqueda Bi-objetivo aplicado a la consulta Skyline, con el fin de optimizar el rendimiento de ejecución sobre consultas acotadas.
- Evaluar el desempeño del algoritmo implementado mediante pruebas.
- Analizar y comparar los resultados de la experimentación.

2.3 Alcances y limites

Los desarrollos del proyecto se han llevado a cabo en el lenguaje de programación C++ ISO/IEC 14882:2020 (C++20) con el compilador gcc 6.3.0, en un entorno de desarrollo Linux, con el sistema operativo Linux Mint 21.2 Cinnamon con una arquitectura de 64 bits, en una máquina virtual corriendo en MS Windows 11 con una arquitectura de 64 bits. Ésta máquina virtual tiene asignado un procesador de un solo núcleo Intel(R) Core(TM) i3-7130U CPU @2.70GHz 2.71 GHz, con 6GB de RAM de tipo DDR3L.

Para mostrar los datos obtenidos por pantalla se utiliza la librería “iostream”, para la utilización de vectores, matrices y listas se hace uso de la librería “numeric” y “vector”, para la creación y administración de pares de objetos, útiles cuando se deben tratar dos objetos como si fuesen uno solo, se utiliza la librería “utility”, para la creacion y administración de la estructura de datos

heap se usa la librería “algorithm”, para las funciones matemáticas se hace uso de la librería “cmath”, para las mediciones de los tiempos de ejecución de los algoritmos implementados se utilizan las funciones de la librería “chrono”, todas estas librerías estan implementadas en C++. Para hacer uso de las estructuras de datos compactas k²-tree, junto con los Bitmaps que la componen, se hace uso de la librería “kTree.h”, incluida en el directorio de codigo fuente de éste proyecto.

Se ha hecho especial énfasis en el uso de la memoria principal, excluyendo cualquier procesamiento que involucre memoria secundaria. Además, se ha restringido el procesamiento a datos bidimensionales, excluyendo la manipulación de datos en tres o más dimensiones, ésta es la principal razón por la cual se utiliza una estructura de datos compacta k²-tree.

2.4 Metodología de trabajo

Dados los objetivos previamente definidos, el desarrollo del algoritmo se rige por una metodología iterativa incremental, la cual consiste en ir construyendo el producto final de manera progresiva a través de etapas. En cada etapa incremental se agrega una nueva funcionalidad, lo que permite ver resultados de una forma más rápida en comparación con el modelo en cascada, pues es mucho más flexible ya que nos permite el solapamiento de las etapas que componen el desarrollo e implementación del algoritmo.

Este proyecto esta subdividido en 4 etapas, las cuales son:

1. Etapa de investigación: Donde se hace análisis de la estructura compacta k²-tree, la implementación de consultas Skyline, la implementación del algoritmo BBSk²-tree [1], y la implementación de un algoritmo de búsqueda Bi-objetivo [2] aplicable al algoritmo anteriormente mencionado, los documentos referenciados corresponden a la base de éste proyecto.
2. Etapa de definiciones: Donde se identifican y definen las herramientas y metodologías referentes al desarrollo del algoritmo.
3. Etapa de desarrollo: Donde se desarrolla el algoritmo y su implementación, con el fin de optimizar los tiempos y resultados del cálculo de la consulta Skyline sobre una estructura de datos compacta k²-tree.
4. Etapa de experimentación y correcciones: Donde se evalua el desempeño del algoritmo implementado mediante pruebas, para el análisis y comparación de los resultados de la experimentación, junto con las posibles correcciones al algoritmo en caso de ser requeridas.

Capítulo 3

Conceptos preliminares

En el siguiente capítulo se presentan las estructuras de datos compactas Bitmap y k²-tree, la definición de la consulta Skyline y su variante que opera sobre las estructuras mencionadas [1], junto con el análisis de los componentes teóricos de la búsqueda Bi-objetivo [2].

3.1 Estructuras Compactas

Una Estructura de Datos Compacta corresponde a un diseño eficiente que busca utilizar una cantidad mínima de espacio de memoria para almacenar información, mientras mantiene la capacidad de realizar operaciones de manera eficaz [3]. Las Estructuras de Datos Compactas fundamentales para este proyecto corresponden al Bitmap y el k²-tree.

3.1.1 Bitmap

Un Bitmap o mapa de bits corresponde a una representación compacta de datos binarios, esta estructura de datos es comúnmente utilizada para almacenar información sobre elementos en un conjunto, cada elemento de este conjunto está asociado a un solo bit el cual puede tener como valor 0 o 1, indicando con esto la presencia o ausencia de un elemento, respectivamente. Un Bitmap es eficiente en cuanto a espacio, esto debido a que requiere menor cantidad de memoria en comparación a otras estructuras de datos al momento de representar conjuntos de elementos de mayor escala. Esta estructura de datos también nos permite responder a una serie de consultas de manera eficiente [4] junto con la posibilidad de implementar otras Estructuras de Datos Compactas [3].

Dada una secuencia de bits $B_{1,n}$, las operaciones fundamentales sobre un Bitmap se definen [3] como:

Definición 3.1: $Rank_b(B, i)$ entrega la cantidad de bits b encontrados en el prefijo $B_{1,i}$.

Es decir, $Rank_0(B, i)$ entrega la cantidad de bits 0 dentro de $B_{1,n}$, entre las posiciones 1 e i, y análogamente $Rank_1(B, i)$ entrega la cantidad de bits 1 dentro $B_{1,n}$ entre de las posiciones 1 e i.

Definición 3.2: $Select_b(B, i)$ entrega la posición en la cual se encuentra la i-ésima ocurrencia del bit b dentro de la secuencia $B_{1,n}$.

Es decir, $Select_0(B, i)$ entrega la posición en donde se encuentra el i-ésimo bit 0, y análogamente $Select_1(B, i)$ entrega la posición en la cual se encuentra el i-ésimo bit 1.

En la Figura 3.1 se observa un ejemplo gráfico correspondiente a la estructura de un Bitmap, junto con la ejecución de ambas operaciones fundamentales. En resumen, si se busca encontrar la posición en la que se encuentra el segundo bit 1, la consulta $select_1(2)$ responde la posición 3 en la distribución del Bitmap, en cuanto a buscar cuantos bit 0 hay entre la posiciones 1 y 7, la consulta $rank_0(7)$ responde que existen 4 bits 0 en el segmento indicado.

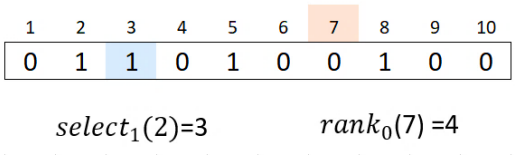

Figura 3.1: Ejemplo de Bitmap y sus operaciones fundamentales.

En términos relacionados a los tiempos de ejecución, la consulta Rank es resuelta en un tiempo constante, por contraparte la consulta Select es resuelta en un tiempo del orden O(log log n). Para agregar, la estructura compacta Bitmap, junto con sus operaciones fundamentales son la base para la construcción de otras Estructuras de Datos Compactas, tales como el k²-tree.

3.1.2 k²-tree

Un k²-tree corresponde a una estructura de datos que acorde a nuestros fines prácticos se utiliza para representar de forma compacta matrices binarias dispersas [5]. Su origen se remonta a 2009 en la investigación "k2-trees for compact web graph representation" [6], donde se introdujo inicialmente con el propósito de representar grafos Web previamente indexados en una matriz de adyacencia. La particularidad de esta estructura de datos radica en su capacidad para comprimir áreas extensas de la matriz que contienen ceros [3], lo que resulta especialmente útil para la compresión eficiente de matrices con una distribución de datos clusterizada.

Dado el antecedente de una matriz $(n\ x\ n)$ donde *n* corresponde a una potencia de *k*. Esta matriz es subdividida en k^2 sub-matrices de $\left(\frac{n}{k}\right) x \left(\frac{n}{k}\right)$ dimensiones, las cuales se contabilizan en orden de izquierda a derecha y de arriba hacia abajo. Cada una de las k^2 submatrices es representado por un bit 1 si contiene 1 o más puntos, y 0 si la submatriz se encuentra íntegramente vacía. De forma recursiva, aquellas submatrices que son representadas con un bit 1 vuelven a ser subdivididas en k^2 submatrices mientras que las submatrices representadas por un bit 0 son ignoradas [6]. Este proceso de subdivisión concluye cuando se encuentra una submatriz completamente vacía o cuando se alcanzan las celdas unitarias, las cuales no puede seguir siendo subdivididas. En el caso particular donde *n* no corresponde a una potencia de *k*, la matriz puede ser extendida de $(n\ x\ n)$ a $(n'x\ n')$ donde *n'* corresponde a una potencia de *k*, el espacio extra generado por esta extensión es completado con ceros. Esta estructura de datos es representada mediante la utilización de dos Bitmaps, llamados T y L. En el Bitmap T se almacenan todos lo bits de los nodos a excepción de los que se encuentran en el último nivel, pues estos son almacenados en el Bitmap L. Para el almacenamiento de los nodos en los Bitmaps se realiza un recorrido en anchura sobre el k²-tree.

En la Figura 3.2 se observa la estructura de la matriz de adyacencia junto con la representación gráfica del k²-tree, en este caso el valor de $k = 2$ y $n = 4$.

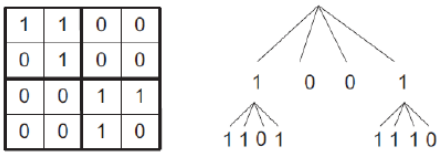

Figura 3.2: Ejemplo de k²-tree.

En la Figura 3.3 se observa la estructura de una matriz binaria, cuyo tamaño $n = 11$ fue extendido a $n' = 16$, el cual si corresponde a una potencia de $k = 2$, junto con la representación conceptual del k²-tree y el Bitmap resultante T:L [1].

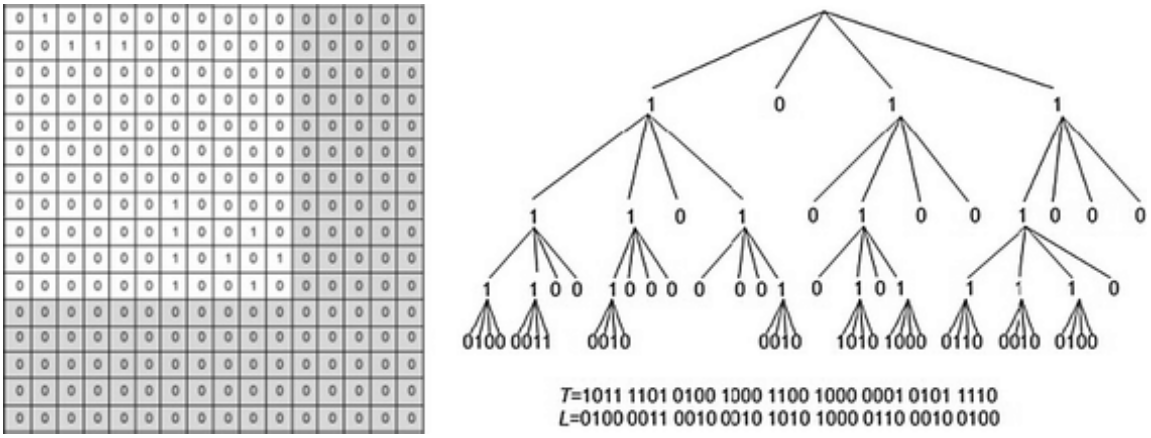

Figura 3.3: Ejemplo de matriz binaria, k²-tree y el Bitmap T:L.

Para representar relaciones binarias como un plano cartesiano, el origen se ubica en la esquina superior izquierda, mientras que las variables x e y son representadas por las columnas y filas del plano respectivamente, comenzando desde la coordenada (0,0) o el origen [5], como se puede observar en la Figura 3.4.

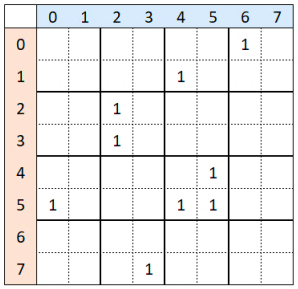

Figura 3.4: Representación gráfica de un k²-tree como un plano cartesiano.

3.1.2.1 Construcción del k²-tree

Para construir el k²-tree se utiliza el algoritmo propuesto en las investigaciones "k2-trees for compact web graph representation" [3] y "Algorithms and compressed data structures for information retrieval" [6], el cual actúa de forma recursiva a partir de una matriz de adyacencia. Con el fin de reducir el tiempo de construcción, se propone utilizar una lista de adyacencia como formato de entrada, esta se utiliza para representar una lista finita de puntos [1].

3.1.2.2 Navegación sobre la estructura k²-tree

Para recorrer la estructura k²-tree es necesario utilizar dos operaciones sobre el Bitmap T, estas operaciones corresponden a Rank y Select [7], ambas operaciones fueron definidas en el ítem 3.1.1 del presente documento.

Dada una secuencia de bits $B_{1,n}$, las operaciones Rank y Select sobre un Bitmap se definen [3] como:

- $Rank_b(B, i)$ entrega la cantidad de bits b encontrados en el prefijo $B_{1,i}$.
- $Select_b(B, i)$ entrega la posición en la cual se encuentra la i-ésima ocurrencia del bit b dentro de la secuencia $B_{1,n}$.

Para obtener el i-ésimo hijo de un nodo n del k²-tree, se utiliza la operación Child(n) [6], esta función se define como:

Definición 3.3: $child_i(n)$ entrega la posición del i-ésimo hijo de un nodo *n* en el Bitmap T:L.

$$child_i(n) = rank_1(T, n)k^2 + i$$

Por ejemplo, en el Bitmap mostrado en la Figura 3.5 se busca encontrar la posición del primer hijo del cuarto nodo, este se calcula:

$$child_1(4) = rank_1(T, 4) * 2^2 + 2 = 3 * 4 + 1 = 13$$

Según lo calculado, el primer hijo del tercer nodo en el Bitmap T:L se encuentra en la posición 13 de T, al acceder a la posición se descubre que corresponde un bit 1, es decir, es un nodo no vacío.

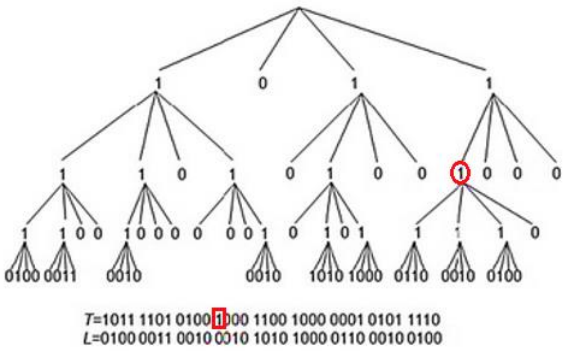

Figura 3.5: k²-tree y el Bitmap T:L que lo componen.

3.2 Consulta Skyline

La consulta Skyline o de Región de Pareto es una operación fundamental para múltiples aplicaciones de toma de decisiones multicriterio, y ha sido objeto de considerable atención en la comunidad de bases de datos [8], pues se trata de un paradigma popular y potente que facilita la extracción de puntos de interés desde un conjunto finito de datos multidimensionales [9].

3.2.1 Definiciones y propiedades de la consulta Skyline

La consulta Skyline, aplicada a un conjunto finito de puntos, cumple el rol de identificar y devolver todos aquellos puntos que no son dominados por otros puntos, a este conjunto solución también se le conoce como conjunto Pareto-óptimo. Un punto se considera dominado por otro punto si es tan bueno o mejor en todas las dimensiones y estrictamente mejor en al menos una de sus dimensiones [10]. La evaluación entre dos puntos para sabes cuál es mejor viene definida por los criterios de selección de los usuarios [11].

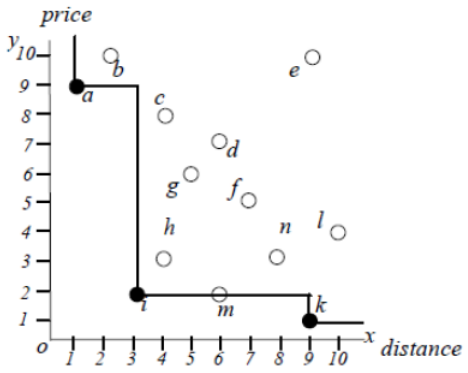

Figura 3.6: Ejemplo de la consulta Skyline sobre un conjunto de puntos.

Por ejemplo, dado un conjunto de puntos, los cuales representan la relación entre precio y distancia de habitaciones de hoteles y su cercanía de la playa, se busca encontrar las habitaciones más baratas y cercanas a la playa, donde la distancia entre la habitación de hotel y la playa corresponde al eje X, y el precio de las habitaciones corresponde al eje Y. En la Figura 3.6 se puede observar que los puntos *a*, *i* y *k* corresponden al conjunto Skyline, mientras que el resto son puntos descartados o dominados por el conjunto Skyline en cuanto a su relación distancia/precio [1].

La definición formal de la consulta Skyline es [9]:

Definición 3.4: Sea *D* un conjunto de puntos d-dimensionales, el resultado de la consulta Skyline *S* es definido como:

$$S = \{p \in D : \nexists q \in D : q \succ p\}$$

La relación de dominancia o de Pareto dominancia $q \succ p$ se define como [12]:

Definición 3.5: Dado dos puntos d-dimensionales *q* y *p*, se dice q domina a *p*, si *q* es mejor o igual a *p* en todas las d-dimensiones y es estrictamente mejor que *p* en al menos una de las d-dimensiones.

Definición 3.6: Sean *q* y *p* dos puntos d-dimensionales, en donde $q = (q_i, \dots, q_d)$ y $p = (p_i, \dots, p_d)$, en donde q_i y p_i representan el valor del atributo de q y p respectivamente en la dimensión i, la relación de dominancia $q \succ p$ se define como [9]:

$$q \succ p \Leftrightarrow \left(\bigwedge_{i=1}^{d} q_i \leq p_i \right) \wedge \left(\bigvee_{i=1}^{d} q_i < p_i \right)$$

Dicha relacion de dominancia o de Pareto dominancia posee 2 propiedades fundamentales:

Propiedad 3.1: Transitividad, dado los puntos p, r, t ∈ D, si p ≻ r ⋀ r ≻ t ⇒ p ≻ t, esta propiedad se puede utilizar para descartar de forma oportuna un solo punto o un conjunto de puntos que están dominados por un punto p, y que a su vez está dominado por un nuevo punto [13].

Propiedad 3.2: Incomparabilidad, dado dos puntos p, r ∈ D, si p no domina a r, y r no domina a p, entonces p y r son incomparables (p ∼ r), esta propiedad ayuda a determinar si uno o más puntos pueden ser Skyline, puesto que dos puntos Skyline son incomparables entre si [13].

3.2.2 Branch and Bound Skyline k²-tree (BBSk)

El algoritmo Branch and Bound Skyline k²-tree (BBSk) fue propuesto en el proyecto "Cálculo de la consulta Skyline sobre Estructuras de Datos Compactas" [1], éste está basado en el algoritmo Branch and Bound Skyline (BBS) y trabaja sobre la estructura compacta k²-tree.

BBSk comienza con el nodo raíz del k²-tree, que corresponde a la matriz completa. Este nodo raíz es subdividido en sus k² submatrices, aquellas submatrices que superen las condiciones de descarte son ingresadas en el heap como nuevos nodos. Este heap, organiza las submatrices de acuerdo a su distancia al origen de coordenadas. De forma iterativa, se evalúa el nodo en el tope del heap H de acuerdo a la dominancia de los puntos Skyline encontrados hasta el momento. Si este nodo no es dominado por dichos puntos, es subdividido en sus k² hijos, se aplican las condiciones de descarte a cada uno de ellos, y quienes las superen ingresan al heap como nuevos nodos o al conjunto Skyline, dependiendo de si corresponden a nodos internos o nodos hojas del k²-tree respectivamente.

Las condiciones de descarte para cada submatriz son fundamentales para mejorar la eficiencia del algoritmo. Mientras más tempranamente sea descartado un nodo, menos regiones que no contienen puntos Skyline son revisadas. El algoritmo BBSk puede implementar más condiciones de descarte que BBS debido a las propiedades del k²-tree y de los Bitmaps que lo componen.

3.2.2.1 Condiciones de descarte

Según BBSk [1], una submatriz puede ser descartada si:

1. Se encuentra dominada por algún punto Skyline encontrado hasta el momento: Sí la submatriz es dominada por algún punto Skyline encontrado hasta el momento, significa que esta no contiene puntos que pertenezcan al conjunto Skyline, por lo cual es descartada.
2. Se encuentra vacía: Esta condición es posible gracias a las propiedades del k²-tree, que permite identificar si una submatriz está vacía de forma rápida, esto se logra accediendo a la ubicación que la representa dentro del Bitmap T:L mediante las operaciones fundamentales para la estructura de datos Bitmap. Si una submatriz se encuentra vacía, no contiene puntos del conjunto Skyline, por lo cual es descartada.
3. Es dominada por alguno de sus hermanos no vacíos: Esta condición hace referencia a una comparación entre las submatrices de un mismo padre, para mantener solo aquellas que no estén dominadas por sus hermanos no vacíos. Si una submatriz es dominada por uno de sus hermanos, quiere decir que no contiene puntos Skyline, por lo tanto, es descartada. Esto reduce significativamente la cantidad de submatrices que ingresan al heap, disminuyendo con esto la cantidad de comparaciones y mejorando la eficiencia del algoritmo BBSk.

   En general, teniendo en cuenta que el origen de los ejes de coordenadas en un k²-tree se encuentra en la esquina superior izquierda, si uno de sus hermanos no vacíos se encuentra en la región sobre y a la izquierda de una submatriz, esta es descartada. Esta comparación puede realizarse con facilidad, accediendo al Bitmaps T:L mediante las operaciones fundamentales para la estructura de datos Bitmap.

De acuerdo a la tercera condición de descarte, en la Figura 3.7 se observa el caso más simple de poda de submatrices, con un k = 2. Se puede verificar que la submatriz de la esquina superior izquierda domina a aquella submatriz ubicada en la esquina inferior derecha, por lo tanto, tres de los cuatro hijos son ingresados al heap H. Si la submatriz dominante tuviese como valor 0, la poda de submatrices no podría realizarse, pero debido a la segunda condición de descarte, siguen siendo tres los hijos que entran al heap H.

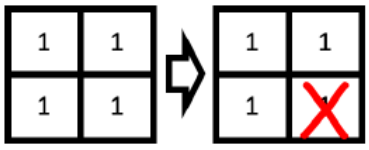

Figura 3.7: Ejemplo de poda en una submatriz de un k²-tree, con k = 2.

3.2.3.2 Algoritmo de Branch and Bound Skyline k²-tree (BBSk)

Al igual que en BBS, se realizan dos evaluaciones de dominancia entre cada submatriz y el conjunto *S*: en la línea 5 y 9. Las condiciones de descarte son aplicadas en orden a sus costos de aplicación. Tras evaluar las condiciones de descarte, en la línea 10 se consulta si la submatriz analizada se trata de un nodo o una hoja. De ser una hoja, ingresa de inmediato al conjunto *S*, en caso de ser un nodo es insertado en el heap para ser evaluado a posterioridad.

```
01: S = ∅
02: Insert root in heap
03: while heap not empty do
04:   Remove top entry n
05:   if n is not dominated by some point in S then
06:     for each child n_i in n do
07:       if n_i is not empty then
08:         if n_i is not dominated by its sibling not empty then
09:           if n_i is not dominated by some point in S then
10:             if n_i is a not a leaf then
11:               Insert n_i into heap
12:             else
13:               Insert n_i into S
14:             end if
15:           end if
16:         end if
17:       end if
18:     end for
19:   end if
20: end while
21: return S
```

El heap se encarga de almacenar las submatrices del k²-tree que pueden contener puntos del conjunto Skyline. Por cada una de las submatrices se almacena el índice que la representa en el Bitmap T:L, las coordenadas cartesianas (x, y) de su esquina superior izquierda (más cercana al origen), junto con el tamaño de la submatriz.

3.2.3.3 Ejemplo de ejecución de la consulta BBSk

Para el ejemplo utilizaremos el k²-tree junto con el Bitmap T:L de la Figura 3.8.

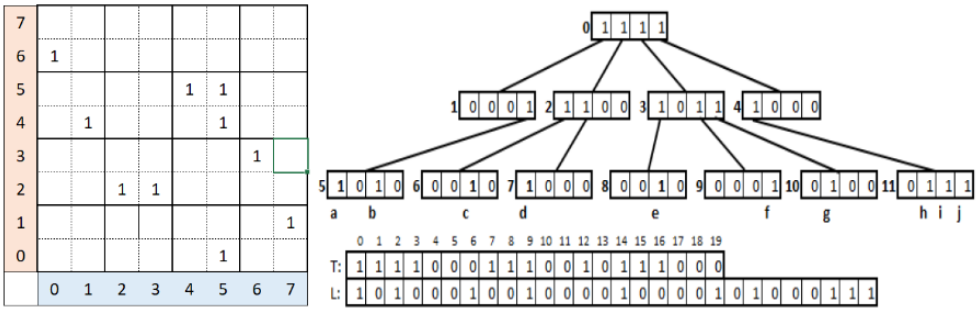

Figura 3.8: k²-tree y junto con el Bitmap T:L para el ejemplo de ejecución de BBSk.

Acción	Contenido del heap	Contenido de *S*
Inicio	(0,(0,0),0)	{}
Expandir 0	(1,(0,0),0), (2,(4,0),4), (3,(0,4),4)	{}
Expandir 1	(5,(2,2),4), (2,(4,0),4), (3,(0,4),4)	{}
Expandir 5	(2,(4,0),4), (3,(0,4),4)	(a,(2,2))
Expandir 2	(6,(4,0),4), (3,(0,4),4), (7,(6,0),6)	(a,(2,2))
Expandir 6	(3,(0,4),4), (7,(6,0),6)	(a,(2,2)), (c,(4,1))
Expandir 3	(8,(0,4),4), (9,(0,6),6), (7,(6,0),6)	(a,(2,2)), (c,(4,1))
Expandir 8	~~(9,(0,6),6)~~, (7,(6,0),6)	(a,(2,2)), (c,(4,1)), (e,(0,5))
Expandir 7	{}	(a,(2,2)), (c,(4,1)), (e,(0,5)), (d,(6,0))

Tabla 3.1: Ejemplo de ejecución de BBSk.

Seguimiento de la ejecución del algoritmo BBSk, para la Figura 3.8 y la Tabla 3.1.

1. Inicialmente dentro del heap solo se encuentra el nodo raíz (nodo 0), y el conjunto *S* se encuentra vacío.
2. Al expandir el nodo 0, se obtienen los nodos 1, 2, 3 y 4. El nodo 4 no es agregado al heap, pues este se encuentra dominado por el nodo 1. El nodo 1 tiene prioridad por sobre sus hermanos debido a que posee una distancia menor desde el origen.
3. Al expandir el nodo 1, se obtiene el nodo 5, el cual es el cuarto hijo del nodo 1. Sus hermanos se encuentran vacíos por lo tanto son descartados y no ingresan al heap. Tanto el nodo 5, 2 y 3 poseen la misma distancia desde el origen, por lo tanto, cualquiera de los tres puede ser expandido.
4. Al expandir el nodo 5, se obtienen cuatro nodos hojas. Primero se evalúa el nodo a, el cual ingresa al conjunto S. Al revisar el nodo b se comprueba que es dominado por el nodo a, por lo tanto, es descartado.
5. Al expandir el nodo 2, aparecen dos nuevos nodos: 6 y 7. El nodo 7 queda al final del heap debido a que su distancia desde el origen es 6, mientras que la distancia desde el origen de su nodo hermano es 4.
6. Al expandir el nodo 6, solo aparece el nodo c, que al no ser dominado por el nodo a y además ser una hoja, también es ingresado al conjunto S.
7. Al expandir el nodo 3, se obtienen los nodos 8, 9 y 10, de los cuales solo 8 y 9 ingresan al heap. El nodo 10 es dominado por su hermano, el nodo 8, entonces es descartado. De los nodos que quedan en el interior, el nodo 8 es el único que posee la distancia menor desde el origen, por lo tanto, es el nodo que aparece en el tope del heap.
8. Al expandir el nodo 8, aparece el nodo e, el cual, al no ser dominado por ningún punto de S y además ser una hoja, es agregado al mismo conjunto.
9. Al revisar el nodo 9, se comprueba de inmediato que es dominado por el recién descubierto nodo e, por lo tanto, es descartado apenas es extraído del tope.
10. El nodo que queda es el 7, del cual se obtiene el nodo d, que al no ser dominado por ningún punto del conjunto S, se convierte en el último punto Skyline en ser descubierto.
11. Termina la ejecución del algoritmo BBSk y se devuelve el conjunto Skyline.

3.3 Aproximación de Subconjuntos Skyline con búsqueda Bi-objetivo

La aproximación de Subconjuntos de regiones de Pareto o Skyline implementado en el algoritmo BOA* fue propuesta en la investigación “Subset Approximation of Pareto Regions with Bi-objective A*” [2], de la mencionada investigación solo se hace énfasis en el análisis de sus componentes teóricos, los cuales nos son de utilidad para el algoritmo que buscamos implementar.

3.3.1 Aproximación de subconjuntos

La aproximación de subconjuntos Skyline [17], consiste en identificar y seleccionar un subconjunto representativo de datos Skyline, en lugar de encontrar el conjunto completo. Esta estrategia se utiliza a menudo para abordar problemas computacionales complejos donde el análisis completo del conjunto de datos puede ser costoso en términos de tiempo o recursos computacionales. Al seleccionar un subconjunto representativo, se puede lograr una solución más rápida o eficiente, aunque a veces con una pérdida aceptable de precisión o exhaustividad. En el contexto de la búsqueda Skyline o la optimización multicriterio [18], la aproximación de subconjuntos puede aplicarse para reducir la complejidad del procesamiento, seleccionando solo un subconjunto representativo de soluciones Skyline en lugar de analizar todo el conjunto. Esto es útil para mejorar la eficiencia computacional y manejar conjuntos de datos grandes.

En un problema de búsqueda de objeto único P_a, donde la función de costo se define como $c = \alpha c_1 + (1-\alpha)c_2$ y $\alpha \in [0,1]$, una solución para P_α pertenece al conjunto de soluciones optimas de Pareto o Skyline.

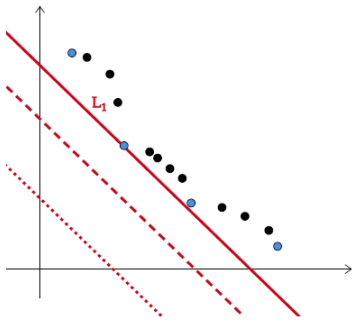

Figura 3.9: Ejemplo de búsqueda Uni-objetivo.

Como se puede observar en la Figura 3.9, el conjunto de puntos negros y azules representan un conjunto de soluciones Pareto-óptimas o Skyline de una instancia de búsqueda Uni-objetivo. Las líneas rojas representan rectas de la forma $\alpha x + (1-\alpha)y = K$ para tres valores diferentes de K. Cuando $K = 0$ la línea atraviesa el origen, y al aumentar el valor de K la línea se acerca a la región de Pareto hasta que la alcanza. Resolver el problema P_α es equivalente a encontrar el valor mínimo de K que hace que la línea sea tangente a la región de Pareto. Con este método solo se pueden encontrar puntos azules.

Teorema 1: Sea $P = (S, E, c, s_0, s_g)$ una búsqueda bi-objetivo, $\alpha \in [0,1]$ y sea P_a el problema de búsqueda de un solo objetivo (S, E, c, s_0, s_g), donde $c = \alpha c_1 + (1-\alpha)c_2$ es una función de costo univariante. Si π es una ruta de costo mínimo para P_a, entonces π pertenece al conjunto Pareto-óptimo o Skyline.

En cuanto a la interpretación geométrica de este enfoque, la búsqueda de una ruta con un costo mínimo dado por $\alpha c_1 + (1-\alpha)c_2$ se puede interpretar como encontrar el valor mínimo de *K* tal que la línea dada por la ecuación $\alpha x + (1-\alpha)y = K$ contiene un punto en el conjunto de Pareto. En otras palabras, encontrar una solución para P_a corresponde a encontrar una línea paralela a la recta $\alpha x + (1-\alpha)y = 1$ que sea tangente al conjunto de Pareto. Para encontrar un subconjunto de soluciones, uno puede probar varios valores de α. Sin embargo, dado que el enfoque está limitado a encontrar puntos que intersecan con tangentes al conjunto de soluciones, está restringido a encontrar como máximo puntos en la envolvente convexa del conjunto de soluciones.

3.3.2 Subconjuntos Skyline a través de la búsqueda Bi-objetivo

La búsqueda Bi-objetivo [19] consiste en un tipo de búsqueda en la que se intenta encontrar soluciones optimas o no dominadas en relación con dos objetivos simultáneamente.
En términos más simples, implica la exploración de soluciones que buscan optimizar dos criterios o metas al mismo tiempo, sin que una solución sea mejor que otra en ambos objetivos. En contextos como la optimización multicriterio o la toma de decisiones basada en múltiples criterios, la búsqueda Bi-objetivo es esencial para identificar soluciones que equilibren dos aspectos o metas diferentes. Por ejemplo, en problemas de planificación, podrías buscar una solución que minimice el tiempo de ejecución y maximice la utilización de recursos al mismo tiempo. En la búsqueda Bi-objetivo, las soluciones no se clasifican como simplemente "mejores" o "peores", sino que se evalúan en términos de su rendimiento relativo en ambos objetivos, lo que da lugar a conjuntos de soluciones no dominadas. Estas soluciones representan compromisos entre los objetivos y ofrecen opciones diversas para el usuario o el tomador de decisiones.

Dado que $\alpha, \beta \in [0,1]$ tal que $\alpha + \beta > 1$, para definir el problema de búsqueda Bi-Objetivo $P_{\alpha,\beta} = (S, E, c, s_0, s_g)$ se define la matriz $M_{\alpha,\beta}$ dada por $M_{\alpha,\beta} = \begin{pmatrix} \alpha & 1-\alpha \\ 1-\beta & \beta \end{pmatrix}$, donde para cada $e \in E$, se define un costo $c_{\alpha,\beta}(e)$ como el resultado de aplicar la matriz $M_{\alpha,\beta}$ a $c(e)$, es decir:

$$M_{\alpha,\beta}\big(c(e)\big) = \big(\alpha c_1(e) + (1-\alpha)c_2(e), (1-\beta)c_1 + \beta c_2(e)\big)$$

Básicamente, $P_{\alpha,\beta}$ es el mismo problema que P pero con su función de costo multiplicada por la matriz $M_{\alpha,\beta}$. Esta nueva instancia $P_{\alpha,\beta}$ posee dos propiedades fundamentales. La primera propiedad es que define una relación de dominancia $\preccurlyeq_{\alpha,\beta}$ en la que *u* es dominado por *v* si y solo si $\alpha u_1 + (1-\alpha)u_2 \leq \alpha v_1 + (1-\alpha)v_2$ y $(1-\beta)u_1 + \beta u_2 \leq (1-\beta)v_1 + \beta v_2$, y $u \neq v$, o de manera más reducida $M_{\alpha,\beta}(u) \preccurlyeq M_{\alpha,\beta}(v)$.

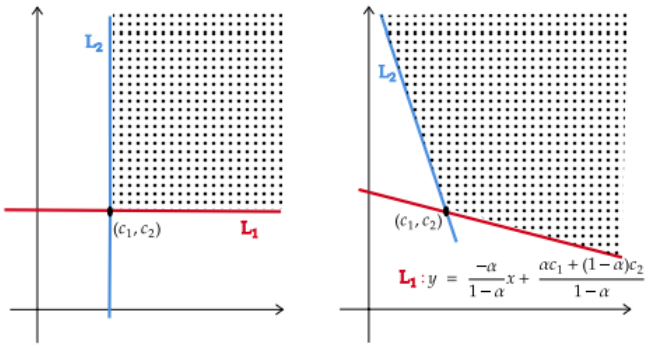

Figura 3.10: Comparación de región para una dominancia tradicional y una Bi-objetivo.

Dado que $L1 = \alpha x + (1-\alpha)y$ y $L2 = (1-\beta)x + \beta y$, y sean π y π' dos caminos tales que $c(\pi) \neq c(\pi')$. Entonces, $\pi \prec_{\alpha,\beta} \pi'$ si y solo si $c(\pi')$ está en la intersección del semiplano por encima de L1 y el semiplano por encima de L2.

La segunda propiedad importante es que el conjunto de soluciones de $P_{\alpha,\beta}$ está contenido en el conjunto de soluciones de *P*, como se indica en el Teorema 2 a continuación. Como consecuencia, todo el conjunto solución encontrado al resolver $P_{\alpha,\beta}$ pertenece al conjunto solución de P.

Teorema 2: Sea $P = (S, E, c, s_0, s_g)$ una instancia de búsqueda bi-objetivo y sea $P_{\alpha,\beta}$ definido como se indicó anteriormente, con $\alpha, \beta \in [0,1]$ y $\alpha + \beta > 1$. Entonces el conjunto solución de $P_{\alpha,\beta}$ pertenece al conjunto solución *P*.

Entonces, las funciones de costo α y β para un punto $p(x,y)$ en un problema de búsqueda Bi-objetivo según el problema planteado anteriormente y teniendo en cuenta las rectas L1 y L2 se definen como:

Definición 3.7: El costo α de $p(x,y)$ es $C_\alpha\big(p(x,y)\big) = \alpha x + (1-\alpha)y$

Definición 3.8: El costo β de $p(x,y)$ es $C_\beta\big(p(x,y)\big) = (1-\beta)x + \beta y$

Por ende, la funcion de costo de un punto $p(x,y)$ en relacion a α y β se define como:

Definición 3.9: La función de costo α, β de $p(x,y)$ es $C_{\alpha,\beta}\big(p(x,y)\big) = \big((C_\alpha(x,y)\big), (C_\beta(x,y)))$

Capítulo 4

Funciones de Costos aplicadas como condición

En el siguiente capítulo se presenta la investigación realizada para la implementación de las funciones de costo de una búsqueda Bi-objetivo [2] como condiciones de comparación y de descarte en el algoritmo Branch and Bound Skyline k²-tree (BBSk) [1].

Los resultados de la investigación nos presentan el análisis de las regiones de dominancia en las cuales actúa un punto, el análisis de las funciones de costo de una búsqueda Bi-objetivo y la implementación de estas funciones como condiciones de comparación y descarte.

4.1 Regiones de dominancia

En el Algoritmo BBSk [1] las regiones dominadas son definidas como todas aquellas regiones en las cuales no existen puntos del conjunto Skyline, pues se encuentra dominada por un punto del conjunto Skyline encontrado hasta el momento de su análisis. Un punto o submatriz es dominada por un punto del conjunto Skyline si este último es tan bueno o mejor en todas las dimensiones y estrictamente mejor en al menos una de sus dimensiones [10].

Como se puede observar en la Figura 4.1 la región dominada de un punto del conjunto Skyline corresponde a la región superior derecha, teniendo en cuenta la representación del k²-tree como un plano cartesiano, como se puede observar en la Figura 3.4.

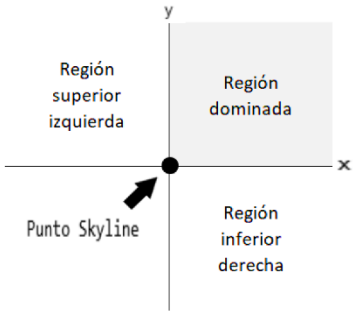

Figura 4.1: Regiones de dominancia para BBS y BBSk.

Las regiones de dominancia para un determinado punto Skyline corresponden a las regiones superior derecha o región dominada, junto con las regiones de dominancia superior izquierda e inferior derecha. Lo que se busca es ampliar la región dominada de un punto Skyline sobre las otras dos regiones de dominancia de acuerdo a dos variables asociadas a los ejes *x* e *y* respectivamente [2], estas variables son α y β cuyos valores en formato decimal están entre 0 y 1, como se puede observar en la Figura 4.2.

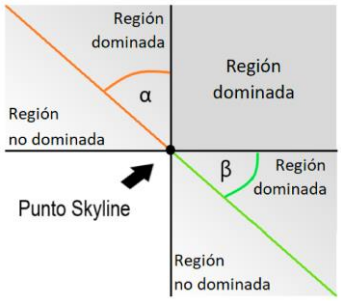

Figura 4.2: Regiones dominancia relacionadas a los valores α y β.

Definición 4.1: Los valores α y β también se pueden interpretar como niveles de importancia o ángulos de apertura para cada uno de los ejes al momento de ampliar e identificar la región dominada por un punto del conjunto Skyline. Además los valores de α y β están relacionados a ángulos respecto a los ejes *x* e *y* según la siguiente definición:

$$\alpha = \frac{\theta_1}{90} \quad \text{y} \quad \beta = \frac{\theta_2}{90}$$

Donde los ángulos de apertura θ_1y θ_2 para α y β no necesariamente deben tener el mismo valor y deben estar comprendidos entre 0° y 90°.

Como se puede observar en la Figura 4.2 la región dominada por un punto del conjunto Skyline aumenta al estar relacionado a las variables α y β, además de dominar la región superior derecha, también domina una región delimitada por la relación entre el punto Skyline y el valor de α en la región de dominancia superior izquierda, análogamente en la región de dominancia inferior derecha también posee una región dominada por la relación entre el punto Skyline y el valor de β. Esto teniendo en cuenta la representación del k²-tree como un plano cartesiano, como se puede observar en la Figura 3.4.

En general y de acuerdo al algoritmo BBSk [1], al aumentar la extensión de la región dominada por un punto Skyline, disminuyen las regiones donde se podrían encontrar puntos del conjunto Skyline, con esto también se disminuyen la cantidad de comparaciones y el tiempo de respuesta de la consulta.

4.2 Funciones de Costos

Para relacionar un punto determinado con las valores de α y β, se hace uso de las funciones de costo definidas para una búsqueda Bi-objetivo, las cuales representan la influencia de los "niveles de importancia" sobre un punto determinado o de acuerdo a lo planteado anteriormente, amplían la región de dominio de un punto.

Estas funciones corresponden a los costos de un punto en relación a valores de α y β, las cuales fueron analizadas en el Ítem 3.3.2 Subconjuntos Skyline a través de la búsqueda Bi-objetivo, pero a diferencia de lo planeado en dicho ítem, utilizamos las funciones de forma invertida, es decir,

que para el $C_\alpha(x,y)$ utilizamos la definición del $C_\beta(x,y)$ y para el $C_\beta(x,y)$ utilizamos la definición del $C_\alpha(x,y)$, esto debido a que la recta L1 de relacion de un punto determinado con el valor β y la recta L2 de relación de un punto determinado con el valor α del Ítem 3.3.2 Subconjuntos Skyline a través de la búsqueda Bi-objetivo son exactamente contrarias a las rectas definidas en el Sección 4.1 Regiones de dominancia respecto a la relacion de un punto determinado y los valores α y β.

Otro aspecto importante a en tener cuenta, es que a diferencia de lo analizado en el Ítem 3.3.2 Subconjuntos Skyline a través de la búsqueda Bi-objetivo, no se usa de la condición $\alpha + \beta > 1$, sino $\alpha + \beta \leq 1$, dado que en términos prácticos y funcionales es exactamente igual a lo analizado, donde $\alpha + \beta = 0$ es equivalente a la región de dominio de BBSk tradicional y $\alpha + \beta = 1$ corresponde a una recta, esto se definió así dado que nos permite tener una idea conceptual de ir aumentando la región de dominio de un punto en relación a los valores de α y β.

4.2.1 Funciones de Costos α y β en relación a un punto determinado

Dado un punto $p_i(x,y)$ en relación a las valores de α y β, y teniendo en cuanto lo definido en el Sección 4.1 Regiones de dominancia, las funciones de costo se definen como:

Definición 4.2: El costo $C_\alpha(p_i)$ de un punto determinado con el valor α es:

$$C_a(p_i) = (1-\alpha)x + \alpha y$$

Definición 4.3: El costo $C_\beta(p_i)$ de un punto determinado con el valor β es:

$$C_b(p_i) = \beta x + (1-\beta)y$$

Definición 4.4: La función de costo $C_{\alpha,\beta}(p_i)$ de un punto determinado con los valores α y β es:

$$C_{\alpha,\beta}(p_i) = (C_\alpha(p_i), C_\beta(p_i))$$

4.2.2 Casos notables

De acuerdo a las funciones de costos previamente definidas, se pueden evaluar varios casos notables para ver cómo interactúa un puntos $p_i(x,y)$ con los valores α y β.

Caso notable N°1: Dado un punto $p_i(x,y)$, $\alpha = 0$ y $\beta = 0$:

$$C_a(p_i) = (1-0)x + (0)y = x$$
$$C_b(p_i) = (0)x + (1-0)y = y$$
$$C_{\alpha,\beta}(p_i) = (x,y)$$

Se puede observar que los costos para un punto $p_i(x,y)$ cuando $\alpha = 0$ y $\beta = 0$ dan como resultado las coordenadas cartesianas del punto evaluado.

Caso notable N°2: Dado un punto $p_i(x, y)$, $\alpha = 1$ y $\beta = 1$:

$$C_a(p_i) = (1-1)x + (1)y = y$$
$$C_b(p_i) = (1)x + (1-1)y = x$$
$$C_{\alpha,\beta}(p_i) = (y, x)$$

Se puede observar que los costos para un punto $p_i(x, y)$ cuando $\alpha = 1$ y $\beta = 1$ dan como resultado las coordenadas cartesianas invertidas del punto evaluado.

Caso notable N°3: Dado un punto $p_i(x, y)$, $\alpha = 1$ y $\beta = 0$:

$$C_a(p_i) = (1-1)x + (1)y = y$$
$$C_b(p_i) = (0)x + (1-0)y = y$$
$$C_{\alpha,\beta}(p_i) = (y, y)$$

Se puede observar que los costos para un punto $p_i(x, y)$ cuando $\alpha = 1$ y $\beta = 0$ dan como resultado que ambos costos son iguales a la coordenada *y*.

Caso notable N°4: Dado un punto $p_i(x, y)$, $\alpha = 0$ y $\beta = 1$:

$$C_a(p_i) = (1-0)x + (0)y = x$$
$$C_b(p_i) = (1)x + (1-1)y = x$$
$$C_{\alpha,\beta}(p_i) = (x, x)$$

Se puede observar que los costos para un punto $p_i(x, y)$ cuando $\alpha = 0$ y $\beta = 1$ dan como resultado que ambos costos son iguales a la coordenada *x*.

4.3 Funciones de Costos como condición comparativa

En BBSk las condiciones comparativas entre puntos responden a una evaluación de las distancias respecto a sus posiciones y el origen de coordenadas del plano cartesiano, es decir, para la distancia Manhattan se suman las coordenadas *x* e *y* de un punto, mientras que para la distancia Euclidiana se suman los cuadrados de las coordenadas *x* e *y* de un punto [1]. Esto no nos es de utilidad para lo que se implementa dada la posibilidad de ingresar un punto al conjunto Skyline y que éste sea dominado por otro punto encontrado posteriormente, para prevenir que esto suceda se hace uso de las funciones de costos analizadas en el Ítem 3.3.2 Subconjuntos Skyline a través de la búsqueda Bi-objetivo, como condición comparativa al momento de reorganizar el heap, pues esta evalúa los costos de los puntos en relación a los valores de α y β para los ejes *x* e *y*.

4.3.1 Condición comparativa cost-Manhattan

Esta condición comparativa corresponde a la evaluación de la suma de los costos de dos puntos en relación a los valores α y β, y se define como:

Definición 4.5: Dado un punto $p_i(x_i, y_i)$, su costo Manhattan es $C_M(p_i) = C_\alpha(p_i) + C_\beta(p_i)$, entonces dado dos puntos $p_1(x_1, y_1)$ y $p_2(x_2, y_2)$, con α y β dentro del rango [0,1].

Si $C_M(p_1) > C_M(p_2)$ o más específicamente $C_\alpha(p_1) + C_\beta(p_1) > C_\alpha(p_2) + C_\beta(p_2)$, entonces se puede decir que el costo Manhattan de p_2 es menor que el costo de p_1, por lo tanto, p_2 va antes que p_1 al reorganizar el heap.

En resumen, esto consiste en evaluar qué punto posee un costo Manhattan menor, y dicho punto va antes que el otro al reorganizar el heap.

4.3.1.1 Ejemplo de comparación de dos puntos con cost-Manhattan

Dado dos puntos; $p_1(5,5)\ y\ p_2(4,8)$, $\alpha = 0.5$ y $\beta = 0.2$ la comparación de costos Manhattan entre estos dos puntos es:

$$C_a(p_1) = (1 - 0.5) * (5) + (0.5) * (5) = 5$$
$$C_b(p_1) = (0.2) * (5) + (1 - 0.2) * (5) = 5$$
$$C_M(p_1) = 5 + 5 = 10$$

$$C_a(p_2) = (1 - 0.5) * (4) + (0.5) * (8) = 6$$
$$C_b(p_2) = (0.2) * (4) + (1 - 0.2) * (8) = 7{,}2$$
$$C_M(p_2) = 6 + 7{,}2 = 13{,}2$$

Entonces, se puede observar que el valor del costo Manhattan del punto p_1 (10) es menor o igual que el costo Manhattan del punto p_2 (13,2), por cual el punto p_1 va antes que el punto p_2 al reorganizar el heap.

4.3.2 Condición comparativa cost-Euclidean

Esta condición comparativa corresponde a la evaluación de la suma de los cuadrados de los costos de dos puntos en relación a las valores α y β, y se define como:

Definición 4.6: Dado un punto $p_i(x_i, y_i)$, su costo Euclidean es $C_E(p_i) = C_\alpha(p_i)^2 + C_\beta(p_i)^2$, entonces dado dos puntos $p_1(x_1, y_1)$ y $p_2(x_2, y_2)$, con α y β dentro del rango [0,1].

Si $C_E(p_1) > C_E(p_2)$ o más específicamente $C_\alpha(p_1)^2 + C_\beta(p_1)^2 > C_\alpha(p_2)^2 + C_\beta(p_2)^2$, entonces se puede decir que el costo Euclidean de p_2 es menor que el costo de p_1, por lo tanto, p_2 va antes que p_1 al reorganizar el heap.

Otra cosa importante que agregar es que, aunque la verdadera evaluación Euclidiana incluye una raíz cuadrada, para propósitos prácticos no se utilizará, esto con el fin de facilitar el procesamiento computacional obteniendo el mismo efecto.

En resumen, esto consiste en evaluar qué punto posee un costo Euclidean menor, y dicho punto va antes que el otro al reorganizar el heap.

3.2.2.1 Ejemplo de comparación de dos puntos con cost-Euclidean

Dado dos puntos; $p_1(5,5)\ y\ p_2(4,8)$, $\alpha = 0.5$ y $\beta = 0.2$ la comparación de costos Euclidean entre estos dos puntos es:

$$C_a(p_1) = (1 - 0.5) * (5) + (0.5) * (5) = 5$$
$$C_b(p_1) = (0.2) * (5) + (1 - 0.2) * (5) = 5$$
$$C_E(p_1) = 5^2 + 5^2 = 50$$

$$C_a(p_2) = (1 - 0.5) * (4) + (0.5) * (8) = 6$$
$$C_b(p_2) = (0.2) * (4) + (1 - 0.2) * (8) = 7{,}2$$
$$C_E(p_2) = 6^2 + 7{,}2^2 = 87{,}84$$

Entonces se puede observar que el valor del costo Euclidean del punto p_1 (50) es menor o igual que el costo Euclidean del punto p_2 (87,84), por cual el punto p_1 va antes que el punto p_2 al reorganizar el heap.

4.4 Función de Costos como condición de descarte

Para BBSk [1] una de las condiciones de descarte es que un nodo o submatriz se encuentre dominada por algún punto del conjunto Skyline encontrado hasta el momento de su evaluación. Sí una submatriz es dominada, significa que esta no contiene puntos que pertenezcan al conjunto Skyline, por lo cual es descartada. Pero esta comparación para el algoritmo BBSk se realiza evaluando las coordenadas cartesianas de ambos elementos y eso no nos es útil para evaluar la dominancia de un punto ahora que utilizamos los valores de α y β como niveles de importancia en los ejes *x* e *y*, los cuales amplían la región dominada por punto y por ende el conjunto Skyline. Para ello utilizamos las funciones de costos sobre ambos elementos y luego, de igual manera que en BBSk evaluábamos coordenadas, ahora evaluamos los costos α y β de ambos elementos.

Entonces, la relación de dominancia o de Pareto dominancia $p_1 \succ p_2$ se define como:

Definición 4.7: Dado dos puntos bidimensionales p_1 y p_2, con $\alpha + \beta \leq 1$, se dice que p_1 domina a p_2, si p_1 es mejor o igual a p_2 en ambos costos α y β, y ademas es estrictamente mejor que p_2 en al menos uno de los costos α y β.

Definición 4.8: Sean p_1 y p_2 dos puntos bidimensionales, en donde $C_{\alpha,\beta}(p_1) = (C_\alpha(p_1), C_\beta(p_1))$, $C_{\alpha,\beta}(p_2) = (C_\alpha(p_2), C_\beta(p_2))$ y $\alpha + \beta \leq 1$, en donde $C_\alpha(p_1)$ y $C_\beta(p_1)$ representan el valor de los

costos α y β del punto p_1, y $C_\alpha(p_2)$ y $C_\beta(p_2)$ representan el valor de los costos α y β del punto p_2, la relación de dominancia $p_1 \succ p_2$ se define como:

$$p_1 \succ p_2 \Leftrightarrow \big(C_\alpha(p_1) \leq C_\alpha(p_2) \wedge C_\beta(p_1) \leq C_\beta(p_2)\big) \wedge \big(C_\alpha(p_1) \leq C_\alpha(p_2) \vee C_\beta(p_1) \leq C_\beta(p_2)\big)$$

Propiedad 4.1: Transitividad, dado los puntos $p_1, p_2, p_3 \in$ D, si $p_1 \succ p_2 \wedge p_2 \succ p_3 \Rightarrow p_1 \succ p_3$, esta propiedad se puede utilizar para descartar de forma oportuna un solo punto o un conjunto de puntos que están dominados por un punto p_1, y que a su vez está dominado por un nuevo punto.

Propiedad 4.2: Incomparabilidad, dado dos puntos $p_1, p_2 \in$ D, si p_1 no domina a p_2, y p_2 no domina a p_1, entonces p_1 y p_2 son incomparables ($p_1 \sim p_2$), esta propiedad ayuda a determinar si uno o más puntos pueden ser Skyline, puesto que dos puntos Skyline son incomparables entre sí.

4.4.1 Ejemplo de descarte por comparación de costos

Dado un punto Skyline $S(2,2)$ y una submatriz $N(0,4)$, con $\alpha = 0.5$ y $\beta = 0.0$, se busca evaluar si la submatriz es dominada por el punto Skyline en relación a los valores de α y β.

Primeramente, se evalúan los costos para ambos elementos.

$$C_a(S) = (1 - 0.5) * (2) + (0.5) * (2) = 2$$
$$C_b(S) = (0) * (2) + (1 - 0) * (2) = 2$$
$$C_{\alpha,\beta}(S) = (2,2)$$

$$C_a(N) = (1 - 0.5) * (0) + (0.5) * (4) = 2$$
$$C_b(N) = (0) * (0) + (1 - 0) * (4) = 4$$
$$C_{\alpha,\beta}(N) = (2,4)$$

Entonces se puede concluir que la submatriz *N*(0,4) es dominada por el punto Skyline S(2,2), debido a que el costo α de S(2,2) es igual al costo α de la submatriz N(0,4), pero el costo β de S(2,2) es menor al costo β de la submatriz N(0,4), por lo tanto la submatriz es descartada.

4.5 Conclusiones de la implementación de las Funciones de Costos

De acuerdo a lo analizado en éste capítulo, se pudo corroborar que las funciones de costos de una búsqueda Bi-objetivo, $C_\alpha(p_i)$ y $C_\beta(p_i)$, que relacionan puntos con los valores α y β, nos permite mantener el heap organizado según los valores de costos α y β de los elementos que lo componen, y ademas, la condicion de descarte por comparación de la funcion de costos $C_{\alpha,\beta}(p_i)$ nos permite aumentar la región de dominio de un punto Skyline en relación a los valores de α y β, con todo esto se busca, disminuir las regiones donde se podrían encontrar puntos del conjunto Skyline junto con disminuir la cantidad de comparaciones y el tiempo de respuesta de la consulta.

Capítulo 5

Branch and Bound Skyline k²-tree cost (BBSkc)

En el siguiente capítulo se presenta la propuesta de esta tesis para responder la consulta Skyline sobre datos almacenados en un k²-tree, utilizando los principios de la búsqueda Bi-objetivo como condición de comparación y de descarte.

El algoritmo propuesto corresponde a una adaptación del algoritmo BBSk presentado en [1]. El BBS k²-tree cost (BBSkc), que de forma similar a BBSk, trabaja con los nodos del k²-tree almacenándolos en un heap que los organiza de acuerdo a los valores de sus costos α y β. También se utilizaron condiciones de poda que permiten reducir la cantidad de nodos a revisar, siendo un aporte significativo en la eficiencia de esta consulta.

BBS k²-tree cost (BBSkc), de igual manera que BBSk comienza extrayendo la raíz del k²-tree, que corresponde a la matriz completa. Este nodo raíz es subdividido en sus k² submatrices, de las cuales solo aquellas que superen las condiciones de descarte ingresan al heap. Este heap, a diferencia de como ocurre en BBSK, ordena las submatrices de acuerdo a sus costos α y β. De forma iterativa, se procede a evaluar el nodo en el tope del heap de acuerdo a la dominancia de los puntos Skyline encontrados hasta el momento. Si este nodo en el tope del heap no es dominado por el conjunto Skyline y además corresponde a un nodo hoja, es ingresado en el conjunto Skyline. Si no es dominado por dichos puntos y corresponde a una submatriz, se subdivide en sus k² hijos, se aplican las condiciones de descarte por cada uno de ellos, y quienes las superen ingresan al heap para su posterior análisis, mientras que los hijos que sean dominados por el conjunto Skyline, son descartados.

5.1 Condiciones de comparación

Para la comparación de nodos se usa la condición analizada en el Sección 4.3 Funciones de costo como condición comparativa, lo que nos permite mantener el heap organizado de acuerdo a los valores de los costos α y β de los elementos en él. Esto nos permite evitar el ingreso de un punto al conjunto Skyline y que este sea dominado por otro punto encontrado posteriormente, ya que el tope del heap es el elemento con el costo menor en relación a su distancia del origen con los valores α y β.

Es resumen: Dado dos puntos $p_1(x_1, y_1)$ y $p_2(x_2, y_2)$, con $\alpha + \beta \leq 1$

Si $C_\alpha(p_1) + C_\beta(p_1) \leq C_a(p_2) + C_\beta(p_2)$, p_1 va antes que p_2 en el heap, debido a que su costo Manhattan es menor, esto para la comparación de costos Manhattan.

Si $C_\alpha(p_1)^2 + C_\beta(p_1)^2 \leq C_a(p_2)^2 + C_\beta(p_2)^2$, p_1 va antes que p_2 en el heap, dado que su costo Euclidean es menor, esto para la comparación de costos Euclidean.

5.2 Condiciones de descarte

Para la condición de descarte se utiliza la condicion analizada en el Sección 4.4 Función de costo como condición de descarte, lo que nos permite ampliar la región de dominio de los nodos en relación a los valores α y β. Esta condición, ademas de extender la región de dominio, nos permite disminuir las regiones donde se podrían encontrar puntos del conjunto Skyline junto con disminuir la cantidad de comparaciones y el tiempo de respuesta de la consulta.

Para BBSkc, una submatriz o nodo puede ser descartado si:

1. Se encuentra dominada por algún punto Skyline encontrado hasta el momento: Esta condición es similar a la definida para BBSk, pero a diferencia en BBSkc se comparan los costos α y β de las submatrices o nodos en vez de sus coordenadas cartesianas respecto al origen, esto fue analizado en la Sección 4.4 Función de costo como condición de descarte. En resumen, sí la submatriz es dominada por algún punto Skyline encontrado hasta el momento, significa que esta no contiene puntos que pertenezcan al conjunto Skyline, por lo cual es descartada.
2. Se encuentra vacía: Esta condición es posible gracias a las propiedades del k²-tree, que permite identificar si una submatriz está vacía de forma rápida, esto se logra accediendo a la ubicación que la representa dentro de los Bitmaps T y L mediante las operaciones fundamentales para la estructura de datos Bitmap. Si una submatriz se encuentra vacía, no contiene puntos del conjunto Skyline, por lo cual es descartada.
3. Es dominada por alguno de sus hermanos no vacíos: Esta condición es definida en el algoritmo BBSk [1] y hace referencia a una comparación entre las submatrices de un mismo padre, para mantener solo aquellas que no estén dominadas por sus hermanos no vacíos. Para esta condición se utiliza la definición de descarte de BBSk, ya que no tiene sentido utilizar la condición de descarte por comparación de costos, dado que una submatriz en una estructura k²-tree tiene solo 4 submatrices hijas. Si una submatriz es dominada por uno de sus hermanos, quiere decir que no contiene puntos Skyline, por lo tanto, es descartada. Esto reduce significativamente la cantidad de submatrices que ingresan al heap, disminuyendo con esto la cantidad de comparaciones y mejorando la eficiencia del algoritmo BBSk.

   En general, teniendo en cuenta que el origen de los ejes de coordenadas en un k²-tree se encuentra en la esquina superior izquierda, si uno de sus hermanos no vacíos se encuentra en la región sobre y a la izquierda de una submatriz, esta es descartada. Esta comparación puede realizarse con facilidad, accediendo a los Bitmaps T y L mediante las operaciones fundamentales para la estructura de datos Bitmap.

5.3 Algoritmo de Branch and Bound Skyline k²-tree cost (BBSkc)

De la misma forma que en BBSk [1], se realizan dos evaluaciones de dominancia entre cada submatriz y el conjunto *S*: en la línea 5 y 12. A diferencia de BBSk, todo nodo es ingresado al heap

(línea 13), al momento de ser extraído y evaluado respecto al conjunto *S*, se verifica si corresponde a una hoja o no (línea 06), de ser una hoja es ingresado al conjunto *S* (línea 07), de ser un nodo se analizan sus submatrices.

```
01: S = ∅
02: Insert root in heap
03: while heap not empty do
04:   Remove top entry n
05:   if n is not dominated by some point in S then
06:     if n is a leaf then
07:       Insert n into S
08      else
09:       for each child n_i in n do
10:         if n_i = is not empty then
11:           if n_i is not dominated by its sibling not empty then
12:             if n_i is not dominated by some point in S then
13:               Insert n_i into heap
14:             end if
15:           end if
16:         end if
17:       end for
18:     end if
19:   end if
20: end while
21: return S
```

Al igual que en BBSk [1] el heap se encarga de almacenar los nodos del k²-tree que pueden contener puntos del conjunto Skyline. Por cada uno de las nodos se almacena el índice que lo representa en el Bitmap T:L, las coordenadas cartesianas de su esquina superior izquierda (más cercana al origen), junto con el tamaño del nodo.

5.4 Ejemplo de ejecución de la consulta BBSkc

Para el ejemplo utilizaremos el k²-tree junto con los Bitmaps T y L de la Figura 5.1 con los valores $\alpha = 0.3$ y $\beta = 0.5$.

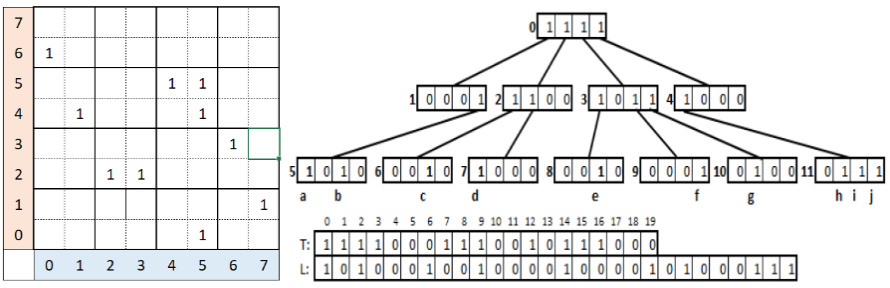

Figura 5.1: k²-tree junto con el Bitmap T:L para el ejemplo de ejecución de BBSkc.

Acción	Contenido del heap	Contenido de *S*
Inicio	(0,(0,0),0)	{}
Expande 0	(1,(0,0),0), (3,(0,4),3.2), (2,(4,0),4.8)	{}
Expande 1	(3,(0,4),3.2), (5,(2,2),4), (2,(4,0),4.8)	{}
Expande 2	(6,(0,4),3.2), (5,(2,2),4), (3,(4,0),4.8), (7,(0,6),4.8)	{}
Expande 6	(5,(2,2),4), (C,(1,4),4.4), (3,(4,0),4.8), (7,(0,6),4.8)	{}
Expande 5	(A,(2,2),4), (C,(1,4),4.4), (3,(4,0),4.8), (7,(0,6),4.8), (B,(3,2),5.2)	{}
Expande A	(C,(1,4),4.4), (3,(4,0),4.8), (7,(0,6),4.8), (B,(3,2),5.2)	{}
Expande C	(3,(4,0),4.8), (7,(0,6),4.8), B,(3,2),5.2)	(A,(2,2),4), (C,(1,4),4.4)
Expande 3	(7,(0,6),4.8), B,(3,2),5.2)	(A,(2,2),4), (C,(1,4),4.4)
Expande 7	(D,(0,6),4.8), (B,(3,2),5.2)	(A,(2,2),4), (C,(1,4),4.4)
Expande D	(B,(3,2),5.2)	(A,(2,2),4), (C,(1,4),4.4), (D,(0,6),4.8)
Expande B	{}	(A,(2,2),4), (C,(1,4),4.4), (D,(0,6),4.8)

Tabla 5.1: Ejemplo de ejecución de BBSkc

Seguimiento de la ejecución del algoritmo BBSkc, correspondiente a la Figura 5.1 y la Tabla 5.1.

1. Inicialmente dentro del heap solo se encuentra el nodo raíz (nodo 0), y el conjunto *S* se encuentra vacío.
2. Al expandir el nodo 0, se obtienen los nodos 1, 2, 3 y 4. El nodo 4 no es agregado al heap, pues este se encuentra dominado por el nodo 1. El nodo 1 tiene prioridad por sobre sus hermanos debido a que posee un costo menor que lo demás nodos. El nodo 3 es ingresado antes que el nodo 2 debido a sus costos en relación a α y β.
3. Al expandir el nodo 1, se obtiene el nodo 5, el cual es el cuarto hijo del nodo 1. Sus hermanos se encuentran vacíos por lo tanto son descartados y no ingresan al heap.
   El nodo 5 posee un costo menor que el nodo 3 y mayor que el nodo 2, por lo cual es ingresado después del nodo 2.
4. Al expandir el nodo 2, se obtienen los nodos 6 y 7, los cuales son el primer y segundo hijo del nodo 2. Sus hermanos se encuentran vacíos por lo tanto son descartados y no ingresan al heap. El nodo 6 posee menor costo que los demás nodos en el heap por lo cual es ingresado al principio de él, mientras que el nodo 7 posee un costo mayor que los demás nodos en el heap, por lo tanto, es ingresado al final del heap.
5. Al expandir el nodo 6, se obtiene el nodo C, el cual es el tercer hijo del nodo 6. Sus hermanos se encuentran vacíos por lo tanto son descartados y no ingresan al heap. El nodo C posee un costo mayor que el nodo 5 y menor que el nodo 3, por lo tanto, es agregado después del nodo 5.

6. Al expandir el nodo 5, se obtienen los nodos A y B, lo cuales corresponden al primer y tercer hijo del nodo 5 respectivamente. Sus hermanos se encuentran vacíos por lo tanto son descartados y no ingresan al heap. El nodo A posee un costo menor al resto del heap por lo cual es agregado al principio de él, mientras que el nodo B posee un costo mayor al resto del heap por lo cual es agregado al final.
7. Al evaluar el nodo A, se descubre que corresponde a un nodo hoja que no es dominado por el conjunto Skyline, entonces es agregado al conjunto Skyline.
8. Al evaluar el nodo C, se descubre que corresponde a un nodo hoja que no es dominado por el conjunto Skyline, entonces es agregado al conjunto Skyline.
9. Al evaluar el nodo 3 respecto al conjunto Skyline, se descubre que este se encuentra dominado, entonces es descartado.
10. Al expandir el nodo 7, se obtiene el nodo D, el cual no es dominado por el conjunto Skyline, entonces es agregado al heap. Sus hermanos se encuentran vacíos por lo tanto son descartados y no ingresan al heap. El nodo D posee un costo menor que el nodo B, entonces lo agrega en el tope del heap.
11. Al evaluar el nodo D, se descubre que corresponde a un nodo hoja que no es dominado por el conjunto Skyline, entonces es agregado al conjunto Skyline.
12. Al evaluar el nodo B respecto al conjunto Skyline, se descubre que este se encuentra dominado, entonces es descartado.
13. La ejecución del algoritmo termina, pues el heap se encuentra vacío.
14. Entrega el conjunto Skyline.

Como se puede observar en la Tabla 5.1, los nodos fueron ingresados en el conjunto Skyline desde el nodo con menor costo en relación α y β, hasta el nodo con costo mayor, con esto evitamos el ingreso de nodos que luego puedan ser dominados por otros puntos Skyline ingresado posteriormente. Además, se pudo constatar que a diferencia de la ejecución de BBSk en el Ítem 3.2.3.3 Ejemplo de ejecución de la consulta BBSk, el nodo *e* se encontraba dominado, por lo cual no fue ingresado en el conjunto Skyline.

Capítulo 6

Implementación y experimentación

En el siguiente capítulo se presenta la implementación y entorno de pruebas de los algoritmos, los bancos de datos utilizados en la experimentación, los resultados de la experimentación y una conclusión respectos a estos.

6.1 Implementación y entorno de pruebas

La implementación de los algoritmos es ejecutada en el lenguaje de programación C++ ISO/IEC 14882:2020 (C++20) y compilados con gcc 6.3.0, en un entorno de desarrollo Linux, con el sistema operativo Linux Mint 21.2 Cinnamon con una arquitectura de 64 bits, en una máquina virtual corriendo en MS Windows 11 con una arquitectura de 64 bits. Ésta máquina virtual tiene asignado un procesador de un mono-núcleo Intel(R) Core(TM) i3-7130U CPU @2.70GHz 2.71 GHz, con 6GB de RAM de tipo DDR3L.

Para la generación de datos se utiliza un programa que genera una n cantidad de coordenadas aleatorias sin duplicados, dentro de un rango de valores que van desde 0 a un valor límite R, estas coordenadas son almacenadas en un archivo de texto. Además, se utiliza una librería implementada en [3] la cual genera un k^2-tree junto con sus Bitmaps T y L, a partir de las coordenadas dentro de un archivo de texto, en este caso el archivo de texto plano anteriormente mencionado.

Para mostrar los datos obtenidos por pantalla se utiliza la librería “iostream”, para la utilización de vectores, matrices y listas se hace uso de la librería “numeric” y “vector”, para la creación y administración de pares de objetos, útiles cuando se deben tratar dos objetos como si fuesen uno solo, se utiliza la librería “utility”, para la creacion y administración de la estructura de datos heap se usa la librería “algorithm”, para las funciones matemáticas se hace uso de la librería “cmath”, para las mediciones de los tiempos de ejecución de los algoritmos implementados se utilizan las funciones de la librería “chrono”, todas estas librerías estan implementadas en C++. Para hacer uso de las estructuras de datos compactas k^2-tree, junto con los Bitmaps que la componen, se hace uso de la librería “kTree.h”, incluida en el directorio de codigo fuente de éste proyecto.

El algoritmo implementado BBSkc, da la posibilidad de escoger entre dos funciones de comparación para organizar el heap: cost-Manhattan y cost-Euclidean, ambas fueron definidas en la Sección 5.1 Condiciones de comparación.

6.2 Bancos de datos experimentales

Los bancos de datos sintéticos son generados utilizando el programa mencionado en la Sección 6.1 Implementación y entorno de pruebas. Los valores n y R son entregados por parámetro, junto con el nombre del archivo de texto plano en el cual son almacenados los datos generados. Los conjuntos de datos generados contienen una cantidad de puntos definida en 100.000 y 1.000.000 puntos, en un rango que oscila desde 100.000 a 1.000.000.000. Como se puede observar en la Tabla 6.1, la cantidad de puntos y el rango son limitados por las capacidades del entorno de desarrollo y pruebas. Ademas, para aclarar, archivo sin indexar se refiere al tamaño del contenedor de puntos en formato de texto plano mientras que archivo indexado se refiere al tamaño del objeto k^2-tree generado a partir del archivo en texto plano.

identificador	Puntos	Rango	Archivo sin indexar (KB)	Archivo indexado (KB)
dataset014	100000	100000	1177,8	425,9
dataset 015	100000	1000000	1377,8	600,3
dataset 016	100000	10000000	1577,8	775,0
dataset 017	100000	100000000	1777,6	949,3
dataset 018	100000	1000000000	1977,7	1123,5
dataset 019	1000000	1000000	13777,4	5130,5

Tabla 6.1: Bancos de datos experimentales.

Como se puede observar, en la Tabla 6.1 se presenta el tamaño en memoria de los archivos que almacenan los puntos de cada conjunto sin indexar vs. su correspondiente representación en el k^2-tree. Se puede ver que aquellos archivos que almacenan la representación del k^2-tree poseen un tamaño inferior en comparación con los archivos que almacenan los datos sin compactar, esto debido a que la representación k^2-tree almacena los datos en formato binario.

6.3 Experimentación

En la siguiente sección se presentan los resultados experimentales sobre el algoritmo implementado. Estos resultados muestran tanto el tiempo de ejecución como la cantidad de resultados de la consulta Skyline. Por cada una de las pruebas sobre los bancos de datos se ejecutaron 100 ciclos de la consulta Skyline, tomando el tiempo de ejecución de cada uno de los 100 ciclos, esto para obtener un tiempo mínimo y máximo, junto con un tiempo promedio de ejecución. Dado que de acuerdo a lo concluido en [1] el algoritmo BBSk con la condicion de comparación de distancia Manhattan es mucho más rápida que su variante con comparación de distancia Euclidiana, se hará pruebas con el algoritmo BBSk con dicha condición de comparación, análogamente haremos las comparaciones con el algoritmo BBSkc con la condición de comparación de costos Manhattan, esto con el fin de que las pruebas sean lo más simétricas posibles y con ello facilitar la comprensión de los resultados de la experimentación.

Primero se ejecutará la consulta BBSk con las condiciones de comparación de distancias Manhattan, sobre los dataset para verificar el tiempo de ejecución base a comparar, luego se ejecutará la consulta BBSkc con la condición de comparación de costos Manhattan, con $\alpha + \beta \leq 1$ y variando los valores de α y β, lo cual en pocas palabras nos muestra la idea conceptual de ir expandiendo la región de dominio de un punto, partiendo desde $\alpha + \beta = 0$, lo cual da como resultado el conjunto Skyline completo pues es equivalente al BBSk base, hasta $\alpha + \beta = 1$ lo que es equivalente a una recta, ésta distribución debiese de devolver un subconjunto de un solo elemento del conjunto Skyline completo, pues ésta recta relacionada con el punto óptimo descarta los demás puntos candidatos al conjunto Skyline.

6.3.1 Definiciones previas a la experimentación

En el siguiente ítem se definen los conceptos utilizados en la experimentación.

- BBSKM: Algoritmo BBSk con condición comparativa de distancia Manhattan.
- BBSKCM: Algoritmo BBSkc con condición comparativa de costos Manhattan.
- Min: Tiempo mínimo de ejecución en microsegundos.
- Max Tiempo máximo de ejecución en microsegundos.
- Promedio: Tiempo promedio de ejecución en microsegundos.
- S: Conjunto y subconjunto Skyline resultante de la operación de los algoritmos.

6.3.2 Resultados experimentales

Pruebas sobre el banco de datos: dataset014, con n = 100000 y rango = [0,100000]

Algoritmo	α	β	$\alpha + \beta$	Min	Max	Promedio	S
BBSKM	-	-	-	605	3493	835	12
BBSKCM	0.0	0.0	0.0	734	2482	938	12
BBSKCM	0.01	0.01	0.02	174	1753	295	4
BBSKCM	0.03	0.07	0.1	91	623	117	2
BBSKCM	0.09	0.11	0.2	58	465	88	2
BBSKCM	0.14	0.16	0.3	78	128	84	2
BBSKCM	0.25	0.15	0.4	47	137	72	1
BBSKCM	0.19	0.31	0.5	55	159	68	2
BBSKCM	0.35	0.25	0.6	42	100	45	1
BBSKCM	0.40	0.30	0.7	41	100	45	1
BBSKCM	0.51	0.29	0.8	41	67	42	1
BBSKCM	0.15	0.75	0.9	54	137	74	2
BBSKCM	0.50	0.50	1.0	42	101	47	1
BBSKCM	0.70	0.30	1.0	40	127	47	1
BBSKCM	0.30	0.70	1.0	46	106	48	1

Tabla 6.2: Resultados de la experimentación sobre el dataset014.

Como se puede observar en la Tabla 6.2, el tiempo promedio de ejecución del algoritmo BBSk sobre el dataset014 es de 835 microsegundos con un conjunto Skyline de 12 elementos, el tiempo promedio del algoritmo BBSkc con $\alpha + \beta = 0$ es de 938 microsegundos con también 12 elementos Skyline, y a medida que los valores de α y β aumentan, y con esto también aumenta el valor de la relación $\alpha + \beta$, lo tiempos disminuyen drásticamente hasta un tiempo de ejecución casi constante de 45 microsegundos. También se puede observar que a medida que la relación $\alpha + \beta$ aumenta, el subconjunto Skyline se reduce hasta estar compuesto por un solo elemento.

Pruebas sobre el banco de datos: dataset015, con n = 100000 y rango = [0,1000000]

Algoritmo	α	β	$\alpha + \beta$	Min	Max	Promedio	S
BBSKM	-	-	-	2184	5195	2685	13
BBSKCM	0.0	0.0	0.0	2724	5161	3268	13
BBSKCM	0.03	0.07	0.1	94	509	111	3
BBSKCM	0.09	0.11	0.2	75	379	83	2
BBSKCM	0.14	0.16	0.3	64	96	67	2
BBSKCM	0.25	0.15	0.4	66	192	69	2
BBSKCM	0.19	0.31	0.5	60	141	65	2
BBSKCM	0.35	0.25	0.6	59	102	62	2
BBSKCM	0.40	0.30	0.7	59	277	64	2
BBSKCM	0.51	0.29	0.8	59	102	63	2
BBSKCM	0.15	0.75	0.9	43	80	45	1
BBSKCM	0.50	0.50	1.0	48	73	50	1
BBSKCM	0.70	0.30	1.0	44	114	46	1
BBSKCM	0.30	0.70	1.0	43	91	45	1

Tabla 6.3: Resultados de la experimentación sobre el dataset015.

Como se puede observar en la Tabla 6.3, el tiempo promedio de ejecución del algoritmo BBSk sobre el dataset015 es de 2685 microsegundos con un conjunto Skyline de 13 elementos, el tiempo promedio del algoritmo BBSkc con $\alpha + \beta = 0$ es de 3268 microsegundos con también 13 elementos Skyline, y a medida que los valores de α y β aumentan, y con esto también aumenta el valor de la relación $\alpha + \beta$, lo tiempos disminuyen drásticamente hasta un tiempo de ejecución casi constante de 58 microsegundos. También se puede observar que a medida que la relación $\alpha + \beta$ aumenta, el subconjunto Skyline se reduce hasta estar compuesto por un solo elemento.

Pruebas sobre el banco de datos: dataset016, con n = 100000 y rango = [0,10000000]

Algoritmo	α	β	$\alpha + \beta$	Min	Max	Promedio	S
BBSKM	-	-	-	2623	5360	3075	18
BBSKCM	0.0	0.0	0.0	3342	5459	3891	18
BBSKCM	0.03	0.07	0.1	155	953	196	4
BBSKCM	0.09	0.11	0.2	121	576	154	4

BBSKCM	0.14	0.16	0.3	82	303	94	2
BBSKCM	0.25	0.15	0.4	52	547	81	1
BBSKCM	0.19	0.31	0.5	73	594	95	2
BBSKCM	0.35	0.25	0.6	69	157	78	1
BBSKCM	0.40	0.30	0.7	48	117	54	1
BBSKCM	0.51	0.29	0.8	48	498	54	1
BBSKCM	0.15	0.75	0.9	79	652	133	2
BBSKCM	0.50	0.50	1.0	48	73	49	1
BBSKCM	0.70	0.30	1.0	47	181	61	1
BBSKCM	0.30	0.70	1.0	50	118	53	1

Tabla 6.4: Resultados de la experimentación sobre el dataset016.

Como se puede observar en la Tabla 6.4, el tiempo promedio de ejecución del algoritmo BBSk sobre el dataset016 es de 3075 microsegundos con un conjunto Skyline de 18 elementos, el tiempo promedio del algoritmo BBSkc con $\alpha + \beta = 0$ es de 3891 microsegundos con también 18 elementos Skyline, y a medida que los valores de α y β aumentan, y con esto también aumenta el valor de la relación $\alpha + \beta$, lo tiempos disminuyen drásticamente hasta un tiempo de ejecución casi constante de 61 microsegundos. También se puede observar que a medida que la relación $\alpha + \beta$ aumenta, el subconjunto Skyline se reduce hasta estar compuesto por un solo elemento.

Pruebas sobre banco de datos: dataset017, con n = 100000 y rango = [0,100000000]

Algoritmo	α	β	$\alpha + \beta$	Min	Max	Promedio	S
BBSKM	-	-	-	2451	5455	2851	13
BBSKCM	0.0	0.0	0.0	3084	4924	3524	13
BBSKCM	0.03	0.07	0.1	175	945	232	5
BBSKCM	0.09	0.11	0.2	160	1268	224	5
BBSKCM	0.14	0.16	0.3	139	396	157	4
BBSKCM	0.25	0.15	0.4	109	194	118	3
BBSKCM	0.19	0.31	0.5	88	441	146	2
BBSKCM	0.35	0.25	0.6	84	255	110	2
BBSKCM	0.40	0.30	0.7	81	560	106	2
BBSKCM	0.51	0.29	0.8	82	382	98	2
BBSKCM	0.15	0.75	0.9	65	422	76	1
BBSKCM	0.50	0.50	1.0	55	368	76	1
BBSKCM	0.70	0.30	1.0	63	309	79	1
BBSKCM	0.30	0.70	1.0	55	291	59	1

Tabla 6.5: Resultados de la experimentación sobre el dataset 017.

Como se puede observar en la Tabla 6.5, el tiempo promedio de ejecución del algoritmo BBSk sobre el dataset017 es de 2851 microsegundos con un conjunto Skyline de 13 elementos, el tiempo promedio del algoritmo BBSkc con $\alpha + \beta = 0$ es de 3524 microsegundos con también 13

elementos Skyline, y a medida que los valores de α y β aumentan, y con esto también aumenta el valor de la relación $\alpha + \beta$, lo tiempos disminuyen drásticamente hasta un tiempo de ejecución casi constante de 86 microsegundos. También se puede observar que a medida que la relación $\alpha + \beta$ aumenta, el subconjunto Skyline se reduce hasta estar compuesto por un solo elemento.

Pruebas sobre el banco de datos: dataset018, con n = 100000 y rango = [0,1000000000]

Algoritmo	α	β	$\alpha + \beta$	Min	Max	Promedio	S
BBSKM	-	-	-	2158	5787	2473	7
BBSKCM	0.0	0.0	0.0	2545	6591	3305	7
BBSKCM	0.03	0.07	0.1	179	891	228	3
BBSKCM	0.09	0.11	0.2	122	736	152	3
BBSKCM	0.14	0.16	0.3	122	771	167	3
BBSKCM	0.25	0.15	0.4	92	545	117	2
BBSKCM	0.19	0.31	0.5	120	553	129	3
BBSKCM	0.35	0.25	0.6	85	566	102	2
BBSKCM	0.40	0.30	0.7	59	133	62	1
BBSKCM	0.51	0.29	0.8	57	98	60	1
BBSKCM	0.15	0.75	0.9	94	614	112	2
BBSKCM	0.50	0.50	1.0	58	427	75	1
BBSKCM	0.70	0.30	1.0	55	217	67	1
BBSKCM	0.30	0.70	1.0	58	109	62	1

Tabla 6.6: Resultados de la experimentación sobre el dataset 018.

Como se puede observar en la Tabla 6.6, el tiempo promedio de ejecución del algoritmo BBSk sobre el dataset018 es de 2473 microsegundos con un conjunto Skyline de 7 elementos, el tiempo promedio del algoritmo BBSkc con $\alpha + \beta = 0$ es de 3305 microsegundos con también 7 elementos Skyline, y a medida que los valores de α y β aumentan, y con esto también aumenta el valor de la relación $\alpha + \beta$, lo tiempos disminuyen drásticamente hasta un tiempo de ejecución casi constante de 81 microsegundos. También se puede observar que a medida que la relación $\alpha + \beta$ aumenta, el subconjunto Skyline se reduce hasta estar compuesto por un solo elemento.

Pruebas sobre el banco de datos: dataset019, con n = 1000000 y rango = [0,1000000]

Algoritmo	α	β	$\alpha + \beta$	Min	Max	Promedio	S
BBSKM	-	-	-	2610	3782	2845	12
BBSKCM	0.0	0.0	0.0	3245	5681	3637	12
BBSKCM	0.03	0.07	0.1	179	686	223	5
BBSKCM	0.09	0.11	0.2	124	390	148	5
BBSKCM	0.14	0.16	0.3	106	1126	165	4
BBSKCM	0.25	0.15	0.4	101	429	120	4
BBSKCM	0.19	0.31	0.5	88	661	117	3

BBSKCM	0.35	0.25	0.6	74	547	90	2
BBSKCM	0.40	0.30	0.7	60	169	84	1
BBSKCM	0.51	0.29	0.8	59	130	72	1
BBSKCM	0.15	0.75	0.9	57	141	70	1
BBSKCM	0.50	0.50	1.0	50	94	51	1
BBSKCM	0.70	0.30	1.0	58	116	62	1
BBSKCM	0.30	0.70	1.0	67	411	93	1

Tabla 6.7: Resultados de la experimentación sobre el dataset 019.

Como se puede observar en la Tabla 6.7, el tiempo promedio de ejecución del algoritmo BBSk sobre el dataset019 es de 2845 microsegundos con un conjunto Skyline de 12 elementos, el tiempo promedio del algoritmo BBSkc con $\alpha + \beta = 0$ es de 3637 microsegundos con también 12 elementos Skyline, y a medida que los valores de α y β aumentan, y con esto también aumenta el valor de la relación $\alpha + \beta$, lo tiempos disminuyen drásticamente hasta un tiempo de ejecución casi constante de 83 microsegundos. También se puede observar que a medida que la relación $\alpha + \beta$ aumenta, el subconjunto Skyline se reduce hasta estar compuesto por un solo elemento.

Análisis gráfico de los resultados experimentales:

Como se puede observar en la Figura 6.1, el algoritmo BBSkc es mucho más rápido al momento de identificar el subconjunto Skyline de acuerdo a los dos criterios α y β del usuario, respecto al algoritmo BBSk al momento de identificar el conjunto Skyline completo, llegando hasta un tiempo de ejecución casi constante, independientemente de la cantidad de puntos en la matriz de adyacencia.

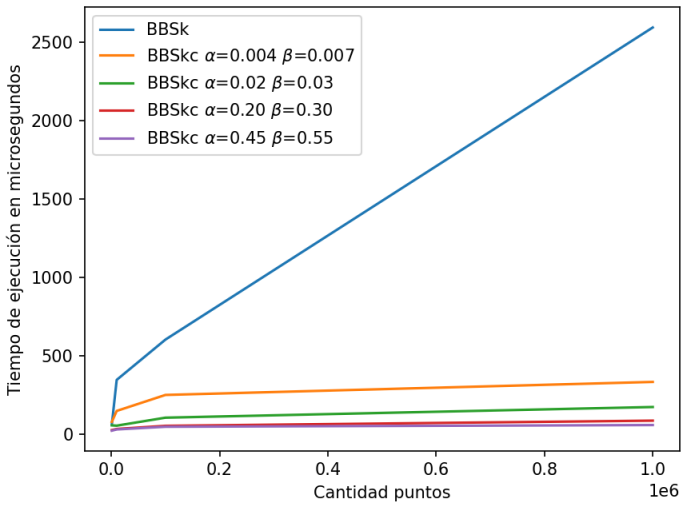

Figura 6.1: Gráfico Tiempo de ejecución vs Cantidad de puntos.

Como se puede observar en la Figura 6.6, el algoritmo BBSkc es mucho más rápido al momento de identificar el subconjunto Skyline de acuerdo a los dos criterios α y β del usuario, respecto al algoritmo BBSk al momento de identificar el conjunto Skyline completo, llegando hasta un tiempo de ejecución casi constante, independientemente de las dimensiones de la matriz de adyacencia.

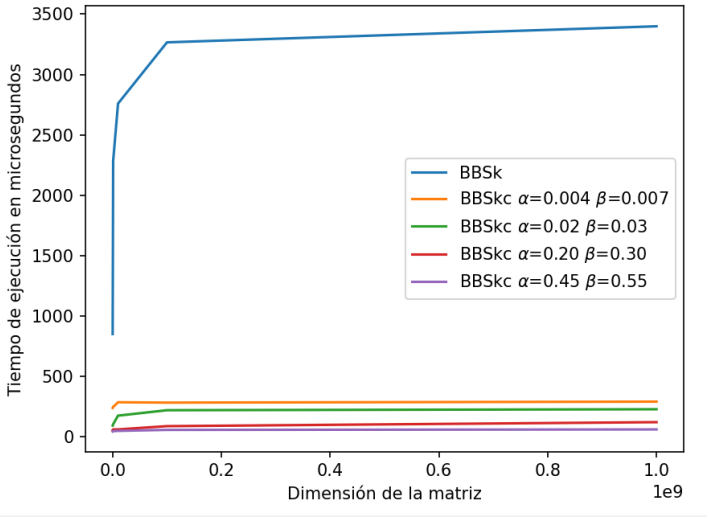

Figura 6.2: Gráfico Tiempo de ejecución vs Dimensión de la matriz.

6.4 Conclusiones de los resultados experimentales

En lo que respecta a la consulta Skyline y de acuerdo a los resultados de la experimentación, BBSkc resulta ser un algoritmo mucho más eficiente que BBSk cuando $\alpha + \beta > 0$. Para BBSkc cuando $\alpha + \beta = 0$, que es equivalente a BBSk en términos de dominancia, los tiempos de ejecución son similares aunque un poco mayores, esto puede deberse a que BBSkc requiere un par de operaciones matemáticas extra al momento de evaluar la dominancia de los elementos, pero cuando $\alpha + \beta > 0$, los tiempos de ejecución fueron reducidos drásticamente. BBSkc es un algoritmo que en la mayor parte de los experimentos no supera los 5000 microsegundos para identificar un subconjunto Skyline que cumpla con los criterios definidos, algo así como 5 milisegundos o 0.005 segundos en tiempos de ejecución, por lo cual en el peor de los casos se puede decir que BBSkc es equivalente a BBSk, pero cuando $\alpha + \beta > 0$, BBSkc es drásticamente más rápido al momento de identificar el subconjunto Skyline.

Capítulo 7

Conclusiones y trabajo futuro

En éste proyecto se diseñó, implementó, analizó y evaluó el algoritmo BBSkc, el cual nos permite responder a la consulta Skyline utilizando los principios de la aproximación de Subconjuntos relacionada a una busqueda Bi-objetivo, todo esto sobre datos almacenados en una estructura compacta k^2-tree.

El algoritmo presentado en esta tesis obtuvo resultados más que satisfactorios tras los experimentos realizados, los tiempos de respuesta han sido reducidos de forma considerable en comparación a los obtenidos de la ejecución del algoritmo BBSk, aunque por otro lado el uso de la memoria principal y la estructura compacta k^2-tree constituyen una considerable ventaja que se debe tener en cuenta a la hora de aplicarlos. También cabe recalcar que, al momento de implementar el algoritmo propuesto en el presente proyecto, además de la labor de investigación y desarrollo, se estableció una sólida base teórica y analítica sobre la estructura de datos compacta k^2-tree junto con lograr demostrar que el uso de estructuras de datos compactas como el k^2-tree, puede ser de utilidad sin ningún tipo de problema en escenarios prácticos, como el de búsqueda bi-objetivo sobre datos indexados. Por otro lado, los resultados de la consulta Skyline con búsqueda Bi-objetivo mediante la aproximación de subconjuntos, no solo corresponden a una mejora de la eficiencia y la precisión en la identificación de objetos de interés de acuerdo a una serie de criterios establecidos por el usuario, sino que también abren nuevas perspectivas para la aplicación práctica de este algoritmo en diversos contextos y problemas.

Los resultados del proyecto tienen un impacto significativo en la mejora de la toma de decisiones basada en datos complejos, posicionando esta aproximación como una herramienta valiosa y adaptable en el ámbito de la informática y la toma de decisiones.

Como trabajo futuro se puede hacer hincapié en la implementación de otras consultas de proximidad espacial sobre puntos, y la comparación de este algoritmo con otro que maneje los puntos en otra estructura compacta, tal como el Wavelet tree, el cual podría mejorar aún más el rendimiento de las consultas Skyline.

Todos los códigos de implementación, pruebas y desarrollo, junto con los bancos de datos utilizados en la etapa experimental están incluidos en el repositorio del proyecto[1], disponibles para ser utilizados en futuros trabajos, adaptaciones, modificaciones o para ampliar las funcionalidades del algoritmo BBSkc.

[1] Repositorio del Proyecto: github.com/ReyesCidBayron/BranchAndBoundSkylineK2treeCost.git

Bibliografía

[1] Dr. Gilberto Gutiérrez y Dr. Rodrigo Torres, Martita Paulina Muñoz Candia.: Cálculo de la consulta Skyline sobre Estructuras de Datos Compactas (2018)

[2] Nicolás Rivera, Jorge A. Baier, Carlos Hernández.: Subset Approximation of Pareto Regions with Bi-objective A* (2022)

[3] Ladra. S.: Algorithms and compressed data structures for information retrieval (2011)

[4] Quijada Fuentes, C.F.: Ampliación de las Capacidades de Estructuras de Datos Compactas (2017)

[5] Navarro, G.: Compact data structures: A practical approach. Cambridge University Press (2016)

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

[7] Rodrigo González, Szymon Grabowski, Veli Mäkinen, y Gonzalo Navarro.: Practical implementation of rank and select queries (2005)

[8] Gao, Y., Chen, G., Chen, L., Chen, C.: An i/o optimal and scalable skyline query algorithm. In: British National Conference on Databases. pp. 127–139. Springer (2006)

[9] Bartolini, I., Ciaccia, P., Patella, M.: Efficient sort-based skyline evaluation. ACM Transactions on Database Systems (TODS) 33(4), 31 (2008)

[10] Eng, P.K., Ooi, B.C., Tan, K.L.: Indexing for progressive skyline computation. Data & Knowledge Engineering 46(2), 169–201 (2003)

[11] Hose, K., Vlachou, A.: A survey of skyline processing in highly distributed environments. The VLDB Journal, The International Journal on Very Large Data Bases 21(3), 359–384 (2012)

[12] Peng, P., Chi-Wing, R.: Skyline queries and Pareto optimality

[13] Kalyvas, C., Tzouramanis, T.: A survey of skyline query processing. arXiv preprint ar-Xiv: 1704.01788 (2017)

[14] Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: Proceedings of the 2003 ACM SIGMOD international conference on Management of data. pp. 467–478. ACM (2003)

[15] Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Transactions on Database Systems (TODS) 30(1), 41–82 (2005)

[16] Jung, H., Han, H., Yeom, H.Y., Kang, S.: A fast and progressive algorithm for skyline queries with totally-and partially-ordered domains. Journal of Systems and Software 83(3), 429–445 (2010)

[17] Warburton, A.: Approximation of Pareto optima in multiple-objective, shortest-path problems. Operations Research, 35(1): 70–79. (1987)

[18] Perny, P., and Spanjaard, O.: Near Admissible Algorithms for Multiobjective Search. In proceedings of the 20th European Conference on Artificial Intelligence (ECAI), volume 178, 490–494 (2008)

[19] Davoodi, M.: Bi-objective path planning using deterministic algorithms. Robotics and Autonomous Systems, 93:105–115 (2017)

[20] Goldin, B.; and Salzman.: Approximate Bi-Criteria Search by Efficient Representation of Subsets of the Pareto-Optimal Frontier. In Biundo, S.; Do, M.; Goldman, R.; Katz, M.; Yang, Q.; and Zhuo, H. H., eds., Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS), 149–158. AAAI Press. (2021)

Linkografia

- Repositorio del Proyecto

github.com/ReyesCidBayron/BranchAndBoundSkylineK2treeCost.git

Anexos

Código fuente de las funciones de comparación de costos

```
// Define a Alpha y Beta como variables globales
float Alpha;
float Beta;

// Función de comparación para el montículo (heap) en relación a la Función de Costos Manhattan
bool cmpCM(pair<pair<int,int>, pair<int, int>> & a, pair<pair<int,int>, pair<int, int>> & b){
        double aCostAlpha = (((1-(Alpha))*(a.first.first))+(Alpha)*(a.first.second)));
        double aCostBeta = (((Beta)*(a.first.first))+((1-(Beta))*(a.first.second)));
        double bCostAlpha = (((1-(Alpha))*(b.first.first))+((Alpha)*(b.first.second)));
        double bCostBeta = (((Beta)*(b.first.first))+((1-(Beta))*(b.first.second)));
        return (aCostAlpha+aCostBeta)>=(bCostAlpha+bCostBeta);
}

// Función de comparación para el montículo (heap) en relación a la Función de Costos Euclidean
bool cmpCE(pair<pair<int,int>, pair<int, int>> & a, pair<pair<int,int>, pair<int, int>> & b){
        double aCostAlpha = (((1-(Alpha))*(a.first.first))+(Alpha)*(a.first.second)));
        double aCostBeta = (((Beta)*(a.first.first))+((1-(Beta))*(a.first.second)));
        double bCostAlpha = (((1-(Alpha))*(b.first.first))+((Alpha)*(b.first.second)));
        double bCostBeta = (((Beta)*(b.first.first))+((1-(Beta))*(b.first.second)));
        return pow(aCostAlpha, 2)+pow(aCostBeta, 2)>=pow(bCostAlpha, 2)+pow(bCostBeta, 2);
}
```

Código fuente del algoritmo BBSkc

```
// Define a Alpha y Beta como variables globales
float Alpha;
float Beta;

// Función principal del algoritmo BBSkc
vector<pair<int, int>> * BBSKC::skylineCost(MREP * rep, float alpha, float beta){
	Alpha = alpha;
	Beta = beta;
	vector<pair<int, int>>*skyline=new vector<pair<int, int>>();
	vector<pair<pair<int,int>, pair<int, int>>> heap;
	heap.push_back(make_pair(make_pair(0, 0), make_pair(rep->numberOfNodes, -1)));
	make_heap(heap.begin(), heap.end(), cmpCE);
	pair<pair<int, int>, pair<int, int>> r = make_pair(make_pair(0, 0),
	make_pair(rep->numberOfNodes, rep->numberOfNodes));
	pair<pair<int,int>, pair<int, int>> n, n_i;
	bool isDominatedBySkyline, isALeaf, subMatrixEmpty, isPruned, isSkyline;
	int firstChild, minX, minY, nChild, iChild;
	double aCostAlpha, aCostBeta, bCostAlpha, bCostBeta;
	while(heap.size() != 0){
		n = heap.front();
		pop_heap(heap.begin(), heap.end(), cmpCE);
		heap.pop_back();
		aCostAlpha = (((1-(alpha))*(n.first.first))+((alpha)*(n.first.second)));
		aCostBeta = (((beta)*(n.first.first))+((1-(beta))*(n.first.second)));
		isDominatedBySkyline = false;
		for(int i=0; i<skyline->size(); i++){
			bCostAlpha = (((1-(alpha))*((*skyline)[i].first))+
				((alpha)*((*skyline)[i].second)));
			bCostBeta = (((beta)*((*skyline)[i].first))+
				((1-(beta))*((*skyline)[i].second)));
			if((aCostAlpha>=bCostAlpha && aCostBeta>=bCostBeta) &&
				(aCostAlpha>bCostAlpha || aCostBeta>bCostBeta)){
				isDominatedBySkyline = true;
				break;
			}
		}
		isSkyline = false;
		if(!isDominatedBySkyline && !isSkyline && n.second.second != -1){
			isALeaf = n.second.second >= rep->bt_len;
			if(isALeaf){
				isSkyline = true;
				skyline->push_back(n.first);
			}
		}
		if(!isDominatedBySkyline && !isSkyline){
			firstChild = 0;
			if(n.second.second != -1){
				firstChild = rank1(rep->btl, n.second.second)*pow(K, 2);
			}
```

```
                    for(uint i=0; i<pow(K, 2); i++){
                            subMatrixEmpty = !isBitSet(rep->btl, firstChild+i);
                            isPruned = false;
                            for(ushort j=0; j<(i/K); j++){
                                    for(ushort l=0; l<(i%K); l++){
                                            isPruned = isPruned ||
                                                    isBitSet(rep->btl, firstChild+l);
                                    }
                            }
                            if(!subMatrixEmpty && !isPruned){
                                    minX = n.first.first+(n.second.first/K)*(i%K);
                                    minY = n.first.second+(n.second.first/K)*ceil(i/K);
                                    nChild = n.second.first / K;
                                    iChild = firstChild + i;
                                    isDominatedBySkyline = false;
                                    aCostAlpha = (((1-(alpha))*(minX)) +
                                            ((alpha)*(minY)));
                                    aCostBeta = (((beta)*(minX)) +
                                            ((1-(beta))*(minY)));
                                    for(int i=0; i<skyline->size(); i++){
                                            bCostAlpha = (((1-
                                                    (alpha))*((*skyline)[i].first)) +
                                                    ((alpha)*((*skyline)[i].second)));
                                            bCostBeta = (((beta)*((*skyline)[i].first)) +
                                                    ((1-(beta))*((*skyline)[i].second)));
                                            if((aCostAlpha>=bCostAlpha &&
                                                    aCostBeta>=bCostBeta) &&
                                                    (aCostAlpha>bCostAlpha ||
                                                    aCostBeta>bCostBeta)){
                                                    isDominatedBySkyline = true;
                                                    break;
                                            }
                                    }
                                    if(!isDominatedBySkyline){
                                            n_i = make_pair(make_pair(minX, minY),
                                            make_pair(nChild, iChild));
                                            heap.push_back(n_i);
                                    push_heap(heap.begin(), heap.end(), cmpCE);
                                    }
                            }
                    }
            }
    }
    return skyline;
}
```