Universidad del Bío-Bío, Chile

Facultad de Ciencias Empresariales

Departamento de Sistemas de Información

Aplicación web que permita manipular datos ráster a través de las operaciones del álgebra de mapas, sobre datos ráster de temperaturas pertenecientes a la región del Bio-Bío, almacenados en una estructura compacta k^3-tree

Proyecto de título presentado por Eric Patricio Herrera Herrera
para obtener el título de Ingeniero Civil en Informática
Dirigido por Dra. Mónica Caniupán Marileo

2018

Resumen

En este proyecto se presenta el desarrollo de una aplicación WEB, que permite representar, analizar y consultar datos espaciales de tipo ráster, implementando para ello el álgebra de mapas sobre datos espaciales ráster almacenados en una estructura de datos compacta k^3-tree. Las estructuras de datos compactas son estructuras de datos que utilizan poco espacio (memoria) y permiten el procesamiento de consultas de manera eficiente (sobre los datos compactados). Estas estructuras de datos han sido utilizadas en diversos escenarios logrando aumentar la capacidad de almacenamiento en memoria principal y logrando eficiencia en el cómputo de consultas. Esto permite procesar grandes cantidades de datos directamente en memoria principal, que es mucho más rápida que la memoria secundaria. Las estructuras de datos compactas se han utilizado en diferentes ámbitos, como por ejemplo, para representar documentos, para representar grafos de enlaces WEB, etc. En este proyecto las estructuras de datos compactas son usadas para representar datos espaciales de tipo ráster, donde como modo de investigación, el contexto que se da en este proyecto es el análisis de temperaturas sobre zonas pertenecientes a la región del Bio-Bío, Chile. Para esto la aplicación representa las zonas geográficas en matrices de $M \times M$, almacenando valores numéricos en su interior, sobre las diferentes ciudades presentes en esta aplicación. El usuario puede realizar una serie de consultas a través de los operadores del algebra de mapas, estas a su vez se dividen en consultas de tipo Local, Focal, Zonal y Global. Mediante los experimentos se muestra que es posible utilizar la estructura compacta k^3-tree, para almacenar datos espaciales de tipo ráster y realizar consultas sobre los datos compactados de manera eficiente en torno a espacio de almacenamiento y tiempos de ejecución de consultas.

Keywords — Datos espaciales de tipo ráster, Álgebra de mapas, Estructura de datos compacta

Agradecimientos

Primero quiero agradecer a mi profesora guía Mónica Caniupán por su confianza y respaldo en este proyecto, a Guillermo de Bernardo, académico de la Universidad de la Coruña, por su importante aporte en facilitar el código de la estructura compacta k^3-tree y contestar dudas claves sobre su funcionamiento, además de proveer datasets de prueba que sirvieron como base fundamental para la adaptación de los rasteres utilizados, a mi familia que ha estado presente en todo momento y ha sido un pilar fundamental para salir adelante en este proceso, a mis amigos que han estado en el momento más indicado para darme apoyo y subir el ánimo en momentos difíciles.

Adicionalmente este proyecto fue financiado por el grupo de investigación Algoritmos y Bases de datos (ALBA) código GI 160119/EF.

Índice de General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

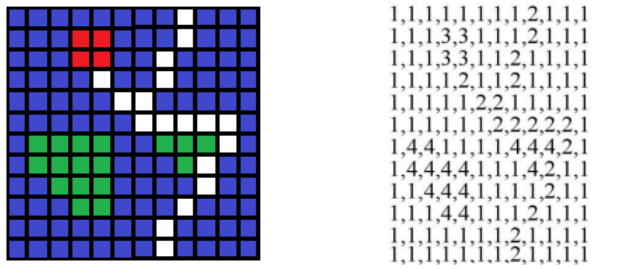
Capítulo 1

Introducción

En la actualidad existen muchas aplicaciones que utilizan datos espaciales de tipo rás-ter, como los Sistemas de información Geográfica (SIG) (Worboys y Duckham, 2004), aplicaciones basadas en la localización como Googly Maps, entre otras. Podemos encontrar los datos espaciales ráster de diferentes maneras, ya sea como rásteres en forma de mapas de superficie, rásteres en forma de mapas de clima, o rásteres en forma de atributos de una entidad. Un ráster consta de una matriz de celdas (o pixeles) organizadas en filas y columnas (o en una cuadrícula) en la que cada celda contiene un valor numérico concreto que representa información, por ejemplo información de tipo geográfico, estas pueden ser en forma de temperaturas, humedad del aire, altura del suelo, etc. Los rásteres pueden ser representados como fotografías aéreas digitales, imágenes de satélite, imágenes digitales o incluso mapas escaneados. Cada celda correspondiente al ráster tiene una coordenada x e y, que hace referencia al espacio en donde se encuentra esa celda dentro del ráster, con el fin de obtener información especifica dentro de una celda de interés. La Figura 1.1 muestra un ejemplo de como tratar este tipo de datos en una matriz de 2 dimensiones de tamaño 12×12. La Figura 1.1(a) muestra la representación que se da a cada una de las celdas dentro de la matriz de la Figura 1.1, asignando un color diferente a lo que se quiera representar, y la Figura 1.1(b) muestra el valor que tiene cada celda con respecto al ráster de la Figura 1.1(a). Si queremos darle un contexto a la Figura 1.1, podemos decir que las celdas en color azul representan agua, las celdas en color verde representan árboles, las celdas en color blanco representan piedras, y las celdas en color rojo representan flores.

La forma tradicional de procesar datos de tipo ráster consiste en el uso de estructuras de datos especiales y algoritmos para llevar los datos desde el disco a la memoria principal, mediante la explotación de la ubicación de los datos en el disco. Sin embargo, es posible, implementar estructuras de datos compactas ad-hoc para datos ráster y de esta forma utilizar de mejor manera la capacidad de almacenamiento de la memoria principal y mejorar el tiempo de respuesta de las consultas

Las estructuras de datos compactas son estructuras de datos que permiten compactar los datos sin perder la capacidad de consultar en su forma compacta, lo que permite mantener su funcionalidad y realizar consultas sobre los datos almacenados de forma eficiente. Una estructura de datos compacta combina, de manera única, una representación compri-



a) Representación de celdas b) Valor de cada celda

Figura 1.1: Representación de datos ráster dentro de una matriz de 2 dimensiones

mida de los datos y los métodos para acceder a dichos datos. Además, estas estructuras permiten procesar grandes conjuntos de datos en memoria principal, donde el tiempo de acceso disminuye con respecto a los niveles más bajos de la jerarquía de memoria (Raman et al., 2001). Las estructuras de datos compactas se han utilizado en diversos escenarios, Por ejemplo, en (Brisaboa et al., 2009) se utilizan para representar grafos de la Web, en (Navarro, 2014) para representar y procesar grandes documentos de texto, en (Vallejos et al., 2017) para representar cubos de datos de Data Warehouses (DWs), en (Brisaboa et al., 2013) se utiliza para mejorar la eficiencia de las consultas en Sistemas de Información Geográfica, entre otros.

En general, las características de los datos ráster son importantes para seleccionar la estructura de datos compacta apropiada que se va a utilizar (por ejemplo, agrupación de valores en áreas homogéneas, cantidad de diferentes valores de la variable ráster, tipo de datos de la variable ráster, entre otros).

En este proyecto proponemos el uso de la estructura de datos compacta k^3-tree, para compactar datos de tipo ráster y realizar consultas a través del algebra de mapas (Shelkhar y Chawla, 2013). Esta estructura de datos compacta fue presentada en (Bernardo, 2014), para compactar conjuntos de datos raster y almacenarlos en una matriz tridimensional. La estructura compacta k^3-tree nos permite realizar un mejor consumo de espacio y tiempo al recuperar el valor en una posición dada, esto nos beneficia en buscar grandes volúmenes de información en poco tiempo con un costo de almacenamiento razonable. En la actualidad esta estructura solo se ha implementado con operaciones de consulta básicas, tales como: operaciones para obtener el valor en una celda, obtener el valor de un conjunto de celdas en un rango espacial, obtener el valor máximo de la variable ráster, entre otras.

El resto del documento se organiza de la siguiente manera: en el Capítulo 3 se presenta el objetivo general y los objetivos específicos de este proyecto, además de los alcances y límites de este. En el capitulo 2 se presentan las consultas espaciales más típicas sobre datos espaciales de tipo ráster, y además presentamos la estructura compacta k^3-tree, explicando detalladamente su funcionamiento, así como los operadores del algebra de mapas. En el capitulo 4 se presentan los algoritmos utilizados para responder a cada uno de los operadores del algebra de mapas. En el capitulo 5 se presentan las herramientas utilizadas para la creación de nuestra aplicación web, así como fotocapturas de su interfaz y ejemplos

prácticos de cada consulta en nuestra página. En el capitulo 6 se presentan las pruebas en torno a espacio de almacenamiento y tiempo de consultas. Finalmente, en el capitulo 7 se presentan las conclusiones del proyecto y las reflexiones acerca de posibles trabajos futuros.

Capítulo 2

Objetivos

En este capitulo se presentan los distintos tipos de objetivos del proyecto, así como sus limites y alcances

2.1. Objetivo General

El objetivo general de este proyecto es implementar una aplicación web en lenguaje de programación PHP, que sea capaz de manipular datos de tipo ráster y realizar consultas, a través de las operaciones del álgebra de mapas, sobre datos ráster almacenados en la estructura de datos compacta k^3-tree. El contexto que se elige es el análisis de datos espaciales tipo ráster que almacenen las temperaturas de zonas pertenecientes a la región del Bio-Bío.

2.2. Objetivos Específicos

Los objetivos específicos del proyecto son los siguientes:

1. Analizar y comprender la estructura compacta de datos k^3-tree.
2. Implementar las operaciones del álgebra de mapas (Shelkhar y Chawla, 2013) para datos de tipo ráster sobre la estructura de datos compacta k^3-tree.
3. Implementar una aplicación Web que permita representar, consultar (a través del álgebra de mapas) datos espaciales de tipo ráster almacenados en una estructura compacta k^3-tree.
4. Seleccionar datos ráster de temperaturas sobre zonas pertenecientes a la región del Bio-Bío.
5. Experimentar con la aplicación propuesta en términos de reducción de espacio y tiempo de ejecución de consultas.

2.3. Alcances y límites

En este proyecto se contempla solamente la utilización de la estructura compacta k^3-tree, no se consideran otras estructuras de datos compactas como Ik^2-tree o Amk^2-tree que también trabajan con datos espaciales de tipo ráster. Solo consideraremos datos de las comunas de la octava región de Chile, estas son: San Pedro de la paz, Concepción, Arauco, Nacimiento, Yungay, Contulmo, Chillan, Los Ángeles, Florida y Cañete. Solo se considera el sistema operativo Ubuntu para las pruebas de la Aplicación WEB. En cuanto a los operadores del algebra de mapas, para las consultas Locales, solo se consideran operadores aritméticos y desigualdades, tales como la suma, resta, multiplicación y división y desigualdades del tipo mayor que, menor que, igual que y distinto que. No se consideran funciones trigonométricas como seno, coseno o tangente. Para las consultas de tipo Global, solo se considera un punto para el cálculo de distancia, con respecto a las celdas del ráster a considerar, no se consideran puntos que estén afuera de la matriz ráster. Para las consultas de tipo Focal solo se considera el calculo de media con respecto a un rango considerando una ventana cuadrada, no consideramos ventanas del tipo circular, Para las consultas zonales solo consideramos la clasificación de valores con respecto a 2 intervalos, no consideramos aplicar operaciones matemáticas sobre las zonas clasificadas, por ejemplo, sumar todos los valores que se encuentren en una zona específica (suma zonal).

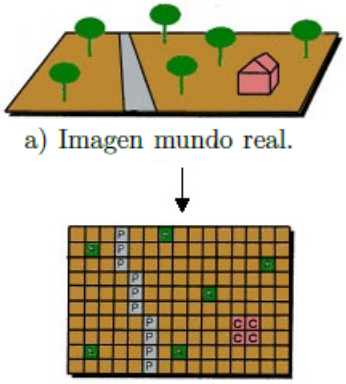
Capítulo 3

Conceptos Preliminares

En este capítulo se presentan las principales consultas sobre datos espaciales de tipo ráster, y además se presenta la estructura de datos compacta k^3-tree, describiendo en detalle cómo se construye y cómo se recorre, mostrando un caso específico para datos de tipo ráster. Finalmente se describen los operadores del álgebra de mapas, que nos servirán para realizar las distintas consultas sobre la estructura compacta k^3-tree.

3.1. Consultas Espaciales

Los datos espaciales de tipo ráster, nos permiten representar mediante celdas cualquier tipo de información o espacio pertenecientes al mundo real, ya sean el estudio de suelos, altitudes, temperaturas con respecto a una región, etc. En la Figura 3.1 podemos ver un ejemplo de cómo representar distintos objetos de la vida real dentro de un datasets ráster.



a) Imagen mundo real.

b) Representación ráster de la Figura 3.1(a).

Figura 3.1: Raster de suelo de la Figura 3.1(a)

La Figura 3.1(a) muestra la imagen de una carretera con árboles y viviendas en sus alrededores, mientras que la Figura 3.1(b) muestra la adaptación en una matriz ráster de

2 dimensiones con respecto a la imagen de la Figura 3.1(a), donde tantos los árboles, el suelo, la carretera y las viviendas son representados con diferentes colores dentro de las celdas de la matriz de la Figura 3.1(b). En este caso cada pixel representa una celda que contiene un valor especifico que puede ser consultado. Para acceder a la información que se encuentran en estos rásteres, existen una serie de consultas que nos permiten saber el valor de cada una de las celdas, o el conjunto de valores con respecto a un rango (Bernardo, 2014), estas se enumeran a continuación:

1. exists (v_1, ..., v_n, w): Verifica si alguna celda en el ráster w contiene cualquiera de los valores v_1, ...,v_n.

2. exists ([v, v_r], w): Verifica si alguna celda en el ráster w contiene algún valor en el intervalo [v, v_r].

3. report (w): Devuelve todos los valores diferentes contenidos dentro del ráster w.

4. select (v_1, ...,v_n, w): Retorna todas las ocurrencias de valores v_1, ..., v_n dentro del ráster w.

5. select ([v, v_r], w): Retorna todas las ocurrencias de valores en el rango [v, v_r] dentro del ráster w

En este proyecto nos centraremos en implementar los operadores del algebra de mapas, para realizar consultas dentro de una estructura de datos compacta k^3-tree, estas consultas pueden ser Locales, Focales, Zonales y Globales explicadas en la sección 3.3.

Ejemplo 3.1 En la Figura 3.2 se muestra el uso del operador Local que muestra las temperaturas mayores a 13 presentes en un ráster que pertenece a la ciudad de Laraquete. Las celdas en color rojo son las que cumplen la condición, mientras que las celdas en color plomo son las temperaturas menores a 13 encontradas.

12	12	12	12	12	12	12	12
12	12	12	12	12	12	12	12
12	12	12	14	13	13	13	13
12	12	13	14	14	14	13	13
13	13	14	15	15	15	14	14
14	14	15	15	15	15	15	15
15	15	15	15	15	15	15	15
15	15	15	15	15	15	15	15

Figura 3.2: Consulta Local aplicado al ráster de la zona de Laraquete

3.2. Estructura compacta k^3-tree

La estructura de datos compacta k^3-tree fue presentada por (Bernardo, 2014) para representar datos espaciales de tipo ráster en un cubo de datos binario tridimensional. Esta estructura se basa en la extensión de la estructura de datos compacta k^2-tree (Brisaboa et al., 2009) , donde se agrega una dimensión extra para su navegación. Cuando se aplica a datasets ráster, el árbol k^3-tree almacena los puntos x,y,z, donde los dos primeros valores representan la posición en el espacio $2D$, y el tercer componente es el valor almacenado en esa celda. Si queremos obtener el valor almacenado en una posición determinada, simplemente fijamos esa posición en el espacio $2D$ (x e y) y luego comprobamos el valor z correspondiente. A diferencia de la estructura compacta k^2-tree que utiliza una matriz de adyacencia para representar los datos, la estructura de datos compacta k^3-tree utiliza un cubo binario de datos tridimensional para representar los valores en 3 dimensiones.

3.2.1. Construcción de un k^3-tree

A partir de una matriz de datos tridimensional se puede construir un k^3-tree. Para la Figura 3.3(a) se muestra una matriz tridimensional con $k = 2$, donde k es la constante que indica la cantidad de particiones que se realizan en el cubo binario, siendo esta última siempre potencia de 2. El árbol k^3-tree se construye siguiendo el mismo procedimiento recursivo utilizado en la estructura compacta k^2-tree. Se crea un árbol de referencia, cuya raíz corresponde a la totalidad del árbol de la matriz, luego la matriz se subdivide y cada una de las matrices resultantes se convierten en un nodo hijo de la raíz. Este nodo se etiqueta con un 1 si la matriz resultante contiene al menos un 1, o 0 en caso contrario. Para todos los nodos que contengan al menos un 1, el proceso de descomposición se repite recursivamente hasta alcanzar los nodos hoja de la matriz. Es importante tener en cuenta el orden de recorrido de las submatrices ,en este caso elegimos un cruce de izquierda a derecha, de arriba a abajo y de adelante hacia atrás de la matriz, clasificando primero por z, luego por y (fila) y finalmente por x (columna). El Ejemplo 3.2 muestra la creación del k^2-tree a partir de un cubo binario de datos.

Ejemplo 3.2 La Figura 3.3 muestra la representación del árbol k^3-tree para una matriz tridimensional de tamaño $8 \times 8 \times 8$ (Figura 3.3(a)), considerando un valor de $k = 2$, y la Figura 3.4 muestra las primeras 4 capas que representan el cubo binario de la Figura 3.3. Dado que $k = 2$ se deben realizar 2^3 particiones en la matriz , lo que genera 8 hijos para la raíz del árbol de la Figura 3.3(b). Como modo de Ejemplo, los datos que se almacenan en el cubo binario representan puntos cartesianos en 3 dimensiones, donde se guardan con valor 1 dentro del cubo binario tridimensional, las demás celdas que no contienen información se representan con 0s, los puntos a considerar son los siguientes : (4,0,0), (5,0,0), (5,1,0), (6,3,0), (7,3,0), (7,3,2), (7,2,3) y (7,3,3) respectivamente. El cubo binario de datos tridimensional de la Figura 3.3(a) representa en color negro las celdas con valor 1 y en color blanco las celdas con valor 0. El nodo raíz del árbol de referencia (Figura 3.3(b)) tiene 8 hijos, que corresponden a las 8 submatrices de la partición, siguiendo el orden explicado en la Sección 3.2. Podemos ver que solo la segunda submatriz contiene

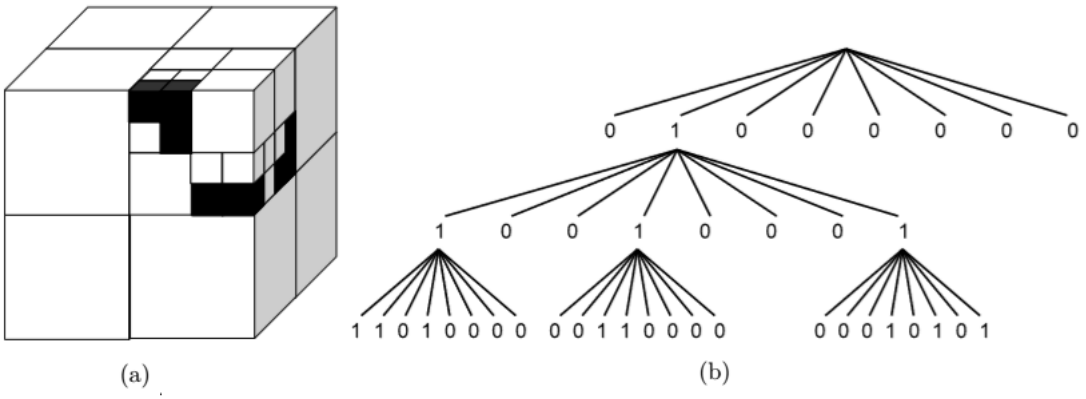


Figura 3.3: Representación del árbol k^3-tree en una matriz de 3 dimensiones

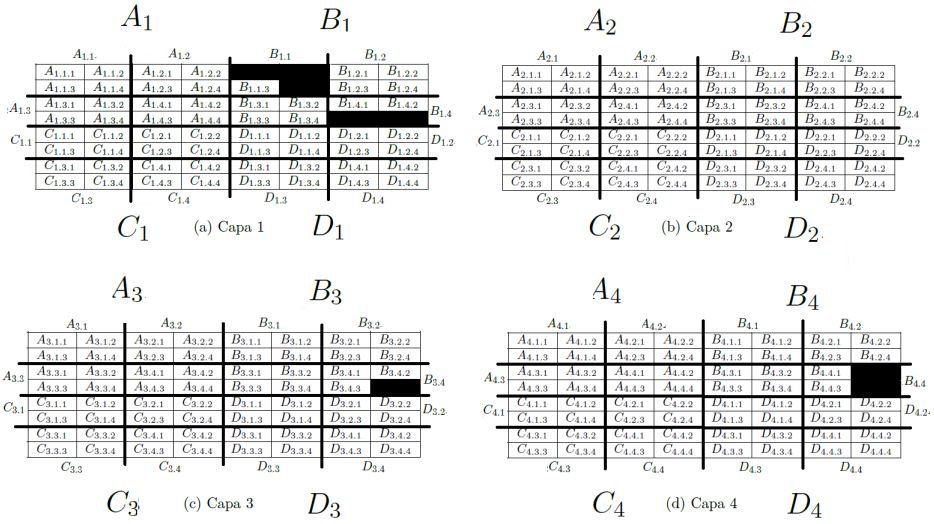


Figura 3.4: Capas que representan al cubo binario de la Figura 3.3

al menos un 1, que corresponde a la partes B_1 , B_2 , B_3 y B_4 de las primeras 4 capas de la Figura 3.4. Por lo tanto, la descomposición continua solo en esa submatriz, y sigue hasta alcanzar las celdas individuales. En la segunda iteración podemos ver que la parte B_1 (Figura 3.4(a)), las submatrices $B_{1,1}$ y $B_{1,4}$ contienen al menos un 1 y en la parte B_3 y B_4 (Figura 3.4(a),3.4(b)), podemos ver las submatrices $B_{3,4}$ y $B_{4,4}$ contienen al menos un 1. El último nivel del árbol corresponde a todas las submatrices $2 \times 2 \times 2$ que contienen al menos un 1. En este caso la submatriz $B_{1,1}$, contienen un 1 en las posiciones $B_{1,1,1}$, $B_{1,1,2}$, $B_{1,1,4}$, la submatriz $B_{1,4}$ contienen un 1 en las posiciones $B_{1,4,3}$ y $B_{1,4,4}$, la submatriz $B_{3,4}$ contienen un 1 en la posición $B_{3,4,4}$ y la submatriz $B_{4,4}$ contienen un 1 en las posiciones $B_{4,4,2}$, $B_{4,4,4}$, todo esto siguiendo el mismo orden y método recursivo para llegar a las celdas

individuales.

A partir del k^3-tree se obtienen 2 bitmaps, el primero T para representar los nodos internos y el segundo L para representar las hojas. En la Figura 3.3 se visualizan los bitmaps T y L generados a partir del k^3-tree de la Figura 3.3(b).

El árbol k^3-tree se utiliza solo como una representación visual, siendo lo importante los dos bitmaps denominados T y L (Árbol y Hojas), los cuales representan a los nodos internos y las hojas del k^2-tree, respectivamente. Esta representación está diseñada para comprimir grandes matrices, en donde existen muchas áreas de 0s (Brisaboa et al., 2009). Debemos destacar que en la implementación, T y L se unen como un solo bitmap.

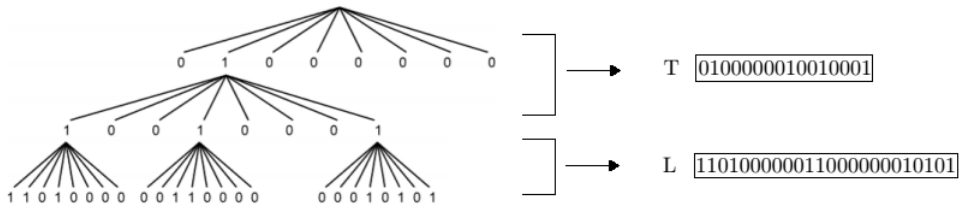


Figura 3.5: k^3-tree para la matriz de 3 dimensiones de la Figura 3.3 y bitmaps generados

3.2.2. Navegación sobre la estructura compacta k^3-tree

Para navegar sobre el árbol k^3-tree, hay que recorrer los bitmaps T y L, que parten desde la posición 1. Para esto es necesario utilizar las operaciones *Rank* y *Select* (Fariña et al., 2009), (González et al., 2005). Dado un bitmap T y una posición p.

- La operación $Rank_1(T, p)$ cuenta la cantidad de 1s (o 0s), hasta una posición p en T.
- La operación $Select_1(T, j)$ retorna la posición de la j-ésima ocurrencia de 1 (o 0) en T.

Para obtener el hijo de un nodo, se utiliza la función $Child_i(x) = Rank(T, x) \times k^3 + i$ que permite obtener el i-ésimo hijo del nodo x en el k^3-tree. En el Ejemplo 4.3 muestra la aplicación de las operaciones *Rank*, *Select* y función $Child_i(x)$ sobre los bitmaps de la Figura 3.5.

Ejemplo 3.3 Considere el k^3-tree de la Figura 3.5 y sus respectivos bitmaps T y L. La operación $Rank(T, 10) = 2$ nos indica que existen dos 1s hasta la posición 10 del bitmap (desde la posición 1). La operación $Select(T, 6) = 18$ nos indica que el sexto 1 está en la posición 18 del bitmap T, pero dado que T tiene 16 posiciones, se continúa la búsqueda en el bitmap L. Supongamos que deseamos obtener el primer hijo (las posiciones parten desde 1, por lo tanto $i = 1$) del segundo hijo de la raíz ($x = 2$) del k^3-tree, se debe aplicar $Child_1(2) = Rank(T, 2) \times k^3 + 1 = 1 \times 2^3 + 1 = 9$, lo que indica que el primer hijo del segundo hijo de la raíz se encuentra en la posición 9 del bitmap T (ver Figura 3.6).

Si se desea obtener todos los hijos del primer hijo del segundo hijo de la raíz, es decir, los hijos del nodo 9, debemos obtener la posición del primer hijo del nodo 9, por lo tanto, se obtiene $Child_1(9) = Rank(T, 9) \times k^3 + 1 = 2 \times 2^3 + 1 = 17$. Esta posición sobrepasa el tamaño del bitmap T ($|T|$) correspondiente a los nodos intermedios del árbol, entonces, el primer hijo del nodo 9 se encuentra en la posición $17 - |T| = 17 - 16 = 1$, y los siete restantes bits corresponden a los siguientes hijos del nodo 9, hasta la posición 8 del bitmap L (ver Figura 3.6).

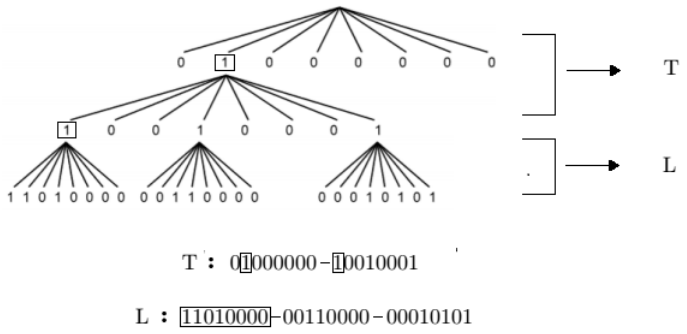


Figura 3.6: k^3-tree para el Ejemplo 3.3

3.2.3. Representación datos ráster sobre estructura compacta k^3-tree

En la sección 3 explicamos el caso de como representar puntos cartesianos en una matriz de 3 dimensiones generado por la estructura compacta k^3-tree, ahora veremos el caso de como representar cualquier dataset ráster dentro de esta estructura (Bernardo, 2014). Consideramos la Figura 3.7 con valores numéricos en su interior. Para representar estos valores dentro de la matriz de 3 dimensiones generado por la estructura compacta k^3-tree, es necesario dividir en matrices diferentes cada uno de los grupos de valores que contiene nuestra matriz ráster. Como podemos en la Figura 3.7, tenemos´numeros con valores de 12, 13, 14 y 15, en este caso vamos asignar estos grupos de valores en matrices de adyacencia diferentes (capas), y etiquetar con un 1 si se encuentran en esa posición, donde en caso contrario se asigna con un 0. Recordemos que cada nodo generado forma k^3 hijos, por lo tanto, si consideramos un $k = 2$ serán 8 hijos por cada iteración hasta llegar al final del árbol y representar cada uno de los valores encontrados en el ráster. La Figura 3.8, Figura 3.9 y Figura 3.10 muestran la construcción de un k^3-tree a partir de las matrices de adyacencia correspondientes, para la primera, segunda y tercera iteración respectivamente.

12	12	12	12	12	12	12	12
12	12	12	12	12	13	12	12
12	12	12	14	13	13	13	13
12	12	13	14	14	14	13	13
13	13	14	15	15	15	14	14
14	14	15	15	15	15	15	15
15	15	15	15	15	15	15	15
15	15	15	15	15	15	15	15

Figura 3.7: Representación de datos ráster con valores numéricos

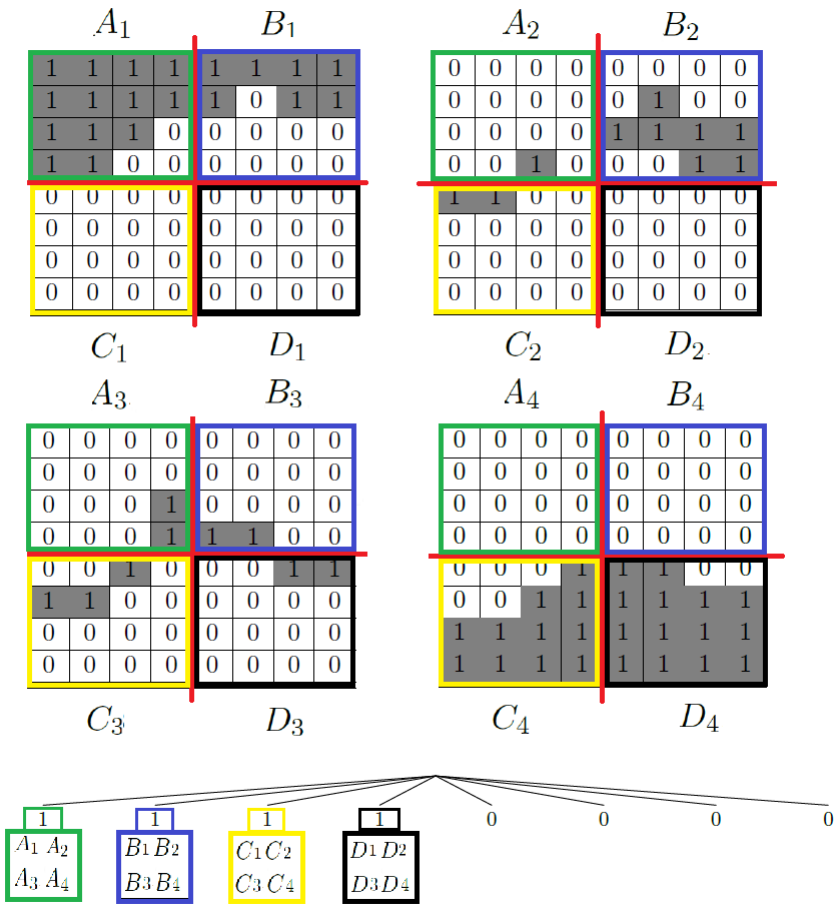


Figura 3.8: Construcción del k^3-tree para el ráster de la Figura 3.7 (Primera iteración)

Empezamos dividiendo en 4 cada una de las matrices representadas en la Figura 3.8

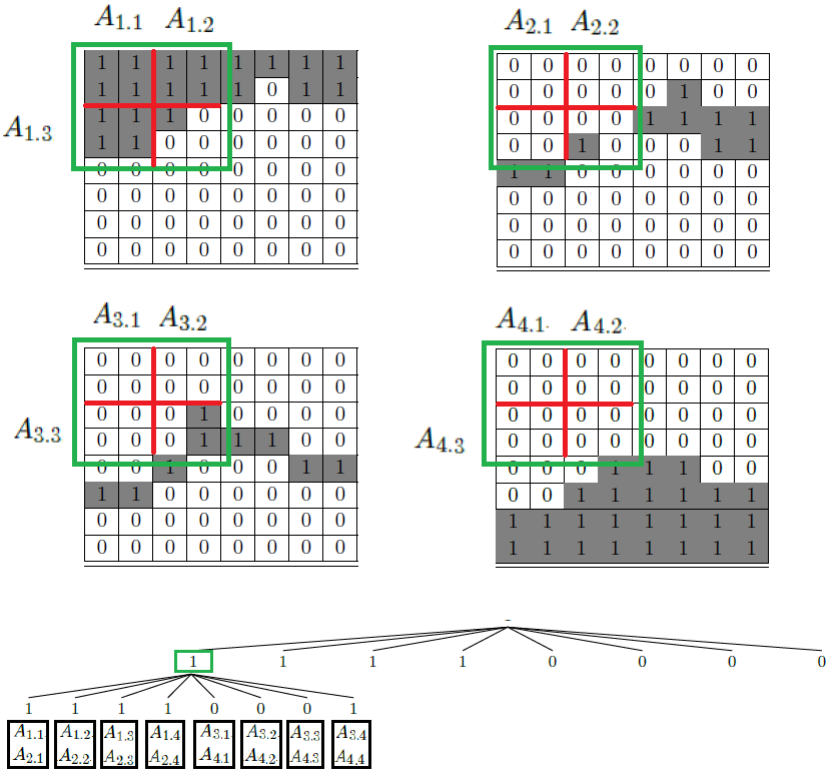


Figura 3.9: Construcción del k^3-tree para las matriz de adyacencia de la Figura 3.7 (Segunda iteración)

donde vemos si A_1, A_2, A_3 y A_4, contiene al menos un 1 en las 4 particiones , si es así se etiqueta con valor de 1 el primer nodo de la raíz del árbol, esto lo hacemos para las particiones B_1, B_2, B_3, B_4, C_1, C_2, C_3, C_4 y D_1, D_2, D_3, D_4. Si estas divisiones contienen al menos un 1, se etiqueta con valor de 1 el nodo generado por la raíz del árbol k^3-tree. La primera iteración nos queda como 1111, pero como son 2^3 hijos que deben generarse, se consideran las 4 matrices restantes que no contienen ningún valor, tal como se muestra en la Figura 3.3, en este caso nos queda como 0000, por lo tanto la primera iteración queda como 11110000. En la Figura 3.9 se muestra la segunda iteración para obtener todos los hijos primer hijo de la raíz del árbol k^3-tree. En esta iteración se consideran si las partes $A_{1,1}$ y $A_{2,1}$ contienen al menos un 1, si es así se considera como hijo del primer hijo de la raíz del árbol. Ahora hacemos lo mismo para las partes $A_{1,2}$ y $A_{2,2}$, $A_{1,3}$ y $A_{2,3}$, $A_{1,4}$ y $A_{2,4}$, $A_{3,1}$ y $A_{4,1}$, $A_{3,2}$ Y $A_{4,2}$, $A_{3,3}$ y $A_{4,3}$ y $A_{3,4}$, $A_{4,4}$. Como vemos en la Figura 3.9 los 8 primeros hijos del primer hijo de la raíz nos queda como 11110001, ahora bien, si queremos saber todos los hijos del segundo hijo de la raíz debemos aplicar el mismo procedimiento de la Figura 3.9 sobre el sector B de la Figura 3.8. Si hacemos esto para todos los hijos de la raíz nos queda como, 11110010 para B, 10001111 para C y 00001111 para D La

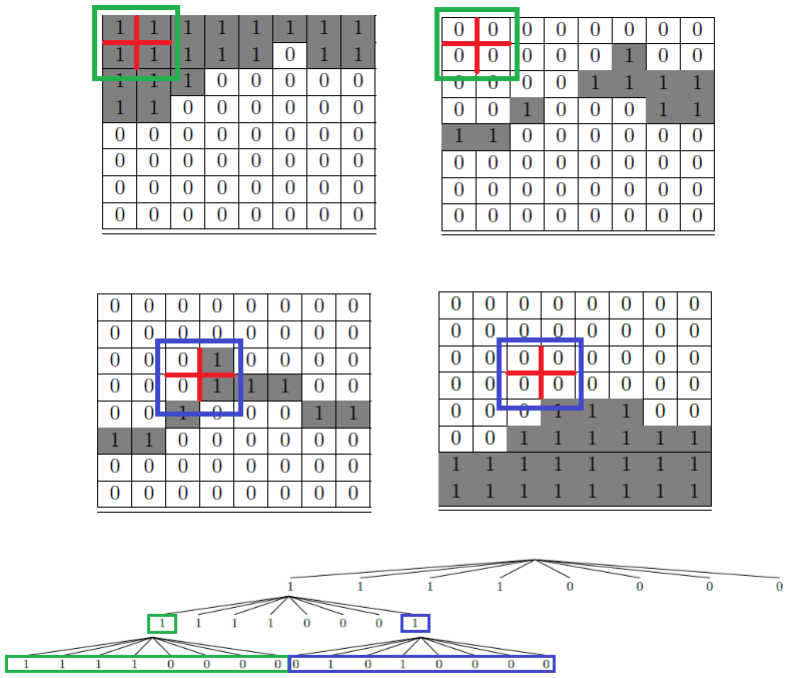


Figura 3.10: Construcción del k^3-tree para las matriz de adyacencia de la Figura 3.7 (Tercera iteración)

Figura 3.10 muestra como acceder a las hojas del árbol k^3-tree, si queremos saber todos los hijos del primer hijo de la raíz del árbol debemos recorrer las matrices de 2x2 generadas en las matrices de adyacencia que se encuentran marcadas en verde, pertenecientes a la Figura 3.10. Esto nos queda como 11110000, ahora bien si queremos saber todos los hijos del octavo hijo del primer hijo de la raíz del árbol, se deben considerar las matrices de 2x2 marcadas en azul, generadas por las matrices de adyacencia de la Figura 3.10, esto nos queda como 01010000. Si quedemos acceder a todas las hojas generadas por todos los hijos del primer hijo de la raíz se debe seguir el mismo procedimiento de la Figura 3.10 para las demás particiones de la matriz de adyacencia, en este caso nos queda como: 11110000 11110000 11110000 10000010 01010000

En la Figura 3.11 se muestra el resultado de repetir el mismo procedimiento recursivo para todas las submatrices generadas por el árbol k^3-tree, donde los bitmaps T y L son respectivamente: [11110000]−[11110001] − [11110010]−[10001111]−[00001111] para el bitmap T y [11110000]−[11110000] − [11110000] − [10000010] − [01010000] − [11100001]−[11110000] − [00001100]−[00001111]−[00110000] − [00001100]−[00110000] − [10000111]−[00001111] − [00001111]−[00001111]−[11000011] − [00001111] − [00001111] − para el bitmap L. Si queremos comprobar que están todos los datos representados en nuestro árbol, solo basta contar la cantidad de 1s que hay en L ,si es igual al numero de datos del ráster , es porque

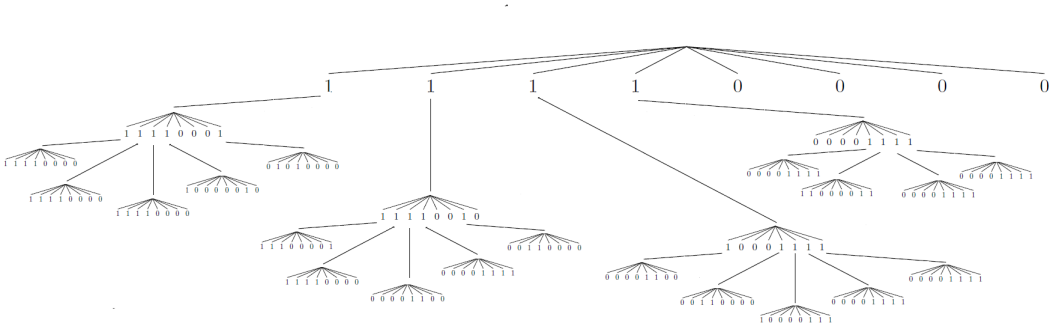


Figura 3.11: Representación de un árbol k^3-tree para la Figura 3.7

están todos los datos representados.

3.3. Operadores del álgebra de mapas

Para realizar consultas sobre datos de tipo ráster existen una serie de operaciones que nos permiten representar todo tipo de consultas, desde sumar una serie de capas para obtener un promedio final, hasta el calculo de una superficie donde una zona X sea predominante. Para esto los sistemas Gis han implementado lo que se llama operadores del álgebra de mapas (Shelkhar y Chawla, 2013),(Bernardo, 2014), que constan en un conjunto de técnicas y procedimientos que, operando sobre una o varias capas en formato ráster, nos permite obtener información derivada, generalmente en forma de nuevas capas de datos. Podemos distinguir las siguientes categorías de operaciones sobre un dataset ráster.

3.3.1. Operaciones de Tipo Local

Las operaciones locales se aplican a una sola celda del ráster, donde cada celda recibe un valor que es función de los valores de esa misma celda en los demás rasters de entrada. Estas operaciones pueden trabajar con uno o varios raster a la vez. La Figura 3.12 describe de una mejor forma esta situación para una consulta de tipo Local. Para este tipo de consultas existen una serie de operadores que pueden usarse, entre los que tenemos:

1. *Aritmético*: Estos operadores se aplican a la suma, resta, multiplicación, división, raíz cuadrada y potencia.
2. *Lógico*: Sentencias como AND, OR, XOR, NOT.
3. *Relacional*: Desigualdades como $>$, $>=$, $<=$, $==$, $!=$.
4. *Trigonométrico*: (sen, cos, tan, ...).

Si tenemos en cuenta los operadores aritméticos, estos pueden tratarse de operaciones sencillas, como la multiplicación por un escalar de una capa de altitudes en metros para

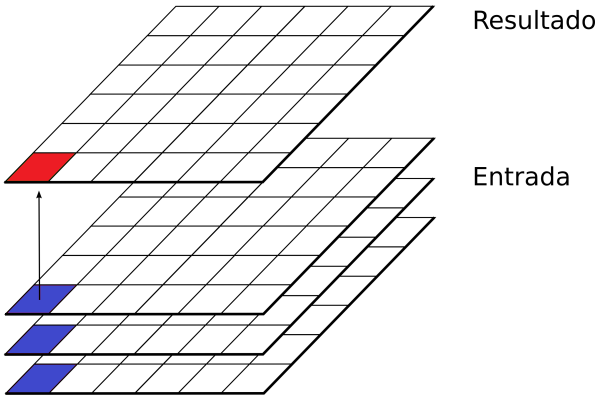


Figura 3.12: Ejemplo de consulta de tipo Local

obtener una capa de altitudes en centímetros o la suma de 12 rásteres de temperatura para obtener el promedio anual de temperaturas de dichos rásteres.

La Figura 3.13 muestra el uso del operador suma (Figura 3.13(a)), resta (Figura 3.13(b)), multiplicación (Figura 3.13(c)) y división (Figura 3.13(d)), y La Figura 3.14 muestra el uso del operador raíz cuadrada (Figura 3.14(a)) y potencia (Figura 3.14(b)), para un valor de 2 sobre todas las celdas del ráster de la Figura 3.7.

12	12	12	12	12	12	12	12
12	12	12	12	12	13	12	12
12	12	12	14	13	13	13	13
12	12	13	14	14	14	13	13
13	13	14	15	15	15	14	14
14	14	15	15	15	15	15	15
15	15	15	15	15	15	15	15
15	15	15	15	15	15	15	15

Raster original

14	14	14	14	14	14	14	14
14	14	14	14	14	15	15	15
14	14	14	16	15	15	15	15
14	14	15	16	16	16	15	15
15	15	16	17	17	17	16	16
16	16	17	17	17	17	17	17
17	17	17	17	17	17	17	17
17	17	17	17	17	17	17	17

a) Suma

10	10	10	10	10	10	10	10
10	10	10	10	10	11	10	10
10	10	10	12	11	11	11	11
10	10	11	12	12	12	11	11
11	11	12	13	13	13	12	12
12	12	13	13	13	13	13	13
13	13	13	13	13	13	13	13
13	13	13	13	13	13	13	13

b) Resta

24	24	24	24	24	24	24	24
24	24	24	24	24	26	24	24
24	24	24	28	26	26	26	26
24	24	26	28	28	28	26	26
26	26	28	30	30	30	28	28
28	28	30	30	30	30	30	30
30	30	30	30	30	30	30	30
30	30	30	30	30	30	30	30

c) Multiplicacíon

6	6	6	6	6	6	6	6
6	6	6	6	6	6.5	6	6
6	6	6	7	6.5	6.5	6.5	6.5
6	6	6.5	7	7	7	6.5	6.5
6.5	6.5	7	7.5	7.5	7.5	7	7
7	7	7.5	7.5	7.5	7.5	7.5	7.5
7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5
7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5

d) División

Figura 3.13: Operador suma, resta, multiplicación, división, sobre el raster de la Figura 3.7

12	12	12	12	12	12	12	12
12	12	12	12	12	13	12	12
12	12	12	14	13	13	13	13
12	12	13	14	14	14	13	13
13	13	14	15	15	15	14	14
14	14	15	15	15	15	15	15
15	15	15	15	15	15	15	15
15	15	15	15	15	15	15	15

Raster original

3.46	3.46	3.46	3.46	3.46	3.46	3.46	3.46
3.46	3.46	3.46	3.46	3.46	3.6	3.46	3.46
3.46	3.46	3.46	3.74	3.6	3.6	3.6	3.6
3.46	3.46	3.6	3.74	3.74	3.74	3.6	3.6
3.6	3.6	3.74	3.87	3.87	3.87	3.74	3.74
3.74	3.74	3.87	3.87	3.87	3.87	3.87	3.87
3.87	3.87	3.87	3.87	3.87	3.87	3.87	3.87
3.87	3.87	3.87	3.87	3.87	3.87	3.87	3.87

a) Raíz cuadrada

1964	1964	1964	1964	1964	1964	1964	1964
1964	1964	1964	1964	1964	169	1964	1964
1964	1964	1964	196	169	169	169	169
1964	1964	169	196	196	196	169	169
169	169	196	225	225	225	196	196
196	196	225	225	225	225	225	225
225	225	225	225	225	225	225	225
225	225	225	225	225	225	225	225

b) Potencia

Figura 3.14: Operador raíz cuadrada y potencia, sobre el raster de la Figura 3.7

3.3.2. Operadores de Tipo Focal

Los operadores de tipo Focal guardan en cada celda un valor que es función de los demás valores en un conjunto de celdas próximas, de un ráster con el mismo tratamiento de datos, estos pueden tratarse de datos altitud, temperatura, o precipitación. El conjunto de celdas próximas a la celda X más ella misma constituye una vecindad. Generalmente se trabaja con vecindades de forma cuadrada y tamaño variable, donde el tamaño se define como el número de celdas que hay en el lado del cuadrado, siempre un número impar para que exista un centro (3, 5, 7, etc.). Los ejemplos más habituales de operador de vecindad son el calculo de media con respecto a una celda y una vecindad. En la Figura 3.15 describe de una mejor forma esta situación para una consulta de tipo Focal.

Ejemplo 3.4 Considerando el ráster de la Figura 3.7, La Figura 3.16 muestra el cálculo de la media considerando una ventana de 3×3 , las celdas que contienen una x significa que no es posible calcular la media.

3.3.3. Operaciones de Tipo Zonal

Las operaciones zonales involucran celdas que comparten la misma zona o categoría, generalmente determinada por el valor de la celda. Son útiles para calcular parámetros de superficie, perímetro, índices de forma, etc, para una zona previamente conocida. La Figura 3.17 describe de una mejor forma esta situación para una consulta de tipo Zonal. Un ejemplo útil seria calcular el área total de las temperaturas mas altas almacenadas en un dataset ráster o clasificar una serie de temperaturas por zonas , ejemplo las zonas con temperaturas de 18 a 20 grados , pertenecerán a un clima caluroso.

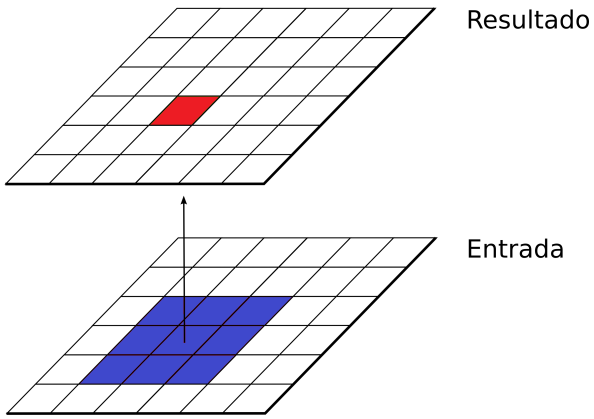


Figura 3.15: Ejemplo de consulta de tipo Focal

x	x	x	x	x	x	x	x
x	12	12.2	12.3	12.5	12.4	12.4	x
x	12.1	12.5	12.8	13.2	13	12.8	x
x	12.5	13.2	13,7	14.1	13.6	13.5	x
x	13.3	13.8	14.4	14.6	14.4	14.2	x
x	14.2	14.5	14.8	15	14.8	14.7	x
x	14.7	14.8	15	15	15	15	x
x	x	x	x	x	x	x	x

Figura 3.16: Consulta Focal con rango 3×3 aplicado al calculo de la media a la Figura 3.7

Ejemplo 3.5 Considerando la imagen de la Figura 3.7, La Figura 3.18 muestra las temperaturas de 10 a 13 grados como clima frió, y las temperaturas de 14 y 15 grados como clima templado.

s

12	12	12	12	12	12	12	12
12	12	12	12	12	13	12	12
12	12	12	14	13	13	13	13
12	12	13	14	14	14	13	13
13	13	14	15	15	15	14	14
14	14	15	15	15	15	15	15
15	15	15	15	15	15	15	15
15	15	15	15	15	15	15	15

Figura 3.18: Filtro de Temperaturas con respecto a una zona en particular

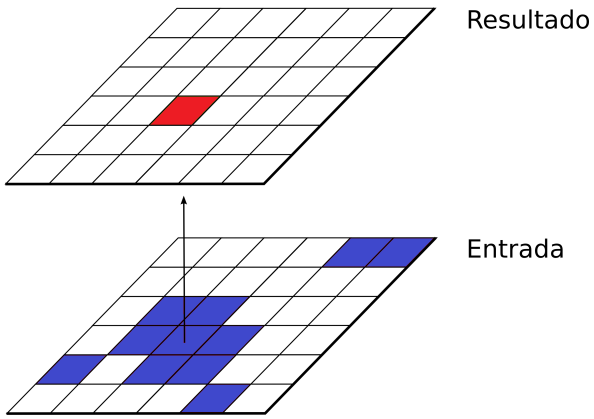


Figura 3.17: Ejemplo de consulta de tipo Zonal

3.3.4. Operadores de Tipo Global

Estos operadores afectan todos los valores del ráster de entrada. se basan en el concepto de distancia. Pueden clasificarse en distintos tipos de operadores globales.

1. *Operador distancia euclidiana*: Permite calcular para todas las celdas su distancia a una serie de celdas en una capa raster.

2. *Operador distancia ponderada*: Permite introducir el concepto de fricción que es el coste de atravesar cada celda. Si queremos determinar el camino más corto a un yacimiento en mitad de una sierra, debería tenerse en cuenta la mayor o menor facilidad para atravesar las diferentes celdas (debido a la pendiente, vegetación, presencia de caminos, etc.)

Ejemplo 3.6 Considerando la imagen de la Figura 3.7, La Figura 3.19 muestra el calculo de distancia mínima, tomando en cuenta la ultima celda de la fila 1 columna 8, como punto a considerar. la distancia que existe entre una celda y otra es 1, por lo tanto, la consulta nos queda de la siguiente manera.

7	6	5	4	3	2	1	0
7	6	5	4	3	2	1	1
7	6	5	4	3	2	2	2
7	6	5	4	3	3	3	3
7	6	5	4	4	4	4	4
7	6	5	5	5	5	5	5
7	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7

Figura 3.19: Cálculo de distancia minima aplicado a la Figura 3.7

Capítulo 4

Algoritmos para consultar sobre estructura compacta k^3-tree

En este capítulo se presentan los distintos algoritmos que responden a cada uno de los operadores del algebra de mapas, introducidos en la sección 3.3 que permiten computar consultas Locales, Focales, Zonales y Globales, correspondientes a cada uno de los operadores de álgebra de mapas, sobre una estructura de datos compacta k^3-tree. Por otra parte, en la sección 4.3 se presentan implementaciones para estos algoritmos que no utilizan estructuras de datos compactas para responder a las consultas antes descritas (sin EDC's), esto con el objetivo de generar escenarios de comparación y experimentación (descritos en el capítulo 6), con respecto a los algoritmos presentados en este proyecto que si trabajan sobre la estructura de datos compacta k^3-tree. Todos los algoritmos presentados en esta sección son de manera genérica, esto significa que pueden ser utilizados bajo cualquier tipo de dataset ráster.

4.1. Algoritmos para consultar sobre estructura compacta k^3-tree

Para realizar consultas de manera exitosa dentro de un k^3-tree R compactado, es necesario explicar el funcionamiento básico de como recuperar celdas dentro de la estructura compacta a considerar. Este método se utiliza en todos los algoritmos que utilizan la estructura de datos compactas k^3-tree para dar solución a cada uno de los operadores del algebra de mapas. En (Bernardo, 2014) se utiliza la función *getcell* como parte de la solución para obtener el valor de una celda determinada en una coordenada (x, y, z) sobre un k^3-tree R compactado.

El algoritmo 1 necesita un k^3-tree R asociado, el tamaño de la matriz tridimensional M, las coordenadas (x, y, z) de la celda a consultar, una constante k que representa el particionamiento del k^3-tree, y una constante p que guarda el posicionamiento de los bitmaps T y L en R.

Ejemplo 4.1 Consideremos que se desea buscar la posición (1,0,0) correspondiente al

Algoritmo 1: FUNCIÓN GETCELL

Input: Dimension matriz M , coordinate x , coordinate y , coordinate z , constant K , variable p , k^3-tree R
Output: cell in coordinates (x,y,z)

```
1  /*Las variables se inician como vacías*/
2  sum=0;
3  r=0;
4  x1=0;
5  y2=0;
6  z3=0;
7  cell=0;
8  if p == 0 then
9      x1 = (x/(M/k);
10     y2 = (y/(M/k);
11     z3 = (z/(M/k);
12     sum = x1 × k^0 + y2 × k^1 + z3 × k^2;
13     r = rank(T, sum) × k^3;
14     Getcell(M/k, x, y, z, k, r);
15 if p >= |T| then
16     r = p − T;
17     x1 = ((x mod M)/k);
18     y2 = ((y mod M)/k);
19     z3 = ((z mod M)/k);
20     sum = x1 × k^0 + y2 × k^1 + z3 × k^2;
21     Cell=access(L, r + sum);
22 else
23     x1 = ((x mod M)/k);
24     y2 = ((y mod M)/k);
25     z3 = ((z mod M)/k);
26     sum = x1 × k^0 + y2 × k^1 + z3 × k^2;
27     r = rank(T, p + sum) × k^3;
28     Getcell(M/k, x, y, z, k, r, R);
29 return cell;
```

raster de la Figura 3.7, mostrado en la sección 3.2.3 con valores numéricos en su interior. En las líneas 2 – 5 definimos las variables locales para recuperar una celda determinada. p debe iniciarse en 0 para empezar a recorrer la primera posición del bitmap T en R. Dependiendo del valor de p existen 3 posibilidades:

1. Si p es igual a 0 significa que estamos en la primera posición del bitmap T de R. Dividimos por $\frac{M}{k}$ cada una de las variables x, y, z ingresadas, el resultado de estas operaciones se almacena en x_1, y_2 y z_3 respectivamente, en este caso, nos queda de la siguiente forma $x_1 = 1/((8/2) = 1/4 = 0{,}25 = 0$ $y_2 = (0(8/2)) = 0$, $z_3 = (0/(8/2)) = 0$. De los resultados obtenidos, solo consideramos la parte entera de estos , en el caso de un número decimal, se aproxima al numero mas proximo , como es el caso de 0.5 que lo aproximamos a 1. Los valores de x_1 , y_2 y z_3 nos quedan respectivamente 0,0,0. En la línea 12 debemos multiplicar por k cada uno de los valores obtenidos, en este caso como estamos trabajando con la estructura compacta k^3 – tree debemos elevar a 2 a 1 y a 0 la constante k En este caso nos queda de la siguiente manera $1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2$ lo que nos da como valor 0. En la línea 12 almacenamos en *sum* el valor de la operación antes descrita, ahora procedemos a

calcular el $rank_1(T, 0) \times k^3 = 1 \times 2^3$, lo que nos queda $1 \times 8 = 8$, con esto estamos listos para calcular el nuevo valor de p , llamamos nuevamente a la función recursiva, lo que nos queda como getcell($\frac{8}{2}, 2, 0, 1, 8, R$).

2. Si $p < 0$ pero menor que T significa que estamos buscando en qué posición de T se encuentra la celda que estamos buscando, calculamos nuevamente x_1 , y_2 , y z_3, donde en vez de dividir por $\frac{n}{k}$, aplicamos el módulo a cada una de estas variables, donde el módulo se define como el resto de la división entre 2 numeros, esto nos queda representado como: $x_1 = \frac{(1 \ mod \ 4)}{2} = \frac{1}{2} = 0$, $y_2 = \frac{(0 \ mod \ 4)}{2} = 0$, $z_3 = \frac{(0 \ mod \ 4)}{2} = 0$. Ahora multiplicamos por k cada uno de los resultados obtenidos, exactamente como en la primera situación, en este caso nos queda como: $0 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 = 0 + 0 + 0 = 0$. Guardamos el *sum* el resultado de la operación, donde procedemos a calcular el $rank_1(T, 8+0) \times 2^3$. Si nos fijamos en el bitmap de la Figura 3.11 nos damos cuenta que $rank_1(T, 8) = 5$ puesto que existen 5 1s hasta la posición 8 del bitmap T , ahora procedemos a multiplicar por 8 el resultado de $rank_1(T, 8)$, lo que nos da un total de 40. Con esto estamos listos para calcular el nuevo valor de p , llamamos nuevamente a la función recursiva, lo que nos queda como Getcell($\frac{4}{2}, 2, 0, 1, 40, R$).

3. Si $p > T$, significa que estamos buscando en que posición de L se encuentra la celda correspondiente, en este caso vemos que la función nos retorna un valor de $p = 40$, que supera al valor de T que es 39 , en la línea 16 guardamos en r el nuevo valor de p que es 1. Ahora debemos calcular nuevamente x_1 , y_2 y z_3 , aplicando el modulo a cada una de estas variables y dividiendo k el resultado correspondiente , esto nos queda de la siguiente manera: $x_1 = \frac{(1 \ mod \ 2)}{2} = \frac{1}{2} = 0,5$, $y_2 = \frac{(0 \ mod \ 2)}{2} = 0$, $z_3 = \frac{(0 \ mod \ 2)}{2} = \frac{1}{2} = 0$, ahora aproximamos el valor de x_1 a su parte entera lo que nos queda como $x_1 = 1$. Ahora multiplicamos por k nuevamente el resultado obtenido por x_1, y_2 , z_3 , en este caso el resultado es de $1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 = 1$, si lo sumamos con el nuevo valor de p nos queda como $1 + 1 = 2$, donde el valor obtenido es la posición donde se encuentra la celda que estamos buscando en L. Una vez calculado este valor, accedemos a la posición con la función access($L, 2$), donde el valor de la celda en la posición $(1, 0, 0)$ es 12.

el Algoritmo retorna la variable *cell*, que corresponde al valor de la celda que se encuentra en la posición $(1, 0, 0)$, correspondiente al k^3-tree R asociado. □

4.2. Operadores del álgebra de mapas sobre estructura compacta k^3-tree

En esta sección se describen todos los algoritmos utilizados para responder a las consultas de tipo Local, Focal, Zonal y Global para datos ráster que se encuentran alojados dentro de una estructura de datos compacta k^3-tree. El Algoritmo 2, Algoritmo 3, y Algoritmo 4 resuelven consultas de tipo Local, el Algoritmo 5 resuelve consultas de tipo Focal, el Algoritmo 6 y Algoritmo 7 resuelven consultas de tipo Zonal y el Algoritmo 8 resuelve

consultas de tipo Global. El algoritmo 2 responde a consultas Locales de tipo aritméticas considerando un k^3-tree R y T asociados. Esta consulta muestra el resultado de sumar, restar, multiplicar y dividir un ráster R y T, dependiendo del operador elegido por el usuario.

20	20	20	20	20	20
20	20	20	20	20	22
20	20	20	21	21	22
20	20	22	21	21	21
22	22	21	23	23	23
21	21	23	23	23	23

Figura 4.1: Representación de datos ráster con valores numéricos (Dimensión 6×6)

Ejemplo 4.2 Supongamos que se desea calcular la resta entre los rasters de la Figura 4.1 y la Figura 3.7. En las líneas 1 y 2 declaramos las variables v11 y v22 que almacenan la celda en una posición (x,y,z), para los rasters compactados R y T respectivamente. En la línea 3 debemos comprobar cual de las 2 dimensiones es mayor, correspondientes a los rasters compactados R y T, si M1 es mayor significa que el tamaño de la matriz resultante v será de dimensión M1, de no ser así, el tamaño de la matriz v será de dimensión M2, en este caso como R tiene una dimensión de 6 y T tiene una dimensión de 8, vamos a considerar esta última por ser mayor. En las líneas 6 y 10 dependiendo de la condición, se almacena la dimensión mayor en limite1 y se crea una matriz v de dimensión $limite1 \times limite1$ lo que nos queda como v[8][8], donde v almacena el resultado entre la operación de los rásteres R y T. En la línea 11 hasta la línea 61 se recorre la matriz en 3 dimensiones considerando la mayor dimensionalidad, en este caso $8 \times 8 \times 8$, dependiendo del operador escogido por el usuario (condicional c), existen 4 posibilidades:

1. Si el operador c es igual SUMA (línea 14) se pregunta en la línea 15 si se está recorriendo dentro del ráster de menor dimensionalidad, en este caso limite2 almacena el valor de M1 = 6. Si se cumple la condición la suma entre los 2 ráster se realiza, se guarda en v11 y v22 las celdas pertenecientes a los 2 ráster en esa posición por medio de la función *getcell* (Algoritmo 1) y se guarda la suma en el arreglo v. Ahora si x e y son mayores a 6 (limite2), significa que estamos considerando solo el ráster T, puesto que el ráster R tiene una dimensión de 6×6. En este punto se necesita saber nuevamente cuál de los 2 rásteres tiene mayor dimensionalidad. En la línea 20 se pregunta si M1 es mayor a M2, si esto se cumple significa que debemos considerar solo los valores que están dentro del ráster R, para ser guardados en v, caso contrario solo vamos a considerar los valores que están dentro del ráster T para ser guardados en v. Como M2 >M1 (8>6), se consideran solo los valores del ráster T y se retorna la matriz v en la línea 62.

2. Si el operador c es igual a RESTA (línea 26), se pregunta en la línea 27 si se está recorriendo dentro del ráster de menor dimensionalidad, en este caso limite2 almacena el valor de M1 = 6. Si se cumple la condición la resta entre los 2 ráster se realiza, se guarda en v11 y v22 las celdas pertenecientes a los 2 ráster en esa posición por medio de la función *getcell* (Algoritmo 1) y se guarda la resta en el arreglo v. Ahora si x e y son mayores a 6 (limite2), significa que estamos considerando solo el ráster T, puesto que el ráster R tiene una dimensión de 6×6. En este punto se necesita saber nuevamente cuál de los 2 rasters tiene mayor dimensionalidad. En la línea 32 se pregunta si M1 es mayor a M2, si esto se cumple significa que debemos considerar solo los valores que están dentro del ráster R, para ser guardados en v, caso contrario solo vamos a considerar los valores que están dentro del ráster T para ser guardados en v. Como M2 >M1 (8>6), se consideran solo los valores del ráster T y se retorna la matriz v en la línea 62.

3. Si el operador c es igual a MULTIPLICACION (línea 38), se pregunta en la línea 39 si se está recorriendo dentro del ráster de menor dimensionalidad, en este caso limite2 almacena el valor de M1 = 6. Si se cumple la condición la multiplicación entre los 2 ráster se realiza, se guarda en v11 y v22 las celdas pertenecientes a los 2 ráster en esa posición por medio de la función *getcell* (Algoritmo 1) y se guarda la multiplicación en el arreglo v. Ahora si x e y son mayores a 6 (limite2), significa que estamos considerando solo el ráster T, puesto que el ráster R tiene una dimensión de 6 x 6. En este punto se necesita saber nuevamente cuál de los 2 rasters tiene mayor dimensionalidad. En la línea 44 se pregunta si M1 es mayor a M2, si esto se cumple significa que debemos considerar solo los valores que están dentro del ráster R, para ser guardados en v, caso contrario solo vamos a considerar los valores que están dentro del raster T para ser guardados en v. Como M2 >M1 (8>6), se consideran solo los valores del ráster T y se retorna la matriz v en la línea 62.

4. Si el operador es c igual a DIVISION (línea 50), se pregunta en la línea 51 si se está recorriendo dentro del roster de menor dimensionalidad, en este caso limite2 almacena el valor de M1 = 6. Si se cumple la condición la división entre los 2 roster se realiza, se guarda en v11 y v22 las celdas pertenecientes a los 2 roster en esa posición por medio de la función *getcell* (Algoritmo 1) y se guarda la división en el arreglo v. Ahora si x e y son mayores a 6 (limite2), significa que estamos considerando solo el roster T, puesto que el roster R tiene una dimensión de 6×6. En este punto se necesita saber nuevamente cuál de los 2 rasters tiene mayor dimensionalidad. En la línea 56 se pregunta si M1 es mayor a M2, si esto se cumple significa que debemos considerar solo los valores que están dentro del roster R, para ser guardados en v, caso contrario solo vamos a considerar los valores que están dentro del roster T para ser guardados en v. Como M2 >M1 (8>6), se consideran solo los valores del ráster T y se retorna la matriz v en la línea 62.

Como podemos ver las 4 condiciones son bastante similares entre sí, donde solo cambia el operador que vamos a usar. La Figura 4.2 muestra el resultado de restar los rásteres de

la Figura 4.1 y la Figura 3.7 respectivamente. □

8	8	8	8	8	8	12	12
8	8	8	8	8	9	12	12
8	8	8	7	9	9	13	13
8	8	9	7	7	7	13	13
9	9	7	8	8	8	14	14
7	7	8	8	8	8	15	15
15	15	15	15	15	15	15	15
15	15	15	15	15	15	15	15

Figura 4.2: Resultado de aplicar la operación resta sobre el ráster de la Figura 4.1 y la Figura 3.7

El algoritmo 3 responde a consultas Locales de tipo aritméticas dentro de un k^3-tree R asociado. Esta consulta muestra el resultado de sumar, restar, multiplicar y dividir un raster, ingresando un numero q y operador elegido por el usuario.

Ejemplo 4.3 Supongamos que se desea calcular la suma de un numero igual a 25 dentro del ráster de la Figura 3.7. En la linea 1 definimos una matriz $v[]$ que guarda los valores alojados dentro de un k^3-tree R asociado. En la linea 2 definimos una variable $v1$ que guarda el valor de una celda alojada en una posición (x, y, z) dentro de un k^3-tree R. En las líneas 3 - 5 recorremos la matriz tridimensional para acceder a las celdas correspondientes. En la línea 6 se llama al Algoritmo 1 para guardar en $v1$ el valor de la celda en la posición (x, y, z) dentro de un ráster R alojado en una estructura compacta k^3-tree. Dependiendo del operador existen 4 posibilidades:

1. Si el operador c es igual a SUMA, en la linea 8 guardamos en el arreglo v el valor de $v1$, que corresponde al valor de R en la posición (x, y, z) En la linea 9 sumamos el valor de q mas el contenido del arreglo v y lo almacenamos en el mismo arreglo v.

2. Si el operador c es igual a RESTA, En la línea 11 guardamos en el arreglo v el valor de $v1$, en este caso el valor de R en la posición (x, y, z), donde en la linea 12 restamos el valor de q con v y guardamos su contenido en el arreglo v

3. Si el operador c es igual a MULTIPLICACIÓN, en la línea 14 guardamos en el arreglo v el valor de $v1$, en este caso el valor de R en la posición (x, y, z), donde en la linea 15 multiplicamos el valor de q por el contenido del arreglo v y lo guardamos en v.

4. Si el operador c es igual a DIVISION, en la línea 17 guardamos en el arreglo v el valor de $v1$, en este caso el valor de R en la posición (x, y, z), donde en la linea 18 dividimos el valor de q por el contenido del arreglo v y lo guardamos en v.

Como podemos apreciar las 4 condiciones son bastante similares entre si, donde solo cambia el operador que vamos a usar, en este caso los valores almacenados en el arreglo v al aplicar el operador suma con un valor de q igual 25 son los que se muestran en la Figura 4.3. □

37	37	37	37	37	37	37	37
37	37	37	37	37	38	37	37
37	37	37	39	38	38	38	38
37	37	38	39	39	39	38	38
38	38	39	40	40	40	39	39
39	39	40	40	40	40	40	40
40	40	40	40	40	40	40	40
40	40	40	40	40	40	40	40

Figura 4.3: Resultado de aplicar la operación suma para un x igual a 25 sobre el ráster de la Figura 3.7

El algoritmo **??** muestra los valores mayores, menores, iguales o distintos, con respecto a un número q a partir de un operador elegido por el usuario, esta función necesita un k^3-tree R asociado, el tamaño M de la matriz tridimensional y el valor del número x.

Ejemplo 4.4 Supongamos que se desea calcular los valores mayores 12 dentro del ráster de la Figura 3.7. En las líneas 2 - 4 recorremos la matriz tridimensional para acceder a las celdas correspondientes. En la línea 5 se llama al Algoritmo 1 para guardar en v1 el valor de la celda en la posición (x, y, z) dentro del ráster R. Dependiendo del operador existen 4 posibilidades:

1. Si el operador c es igual a MAYOR, preguntamos por el valor de q si es mayor a $v1$. Si se cumple la condición guardamos el valor en arreglo v.

2. Si el operador c es igual a MENOR, preguntamos por el valor de q si es menor a $v1$. Si se cumple la condición guardamos el valor en arreglo v.

3. Si el operador c es igual a IGUAL, preguntamos por el valor de q si es igual a $v1$. Si se cumple la condición guardamos el valor en arreglo v.

4. Si el operador c es igual a DISTINTO, preguntamos por el valor de q si es distinto a $v1$. Si se cumple la condición guardamos el valor en arreglo v.

Como podemos ver las 4 condiciones son bastante similares entre si, donde solo cambia el operador que vamos a usar. La Figura 4.5 muestra los valores almacenados en el arreglo v al usar el operador MAYOR. □

El Algoritmo 5, muestra la media con respecto a un rango q ingresado por el usuario.

0	0	0	0	0	0	0	0
0	0	0	0	0	13	0	0
0	0	0	14	13	13	13	13
0	0	13	14	14	14	13	13
13	13	14	15	15	15	14	14
14	14	15	15	15	15	15	15
15	15	15	15	15	15	15	15
15	15	15	15	15	15	15	15

Figura 4.4: Representación de datos ráster con valores numéricos

Ejemplo 4.5 Supongamos que se desea calcular la media con un rango de 5 con respecto a los datos del ráster de la Figura 3.7.

1. En las líneas 1 - 8 definimos las variables que vamos a utilizar. En la linea 10 recorremos todas las celdas del ráster R alojados en la estructura compacta k^3-tree. En la linea 13 llamamos al algoritmo 1 y guardamos en $v1$ la posición (x, y, z) correspondiente a la celda alojada en R. En la linea 15 guardamos en la matriz v el valor de $v1$, al terminar el ciclo, en la linea 16, la matriz v contiene todos los valores del ráster R alojados en la estructura compacta k^3-tree. En la linea 16 guardamos en *dim* el valor del rango ingresado por el usuario, en este caso *dim* tiene el valor de 5.

2. En la linea 17 se inicia un ciclo while, donde se ejecuta siempre y cuando *dim* sea mayor o igual que 3. Cada vez que entre a este ciclo se reducirá en 2 unidades la variable *dim*, donde *cont* almacena las veces que se ha ejecutado el ciclo, en este caso hasta que *dim* sea menor o igual a 3.

3. La variable *cont* almacena los limites que tendrá la matriz para calcular la media en una posición (x, y) en el raster R de la Figura 3.7. En la linea 29 se recorre nuevamente la matriz v, para calcular la media con respecto a una posición (x, y), ahora debemos escoger una celda que cumpla las características necesarias dependiendo del rango ingresado por el usuario, si $q = 3$, serán 9 las celdas representadas para el calculo de media , si $q = 5$ serán 25, en este caso si nos posicionamos en la celda $(0, 0)$ vemos que no es posible realizar el calculo de media, puesto que si vemos la condición de la linea 21, nos damos cuenta que la celda $(0, 0)$ no se encuentra en los intervalos permitidos para realizar la operación, en este caso *xey* deben ser mayores que *cont* y menores a M - *cont*, en este caso las celdas candidatas sen encuentran en las posiciones (2,2), hasta la posición (5,5) de la matriz v. Al llegar a la celda (2,2), vemos que cumple con la condición y procede a realizar el calculo de la media.

4. En la linea 22 se recorre la matriz que rodea a la celda (2,2), en este caso se recorre desde la celda (0,0) hasta la celda (4,4), lo que da un total de 25 valores incluyendo la celda (2,2). En la linea 25 se almacena la suma de todas las variables que se

encuentran en esa matriz. En la linea 24 se guarda en la matriz $focal$ el resultado del calculo de media para la celda (2,2), donde la variable *suma* almacena el resultado del calculo de media, donde al terminar el ciclo for de la linea 25 se divide la suma total por el *rango* al cuadrado y se iguala a cero la variable suma en la línea 26. El algoritmo termina retornando la matriz $focal$, que corresponde a todas las celdas de la matriz v donde fue posible calcular la media con respecto a una posición (x, y).

En la Figura 4.5, vemos el resultado de la matriz $focal$ para el calculo de la media para el ráster de la Figura 3.7. □

x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	12.72	12.92	13.06	13.08	x	x
x	x	13.24	13.48	13.64	13.68	x	x
x	x	13.84	14.08	14.24	14.28	x	x
x	x	14.32	14.52	14.64	14.64	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x

Figura 4.5: Resultado de aplicar la operación Focal para un rango q igual 5 sobre el ráster de la Figura 3.7

El Algoritmo 6, muestra la clasificación por zonas con respecto a los valores del ráster R en la estructura de datos compacta k^3-tree. Este Algoritmo necesita un ráster R compactado y la Dimension de la matriz tridimensional M.

Ejemplo 4.6 Supongamos que se desea clasificar por zonas el ráster de la Figura 3.7. En las líneas 3 - 5 el algoritmo recorre la matriz tridimensional, donde al llegar a la línea 6 se almacena en $v1$ el valor de la celda en la posición (x, y, z) correspondiente al ráster R compactado por la estructura k^3-tree. Este Algoritmo tiene la capacidad de clasificar en 3 tipos de zonas.

1. Si el valor de $v1$ se encuentra entre 0 o 12, entonces se clasifica por Z1, donde Z1 significa que pertenece a una zona de tipo fría.

2. Si el valor de $v1$ se encuentra entre 12 o 20, entonces se clasifica por Z2, donde Z2 significa que pertenece a una zona de tipo templada.

3. Si el valor de $v1$ se encuentra entre 20 o 40, entonces se clasifica por Z3, donde Z3 significa que pertenece a una zona de tipo calurosa.

Finalmente el Algoritmo en la línea 13 retorna el arreglo v con las zonas clasificadas dependiendo del valor de las celdas del ráster R. En la Figura 4.6 se muestra las zonas clasificadas con respecto a la Figura 3.7. □

Z1	Z1	Z1	Z1	Z1	Z1	Z1	Z1
Z1	Z1	Z1	Z1	Z1	Z2	Z1	Z1
Z1	Z1	Z1	Z2	Z2	Z2	Z2	Z2
Z1	Z1	Z2	Z2	Z2	Z2	Z2	Z2
Z2	Z2	Z2	Z2	Z2	Z2	Z2	Z2
Z2	Z2	Z2	Z2	Z2	Z2	Z2	Z2
Z2	Z2	Z2	Z2	Z2	Z2	Z2	Z2
Z2	Z2	Z2	Z2	Z2	Z2	Z2	Z2

Figura 4.6: Clasificación por zonas correspondientes al ráster de la Figura 3.7

El Algoritmo 7 clasifica por zonas ingresando un intervalo p y q a elección del usuario. A diferencia del Algoritmo 6, esta consulta nos permite clasificar considerando un intervalo de valores personalizados para un ráster R compactado a considerar.

Ejemplo 4.7 Supongamos que se desea clasificar por zona "ZC" al ráster de la Figura 4.1 donde p tendrá un valor de 20 y q tendrá un valor de 22. En la línea 1 declaramos la matriz v de tamaño M z M, donde M tendrá un valor de 6. En las líneas 3 – 8 se recorre la matriz tridimensional correspondiente al ráster R compactado. En la línea 6 almacenamos en v1 el valor de la celda del ráster R en una posición (x,y,z) por medio de la función getcell correspondiente al Algoritmo 1. En la línea 7 se comprueba si v1 se encuentra en los intervalos ingresados (20 - 22), si la condición se cumple, se almacena en v el valor de "ZC", caso contrario, se iguala a cero la posición de dicha celda. El algoritmo finaliza en la línea 11 retornando la matriz v correspondiente. En la Figura 4.7 se muestra las zonas clasificadas por "ZC" correspondientes a la Figura 3.7. □

0	0	0	0	0	0
0	0	0	0	0	ZC
0	0	0	ZC	ZC	ZC
0	0	ZC	ZC	ZC	ZC
ZC	ZC	ZC	0	0	0
ZC	ZC	0	0	0	0

Figura 4.7: Resultado de filtrar por zonas con un p y q igual a 20 y 22 respectivamente, con respecto a la Figura 3.7

El Algoritmo 8 calcula la distancia mínima existente entre cada celda considerando un peso z (distancia a considerar), una celda en coordenadas (x,y), y un ráster R compactado por la estructura de datos compacta k^3-tree. Esta consulta nos permite saber a nivel espacial a que distancia se encuentran los objetos representados en un ráster si lo llevamos a un plano de la vida real. Como modo de estudio podemos decir que escogemos una

celda que representa la mayor temperatura de un mapa climático, y queremos saber a que distancia se encuentra de las demás celdas que contienen un menor valor, con esto podemos saber a que distancia se encuentran los focos de mayor o menor temperatura, considerando que estamos analizando un mapa geográfico. Esta consulta no necesita de todos los datos (valores) que presente el ráster compactado a consultar, más bien necesita solo de la celda que estamos considerando para el cálculo de distancia.

Ejemplo 4.8 Supongamos que se desea calcular la distancia mínima existente considerando el ráster R de la Figura 3.7, un peso z igual a 2 y la celda con coordenadas (4,4). Este algoritmo se divide en 2 partes principales. La primera consta de un ciclo for que recorre las celdas que se encuentran en la parte superior a la celda (4,4) (línea 5) y otro ciclo for que recorre las celdas que se encuentran por debajo a la celda (4,4) (línea 37). Comenzaremos explicando el funcionamiento del ciclo for que recorre las celdas que se encuentran en la parte superior a la celda (4,4). En la líneas 1-4 declaramos las variables iniciales a utilizar.

1. En la línea 5 se recorre la matriz v desde la celda (4,4) hasta la celda (0,4). Al encontrarnos en la celda (4,4), la sentencia de la línea 6 nos dice que, si se cumple la condición, significa que estamos en la primera posición a analizar, en este caso asignamos la celda (4,4) con distancia de 0. Ahora, como se cumple la condición de la línea 6, pasamos a recorrer la matriz v en el ciclo for de la línea 8, esta instrucción recorre todas las celdas encontradas al lado izquierdo de la celda (4,4) (celda (4,3), (4,2), (4,1), (4,0)). En este caso se pregunta nuevamente por la primera posición y en la línea 10 asignamos el valor de cero en la celda (4,4). Ahora para el resto de celdas el cálculo de distancia será el resultado de la suma de la celda siguiente más un peso z, tal como se muestra en la línea 12.

2. Ahora si estamos en la segunda iteración del ciclo for de la línea 8, significa que estamos en la celda (4,3), donde si aplicamos la fórmula de la línea 12, nos queda como v[4][3] = V[4][4] + 2 = 2, puesto que la celda (4,4) es igual a cero y solo consideramos el peso z que tiene un valor de 2, ahora si consideramos la celda (4,2) el calculo de distancia nos queda como: v[4][2] = V[4][3] + 2 = 4, esto lo hacemos para las celdas (4,1) y (4,0), donde nos queda una distancia de 6 y 8 respectivamente. Con esto termina el ciclo for de la línea 8 y entramos en la segunda iteración del ciclo for de la línea 5. En este punto la condición de la línea 6 no se cumple, puesto que i es distinto que x (3 distinto que 4), con esto pasamos a analizar la instrucción de la línea 14, donde guardamos en la matriz v la distancia que existe entre la celda (3,4) y (4,4) lo que nos queda como v[3][4] = v[4][4] + 2 = 2, donde la distancia es de 2.

3. Ahora en el ciclo for de la línea 15 se calculan todas las celdas que se encuentran al lado derecho de la celda (3,4), y en el ciclo for de la línea 25 vamos a calcular todas las celdas que se encuentran al lado izquierdo de la celda (3,4). En el primer caso preguntamos en la línea 16 si estamos en la primera iteración del ciclo for de la línea 15, si es así guardamos en la línea 17 en la variable *valor* la distancia de la celda (3,4) con respecto a la celda (4,4), ahora guardamos en *cont*2 el contenido de la variable

valor en la línea 18. Ahora si estamos en la segunda iteración significa que estamos analizando la celda (3,5), como no se cumple la condición de la línea 16, pasamos directamente a la condición de la línea 20, en este caso como c*ont*2 = 2, cumple la condición y almacena en la línea 21 el contenido de la variable *valor* que representa la distancia entre la celda (3,5) y la celda (4,4) que es de 2, ahora en la línea 22 se reduce en *z* la variable *cont*2, lo que nos queda como *cont*2 igual a 0.

4. Ahora en la siguiente iteración consideramos la celda (3,6), en este caso como *cont*2 es igual a cero, pasa a la condición de la línea 23, donde en la línea 24 se almacena el valor de la distancia de la celda (3,6), con respecto a la celda (4,4), lo que nos queda como v[3][6] = v[3][5] + 2 = 4, puesto que la celda (3,5) tiene un valor de 2. En la siguiente iteración tenemos la celda (3,7), donde se aplica la misma formula de la línea 24, lo que nos da una distancia de 6 con respecto a la celda (4,4).

5. Una vez terminado el ciclo for de la línea 15, vamos a entrar al ciclo for de la línea 25 para calcular las celdas que se encuentran al lado izquierdo de la celda (3,4) (celda (3,3), (3,2), (3,1), (3,0)). Al empezar el ciclo preguntamos en la línea 26 si estamos en la primera iteración del ciclo for de la línea 25, si es así guardamos en la línea 27 en la variable *valor* la distancia de la celda (3,4) con respecto a la celda (4,4), ahora guardamos en *cont*2 el contenido de la variable *valor* en la línea 28. Ahora si estamos en la segunda iteración significa que estamos analizando la celda (3,3), como no se cumple la condición de la línea 26, pasamos directamente a la condición de la línea 30, en este caso como *cont*2 = 2, cumple la condición y almacena en la línea 31 el contenido de la variable *valor* que representa la distancia entre la celda (3,3) y la celda (4,4), que es de 2, ahora en la línea 32 se reduce en z la variable *cont*2, lo que nos queda como *cont*2 igual a 0.

6. Ahora en la siguiente iteración consideramos la celda (3,2), en este caso como *cont*2 es igual a cero, pasa a la condición de la línea 33, donde en la línea 34 se almacena el valor de la distancia de la celda (3,2) con respecto a la celda (4,4), lo que nos queda como v[3][2] = v[3][3] + 2 = 4, puesto que la celda (3,3) tiene un valor de 2. En la siguiente iteración tenemos la celda (3,1), donde se aplica la misma fórmula de la línea 34, lo que nos da una distancia de 6 con respecto a la celda (4,4). Para la celda (3,0) se realiza el mismo procedimiento donde la distancia con respecto a la celda (4,4) es de 8.

7. Con esto estamos en la tercera iteración del ciclo for de la línea 5, donde para la celda (2,4), se debe realizar el mismo procedimiento descrito anteriormente, el ciclo for de la línea 15 calcula las celdas que se encuentran al lado derecho de la celda (2,4) (celda (2,5),(2,6),(2,7)), y el ciclo for de la línea 25 calcula las celdas que se encuentran al lado izquierdo de la celda (2,4) (celda (2,3),(2,2),(2,1),(2,0)). Para las iteraciones que quedar del ciclo for de la línea 5, se debe hacer lo mismo para las celdas (1,4) y (0,4). La Figura 4.8 muestra como queda la matriz *v* al terminar todas las iteraciones del ciclo for de la línea 5.

8. En la línea 37 empieza el ciclo for que recorre desde la celda (4,4) hasta la celda (7,4), donde en la línea 38 se pregunta si estamos en la primera iteración, si la condición se cumple se asigna un cero a la posición (4,4) y se recorre todas las celdas que se encuentran a la derecha de la celda (4,4) (celda (4,5),(4,6),(4,7)), esto lo podemos ver reflejado en el ciclo for de la línea 40, donde en la primera iteración la distancia será de 0, y en la segunda iteración se calcula la distancia para la celda (4,5) con respecto a la celda (4,4), esto nos queda como v[4][5] = v[4][4] + 2 = 2 lo que significa que la distancia entre la celda (4,5) y (4,4) es 2.

9. Para las celdas faltantes se hace lo mismo y queda como una distancia de 4 para la celda (4,6) y una distancia de 6 para la celda (4,7), con respecto a la celda (4,4). Ahora nos encontramos en la segunda iteración del ciclo for de la línea 37, donde en la línea 46 se calcula la distancia para la celda (5,4) con respecto a la celda (4,4), esto nos queda como: v[5][4] = v[4][4] + 2 = 2, donde el ciclo for de la línea 48 calcula el valor de las distancias pertenecientes a todas las celdas que se encuentran a la derecha de la celda (5,4) y el ciclo for de la línea 58 calcula el valor de las distancias pertenecientes a todas las celdas que se encuentran a la izquierda de la celda (5,4). Como podemos observar el ciclo for de la línea 15 es equivalente al ciclo for de la línea 48 y el ciclo for de la línea 25 es equivalente al ciclo for de la línea 58, en este caso si aplicamos el funcionamiento del ciclo for de la línea 15 para obtener las distancias pertenecientes a todas las celdas que se encuentran a la derecha de la celda (5,4), nos queda como: distancia de 2 para la celda (5,5), distancia de 4 para la celda (5,6), y distancia de 6 para la celda (5,7).

10. Ahora si aplicamos el funcionamiento del ciclo for de la línea 25 para obtener las distancias pertenecientes a todas las celdas que están a la izquierda de la celda (5,4), nos queda como: distancia de 2 para la celda (5,3), distancia de 4 para la celda (5,2), distancia de 6 para la celda (5,1), y distancia de 8 para la celda (5,1). Con esto nos encontramos en la tercera iteración del ciclo for de la línea 37, donde para calcular las distancias de las celdas (6,4) y (7,4) se realiza lo mismo explicado en el ciclo for de la línea 5.

11. Una vez terminado el ciclo for de la línea 37, vamos a buscar que valor representa la posición (x, y) dentro del ráster R a considerar, en este caso guardamos en $v1$, el valor de la posición (x, y, z) donde se encuentra la celda (x, y) de interés. Para esto se usa al Algoritmo 1 para encontrar la celda correcta (función getcell), donde una vez encontrada en la línea 39 se rompe el ciclo for y retornamos la matriz v en la línea 73. La Figura 4.9 muestra como queda la matriz v al calcular todas las distancias con respecto a la celda (4,4), con un valor de $v1$ igual a 15 y peso z igual a 2. □

4.3. Operadores del álgebra de mapas sin EDC's

En esta sección se presentan soluciones para implementar los operadores del álgebra de mapas que no presentan el uso de la estructura de datos compacta k^3-tree (sin EDC's),

8	8	8	8	8	8	8	8
8	6	6	6	6	6	4	6
8	6	4	4	4	4	4	6
8	6	4	2	2	2	4	6
8	6	4	2	X	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figura 4.8: Resultado de la matriz v al terminar ciclo for perteneciente a la línea 5 del Algoritmo 8

8	8	8	8	8	8	8	8
8	6	6	6	6	6	4	6
8	6	4	4	4	4	4	6
8	6	4	2	2	2	4	6
8	6	4	2	15	2	4	6
8	6	4	2	2	2	4	6
8	6	4	4	4	4	4	6
8	6	6	6	6	6	6	6

Figura 4.9: Resultado de la matriz v al calcular todas las distancias con respecto a la celda (4,4), perteneciente al ráster de la Figura 3.7

esto con el propósito de generar escenarios de experimentación en tiempo de ejecución de consultas. La particularidad de estos algoritmos es que acceden a la información por medio de archivos de texto (formato txt), sin ningún tipo de compactación. El Algoritmo 9, Algoritmo 10 y Algoritmo 11 responden a consultas de tipo local , el Algoritmo 12 responde a consultas de tipo Focal, el Algoritmo 13 y Algoritmo 14 responden a consultas de tipo Zonal y el Algoritmo 15 responde a consultas de tipo Global.

Como podemos ver el Algoritmo 9, Algoritmo 10, Algoritmo 11, Algoritmo 12, Algoritmo 13, Algoritmo 14, Algoritmo 15, solo cambia la obtención de los datos de entrada y el recorrido que se hace a la matriz que almacena los datos, en este caso para dar solución a estos Algoritmos solo es necesario recorrer considerando una fila y columna, no como en el caso anterior que es necesario recorrer la matriz tridimensional considerando coordenadas (x, y, z).

Algoritmo 2: Obtener suma, resta, multiplicación y división con respecto a un ráster R y T

```
   Input: k^3-tree R , k^3-tree T , conditional c , dimensión M1 , dimensión M2
   Output: matriz v[][]
 1 v11 = 0;
 2 v22 = 0;
 3 if M1 > M2 then
 4 |   limite1 = M1;
 5 |   Limite2 = M2;
 6 |   v[limite1][limite1] =← createArrayList();
 7 else
 8 |   limite1 = M2;
 9 |   limite2 = M1;
10 |   v[limite1][limite1] =← createArrayList();
11 for z = 0; z < limite1; z + + do
12 |   for x = 0; x < limite1; x + + do
13 |   |   for y = 0; y < limite1; y + + do
14 |   |   |   if c == SUMA then
15 |   |   |   |   if (x <limite2) AND (y <limite2) then
16 |   |   |   |   |   v11 = getcell(M1, x, y, z, 2, 0, R);
17 |   |   |   |   |   v22 = getcell(M2, x, y, z, 2, 0, T);
18 |   |   |   |   |   v[x][y] = v11 + v22;
19 |   |   |   |   else
20 |   |   |   |   |   if M1 > M2 then
21 |   |   |   |   |   |   v11 = getcell(limite1, x, y, z, 2, 0, R);
22 |   |   |   |   |   |   v[x][y] = v11;
23 |   |   |   |   |   else
24 |   |   |   |   |   |   v12 = getcell(limite1, x, y, z, 2, 0, T);
25 |   |   |   |   |   |   v[x][y] = v12;
26 |   |   |   |   if c == RESTA then
27 |   |   |   |   |   if (x <limite2) AND (y <limite2) then
28 |   |   |   |   |   |   v11 = getcell(limite1, x, y, z, 2, 0, R);
29 |   |   |   |   |   |   v22 = getcell(limite2, x, y, z, 2, 0, T);
30 |   |   |   |   |   |   v[x][y] = v11 − v22;
31 |   |   |   |   |   else
32 |   |   |   |   |   |   if M1 > M2 then
33 |   |   |   |   |   |   |   v11 = getcell(limite1, x, y, z, 2, 0, R);
34 |   |   |   |   |   |   |   v[x][y] = v11;
35 |   |   |   |   |   |   else
36 |   |   |   |   |   |   |   v12 = getcell(limite1, x, y, z, 2, 0, T);
37 |   |   |   |   |   |   |   v[x][y] = v12;
38 |   |   |   |   if c == MULTIPLICACION then
39 |   |   |   |   |   if (x <limite2) AND (y <limite2) then
40 |   |   |   |   |   |   v11 = getcell(M1, x, y, z, 2, 0, R);
41 |   |   |   |   |   |   v22 = getcell(M2, x, y, z, 2, 0, T);
42 |   |   |   |   |   |   v[x][y] = v11 + v22;
43 |   |   |   |   |   else
44 |   |   |   |   |   |   if M1 > M2 then
45 |   |   |   |   |   |   |   v11 = getcell(limite1, x, y, z, 2, 0, R);
46 |   |   |   |   |   |   |   v[x][y] = v11;
47 |   |   |   |   |   |   else
48 |   |   |   |   |   |   |   v12 = getcell(limite1, x, y, z, 2, 0, T);
49 |   |   |   |   |   |   |   v[x][y] = v12;
50 |   |   |   |   |   if c == DIVISION then
51 |   |   |   |   |   |   if (x <limite2) AND (y <limite2) then
52 |   |   |   |   |   |   |   v11 = getcell(limite1, x, y, z, 2, 0, R);
53 |   |   |   |   |   |   |   v22 = getcell(limite2, x, y, z, 2, 0, T);
54 |   |   |   |   |   |   |   v[x][y] = v11 − v22;
55 |   |   |   |   |   |   else
56 |   |   |   |   |   |   |   if M1 > M2 then
57 |   |   |   |   |   |   |   |   v11 = getcell(limite1, x, y, z, 2, 0, R);
58 |   |   |   |   |   |   |   |   v[x][y] = v11;
59 |   |   |   |   |   |   |   else
60 |   |   |   |   |   |   |   |   v12 = getcell(limite1, x, y, z, 2, 0, T);
61 |   |   |   |   |   |   |   |   v[x][y] = v12;
62 return v[][];
```

Algoritmo 3: Obtener suma, resta, multiplicación, división con respecto a un numero Q

```
   Input: k^3-tree R , value q , conditional c , Dimensión M
   Output: matriz v[][]
 1 v[M][M] =← createArrayList();
 2 v1 = 0;
 3 for z = 0; z < M; z++ do
 4     for x = 0; x < M; x++ do
 5         for y = 0; y < M; y++ do
 6             v1 = getcell(M, x, y, z, 2, 0, R);
 7             if c == SUMA then
 8                 v[x][y] = v1;
 9                 v[x][y] = v[x][y] + q;
10             if c == RESTA then
11                 v[x][y] = v1;
12                 v[x][y] = v[x][y] − q;
13             if c == MULTIPLICACION then
14                 v[x][y] = v1;
15                 v[x][y] = v[x][y] × q;
16             if c == DIVISION then
17                 v[x][y] = v1;
18                 v[x][y] = v[x][y]/q;
19 return v[][];
```

Algoritmo 4: Obtener valores mayores, menores, iguales o distintos, con respecto a un numero Q

```
   Input: k^3-tree R , Value q , Conditional c , Dimension M
   Output: matriz v[][]
 1 v[M][M] =← createArrayList();
 2 v1 = 0;
 3 for z = 0; z < M; z++ do
 4     for x = 0; x < M; x++ do
 5         for y = 0; y < M; y++ do
 6             v1 = getcell(M, x, y, z, 2, 0, R);
 7             if c == MAYOR then
 8                 if q > v1 then
 9                     v[x][y] = v1;
10             if c == MENOR then
11                 if q < v1 then
12                     v[x][y] = v1;
13             if c == IGUAL then
14                 if q == v1 then
15                     v[x][y] = v1;
16                 if c == DISTINTO then
17                     if q! = v1 then
18                         v[x][y] = v1;
19 return v[][];
```

Algoritmo 5: Calcular la media con respecto a un rango Q

```
   Input: k^3-tree R , dimension M , range q
   Output: Matriz focal[][] con la media de valores
 1 /*Las variables se inician como vacías*/
 2 focal[M][M] ← createArrayList();
 3 v[][] ← createArrayList();
 4 cont = 0;
 5 suma = 0;
 6 dim = 0;
 7 v1 = 0;
 8 cont = 0;
 9 for z = 0; z < M; z++ do
10     for x = 0; x < M; x++ do
11         for y = 0; y < M; y++ do
12             v1 = Getcell(M, x, y, z, 2, 0, R);
13             if v1 < 0 then
14                 v[x][y] = v1;
15 Dim=rango;
16 while dim >= 3 do
17     Dim=Dim-2;
18     cont=cont+1;
19 for x = 0; x < M; x++ do
20     for y = 0; y < M; y++ do
21         if (x>=cont AND y >= cont) AND (x<M - cont AND y < M − cont) then
22             for k = x − cont; k <= x + cont; k++ do
23                 for t = y − cont; t <= y + cont; t++ do
24                     suma = suma + v[k][t];
25             focal[x][y] = suma/rango × rango;
26             suma = 0;
27 return focal[][];
```

Algoritmo 6: Clasificar valores con respecto a una zona en particular

```
   Input: k^3-tree R, Dimension M
   Output: matriz v[][]
 1 v[M][M] =← createArrayList();
 2 v1 = 0;
 3 for z = 0; z < M; z++ do
 4     for x = 0; x < M; x++ do
 5         for y = 0; y < M; y++ do
 6             v1 = getcell(M, x, y, z, 2, 0, R);
 7             if v1 > 0orv1 <= 12 then
 8                 v[x][y] = Z1;
 9             if v1 > 12orv1 <= 20 then
10                 v[x][y] = Z2;
11             if v1 > 20orv1 <= 40 then
12                 v[x][y] = Z3;
13 return v[][];
```

Algoritmo 7: CLASIFICAR POR ZONAS INGRESANDO UN INTERVALO P Y Q

```
   Input: k^3-tree R, Dimension M , intervalo p , intervalo q
   Output: matriz v[][]
 1 v[M][M] =← createArrayList();
 2 v1 = 0;
 3 for z = 0; z < M; z++ do
 4     for x = 0; x < M; x++ do
 5         for y = 0; y < M; y++ do
 6             v1 = getcell(M, x, y, z, 2, 0, R);
 7             if (v1 >p) AND (v1 <= q) then
 8                 v[x][y] = ZC;
 9             else
10                 v[x][y] = 0;
11 return v[][];
```

Algoritmo 8: Calcular la distancia mínima con respecto a una celda (x,y) y peso z en un raster R

```
   Input: k^3-tree R, dimension M , position x , position y , peso z
   Output: matriz v[][]
 1 v[M][M] =← createArrayList();
 2 v1 = 0;
 3 cont2 = 0;
 4 valor = 0;
 5 for i = x; i >= 0; i − − do
 6     if i == x then
 7         v[i][y] = 0;
 8         for j = y; j >= 0; j − − do
 9             if j == y then
10                 v[x][j] = 0;
11             else
12                 v[x][j] = v[x][j + 1] + z;
13     else
14         v[i][y] = v[i + 1][y] + z;
15         for j = y; j <M; j + + do
16             if j == y then
17                 valor = v[i][j];
18                 cont2 = valor;
19             else
20                 if cont2! = 0 then
21                     v[i][j] = valor;
22                     cont2 = cont2 − z;
23                 else
24                     v[i][j] = v[i][j − 1] + z;
25         for j = y; j >= 0; j − − do
26             if j == y then
27                 valor = v[i][j];
28                 cont2 = valor;
29             else
30                 if cont2! = 0 then
31                     v[i][j] = valor;
32                     cont2 = cont2 − z;
33                 else
34                     v[i][j] = v[i][j + 1] + z;
35 for i = x; i <M; i + + do
36     if i == x then
37         v[i][y] = 0;
38         for j = y; j <M; j + + do
39             if j == y then
40                 v[x][j] = 0;
41             else
42                 v[x][j] = v[x][j − 1] + z;
43     else
44         v[i][y] = v[i − 1][y] + z;
45         for j = y; j <M; j + + do
46             if j == y then
47                 valor = v[i][j];
48                 cont2 = valor;
49             else
50                 if cont2! = 0 then
51                     v[i][j] = valor;
52                     cont2 = cont2 − z;
53                 else
54                     v[i][j] = v[i][j − 1] + z;
55         for j = y; j >= 0; j − − do
56             if j == y then
57                 valor = v[i][j];
58                 cont2 = valor;
59             else
60                 if cont2! = 0 then
61                     v[i][j] = valor;
62                     cont2 = cont2 − z;
63                 else
64                     v[i][j] = v[i][j + 1] + z;
65 for i = 0; i <M; i + + do
66     v1 = getcell(M, x, y, i, 2, 0, R);
67     if v1 > 0ORv1 < 0ANDv1! = null then
68         v1[x][y] = v1;
69         break;
70 return v[][];
```

Algoritmo 9: Obtener suma, resta, multiplicación y división con respecto a un ráster R y T sin EDC's

```
   Input: Archivo txt R , Archivo txt T , conditional c , dimensión M1 , dimensión M2
   Output: matriz v[][]
 1 v11 = 0;
 2 v22 = 0;
 3 if M1 > M2 then
 4 |   limite1 = M1;
 5 |   Limite2 = M2;
 6 |   v[limite1][limite1] =← createArrayList();
 7 else
 8 |   limite1 = M2;
 9 |   limite2 = M1;
10 |   v[limite1][limite1] =← createArrayList();
11 for x = 0; x < limite1; x + + do
12 |   for y = 0; y < limite1; y + + do
13 |   |   if c == SUMA then
14 |   |   |   if (x < limite2)AND(y < limite2) then
15 |   |   |   |   v11 = fopen(R.txt);
16 |   |   |   |   v22 = fopen(T.txt);
17 |   |   |   |   v[x][y] = v11 + v22;
18 |   |   |   else
19 |   |   |   |   if M1 > M2 then
20 |   |   |   |   |   v11 = fopen(R.txt);
21 |   |   |   |   |   v[x][y] = v11;
22 |   |   |   |   else
23 |   |   |   |   |   v22 = fopen(T.txt);
24 |   |   |   |   |   v[x][y] = v12;
25 |   |   if c == RESTA then
26 |   |   |   if (x < limite2)AND(y < limite2) then
27 |   |   |   |   v11 = fopen(R.txt);
28 |   |   |   |   v22 = fopen(T.txt);
29 |   |   |   |   v[x][y] = v11 − v22;
30 |   |   |   else
31 |   |   |   |   if M1 > M2 then
32 |   |   |   |   |   v11 = fopen(R.txt);
33 |   |   |   |   |   v[x][y] = v11;
34 |   |   |   |   else
35 |   |   |   |   |   v22 = fopen(T.txt);
36 |   |   |   |   |   v[x][y] = v12;
37 |   |   if c == MULTIPLICACIÖN then
38 |   |   |   if (x < limite2)AND(y < limite2) then
39 |   |   |   |   v11 = fopen(R.txt);
40 |   |   |   |   v22 = fopen(T.txt);
41 |   |   |   |   v[x][y] = v11 ∗ v22;
42 |   |   |   else
43 |   |   |   |   if M1 > M2 then
44 |   |   |   |   |   v11 = fopen(R.txt);
45 |   |   |   |   |   v[x][y] = v11;
46 |   |   |   |   else
47 |   |   |   |   |   v22 = fopen(T.txt);
48 |   |   |   |   |   v[x][y] = v12;
49 |   |   if c == DIVISIÖN then
50 |   |   |   if (x < limite2)AND(y < limite2) then
51 |   |   |   |   v11 = fopen(R.txt);
52 |   |   |   |   v22 = fopen(T.txt);
53 |   |   |   |   v[x][y] = v11/v22;
54 |   |   |   else
55 |   |   |   |   if M1 > M2 then
56 |   |   |   |   |   v11 = fopen(R.txt);
57 |   |   |   |   |   v[x][y] = v11;
58 |   |   |   |   else
59 |   |   |   |   |   v22 = fopen(T.txt);
60 |   |   |   |   |   v[x][y] = v12;
61 return v[][];
```

Algoritmo 10: Obtener suma, resta, multiplicación y división con respecto a un numero q sin EDC's

```
   Input: Archivo txt R , value q , Conditional c , Dimension M
   Output: matriz v[][]
 1 v[][] =← createArrayList();
 2 v1 = 0;
 3 for x = 0; x < M; x++ do
 4     for y = 0; y < M; y++ do
 5         v1 = open(R.txt);
 6         v[x][y] = v1;
 7         if c == SUMA then
 8             v[x][y] = v[x][y] + q;
 9         if c == RESTA then
10             v[x][y] = v[x][y] − q1;
11         if c == MULTIPLICACION then
12             v[x][y] = v[x][y] ∗ q1;
13         if c == DIVISION then
14             v[x][y] = v[x][y]/q1;
15 return v[][];
```

Algoritmo 11: Obtener valores mayores , menores o iguales con respecto a un numero q sin EDC's

```
   Input: Archivo txt R , Value q , Conditional c , Dimension M
   Output: matriz v[][]
 1 v[][] =← createArrayList();
 2 v1 = 0;
 3 for x = 0; x < M; x++ do
 4     for y = 0; y < M; y++ do
 5         v1 = open(R.txt);
 6         if c == MAYOR then
 7             if v > v1 then
 8                 v[x][y] = v1;
 9         if c == MENOR then
10             if v < v1 then
11                 v[x][y] = v1;
12         if c == IGUAL then
13             if v == v1 then
14                 v[x][y] = v1;
15 return v[][];
```

Algoritmo 12: Calcular la media con respecto a un rango q sin EDC's

```
   Input: Archivo txt R, Dimensión M ,Rango q
   Output: Matriz focal[][] con la media de valores
 1 /*Las variables se inician como vacías*/
 2 focal[][] ← createArrayList();
 3 v[] ← createArrayList();
 4 cont = 0;
 5 suma = 0;
 6 dim = 0;
 7 v1 = 0;
 8 cont = 0;
 9 for x = 0; x < M; x++ do
10     for y = 0; y < M; y++ do
11         v1 = open(R.txt);
12         v[x][y] = v1;
13 Dim=q;
14 while dim >= 3 do
15     Dim=Dim-2;
16     cont=cont+1;
17 for i = 0; i < Fila; i++ do
18     for j = 0; j < Columna; j++ do
19         if (x>=cont AND y >= cont) AND (x<M - cont AND y < M − cont) then
20             for k = i − cont; k <= i + cont; k++ do
21                 for t = j − cont; t <= j + cont; t++ do
22                     suma = suma + v[k][t];
23             focal[i][j] = suma/rango ∗ rango;
24 return focal[][];
```

Algoritmo 13: Clasificar valores con respecto a una zona en particular sin EDC's

```
   Input: Archivo txt R, Dimension M
   Output: matriz v[][]
 1 v[][] =← createArrayList();
 2 v1 = 0;
 3 for x = 0; x < M; x++ do
 4     for y = 0; y < M; y++ do
 5         v1 = open(R.txt);
 6         v[x][y] = v1;
 7         if v1 > 0ANDv1 <= 12 then
 8             v[x][y] = Z1;
 9         if v1 > 12ANDv1 <= 20 then
10             v[x][y] = Z2;
11         if v1 > 20ANDv1 <= 40 then
12             v[x][y] = Z3;
13 return v[][];
```

Algoritmo 14: CLASIFICAR POR ZONAS INGRESANDO UN INTERVALO P Y Q SIN EDC'S

```
   Input: Archivo txt R, Dimension M , intervalo p , intervalo q
   Output: matriz v[][]
 1 v[M][M] =← createArrayList();
 2 v1 = 0;
 3 for x = 0; x < M; x++ do
 4     for y = 0; y < M; y++ do
 5         v1 = open(R.txt);
 6         if (v1 >p) AND (v1 <= q) then
 7             v[x][y] = ZC;
 8         else
 9             v[x][y] = 0;
10 return v[][];
```

Algoritmo 15: Calcular la distancia minima con respecto a una celda (x,y) y peso z en un raster R sin EDC's

```
   Input: Archivo txt R, dimension M , position x , position y , peso z
   Output: matriz v[][]
 1 v[M][M] =← createArrayList();
 2 v1 = 0;
 3 cont2 = 0;
 4 valor = 0;
 5 for i = x; i >= 0; i − − do
 6     if i == x then
 7         v[i][y] = 0;
 8         for j = y; j >= 0; j − − do
 9             if j == y then
10                 v[x][j] = 0;
11             else
12                 v[x][j] = v[x][j + 1] + z;
13     else
14         v[i][y] = v[i + 1][y] + z;
15         for j = y; j <M; j + + do
16             if j == y then
17                 valor = v[i][j];
18                 cont2 = valor;
19             else
20                 if cont2! = 0 then
21                     v[i][j] = valor;
22                     cont2 = cont2 − z;
23                 else
24                     v[i][j] = v[i][j − 1] + z;
25         for j = y; j >= 0; j − − do
26             if j == y then
27                 valor = v[i][j];
28                 cont2 = valor;
29             else
30                 if cont2! = 0 then
31                     v[i][j] = valor;
32                     cont2 = cont2 − z;
33                 else
34                     v[i][j] = v[i][j + 1] + z;
35 for i = x; i <M; i + + do
36     if i == x then
37         v[i][y] = 0;
38         for j = y; j <M; j + + do
39             if j == y then
40                 v[x][j] = 0;
41             else
42                 v[x][j] = v[x][j − 1] + z;
43     else
44         v[i][y] = v[i − 1][y] + z;
45         for j = y; j <M; j + + do
46             if j == y then
47                 valor = v[i][j];
48                 cont2 = valor;
49             else
50                 if cont2! = 0 then
51                     v[i][j] = valor;
52                     cont2 = cont2 − z;
53                 else
54                     v[i][j] = v[i][j − 1] + z;
55         for j = y; j >= 0; j − − do
56             if j == y then
57                 valor = v[i][j];
58                 cont2 = valor;
59             else
60                 if cont2! = 0 then
61                     v[i][j] = valor;
62                     cont2 = cont2 − z;
63                 else
64                     v[i][j] = v[i][j + 1] + z;
65 for i = 0; i <M; 1 + + do
66     for j = 0; j <M; j + + do
67         v1 = fopen(R.txt);
68         if i == xANDj == y then
69             v1[x][y] = v1;
70             break;
71 return v[][];
```

Capítulo 5

Desarrollo e Interfaz de sistema Web

En este capítulo se describen las herramientas utilizadas para la creación del Sistema Web ,además se presentan una serie de capturas de pantalla con el fin de mostrar las distintas funcionalidades que presenta nuestra página Web, que son permitir diversas consultas a través de las operaciones del álgebra de mapas, sobre datos ráster que se encuentran almacenados en una estructura compacta k^3-tree sobre datos de temperatura pertenecientes a la región del Bío-Bío.

5.1. Herramientas utilizadas

A continuación se presentan las herramientas utilizadas para desarrollar el sistema WEB.

5.1.1. HTML

HTML, que significa Lenguaje de Marcado para Hipertextos (HyperText Markup Language) es el elemento de construcción más básico de una página web y se usa para crear y representar virtualmente una página web. Determina el contenido que contiene la página, pero no su funcionalidad. HTML se suele combinar con diferentes lenguajes de programación como javascript (funcionalidad) y CSS (estilos) para la creación de páginas web, ya que técnicamente, no es un lenguaje de programación, puesto que carece de funciones aritméticas, variables, o estructuras de control. HTML se encarga de describir la estructura y organización de una página, así como la forma en como se muestra su contenido (imágenes, videos, texto, entre otros). En la Figura 5.1 podemos visualizar la estructura mas básica que presenta un código HTML (*HyperText Markup Language*[1]).

[1]https://developer.mozilla.org/es/docs/Web/HTML

```html
<!DOCTYPE html>
<html>
<head>
<title>Título de la  Pagina</title>
</head>
<body>

<h1>Esto es un Ejemplo en HTML</h1>
<p>Escribe el texto que quieras.</p>

</body>
</html>

```

Esto es un Ejemplo en HTML

Escribe el texto que quieras.

Figura 5.1: Estructura básica de HTML para visualizar un título y párrafo

5.1.2. PHP

PHP (Hypertext Preprocessor), es un lenguaje interpretado de alto nivel contenido en páginas HTML y ejecutado en el servidor. PHP inicio como una modificación a Perl escrita por Rasmus Lerdorf a finales de 1994. Su primer uso fue el de mantener un control sobre quien visitaba su curriculum en su web. Este lenguaje se suele procesar directamente en un servidor, el cual genera código HTML que puede ser enviado a una aplicación cliente.

Shellexec de PHP permite ejecutar un comando de consola mediante el intérprete de comandos y devolver la salida completa como una cadena, esto nos permite mandar comandos a la consola de Linux por medio del sistema Web desarrollado. En este caso shellexec será la principal herramienta para llamar a los programas alojados en el servidor web. De tal forma que el usuario no se de cuenta que haciendo un simple click en un botón puede ejecutar diversos comandos de Linux de una manera practica y sencilla. A continuación se presenta un ejemplo de *shellexec*: [2].

```php
$cell = shell_exec("/opt/lampp/htdocs/Proyecto/application/controllers/Estructura/bin/Getcell ".$R." ".$x." ".$y." ".$z."");
```

Figura 5.2: Ejemplo de la función shell

En la Figura 5.2 muestra como *shellexec* ejecuta la función *Getcell* para obtener el valor de una celda en una posición x, y, z sobre un ráster R compactado. El valor de dicha celda se almacena en la variable cell. La función *shellexec* recibe como parámetros la ruta donde esta alojada la función *Getcell* mas las variables necesarias para ejecutar dicha función, en este caso las coordenadas x, y, z mas el ráster R a considerar.

5.1.3. Bootstrap

Bootstrap es un framework CSS desarrollado inicialmente (en el año 2011) por Twitter que permite dar forma a un sitio web mediante librerías CSS que incluyen tipografías, botones, cuadros, menús y otros elementos que pueden ser utilizados en cualquier sitio web. Bootstrap es una excelente herramienta para crear interfaces de usuario limpias y totalmente adaptables a todo tipo de dispositivos y pantallas, sea cual sea su tamaño.

[2] http://php.net/manual/es/intro-whatis.php

Además, Bootstrap ofrece las herramientas necesarias para crear cualquier tipo de sitio web utilizando los estilos y elementos de sus librerías [3]

5.1.4. CodeIgniter

CodeIgniter es un framework para el desarrollo de aplicaciones en PHP desarrollada por la empresa EllisLab, que utiliza el MVC (Modelo - vista - Controlador). Esto permite a los programadores o desarrolladores Web mejorar su forma de trabajar, además de dar una mayor velocidad a la hora de crear páginas WEB. El MVC es un patrón de arquitectura de software que separa la lógica de control, la interfaz del usuario y los datos del sistema. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir por un lado define los componentes para la representación de la información y por otro lado la interacción del usuario. Entre las características que presenta CodeIgniter se encuentran en procesar las páginas de una manera más rápida, facilitar la edición del código ya creado, formulación y validación de datos, sencillo de instalar y abundante documentación en la red, además de ser bajo licencia de open source (código abierto). La versión de CodeIgniter utilizada en este proyecto es la 3.1.10. [4]

5.2. Interfaz de la Aplicación

El sistema Web propuesto, es un sitio web creado en lenguaje HTML y PHP, donde utiliza boostrap para dar estilo a los menus y botones, además ocupa el framework Codeigniter para programar en un entorno MVC (modelo – vista – controlador), para contestar a las distintas consultas solicitadas. El sitio cuenta con una interfaz simple de usar, con una pantalla de inicio con 5 botones, divididas en 5 vistas. El primer botón se llama “Local query”, y resuelve consultas locales considerando 2 rásteres de entrada., el segundo botón se llama “Focal query” y resuelve consultas de tipo Focales, el tercer botón se llama “Zonal query” y resuelve consultas de tipo Zonales, el cuarto botón se llama “Global query” y resuelve consultas de tipo Globales, y el quinto botón se llama “Local query 2” y resuelve consultas de tipo Local, considerando un solo ráster de entrada. En la Figura 5.3 se muestra virtualmente la pantalla de inicio de nuestro sitio Web.

Para realizar consultas dentro de la aplicación Web, se debe presionar cualquiera de los 5 botones presentes en nuestra pantalla de inicio (Figura 5.3). A continuación, vamos a mostrar cada una de las vistas que representa las distintas consultas soportadas por la aplicación, además de mostrar un ejemplo práctico para cada una de las consultas soportadas por nuestro sitio web.

La Figura 5.4, muestra la vista que representa el botón “Local query”. Esta vista permite realizar consultas Locales considerando 2 rásteres de entrada. En la parte inferior, se tiene la opción de cargar 2 rásteres a elección y seleccionar una operación entre ellos (suma, resta, multiplicación, división), y en la parte superior podemos seleccionar estas mismas

[3] https://getbootstrap.com/docs/4.1/getting-started/introduction/
[4] https://codeigniter.com/

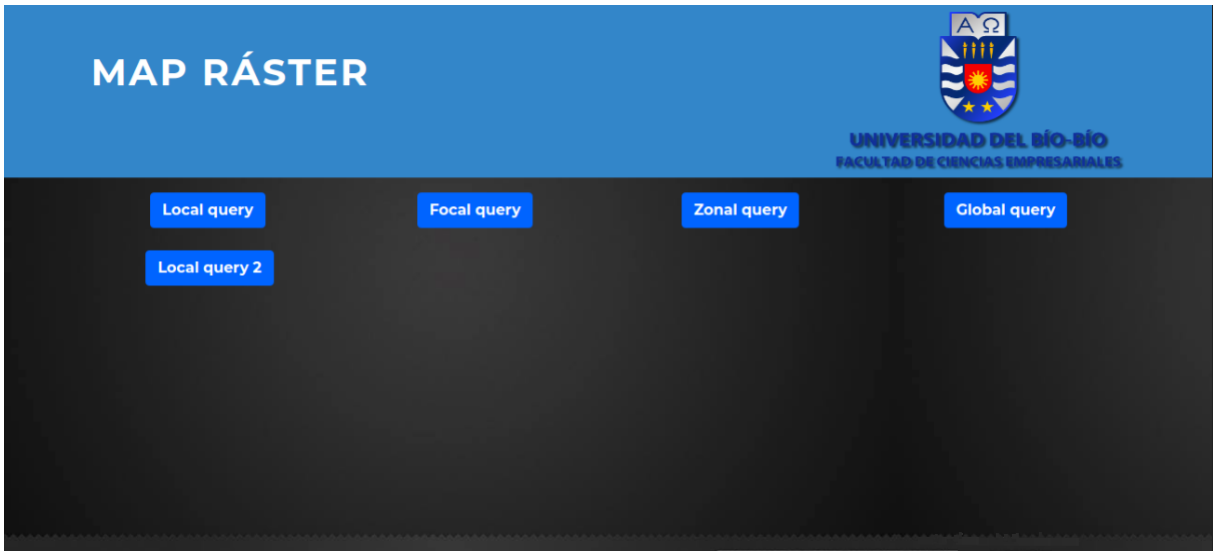


Figura 5.3: Pantalla principal de nuestra Aplicación Web

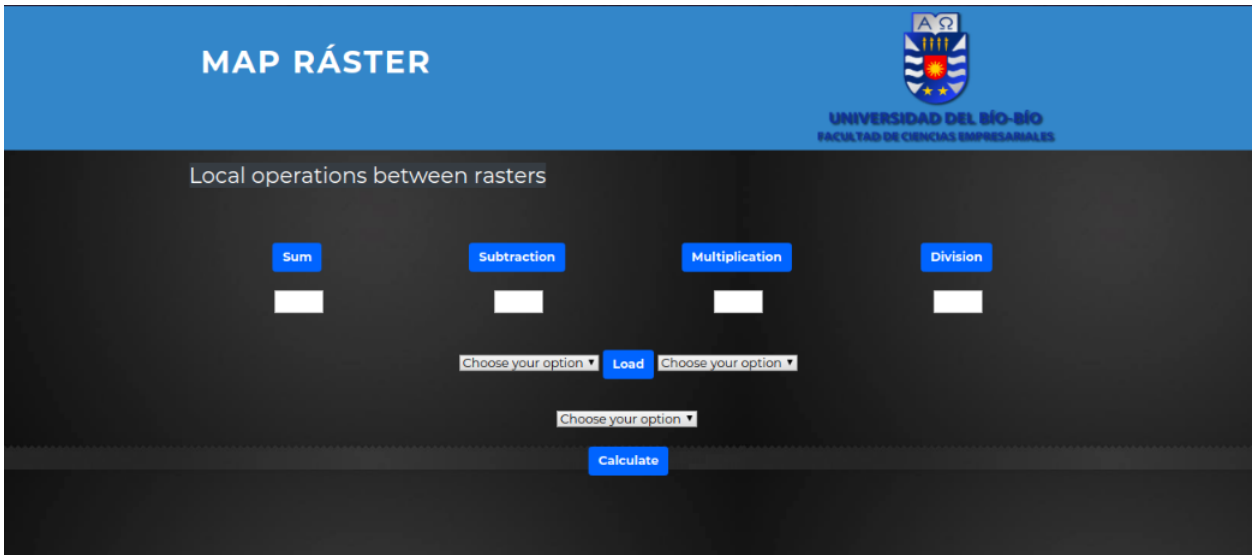


Figura 5.4: Vista para consultas de Tipo Local

opciones, pero solo se aplica al resultado obtenido de los 2 rasters de entrada, mas un numero ingresado por el usuario.

Ejemplo 5.1 Consideremos los rásteres de la Figura 5.5 y Figura 5.6, vamos a calcular la suma y el promedio entre los rásteres de Laraquete y San Pedro presentes en nuestro sitio web. El resultado de la consulta retorna una matriz considerando la mayor dimensionalidad entre los 2 rásteres seleccionados, en este caso Laraquete tiene una dimensión de 8×8

MAP LARAQUETE

	0	1	2	3	4	5	6	7
0	12	12	12	12	12	12	12	12
1	12	12	12	12	12	12	12	12
2	12	12	12	14	13	13	13	13
3	12	12	13	14	14	14	13	13
4	13	13	14	15	15	15	14	14
5	14	14	15	15	15	15	15	15
6	15	15	15	15	15	15	15	15
7	15	15	15	15	15	15	15	15

Figura 5.5: Ráster perteneciente a la ciudad de Laraquete

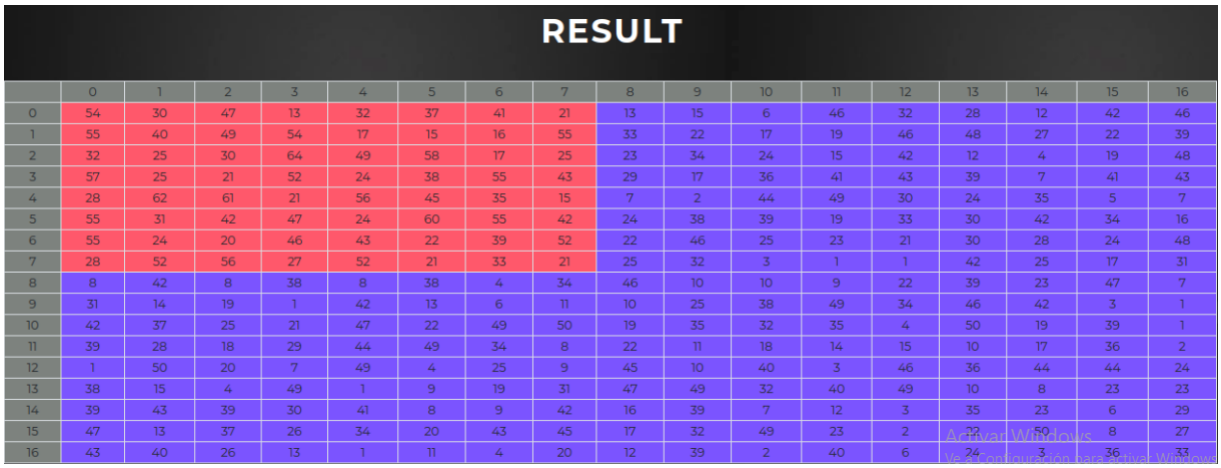
MAP SANPEDRO

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	42	18	35	1	20	25	29	9	13	15	6	46	32	28	12	42	46
1	43	28	37	42	5	3	4	43	33	22	17	19	46	48	27	22	39
2	20	13	18	50	36	45	4	12	23	34	24	15	42	12	4	19	48
3	45	13	8	38	10	24	42	30	29	17	36	41	43	39	7	41	43
4	15	49	47	6	41	30	21	1	7	2	44	49	30	24	35	5	7
5	41	17	27	32	9	45	40	27	24	38	39	19	33	30	42	34	16
6	40	9	5	31	28	7	24	37	22	46	25	23	21	30	28	24	48
7	13	37	41	12	37	6	18	6	25	32	3	1	1	42	25	17	31
8	8	42	8	38	8	38	4	34	46	10	10	9	22	39	23	47	7
9	31	14	19	1	42	13	6	11	10	25	38	49	34	46	42	3	1
10	42	37	25	21	47	22	49	50	19	35	32	35	4	50	19	39	1
11	39	28	18	29	44	49	34	8	22	11	18	14	15	10	17	36	2
12	1	50	20	7	49	4	25	9	45	10	40	3	46	36	44	44	24
13	38	15	4	49	1	9	19	31	47	49	32	40	49	10	8	23	23
14	39	43	39	30	41	8	9	42	16	39	7	12	3	35	23	6	29
15	47	13	37	26	34	20	43	45	17	32	49	23	2	22	50	8	27
16	43	40	26	13	1	11	4	20	12	39	2	40	6	24	3	36	33

Figura 5.6: Ráster perteneciente a la ciudad de San Pedro

mientras que San Pedro tiene una dimensión de 17×17. La matriz resultante se muestra en la Figura 5.7, donde en color rojo se muestra la suma entre Laraquete y San Pedro, y en color azul se muestran las celdas donde no es posible calcular la suma, en este caso solo se consideran los valores de San Pedro por tener una mayor dimensionalidad. Ahora para calcular el promedio entre Laraquete y San pedro vamos a considerar las opciones que se encuentran en la parte superior de la vista "Local query" (Figura 5.4), en este caso se debe presionar el botón "Division" con un valor de 2 y operará directamente sobre el resultado de la suma. En la Figura 5.8 se muestra el resultado del promedio entre Laraquete y San Pedro. □

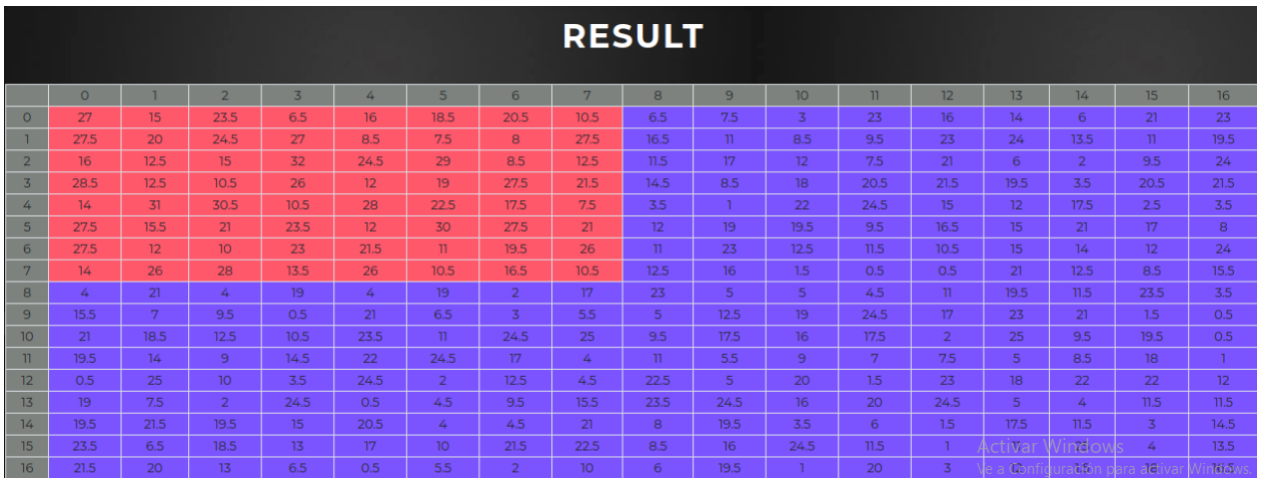
La Figura 5.9, muestra la vista que representa el botón "Focal query". Esta vista permite realizar consultas Focales para el cálculo de la media con respecto a un rango y ráster a considerar. En la parte inferior se tiene la opción de ingresar la dimensión de nuestro rango (o ventana Focal), siempre considerando un número impar para que exista un centro. Una vez ingresado el valor de rango se debe presionar el botón "Calculate" para que muestre el cálculo de la media.



RESULT

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	54	30	47	13	32	37	41	21	13	15	6	46	32	28	12	42	46
1	55	40	49	54	17	15	16	55	33	22	17	19	46	48	27	22	39
2	32	25	30	64	49	58	17	25	23	34	24	15	42	12	4	19	48
3	57	25	21	52	24	38	55	43	29	17	36	41	43	39	7	41	43
4	28	62	61	21	56	45	35	15	7	2	44	49	30	24	35	5	7
5	55	31	42	47	24	60	55	42	24	38	39	19	33	30	42	34	16
6	55	24	20	46	43	22	39	52	22	46	25	23	21	30	28	24	48
7	28	52	56	27	52	21	33	21	25	32	3	1	1	42	25	17	31
8	8	42	8	38	8	38	4	34	46	10	10	9	22	39	23	47	7
9	31	14	19	1	42	13	6	11	10	25	38	49	34	46	42	3	1
10	42	37	25	21	47	22	49	50	19	35	32	35	4	50	19	39	1
11	39	28	18	29	44	49	34	8	22	11	18	14	15	10	17	36	2
12	1	50	20	7	49	4	25	9	45	10	40	3	46	36	44	44	24
13	38	15	4	49	1	9	19	31	47	49	32	40	49	10	8	23	23
14	39	43	39	30	41	8	9	42	16	39	7	12	3	35	23	6	29
15	47	13	37	26	34	20	43	45	17	32	49	23	2	22	50	8	27
16	43	40	26	13	1	11	4	20	12	39	2	40	6	24	3	36	33

Figura 5.7: Resultado de aplicar el operador suma entre el ráster de la Figura 5.5 y la Figura 5.6



RESULT

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	27	15	23.5	6.5	16	18.5	20.5	10.5	6.5	7.5	3	23	16	14	6	21	23
1	27.5	20	24.5	27	8.5	7.5	8	27.5	16.5	11	8.5	9.5	23	24	13.5	11	19.5
2	16	12.5	15	32	24.5	29	8.5	12.5	11.5	17	12	7.5	21	6	2	9.5	24
3	28.5	12.5	10.5	26	12	19	27.5	21.5	14.5	8.5	18	20.5	21.5	19.5	3.5	20.5	21.5
4	14	31	30.5	10.5	28	22.5	17.5	7.5	3.5	1	22	24.5	15	12	17.5	2.5	3.5
5	27.5	15.5	21	23.5	12	30	27.5	21	12	19	19.5	9.5	16.5	15	21	17	8
6	27.5	12	10	23	21.5	11	19.5	26	11	23	12.5	11.5	10.5	15	14	12	24
7	14	26	28	13.5	26	10.5	16.5	10.5	12.5	16	1.5	0.5	0.5	21	12.5	8.5	15.5
8	4	21	4	19	4	19	2	17	23	5	5	4.5	11	19.5	11.5	23.5	3.5
9	15.5	7	9.5	0.5	21	6.5	3	5.5	5	12.5	19	24.5	17	23	21	1.5	0.5
10	21	18.5	12.5	10.5	23.5	11	24.5	25	9.5	17.5	16	17.5	2	25	9.5	19.5	0.5
11	19.5	14	9	14.5	22	24.5	17	4	11	5.5	9	7	7.5	5	8.5	18	1
12	0.5	25	10	3.5	24.5	2	12.5	4.5	22.5	5	20	1.5	23	18	22	22	12
13	19	7.5	2	24.5	0.5	4.5	9.5	15.5	23.5	24.5	16	20	24.5	5	4	11.5	11.5
14	19.5	21.5	19.5	15	20.5	4	4.5	21	8	19.5	3.5	6	1.5	17.5	11.5	3	14.5
15	23.5	6.5	18.5	13	17	10	21.5	22.5	8.5	16	24.5	11.5	1	25	15	4	13.5
16	21.5	20	13	6.5	0.5	5.5	2	10	6	19.5	1	20	3	13	1.5	16	16.5

Figura 5.8: Resultado de aplicar el operador división entre la suma del ráster de la Figura 5.5 y la Figura 5.6

Ejemplo 5.2 Consideremos el ráster de la Figura 5.6, vamos a calcular la media con un rango de 3 sobre la ciudad de San Pedro, el resultado de la consulta de muestra en la Figura 5.10, donde las celdas en "X" en color plomo es donde no es posible calcular la media, las celdas en color verde es donde es posible calcular la media.

La Figura 5.11, muestra la vista que representa el botón "Zonal query". Esta vista permite realizar consultas Zonales para zonas predefinidas y zonas ingresando los intervalos por el usuario. En la parte izquierda de la Figura 5.11, se muestran 3 tipos de zonas en color verde: Zone 1 que filtra las zonas entre los valores de 0 y 15, Zone 2 que filtra las zonas entre los valores de 15 y 30, y Zone 3 que filtra las zonas entre los valores de 30 y

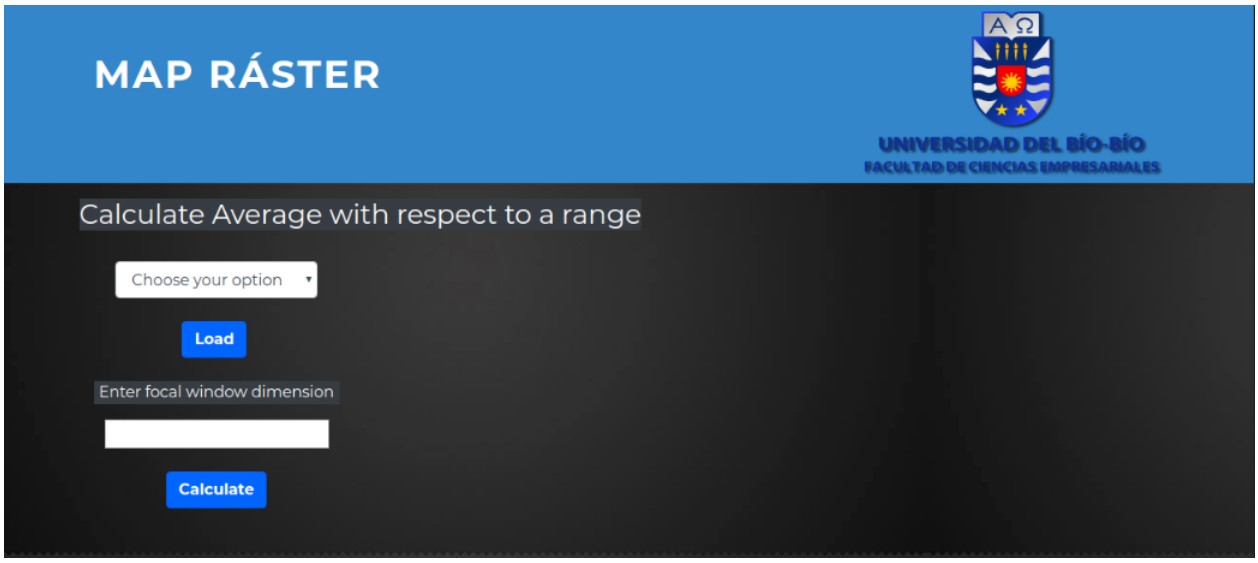
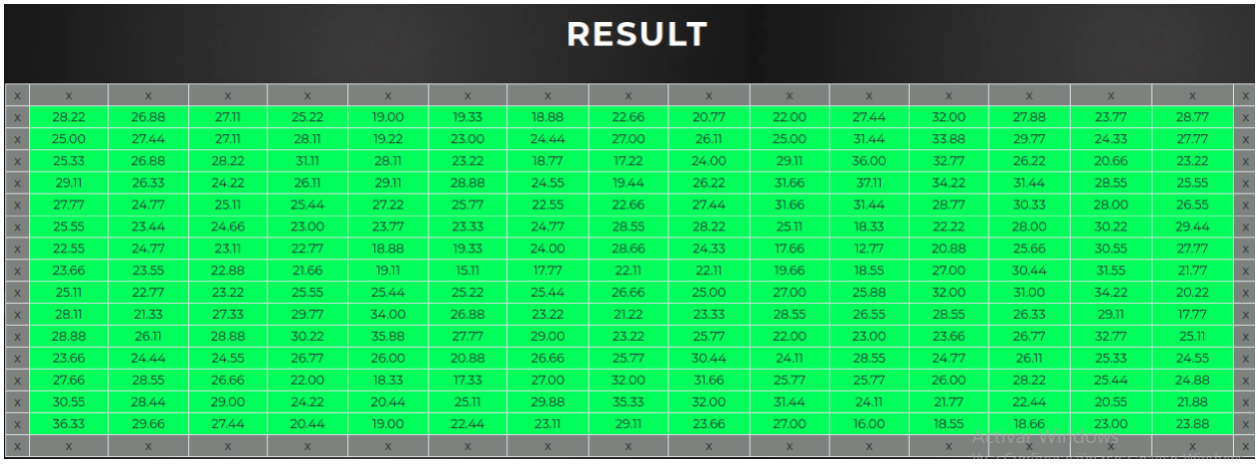


Figura 5.9: Vista para consultas de Tipo Focal



x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	28.22	26.88	27.11	25.22	19.00	19.33	18.88	22.66	20.77	22.00	27.44	32.00	27.88	23.77	28.77	x
x	25.00	27.44	27.11	28.11	19.22	23.00	24.44	27.00	26.11	25.00	31.44	33.88	29.77	24.33	27.77	x
x	25.33	26.88	28.22	31.11	28.11	23.22	18.77	17.22	24.00	29.11	36.00	32.77	26.22	20.66	23.22	x
x	29.11	26.33	24.22	26.11	29.11	28.88	24.55	19.44	26.22	31.66	37.11	34.22	31.44	28.55	25.55	x
x	27.77	24.77	25.11	25.44	27.22	25.77	22.55	22.66	27.44	31.66	31.44	28.77	30.33	28.00	26.55	x
x	25.55	23.44	24.66	23.00	23.77	23.33	24.77	28.55	28.22	25.11	18.33	22.22	28.00	30.22	29.44	x
x	22.55	24.77	23.11	22.77	18.88	19.33	24.00	28.66	24.33	17.66	12.77	20.88	25.66	30.55	27.77	x
x	23.66	23.55	22.88	21.66	19.11	15.11	17.77	22.11	22.11	19.66	18.55	27.00	30.44	31.55	21.77	x
x	25.11	22.77	23.22	25.55	25.44	25.22	25.44	26.66	25.00	27.00	25.88	32.00	31.00	34.22	20.22	x
x	28.11	21.33	27.33	29.77	34.00	26.88	23.22	21.22	23.33	28.55	26.55	28.55	26.33	29.11	17.77	x
x	28.88	26.11	28.88	30.22	35.88	27.77	29.00	23.22	25.77	22.00	23.00	23.66	26.77	32.77	25.11	x
x	23.66	24.44	24.55	26.77	26.00	20.88	26.66	25.77	30.44	24.11	28.55	24.77	26.11	25.33	24.55	x
x	27.66	28.55	26.66	22.00	18.33	17.33	27.00	32.00	31.66	25.77	25.77	26.00	28.22	25.44	24.88	x
x	30.55	28.44	29.00	24.22	20.44	25.11	29.88	35.33	32.00	31.44	24.11	21.77	22.44	20.55	21.88	x
x	36.33	29.66	27.44	20.44	19.00	22.44	23.11	29.11	23.66	27.00	16.00	18.55	18.66	23.00	23.88	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figura 5.10: Resultado consulta Focal aplicada al ráster de la Figura 5.6

50. Para filtrar las 3 zonas predefinidas a la vez, se debe presionar el botón “Calculate Z1 Z2 Z3”. Para filtrar zonas ingresando los intervalos por el usuario, se debe ir a la parte derecha de la Figura 5.11 y en los campos vacíos se ingresa los intervalos 1 y 2, una vez listos los parámetros se debe presionar el botón ’Calculate’para obtener el filtro por zonas ingresando parámetros por el usuario.

Ejemplo 5.3 Consideremos el ráster de la Figura 5.6, vamos a filtrar las zonas predefinidas y las zonas considerando un intervalo de 13 y 27, ingresado por el usuario. En la Figura 5.12, se muestra el resultado de filtrar por Zone 1, Zone 2, Zone 3, donde las celdas en color rojo con valor Z1 son equivalentes a la Zone 1, las celdas en color azul con valor Z2 son

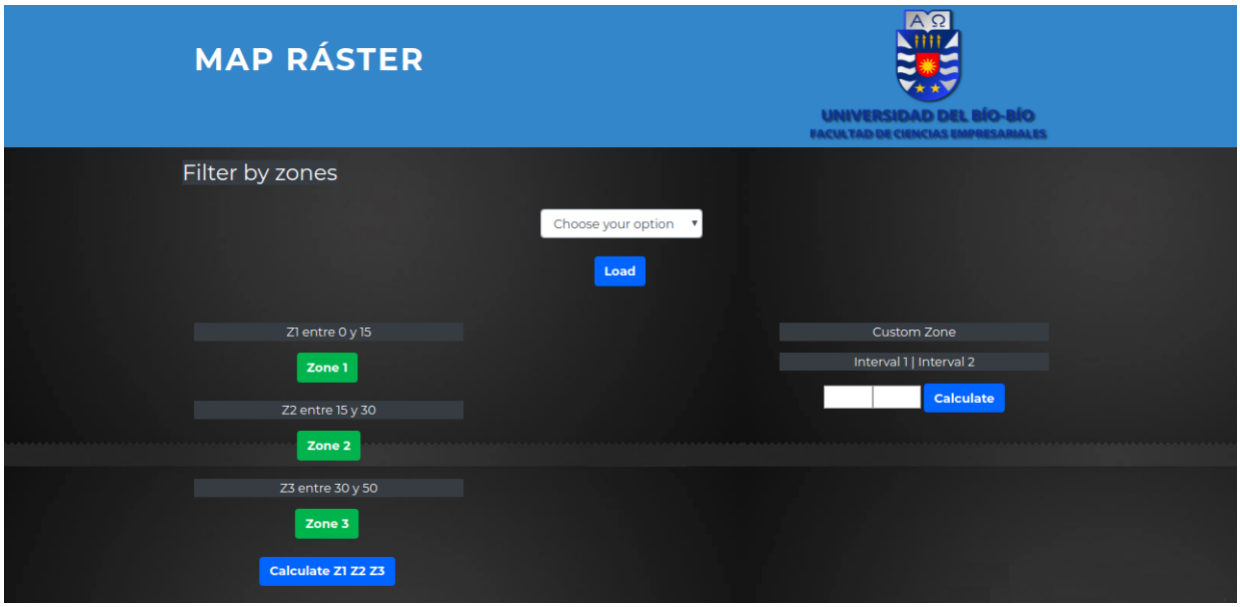


Figura 5.11: Vista para consultas de Tipo Zonal

equivalentes a la Zone 2, y las celdas en color verde con valor Z3 son equivalentes a la Zone 3. En la Figura 5.13 se muestra en color morado las celdas en ZC que se encuentran dentro del intervalo 13 - 27. Las celdas en color plomo son las que no cumplen esta condición.

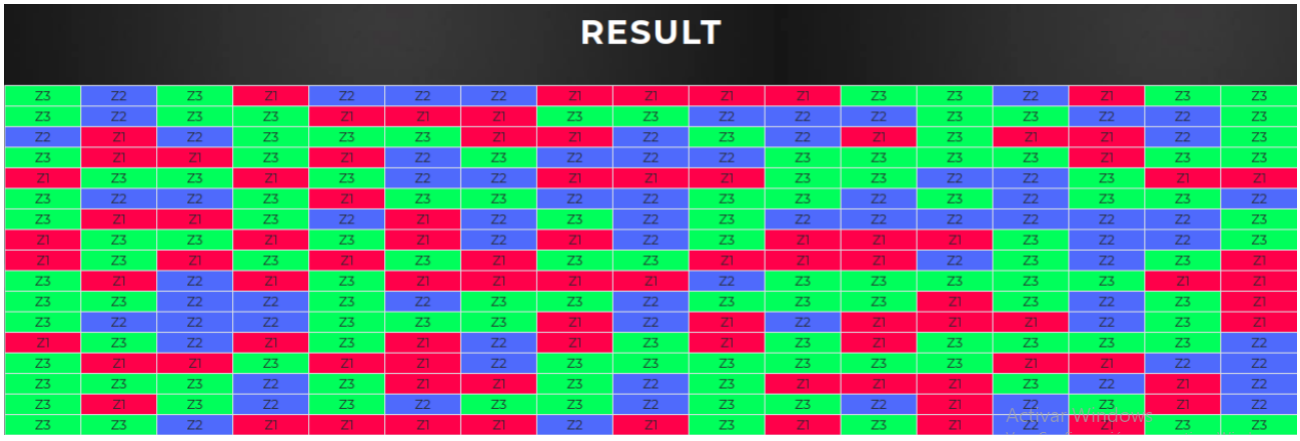


Figura 5.12: Resultado de filtrar por Zone 1, Zone 2, Zone 3, al ráster de la Figura 5.6

La Figura 5.14, muestra la vista que representa el botón “Global query”. Esta vista permite realizar consultas Globales considerando un punto (posición x e y), peso y ráster a considerar. En la parte izquierda de la Figura 5.14, se muestran los campos para ingresar la posición (x,y) de la celda a consultar y el peso que representa la distancia entre cada

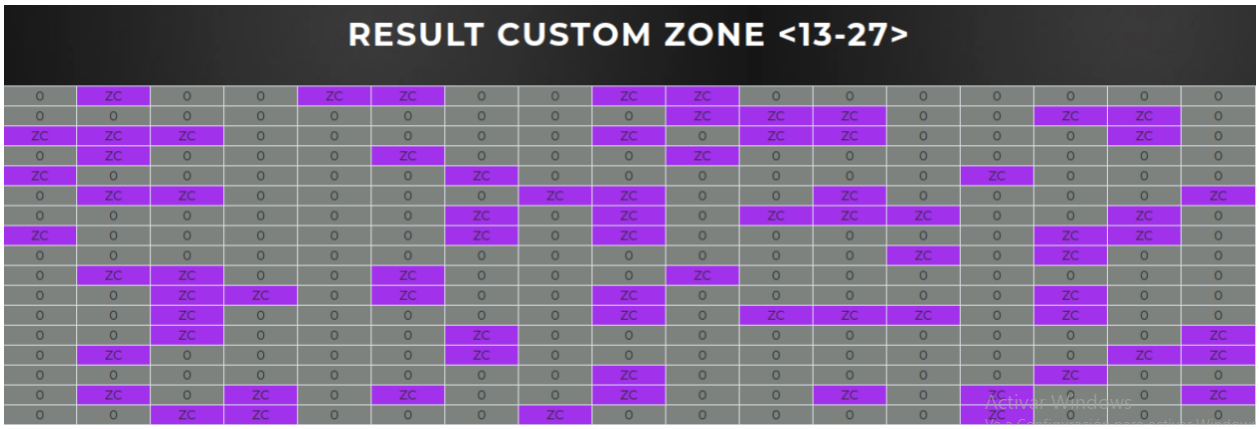


Figura 5.13: Resultado de filtrar por zonas en un intervalo de 13-27, al ráster de la Figura 5.6

celda. Finalmente se debe presionar el botón "Calculate" para mostrar las distancias que existen con respecto a un punto de un ráster seleccionado, considerando un peso ingresado por el usuario.

Figura 5.14: Vista para consultas de Tipo Global

Ejemplo 5.4 Consideremos el ráster de la Figura 5.6, vamos a mostrar las distancias existentes para la celda 10,15 con un valor de 39, considerando un peso de 1 entre cada celda. En la Figura 5.15 se muestra el resultado de mostrar las distancias, donde la celda 10,15 se muestra en rojo, y en verde se muestran las demás celdas que representan la distancia existente sobre ese punto.

RESULT

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	15	14	13	12	11	10	10	10	10	10	10	10	10	10	10	10	10
1	15	14	13	12	11	10	9	9	9	9	9	9	9	9	9	9	9
2	15	14	13	12	11	10	9	8	8	8	8	8	8	8	8	8	8
3	15	14	13	12	11	10	9	8	7	7	7	7	7	7	7	7	7
4	15	14	13	12	11	10	9	8	7	6	6	6	6	6	6	6	6
5	15	14	13	12	11	10	9	8	7	6	5	5	5	5	5	5	5
6	15	14	13	12	11	10	9	8	7	6	5	4	4	4	4	4	4
7	15	14	13	12	11	10	9	8	7	6	5	4	3	3	3	3	3
8	15	14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2
9	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1
10	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	39	1
11	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1
12	15	14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2
13	15	14	13	12	11	10	9	8	7	6	5	4	3	3	3	3	3
14	15	14	13	12	11	10	9	8	7	6	5	4	4	4	4	4	4
15	15	14	13	12	11	10	9	8	7	6	5	5	5	5	5	5	5

Figura 5.15: Resultado consulta Global aplicada al ráster de la Figura 5.6

La Figura 5.16, muestra la vista que representa el botón “Local query 2”. Esta vista permite realizar consultas Locales considerando un ráster de entrada. En la parte izquierda de la Figura 5.16, se muestran los condicionales para saber si las celdas de un ráster son mayor, menor, igual o distinto a un número ingresado por el usuario. En la parte derecha de la Figura 5.16 se muestran las operaciones de suma, resta, multiplicación y división, considerando un ráster y un número ingresado por el usuario.

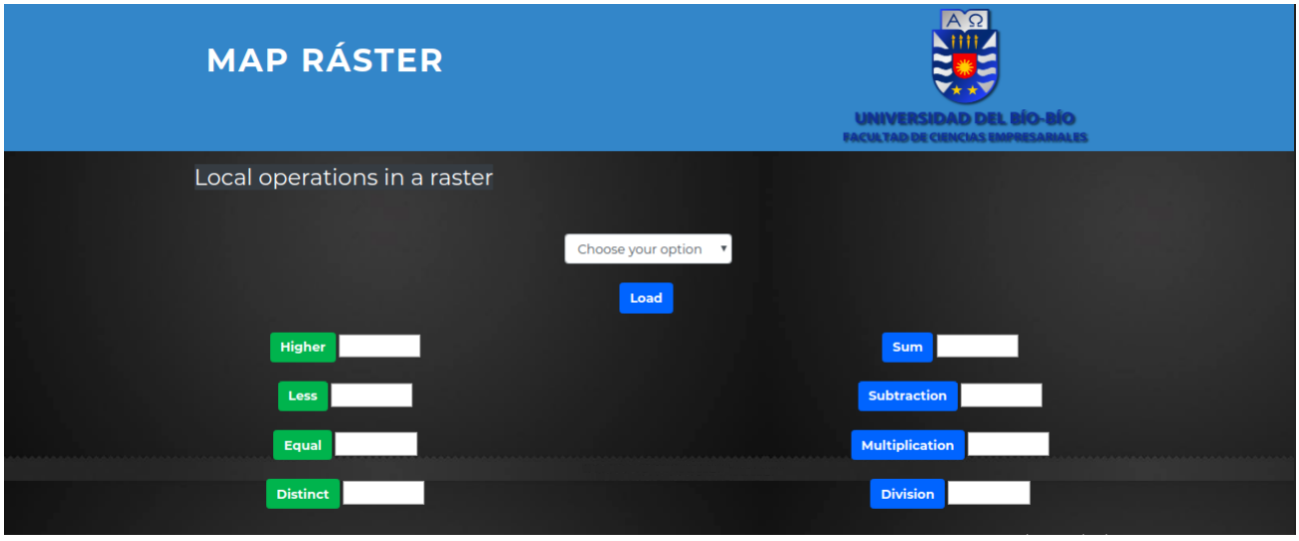


Figura 5.16: Vista para consultas de Tipo Local 2

Ejemplo 5.5 Consideremos el ráster de la Figura 5.6, vamos a mostrar los valores menores a 12 y multiplicar por un valor de 500 cada una de las celdas presentes en el ráster de San

Pedro. En la Figura 5.17 se muestra el resultado de los valores menores a 12 en color rojo, donde las celdas en color plomo son las que no cumplen la condición. En la Figura 5.18 se muestra el resultado de multiplicar por 500 cada una de las celdas pertenecientes al ráster de la Figura 5.6.

RESULT

42	18	35	1	20	25	29	9	13	15	6	46	32	28	12	42	46
43	28	37	42	5	3	4	43	33	22	17	19	46	48	27	22	39
20	13	18	50	36	45	4	12	23	34	24	15	42	12	4	19	48
45	13	8	38	10	24	42	30	29	17	36	41	43	39	7	41	43
15	49	47	6	41	30	21	1	7	2	44	49	30	24	35	5	7
41	17	27	32	9	45	40	27	24	38	39	19	33	30	42	34	16
40	9	5	31	28	7	24	37	22	46	25	23	21	30	28	24	48
13	37	41	12	37	6	18	6	25	32	3	1	1	42	25	17	31
8	42	8	38	8	38	4	34	46	10	10	9	22	39	23	47	7
31	14	19	1	42	13	6	11	10	25	38	49	34	46	42	3	1
42	37	25	21	47	22	49	50	19	35	32	35	4	50	19	39	1
39	28	18	29	44	49	34	8	22	11	18	14	15	10	17	36	2
1	50	20	7	49	4	25	9	45	10	40	3	46	36	44	44	24
38	15	4	49	1	9	19	31	47	49	32	40	49	10	8	23	23
39	43	39	30	41	8	9	42	16	39	7	12	3	35	23	6	29
47	13	37	26	34	20	43	45	17	32	49	23	2	22	50	8	27
43	40	26	13	1	11	4	20	12	39	2	40	6	24	3	36	33

Figura 5.17: Resultado de aplicar el operador Menor sobre el ráster de la Figura 5.6

RESULT

21000	9000	17500	500	10000	12500	14500	4500	6500	7500	3000	23000	16000	14000	6000	21000	23000
21500	14000	18500	21000	2500	1500	2000	21500	16500	11000	8500	9500	23000	24000	13500	11000	19500
10000	6500	9000	25000	18000	22500	2000	6000	11500	17000	12000	7500	21000	6000	2000	9500	24000
22500	6500	4000	19000	5000	12000	21000	15000	14500	8500	18000	20500	21500	19500	3500	20500	21500
7500	24500	23500	3000	20500	15000	10500	500	3500	1000	22000	24500	15000	12000	17500	2500	3500
20500	8500	13500	16000	4500	22500	20000	13500	12000	19000	19500	9500	16500	15000	21000	17000	8000
20000	4500	2500	15500	14000	3500	12000	18500	11000	23000	12500	11500	10500	15000	14000	12000	24000
6500	18500	20500	6000	18500	3000	9000	3000	12500	16000	1500	500	500	21000	12500	8500	15500
4000	21000	4000	19000	4000	19000	2000	17000	23000	5000	5000	4500	11000	19500	11500	23500	3500
15500	7000	9500	500	21000	6500	3000	5500	5000	12500	19000	24500	17000	23000	21000	1500	500
21000	18500	12500	10500	23500	11000	24500	25000	9500	17500	16000	17500	2000	25000	9500	19500	500
19500	14000	9000	14500	22000	24500	17000	4000	11000	5500	9000	7000	7500	5000	8500	18000	1000
500	25000	10000	3500	24500	2000	12500	4500	22500	5000	20000	1500	23000	18000	22000	22000	12000
19000	7500	2000	24500	500	4500	9500	15500	23500	24500	16000	20000	24500	5000	4000	11500	11500
19500	21500	19500	15000	20500	4000	4500	21000	8000	19500	3500	6000	1500	17500	11500	3000	14500
23500	6500	18500	13000	17000	10000	21500	22500	8500	16000	24500	11500	1000	11000	25000	4000	13500
21500	20000	13000	6500	500	5500	2000	10000	6000	19500	1000	20000	3000	12000	1500	18000	16500

Figura 5.18: Resultado de aplicar el operador multiplicación sobre el ráster de la Figura 5.6

Capítulo 6

Experimentación

En este capítulo se presenta la experimentación realizada a partir de las distintas pruebas sobre la estructura compacta de datos k^3-tree, como lo son pruebas de espacio de almacenamiento y tiempos de respuesta de consultas para cada uno de los operadores del algebra de mapas presentados en la sección 3.3. En la Sección 6.1 se describen los datos utilizados para realizar las distintas pruebas sobre la estructura compacta k^3-tree, En las Secciones 6.2 y 6.3 se presentan los experimentos de espacio de almacenamiento y tiempo de ejecución de consultas.

Todos los experimentos se corrieron en computador con las siguientes especificaciones de hardware: procesador Intel(R) Core(TM) i5 5200U 2.20GHZ (4 núcleos), memoria RAM de 8GB DDR3 a 3000 Mhz, almacenamiento de 1 Tb HDD de 5400 RPM y sistema operativo Ubuntu 16.06 de 64 bits.

6.1. Descripción de los Datos de Prueba

Para la experimentación de los algoritmos, se utilizaron solo datos de origen sintético. En cuanto a estos conjuntos de datos fueron generados a partir de matrices aleatorias de diferentes dimensionalidades, con distribución de datos normal (Gaussiana) (ver Figura 6.1) y distribución de datos generadas de manera aleatoria (ver Figura 6.2). Estos conjuntos de datos fueron obtenidos desde el portal *Pinetools* [1], donde se considera una media de 100 (valor escalar numérico) y desviación estándar de 5 para los datos de distribución normal , y un limite de 100 (valor escalar numérico) para los datos con distribución aleatoria, en cada caso, solo se aumenta el tamaño de celdas a considerar en cada dataset. Para los experimentos solo se consideran matrices cuadradas de dimensiones de $M \times M$. Se generan 10 data sets.

[1] https://pinetools.com/es/generador-numeros-aleatorios

87	88	88	89	90	91	91	92
92	92	93	94	95	95	95	95
96	96	97	97	97	98	98	98
99	99	99	100	100	100	100	100
101	101	101	101	102	102	102	102
102	102	103	103	103	103	103	104
104	104	104	104	104	104	105	105
106	106	106	106	107	109	109	110

Figura 6.1: Matriz de 8×8 con valores de distribución normal (media = 100 y desviacion estandar = 5)

19	97	53	62	3	52	8	14
45	87	29	2	26	20	88	98
38	40	42	51	44	100	10	83
80	91	78	16	35	71	58	56
43	24	86	36	1	28	94	77
32	49	37	81	63	84	95	93
74	70	73	72	0	89	82	21
99	34	4	31	17	59	96	22

Figura 6.2: Matriz de 8×8 con valores aleatorios en su interior

6.2. Pruebas para Medir el Espacio de Almacenamiento

En esta sección se describe el espacio utilizado por la estructura de datos compacta k^3-tree en comparación a almacenar los datos sin recibir ningún tipo de compactación. Para medir la eficiencia en cuanto al espacio de almacenamiento ocupado (expresado en bytes), se realizaron experimentos con datos de origen sintético representando una distribución normal y una distribución de tipo aleatoria. En este caso, se compara el tamaño inicial del raster en relación al tamaño compactado por la estructura compacta k^3-tree. Para cada matriz se considera el numero valores que almacena, desde n = 289 hasta n = 4489, ordenados de menor a mayor valor. De esta manera, los datos representados en la Tabla 6.1 muestran los resultados de compactación para los datasets de origen ficticio, mostrando el tamaño original y compactado respectivamente.

Como se puede apreciar en la Tabla 6.1, la compactación mejora a medida que se tiene una mayor cantidad de elementos, en concreto, superando los 4000 datos en un ráster de prueba, se puede llegar hasta un 90 porciento de compactación, con respecto al ráster original. De los datos aleatorios obtenidos en la Tabla 6.1, podemos ver que a medida que se tiene una mayor cantidad de elementos este no supera mas de 50 porciento de compactación,

Data set	Dimension	celdas	# Espacio de almacenamiento					
			Distribución normal			Distribución aleatoria		
			Original	k^3-tree	Porcentaje	Original	k^3-tree	porcentaje
data set 1	17×17	289	1140 byte	407 bytes	64.3	1140 bytes	817 bytes	28.34
data set 2	25×25	625	3880 bytes	642 bytes	83.46	3880 bytes	1611 bytes	58.48
data set 3	53×53	2809	9256 bytes	1691 bytes	81.74	9256 bytes	4412 bytes	52.34
data set 4	62×62	3844	13104 bytes	2421 bytes	86.76	13104 bytes	5421 bytes	58.64
data set 5	49×49	2401	8800 bytes	1038 bytes	88.21	8800 bytes	4390 bytes	50.12
data set 6	43×43	1849	7344 bytes	1739 bytes	76.33	7344 bytes	3411 bytes	53.56
data set 7	47×47	2209	8856 bytes	1172 bytes	86.77	8856 bytes	4312 bytes	51.31
data set 8	67×67	4489	17784 bytes	1619 bytes	90.9	17784 bytes	9451 bytes	46.86
data set 9	39×39	1521	6232 bytes	954 bytes	84.7	6232 bytes	3423 bytes	45.08
data set 10	57×57	3249	11100 bytes	1735 bytes	84.37	11100 bytes	4912 bytes	55.75

Tabla 6.1: Espacio de almacenamiento utilizado por datasets ráster vs Espacio de almacenamiento utilizado por la estructura compacta k^3-tree (distribución normal y distribución aleatoria

con respecto a los resultados obtenidos de la Tabla 6.1. Una de las razones de esto es por la forma que tiene la estructura de agrupar los datos en la matriz tridimensional, si vemos el particionamiento explicando en la sección 3.2.3, nos damos cuenta que a mayor numero de datos distintos , mayor será el numero de capas utilizado por la estructura compacta k^3-tree. La Figura 6.3)(a) muestra la gráfica del espacio utilizado para una distribución de datos normal y la Figura 6.3)(b) sobre conjuntos de datos aleatorios.

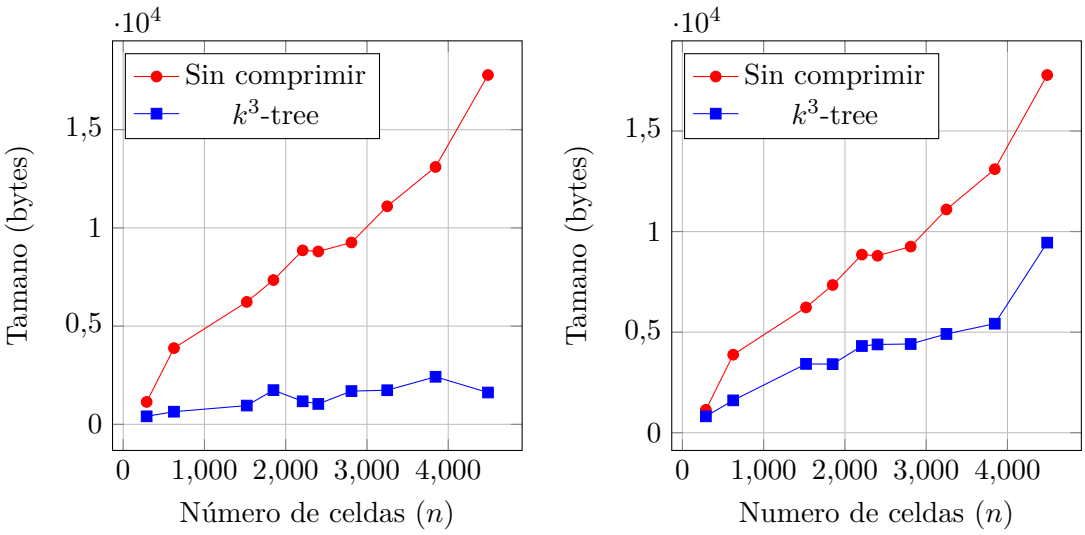
6.3. Experimentación para medir tiempos de ejecución

Para medir los tiempos de ejecución se tomaron en cuenta 2 factores (i) El tiempo de obtención de celdas, (ii) El tiempo de ejecución para los algoritmos encargados de contestar las consultas Locales, Focales, Zonales y Globales, correspondientes a cada uno de los operadores del algebra de mapas, considerando el uso y no uso de estructuras de datos compactas, sobre datos con distribución normal como aleatoria, presentados en la sección 6.2. Para obtener los tiempos correspondientes se ocupa la función microtime() de php expresada en milisegundos (ms), donde los resultados fueron los siguientes.

6.3.1. Tiempos de obtención de celdas

Para medir el tiempo de obtención de celdas se compara el uso de la estructura compacta k^3-tree con respecto a buscar los datos directamente desde un archivo de texto, esto se presenta cuando un usuario quiere visualizar un ráster en la aplicación Web, sin aplicar ningún tipo de consulta. En la Tabla 6.2 se muestran los tiempos de obtención de celdas para las ciudades descritas en la sección 6.2 considerando el uso y no uso de la estructura de datos compacta k^3-tree.

De los resultados obtenidos de la Tabla 6.2 podemos concluir que la estructura compacta de datos k^3-tree se demora mas en obtener las celdas desde una distribución de datos aleatoria, sobre una distribución de datos normal. Podemos ver que la diferencia de obtención de celdas desde un archivo txt no es tan notoria con respecto al uso de la estructura de datos compacta. Según los gráficos obtenidos de la Figura 6.4(a) y Figura 6.4(b) nos



(a)Compactación distribución normal (b)Compactación distribución aleatoria

Figura 6.3: Uso de memoria de la estructura compacta k^3-tree v/s datos sin compactar

damos cuenta que al tener mas de 4000 datos dentro de una matriz, existe una diferencia de 2 ms con respecto a obtener las celdas desde un k^3-tree, con respecto a un archivo de texto. También podemos ver que a medida que aumenta el numero de celdas, mayor es el tiempo de obtención de estas, tanto para datos alojados dentro de un archivo txt como datos alojados dentro de la estructura compacta k^3-tree.

6.3.2. Experimentación para medir tiempo de consultas

Para cada consulta se considera un particionamiento de $k = 2$ para una matriz de datos tridimensional. La Tabla 6.3 y Tabla 6.4 muestran, respectivamente, los tiempos de ejecución para las consultas de Tipo local 1 (suma, resta, multiplicación y división entre 2 rasters), considerando los data sets de rasters presentados en este proyecto, bajo distribución normal y distribución aleatoria. La Figura 6.5 y Figura 6.6 muestran las gráficas para estas consultas de Tipo Local 1. La Tabla 6.5 y Tabla 6.6 muestran, respectivamente, los tiempos de ejecución para las consultas de Tipo local 2 (suma, resta, multiplicación, división , entre un raster y un escalar igual a 100), bajo datasets con distribución normal y distribución aleatoria. La Figura 6.7 y Figura 6.8, muestran las gráficas para la consulta de Tipo Local 2. La Tabla 6.7 y Tabla 6.8 muestran, respectivamente, los tiempos de ejecución para las consultas de Tipo local 3 (mayor, menor, igual y distinto, considerando un escalar igual a 100), bajo datasets con distribución normal y distribución aleatoria. La Figura 6.9 y Figura 6.10, muestran las gráficas para la consulta de Tipo Local 3. La Tabla 6.9 y Tabla 6.10 muestran, respectivamente, los tiempos de ejecución para las consultas de Tipo Focal, considerando un rango igual a 3, 7, y 11, para el calculo de la media, bajo datasets con distribución normal y distribución aleatoria. La Figura 6.11 y Figura 6.12,

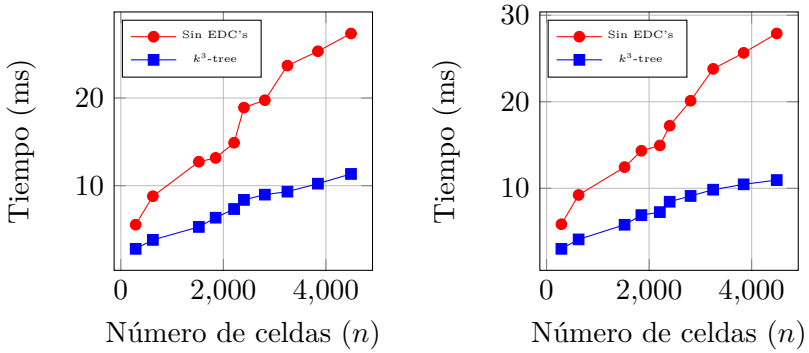
Data set	Celdas	# Tiempos de obtención de celdas (ms)			
		Distribución normal		Distribución aleatoria	
		Sin EDC's	k^3-tree	Sin EDC's	k^3-tree
data set 1	289	5.55	2.80	5.85	3.01
data set 2	625	8.80	3.82	9.23	4.10
data set 3	2809	19.75	8.98	20.12	9.12
data set 4	3844	25.33	10.23	25.65	10.46
data set 5	2401	18.91	8.38	17.23	8.45
data set 6	1849	13.17	6.35	14.34	6.90
data set 7	2209	14.90	7.34	14.95	7.25
data set 8	4489	27.36	11.34	27.89	10.95
data set 9	1521	12.73	5.30	12.45	5.79
data set 10	3249	23.70	9.32	23.80	9.84

Tabla 6.2: Tiempos de ejecución en milisegundos (ms), para obtención de celdas (distribución normal y distribución aleatoria)

muestran las gráficas para la consulta de Tipo Focal. La Tabla 6.11 y Tabla 6.12 muestran, respectivamente, los tiempos de ejecución para las consultas de Tipo Zonal, considerando intervalos de (10,20),(10,40), y (10,60), bajo datasets con distribución normal y distribución aleatoria. La Figura 6.13 y Figura 6.14, muestran las gráficas para la consulta de Tipo Zonal. La Tabla 6.13 y Tabla 6.14 muestran, respectivamente, los tiempos de ejecución para las consultas de Tipo Global, Considerando una posición x, y de (0,0), (5,5), (10,10), bajo datasets con distribución normal y distribución aleatoria. La Figura 6.15 y Figura 6.16, muestran las gráficas para la consulta de Tipo Global, considerando una distribución normal y aleatoria.

Data set	Celdas (n)	# Tiempos de ejecución							
		suma		resta		multiplicación		división	
		Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree
data set 1	289	12.45	6.23	12.42	6.24	11.92	6.12	12.93	6.99
data set 2	625	14.3	7.23	14.64	7.48	14.93	7.92	13.23	7.23
data set 3	2809	29.84	16.42	30.13	14.43	30.45	15.99	29.34	15.21
data set 4	3844	34.21	20.94	35.34	18.54	36.01	19.93	36.23	19.43
data set 5	2401	30.08	13.23	29.43	12.98	29.32	15.01	25.23	14.47
data set 6	1849	19.64	10.23	19.34	9.93	20.23	11.01	20.2	11.23
data set 7	2209	22.23	11.91	21.35	10.12	22.03	12.23	21.23	12.73
data set 8	4489	39.30	21.01	40.32	20.91	39.34	21.04	40.34	21.91
data set 9	1521	12.73	8.24	15.34	8.21	15.03	10.12	16.9	9.23
data set 10	3249	33.27	19.45	32.45	16.48	33.23	17.43	31.45	17.32

Tabla 6.3: Tiempos de ejecución en milisegundos (ms) para suma, resta, multiplicación y división entre rásteres (distribución normal)

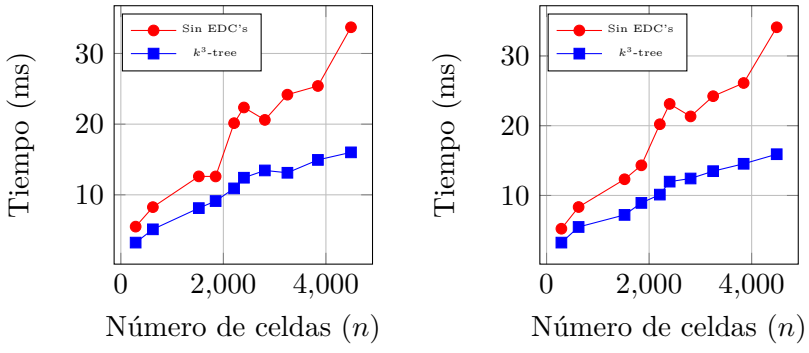


(a) Distribución normal (b) Distribución aleatoria

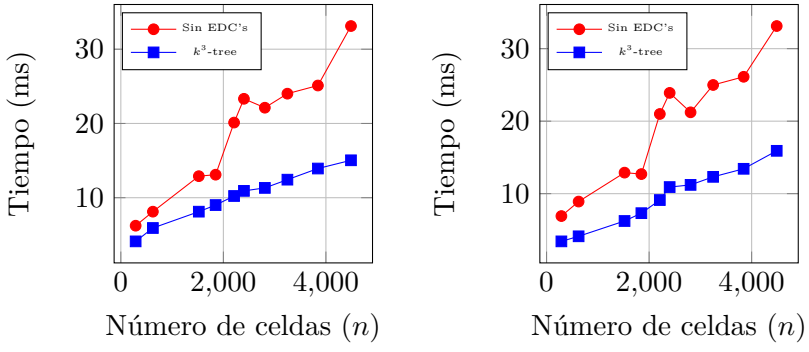
Figura 6.4: Tiempos de ejecución para obtención de celdas

Data set	Celdas (n)	# Tiempos de ejecución							
		suma		resta		multiplicación		división	
		Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree
data set 1	289	12.34	7.23	12.34	7.24	12.34	7.12	12.45	7.99
data set 2	625	14.34	8.23	14.64	8.48	13.34	8.92	13.34	8.23
data set 3	2809	29.84	16.23	30.13	14.01	31.34	16.34	29.34	16.21
data set 4	3844	33.92	21.94	35.34	18.34	35.56	19.93	36.67	19.12
data set 5	2401	31.54	13.99	29.43	12.98	28.25	15.01	25.34	15.12
data set 6	1849	19.54	10.11	20.24	9.93	18.34	11.45	20.23	11.56
data set 7	2209	23.34	12.91	21.54	11.01	22.45	12.34	21.34	13.34
data set 8	4489	38.90	22.01	40.32	20.95	39.04	21.11	40.75	21.23
data set 9	1521	13.34	8.98	15.12	9.01	15.45	10.01	16.54	10.23
data set 10	3249	33.44	20.12	32.45	16.65	33.75	17.23	31.45	17.98

Tabla 6.4: Tiempos de ejecución para suma, resta, multiplicación y división entre rásteres (distribución aleatoria)

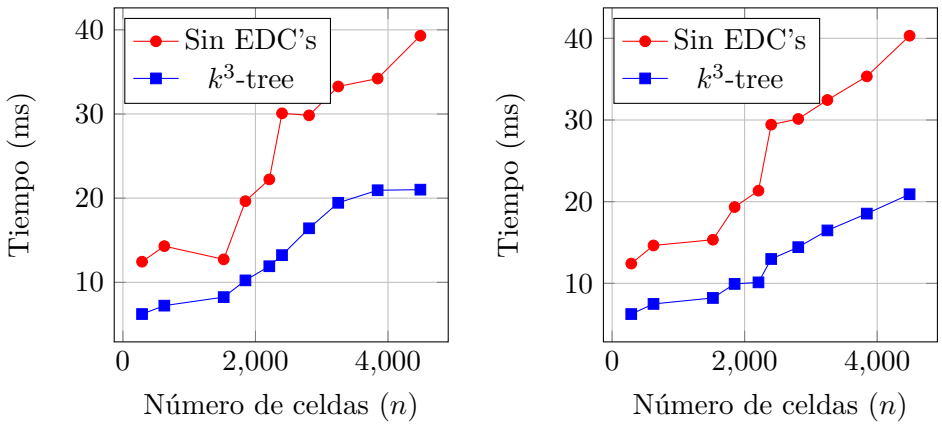


(a) Suma escalar = 100 (b) Resta escalar = 100



(a) Multiplicación escalar = 100 (b) División escalar = 100

Figura 6.7: Tiempos de ejecución en milisegundos (ms) para Consulta Local 2 (distribución normal)



(a) Tiempos de ejecución suma

(b) Tiempos de ejecución resta

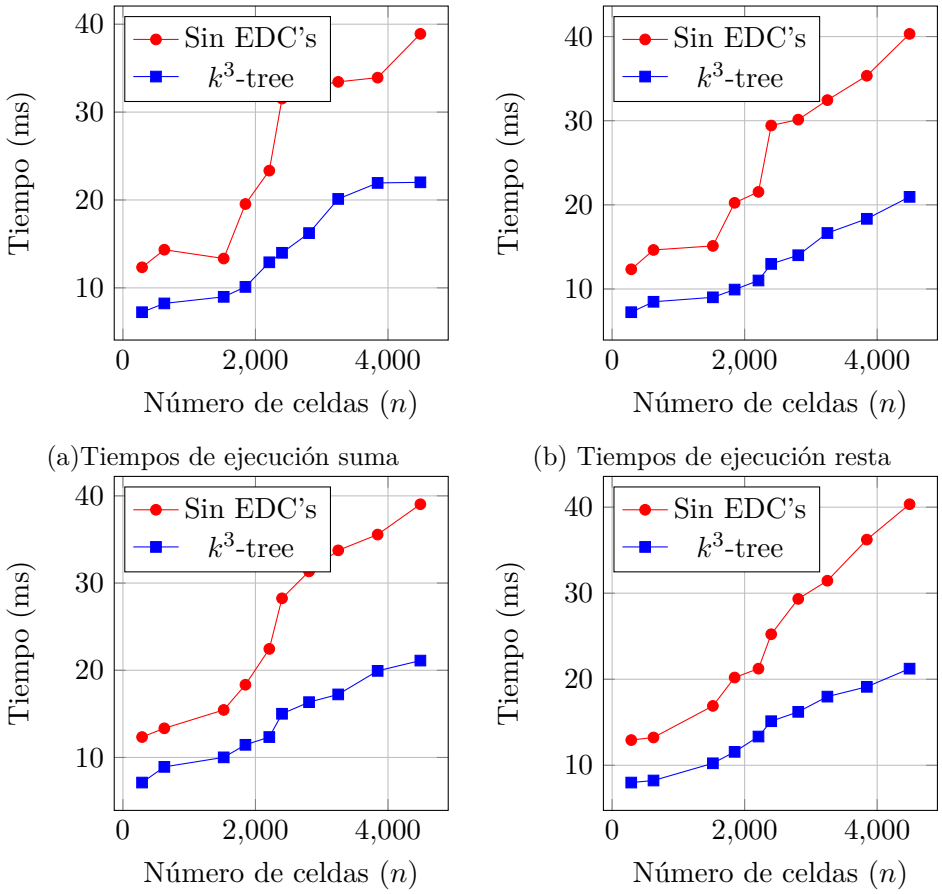
(c) Tiempos de ejecución multiplicación

(d) Tiempos de ejecución división

Figura 6.5: Tiempos de ejecución para Consulta Local 1 (distribución normal)

De los resultados obtenidos de la consulta Local 1 (ver Algoritmo 2 y Algoritmo 9), podemos decir que a medida que aumenta el tamaño de los 2 rasters de entrada (dimension), la estructura de datos compacta k^3-tree demora mas en realizar operaciones entre los rasters seleccionados. Dado los resultados mostrados en la Tabla 6.3 y Tabla 6.4, al comparar el raster del data set 1 (n = 289) , con el resto de ciudades seleccionadas, se puede ver que la estructura compacta k^3-tree redujo al menos la mitad del tiempo en comparación a los algoritmos utilizados que no cuentan con la estructura compacta de datos k^3-tree (ver Algoritmo 9). Al comparar con 2 tipos de datasets, se pudo ver que por diferencias muy leves la estructura comparta k^3-tree tiene un mejor desempeño en realizar las pruebas con datasets con distribución normal sobre los datasets con distribución aleatoria mostrando, diferencias de aproximadamente 1 milisegundo (ms), entre cada prueba realizada. Esto se puede apreciar de mejor forma en la Gráfica de la Figura 6.5 y Figura 6.6, respectivamente.

De los resultados obtenidos de la consulta Local 2 (ver Algoritmo 3 y Algoritmo 10) tanto las operaciones de suma, resta, multiplicación y división , tienen tiempos de ejecución



(a)Tiempos de ejecución suma

(b) Tiempos de ejecución resta

(c) Tiempos de ejecución multiplicación

(d) Tiempos de ejecución división

Figura 6.6: Tiempos de ejecución en milisegundos (ms) para Consulta Local, entre rásteres (distribución aleatoria)

similares al aplicar un escalar de 100, considerando el uso de la estructura compacta k^3-tree. Si comparamos su desempeño con el Algoritmo 10), al ver la Tabla 6.5 y Tabla 6.6, nos damos cuenta que la estructura compacta k^3-tree reduce aproximadamente la mitad del tiempo de ejecución. También se aprecia que a medida que aumenta la dimension de los rasters de entrada, el tiempo de ejecución es mayor, aumentando en promedio 2 milisegundos(ms) cada 500 celdas analizadas. Al comparar con 2 tipos de datasets, se pudo observar que por diferencias muy leves la estrictura comparta k^3-tree tiene un mejor desempeño al realizar las pruebas con datasets con distribución normal sobre los datasets con distribución aleatoria, mostrando diferencias de aproximadamente 1 milisegundo (ms), entre cada prueba realizada.Esto se puede apreciar de mejor forma en la Gráfica de la Figura 6.7 y Figura 6.8, respectivamente.

De los resultados obtenidos de la consulta Local 3 (ver Algoritmo 4 y Algoritmo 11, podemos decir que tanto las operaciones de mayor, menor, igual, distinto, tienen tiempos

		# Tiempos de ejecución							
		suma = 100		resta = 100		multiplicación = 100		división = 100	
Data set	Celdas (n)	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree
data set 1	289	5.50	3.24	5.23	3.24	6.23	4.12	6.91	3.42
data set 2	625	8.26	5.12	8.34	5.48	8.12	5.92	8.91	4.12
data set 3	2809	20.60	12.42	21.33	12.43	22.12	11.32	21.22	11.21
data set 4	3844	25.40	14.94	26.12	14.54	25.11	13.93	26.12	13.43
data set 5	2401	22.36	11.23	23.12	11.98	23.32	10.92	23.9	10.91
data set 6	1849	12.60	9.12	12.32	8.93	13.1	9.01	12.7	7.31
data set 7	2209	20.14	10.91	20.21	10.12	20.12	10.23	20.99	9.12
data set 8	4489	33.71	16.01	34.11	15.91	33.12	15.04	33.12	15.91
data set 9	1521	12.61	8.12	12.32	7.21	12.9	8.12	12.9	6.23
data set 10	3249	24.17	13.45	24.23	13.48	24.01	12.43	24.99	12.32

Tabla 6.5: Tiempos de ejecución para Consulta Local 2 (suma, resta, multiplicación y división), considerando escalar de 100 (distribución normal)

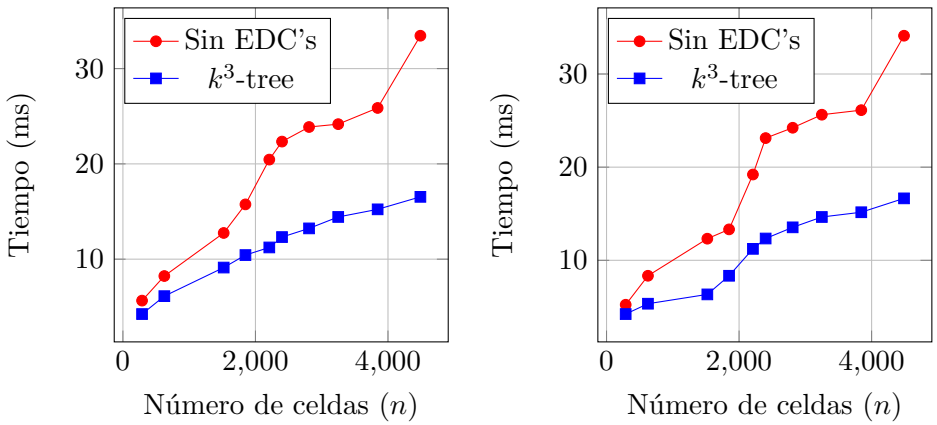
		# Tiempos de ejecución							
		suma = 100		resta = 100		multiplicación = 100		división = 100	
Data set	Celdas (n)	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree
data set 1	289	5.65	4.24	5.23	4.23	6.23	4.12	6.91	4.12
data set 2	625	8.23	6.12	8.34	5.34	8.12	5.92	8.91	5.11
data set 3	2809	23.87	13.23	24.23	13.54	24.12	11.32	24.23	12.21
data set 4	3844	25.87	15.23	26.12	15.16	26.11	13.93	27.23	14.43
data set 5	2401	22.34	12.32	23.12	12.34	23.32	10.92	23.54	11.91
data set 6	1849	15.75	10.43	13.32	8.34	13.1	9.01	13.43	7.23
data set 7	2209	20.45	11.23	19.21	11.23	20.12	10.23	20.23	9.23
data set 8	4489	33.45	16.54	34.11	16.65	33.12	15.04	33.23	16.91
data set 9	1521	12.75	9.12	12.32	6.34	12.9	8.12	10.23	6.10
data set 10	3249	24.12	14.43	25.63	14.65	25.01	12.43	25.54	13.32

Tabla 6.6: Tiempos de ejecución para Consulta Local 2 (suma, resta, multiplicación y división), considerando escalar de 100 (distribución aleatoria)

		# Tiempos de ejecución							
		mayor = 100		menor = 100		igual = 100		distinto = 100	
Data set	Celdas (n)	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree
data set 1	289	6.55	3.12	6.23	3.74	6.23	3.84	6.21	3.43
data set 2	625	9.80	4.54	9.11	4.36	9.11	4.93	9.21	5.35
data set 3	2809	20.75	10.93	20.01	10.93	20.2	10.74	20.47	10.35
data set 4	3844	28.33	14.23	27.11	15.83	29.11	14.75	29.12	14.99
data set 5	2401	19.91	10.23	19.12	10.36	20.1	9.64	20.23	9.46
data set 6	1849	16.17	7.12	15.12	7.34	16.11	7.94	16.22	8.19
data set 7	2209	15.90	8.12	16.22	7.93	16.11	8.64	16.91	8.35
data set 8	4489	30.36	15.12	31.22	16.01	31.22	15.37	31.23	16.01
data set 9	1521	13.73	6.12	12.9	5.23	11.21	5.84	13.73	7.01
data set 10	3249	25.70	13.12	24.71	14.74	25.21	12.84	25.23	13.47

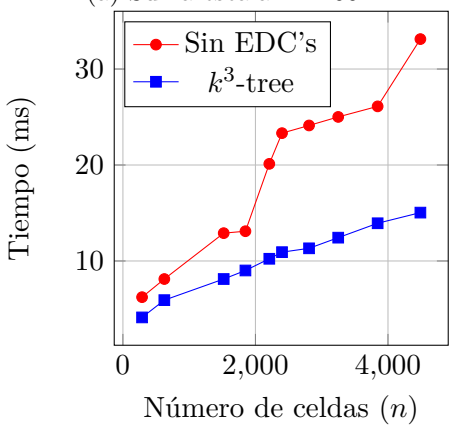
Tabla 6.7: Tiempos de ejecución para Consulta Local 3 (mayor, menor, igual y distinto), considerando escalar de 100 (distribución normal)

de ejecución similares al aplicar el condicional con un escalar de 100, acercándose a los tiempos de obtener todas las celdas, al usar la estructura compacta de datos k^3-tree. Si comparamos el desempeño del Algoritmo 4, al ver la Tabla 6.7 y Tabla 6.8, nos damos cuenta que la estructura compacta k^3-tree reduce aproximadamente la mitad del tiempo de ejecución con respecto al Algoritmo 11. También se aprecia que a medida que aumenta la dimension de los rasters de entrada, el tiempo de ejecución es mayor, aumentando en promedio 1 milisegundos(ms) cada 300 celdas analizadas. Al comparar con 2 tipos de

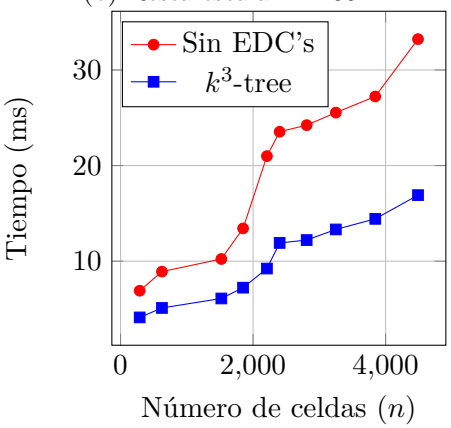


(a) Suma escalar = 100

(b) Resta escalar = 100



(c) Multiplicación escalar = 100



(d) División escalar = 100

Figura 6.8: Tiempos de ejecución en milisegundos (ms) para Consulta Local 2 (distribución aleatoria)

Data set	Celdas (n)	# Tiempos de ejecución							
		mayor = 100		menor = 100		igual = 100		distinto = 100	
		Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree
data set 1	289	6.98	4.19	6.65	4.23	7.11	4.99	7.01	4.54
data set 2	625	10.01	5.12	10.56	5.23	9.45	5.23	8.91	5.23
data set 3	2809	20.23	11.34	21.23	11.34	21.22	11.11	20.95	12.95
data set 4	3844	28.23	15.93	28.11	16.23	29.45	14.99	27.45	15.11
data set 5	2401	19.23	10.94	18.34	10.94	20.54	10.23	20.26	11.45
data set 6	1849	16.01	8.23	14.76	8.23	16.43	8.01	15.94	9.12
data set 7	2209	16.98	9.91	16.11	9.23	17.23	9.11	17.12	10.65
data set 8	4489	30.11	16.23	32.30	17.12	31.11	15.45	31.23	16.99
data set 9	1521	13.32	7.23	12.99	6.12	11.23	6.23	12.34	7.23
data set 10	3249	24.01	14.23	25.54	15.23	25.34	13.23	24.65	14.23

Tabla 6.8: Tiempos de ejecución para Consulta Local 3 (mayor, menor, igual y distinto), considerando escalar de 100 (distribución aleatoria)

		# Tiempos de ejecución					
		rango = 3		rango = 7		rango = 11	
Data set	Celdas (n)	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree
data set 1	289	6.36	3.12	12.93	5.91	15.38	6.23
data set 2	625	9.00	6.34	16.52	8.12	21.050	10.45
data set 3	2809	32.15	17.89	46.88	22.12	61.40	32.43
data set 4	3844	54.83	26.54	61.95	29.53	93.73	41.64
data set 5	2401	30.88	15.23	37.34	19.73	69.48	31.23
data set 6	1849	21.39	11.34	29.17	15.43	41.78	20.93
data set 7	2209	27.20	13.53	33.52	17.54	55.76	26.48
data set 8	4489	58.63	30.21	88.68	37.84	116.60	49.23
data set 9	1521	17.7	9.23	25.12	12.43	40.13	19.23
data set 10	3249	45.61	23.23	61.62	28.43	86.18	37.65

Tabla 6.9: Tiempos de ejecución para Consulta Focal con respecto al calculo de media (distribución normal)

		# Tiempos de ejecución					
		rango = 3		rango = 7		rango = 11	
Data set	Celdas (n)	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree
data set 1	289	6.23	4.12	12.44	6.91	15.23	7.23
data set 2	625	9.23	6.78	16.75	9.12	22.23	11.34
data set 3	2809	32.34	18.56	45.34	23.65	67.34	33.76
data set 4	3844	55.34	26.43	63.34	30.23	93.73	41.97
data set 5	2401	31.45	15.97	37.65	19.98	62.23	31.76
data set 6	1849	22.23	11.78	29.23	15.64	46.35	21.34
data set 7	2209	27.95	14.56	34.23	18.34	55.34	26.87
data set 8	4489	58.87	30.76	89.12	37.54	116.60	50.45
data set 9	1521	18.34	9.98	26.34	13.43	41.34	20.32
data set 10	3249	46.34	23.43	60.13	29.34	70.45	38.45

Tabla 6.10: Tiempos de ejecución para Consulta Focal para el calculo de media (distribución aleatoria)

		# Tiempos de ejecución					
		p = 10, q = 20		p = 10, q = 40		p = 10, q = 60	
Data set	Celdas (n)	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree
data set 1	289	6.44	3.54	6.91	3.98	6.74	3.92
data set 2	625	7.29	4.45	8.12	4.13	8.21	4.95
data set 3	2809	20.76	13.45	23.23	13.43	22.12	12.98
data set 4	3844	26.83	16.12	29.12	16.23	28.12	17.23
data set 5	2401	20.97	12.12	20.24	12.43	21.23	12.74
data set 6	1849	15.51	8.23	15.34	8.12	15.63	8.43
data set 7	2209	23.32	11.92	21.74	12.23	23.73	12.34
data set 8	4489	28.06	18.23	30.12	18.48	30.22	18.93
data set 9	1521	13.61	7.33	14.01	7.93	14.83	8.01
data set 10	3249	28.95	15.34	28.12	14.34	28.12	14.93

Tabla 6.11: Tiempos de ejecución para Consulta Zonal (distribución normal)

datasets, se pudo observar que por diferencias muy leves la estrictura comparta k^3-tree tiene un mejor desempeño al realizar las pruebas con datasets con distribución normal sobre los datasets con distribución aleatoria mostrando diferencias de aproximadamente 1

Data set	Celdas (n)	# Tiempos de ejecución					
		p = 10, q = 20		p = 10, q = 40		p = 10, q = 60	
		Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree
data set 1	289	6.21	4.12	6.83	4.98	6.91	5.84
data set 2	625	7.12	5.12	8.45	5.13	8.23	46.34
data set 3	2809	26.11	14.01	23.23	14.34	25.34	14.12
data set 4	3844	27.12	17.12	29.23	17.45	29.06	17.23
data set 5	2401	25.31	12.93	21.01	13.12	24.23	13.23
data set 6	1849	15.23	9.23	16.23	9.23	16.23	10.34
data set 7	2209	24.12	10.92	20.92	12.23	23.93	12.91
data set 8	4489	28.43	19.01	30.93	18.90	30.22	19.93
data set 9	1521	14.23	7.11	15.34	7.34	15.23	9.23
data set 10	3249	28.23	16.23	27.12	16.23	27.23	15.95

Tabla 6.12: Tiempos de ejecución para Consulta Zonal (distribución aleatoria)

Data set	Celdas (n)	# Tiempos de ejecución					
		x = 0, y = 0		x = 5, y = 5		x = 10, y = 10	
		Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree
data set 1	289	6.49	5.23	10.14	7.12	6.24	5.12
data set 2	625	16.13	14.12	16.43	14.12	16.12	15.12
data set 3	2809	30.02	29.1.12	28.70	28.12	29.12	30.12
data set 4	3844	36.07	35.12	36.59	35.12	37.12	34.12
data set 5	2401	33.49	32.12	29.32	29.12	31.23	29.12
data set 6	1849	28.78	25.12	23.80	20.12	25.12	23.12
data set 7	2209	28.75	26.12	31.14	29.12	29.98	27.12
data set 8	4489	44.21	40.12	45.75	42.12	44.12	42.12
data set 9	1521	20.04	17.92	28.70	25.12	22.12	23.34
data set 10	3249	39.80	38.12	34.96	33.12	37.12	32.12

Tabla 6.13: Tiempos de ejecución para Consulta Global (distribución normal)

Data set	Celdas (n)	# Tiempos de ejecución					
		x = 0, y = 0		x = 5, y = 5		x = 10, y = 10	
		Sin EDC's	k^3-tree	Sin EDC's	k^3-tree	Sin EDC's	k^3-tree
data set 1	289	6.23	5.34	10.33	7.12	6.23	5.23
data set 2	625	16.34	14.23	15.32	14.96	17.23	15.12
data set 3	2809	33.11	32.23	33.23	31.53	32.65	31.23
data set 4	3844	39.12	39.12	37.23	35.23	37.14	34.75
data set 5	2401	32.23	31.45	32.23	30.23	31.65	29.76
data set 6	1849	28.23	26.12	24.23	25.65	24.75	24.53
data set 7	2209	29.23	27.23	29.23	29.64	27.34	28.43
data set 8	4489	44.21	43.13	43.11	42.33	43.43	42.57
data set 9	1521	20.23	17.23	20.23	23.34	23.34	23.34
data set 10	3249	38.23	38.12	34.12	33.43	35.34	32.66

Tabla 6.14: Tiempos de ejecución para Consulta Global (distribución aleatoria)

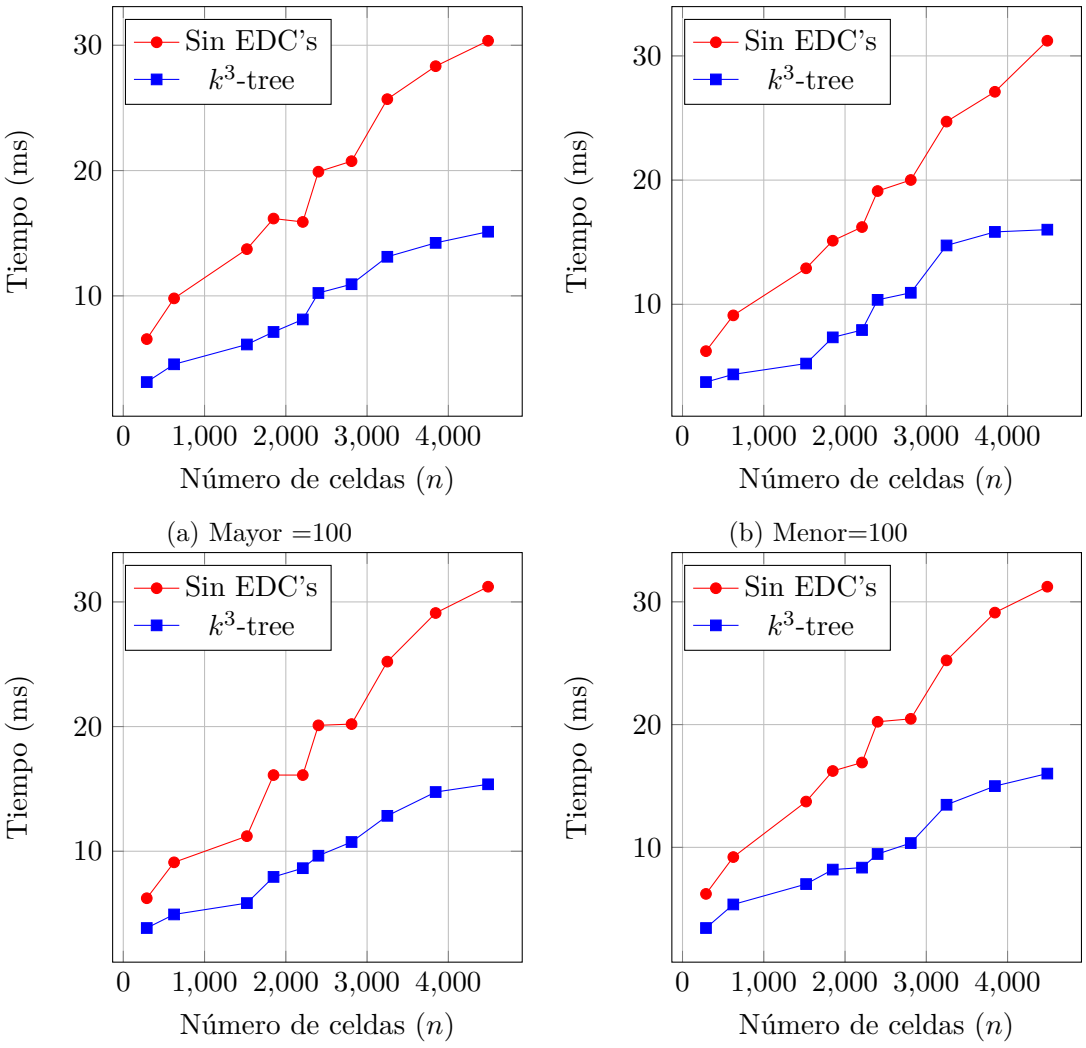
milisegundo (ms), entre cada prueba realizada. Esto se puede apreciar de mejor forma en la Figura 6.9 y Figura 6.10, respectivamente.

De los resultados obtenidos de la consulta Focal (ver Algoritmo 5 y Algoritmo 12) podemos decir que a medida que aumenta la dimension del raster seleccionado, y el rango ingresado por el usuario, aumenta en forma progresiva el tiempo de ejecución que tarda la

estructura compacta k^3-tree en realizar la consulta. Como se puede ver en La Tabla 6.9 y Tabla 6.10, al tener un rango de 3, 7, y 11, el tiempo subió en promedio 20 milisegundos(ms), considerando la ciudad de Los Angeles (n = 4489) y 3 milisegundos (ms), considerando la ciudad de San pedro (n = 289). Aun así la estructura compacta de datos k^3-tree logra reducir aproximadamente la mitad del tiempo en comparación con utilizar los algoritmos que no ocupan la estructura de datos compacta (ver algoritmo 12). Al comparar con 2 tipos de datasets, se pudo observar que por diferencias muy leves la estrictura comparta k^3-tree tiene un mejor desempeño al realizar las pruebas con datasets con distribución normal sobre los datasets con distribución aleatoria, mostrando diferencias de aproximadamente 1 milisegundo (ms), entre cada prueba realizada. Esto se puede apreciar de mejor forma en la Gráfica de la Figura 6.9 y Figura 6.10, respectivamente.

De los resultados obtenidos de Consulta Zonal (ver Algoritmo 7 y Algoritmo 14), podemos decir que al realizar las pruebas con 3 tipos de intervalos diferentes (10.20) , (10,40) , (10,60), los tiempos de ejecución varían entre medio milisegundo(ms), al usar la estructura compacta de datos k^3-tree. Si comparamos el desempeño del Algoritmo 7, al ver La Tabla 6.11 y Tabla 6.12, nos damos cuenta que la estructura compacta k^3-tree reduce aproximadamente la mitad del tiempo de ejecución, con respecto al Algoritmo 14). También se aprecia que a medida que aumenta la dimension de los rasters de entrada, el tiempo de ejecución es mayor, aumentando en promedio 2 milisegundos(ms) cada 300 celdas analizadas. Al comparar con 2 tipos de datasets, se pudo observar que por diferencias muy leves la estructura comparta k^3-tree tiene un mejor desempeño al realizar las pruebas con datasets con distribución normal sobre los datasets con distribución aleatoria, mostrando diferencias de aproximadamente 1 milisegundo (ms), entre cada prueba realizada. Esto se puede apreciar de mejor forma en la Gráfica de la Figura 6.13 y Figura 6.14, respectivamente.

De los resultados obtenidos de la consulta Global podemos decir que al comparar los 2 tipos de Algoritmos presentados (ver Algoritmo 8 y Algoritmo 15), los tiempos de ejecución fueron bastante similares, esto lo podemos ver en la Tabla 6.13 y Tabla 6.14, puesto que como se dijo en el capitulo 4.1, tanto el Algoritmo 8 y el Algoritmo 8, solo necesitan de la celda que se va a consultar, en este caso las coordenadas (x,y) en el raster seleccionado. Sin embargo, los tiempos mostrados corresponden al tiempo que toma el algoritmo en construir la matriz de distancias, mas la búsqueda de la celda a analizar, si comparamos el hecho de buscar la celda en el Algoritmo 8, el tiempo de búsqueda fue menor en comparación con el tiempo presentado en el algoritmo 15, las diferencias entre uno y otro son de 3 a 4 milisegundos(ms), con respecto aumenta el numero de celdas. Si consideramos el hecho de comparar con distintos puntos (0,0), (5,5), (10,10), estos pueden presentar diferencias pequeñas pero siempre se mantendrá un tiempo similar de búsqueda, si consideramos un raster con el mismo numero de celdas, a medida que las celdas aumentan, el tiempo de ejecución sera mayor, sin importar el punto que se quiera analizar. Con respecto al uso de datasets con distribución normal y aleatoria, no hay diferencias significativas puesto que solo se necesita el punto (x, y) en el raster a analizar. Esto se puede apreciar de mejor forma en la Gráfica de la Figura 6.15 y Figura 6.16, respectivamente.



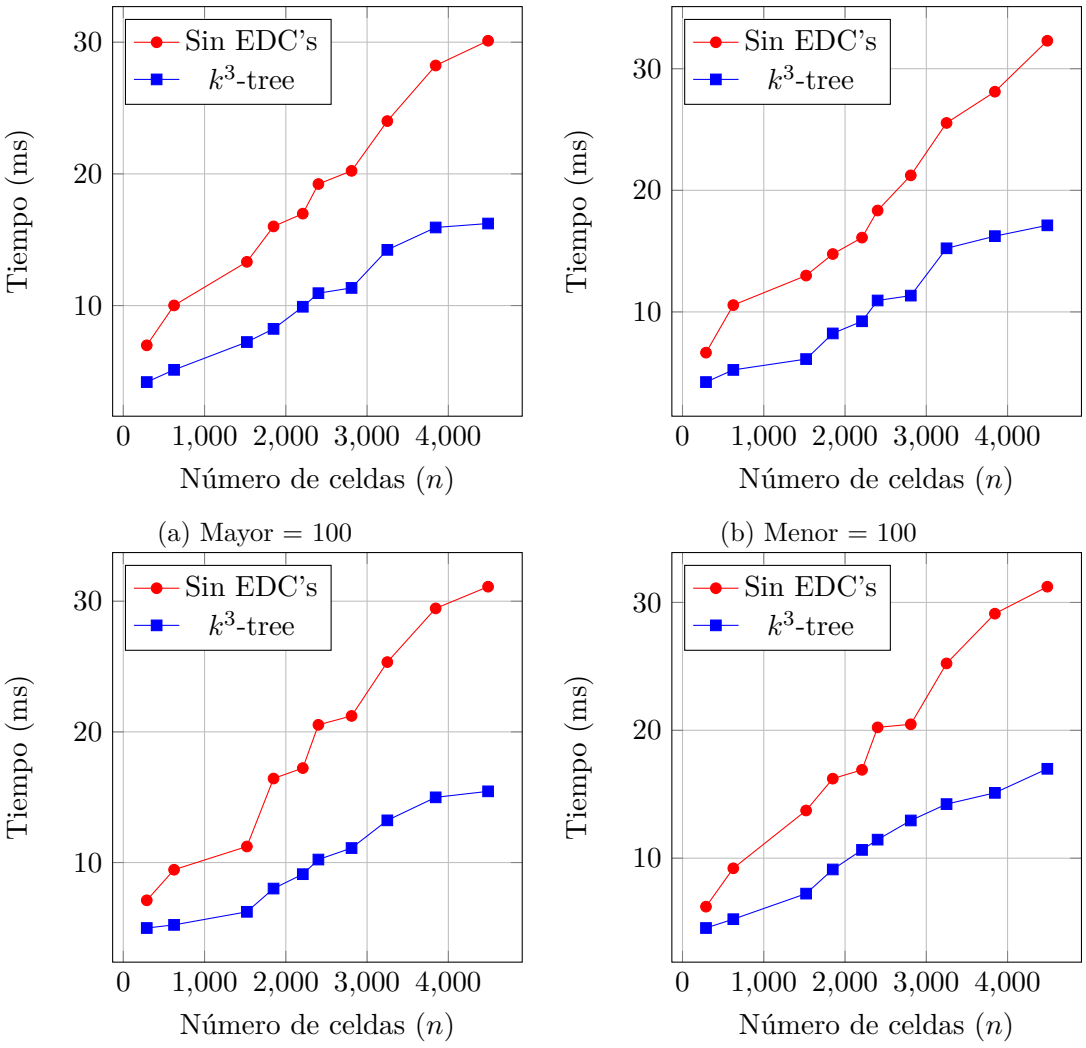
(a) Mayor =100

(b) Menor=100

(c) Igual =100

(d) Distinto =100

Figura 6.9: Tiempos de ejecución en milisegundos (ms) para Consulta Local 3 (distribución normal)



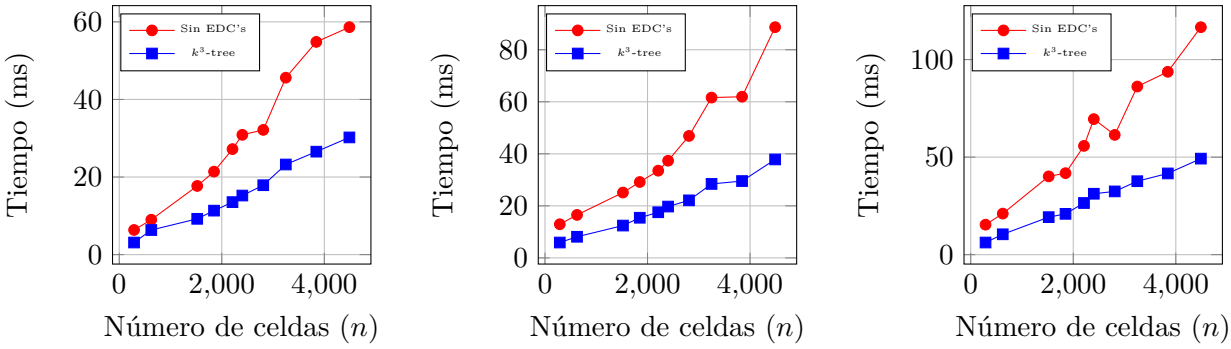
(a) Mayor = 100

(b) Menor = 100

(c) Mayor = 100

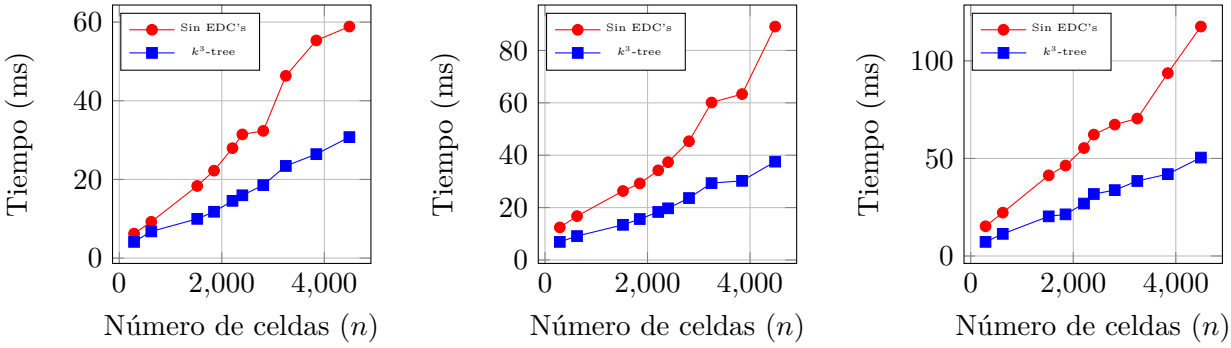
(d) Distinto = 100

Figura 6.10: Tiempos de ejecución en milisegundos (ms) para Consulta Local 3 (distribución aleatoria)



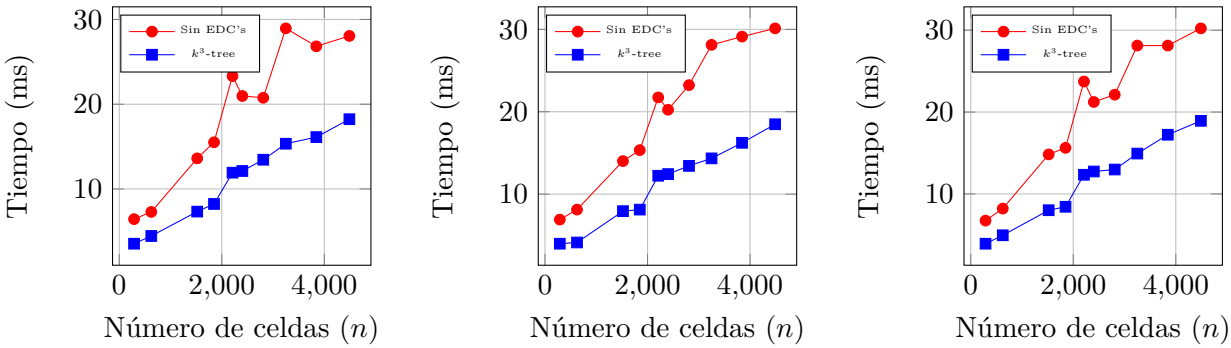
(a) Rango = 5 (b) Rango = 7 (c) Rangco = 11

Figura 6.11: Tiempos de ejecución en milisegundos (ms) para Consulta Focal (distribución normal)



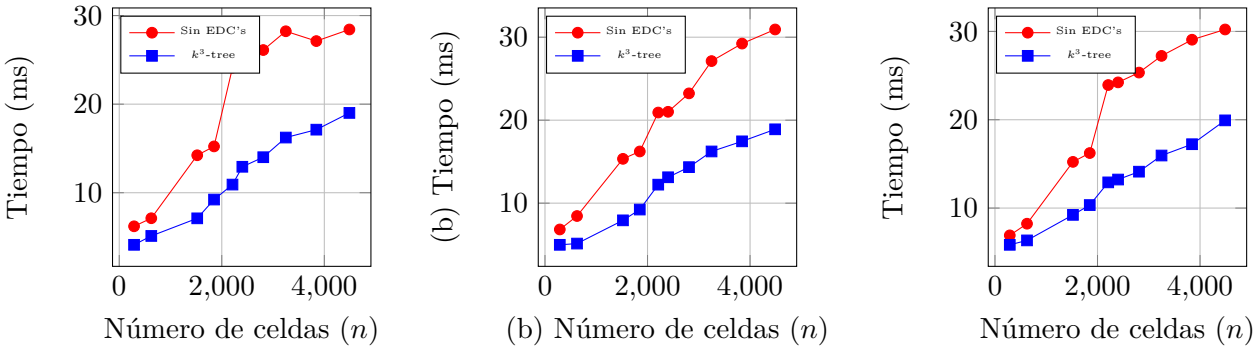
(a) Rango = 5 (b) Rango = 7 (c) Rangco = 11

Figura 6.12: Tiempos de ejecución en milisegundos (ms) para Consulta Focal (distribución aleatoria)



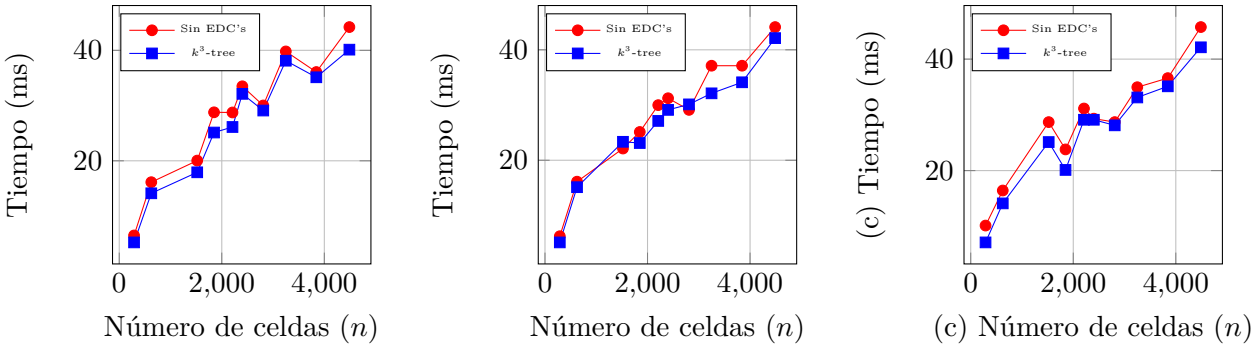
(a) p = 10 , q = 20 (b) p = 10 , q = 40 (a) p = 10 , q = 60

Figura 6.13: Tiempos de ejecución en milisegundos (ms) para Consulta Zonal (distribución normal)



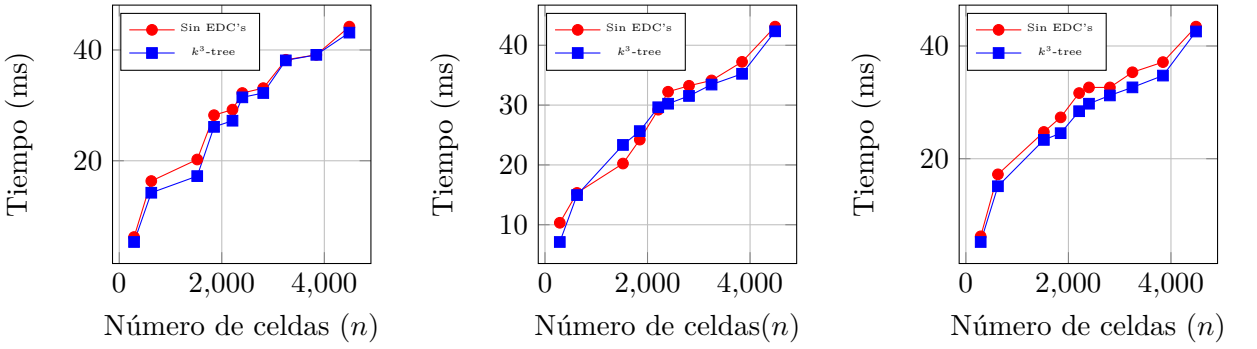
(a) p = 10 , q = 20 (b) p = 10 , q = 40 (c) p = 10 , q = 60

Figura 6.14: Tiempos de ejecución en milisegundos (ms) para consulta Zonal (distribución aleatoria)



(a) $x = 0,\ y = 0$ (b) $x = 5, y = 5$ (c) $x = 10, y = 10$

Figura 6.15: Tiempos de ejecución en milisegundos (ms) para consulta Global (distribución normal)



(a) $x = 0,\ y = 0$ (b) $x = 5, y = 5$ (c) $x = 10, y = 10$

Figura 6.16: Tiempos de ejecución en milisegundos (ms) para consulta Global (distribución aleatoria)

Capítulo 7

Conclusiones y Trabajos Futuros

En este proyecto se presentó una aplicación Web la cual permite resolver consultas sobre datasets de tipo ráster, aplicando los operadores del álgebra de mapas, sobre datos ráster alojados en una estructura de datos compacta k^3-tree. El contexto real que se dio en esta oportunidad fue representar zonas pertenecientes a la región del Bio-Bío, con datos de temperatura en su interior. Dado los experimentos que se realizaron, podemos concluir que las estructuras de datos compactas son una alternativa a considerar para almacenar información sobre datasets de tipo ráster, dado la forma de compactar que presentan. Estas estructuras son capaces de comprimir grandes cantidades de datos ocupando un espacio muy reducido, donde al hacer las pruebas de almacenamiento, nos dimos cuenta que la estructura compacta de datos k^3-tree puede llegar incluso a compactar un 90 por ciento de los datos, al sobrepasar las 4000 celdas en un ráster de entrada, todo esto sin perder la velocidad de acceso a información.

Mediante el uso de la estructura compacta de datos k^3-tree, se logró demostrar que es posible implementar consultas que respondan a los operadores del algebra de mapas. Es este proyecto se analizaron consultas locales, Focales, Zonales y Globales, haciendo uso de ejemplos prácticos bajo datasets con distribución de datos en un rango de 1 a 10 y con distribución de datos de forma aleatoria, variando en el número de celdas de cada ráster, para ver en cada situación el tiempo de ejecución de consulta que presenta cada uno de estos operadores, logrando reducir en algunos casos hasta la mitad del tiempo de ejecución con respecto a obtener los datos directamente desde un archivo de texto. Se pudo observar que las consultas con más tiempo de ejecución son las consultas encargadas de calcular la media con respecto a un rango q, ingresado pro el usuario, donde a medida que se aumenta el valor de q, el tiempo de consulta aumentaba progresivamente debido al incremento del numero de celdas a analizar.

Además, se pudo demostrar que a mayor número de celdas, mayor será la compactación mostrada por la estructura compacta k^3-tree. Esto a su vez depende de la cantidad de datos distintos que presente el dataset de prueba, ya que, si se quiere representar un ráster con una gran cantidad de celdas y valores aleatorios en su interior, la estructura de datos compacta k^3-tree necesita de más capas para representar estos valores. Según los experimentos de tiempo de ejecución de consultas puede afectar en la velocidad de acceso a información

sobre los datos compactados.

Como trabajos futuros podemos comparar la estructura de datos compacta k^3-tree con otras estructuras de datos compactas que trabajen con datos espaciales de tipo ráster, como, por ejemplo, la estructura compacta de datos k^2-raster (Silva, 2017), comparando tiempos de ejecución de consulta y espacio de compactación de archivos. Si queremos seguir trabajando con la estructura de datos compacta k^3-tree, podríamos analizar el espacio de almacenamiento y tiempo de consultas para rásteres con mayor dimensionalidad o probar datasets que tengan mas de un tipo de valor espacial para analizar, como por ejemplo rasters que representen el suelo de una superficie, con valores de profundidad, humedad, calor, etc.

Referencias

Guillermo Bernardo. *Compact Data Structures for Large and Complex Datasets.* Tesis Doctoral, Universidad de la Coruña, 2014.

Nieves Brisaboa, Susana Ladra, y Gonzalo Navarro. k2-trees for compact web graph representation. págs. 18–30. 2009.

Nieves R. Brisaboa, Miguel R. Luaces, Gonzalo Navarro, y Diego Seco. Space-efficient representations of rectangle datasets supporting orthogonal range querying. *Inf. Syst.*, 38(5):635–655, 2013.

Antonio Fariña, Susana Ladra, Oscar Pedreira, y Ángeles S. Places. Rank and select for succinct data structures. *Electronic Notes Theoretical Computer Science*, 236:131–145, 2009.

Rodrigo González, Szymon Grabowski, Veli Mäkinen, y Gonzalo Navarro. Practical implementation of rank and select queries. En *In Poster Proceedings Volume of 4th Workshop on Efficient and Experimental Algorithms*, págs. 27–38. 2005.

Gonzalo Navarro. Wavelet trees for all. *Journal of Discrete Algorithms*, 25:2–20, 2014.

Rajeev Raman, Venkatesh Raman, y S. Srinivasa Rao. Succinct dynamic data structures. En *Algorithms and Data Structures*, págs. 426–437. Springer Berlin Heidelberg, 2001.

Shashi Shelkhar y Sanjay. Chawla. *Spatial Databases a Tour.* Prentice Hall, 2013.

Fernando Silva. *New data structures and algorithms for the efficient management of large spatial datasets.* Tesis Doctoral, Universidad de la Coruña, 2017.

Cristian Vallejos, Mónica Caniupán, y Gilberto Gutiérrez. K2-treaps to represent and query data warehouses into main memory. En *Proceedings of the 36th International Conference of the Chilean Computer Society*, SCCC 2017, págs.–. 2017.

Michael Worboys y Matt Duckham. *GIS: A Computing Perspective, 2Nd Edition.* CRC Press, Inc., 2004.