Compact Data Structures to Represent and Query Data Warehouses into Main Memory

C. Vallejos, M. Caniupán and G. Gutiérrez

Abstract— **In this paper we propose the use of compact data structures to represent and process Data Warehouses (DWs) into main memory. Compact data structures are data structures that allow compacting the data without losing the capacity of querying the data in their compact form. A DW is a data repository to store historical data for decision support, and consists of *dimensions* and *facts*. The dimensions are abstract concepts that groups data with a similar meaning, usually, they are modelled as hierarchies of levels, which contain elements. The facts are quantitative data associated to dimensions. A data cube is a way to retrieve facts at different levels of granularity, which is achieved by navigation on dimensions hierarchies. Since a DW can store terabytes of data, the efficient processing of data cubes is key in OLAP (On-line Analytical Processing). We show that by using a compact representation of DWs we can improve the use of space in main memory, and achieve better performance for query processing. In this paper we extend a previous work to process aggregate queries with aggregate functions MAX, MIN, COUNT and AVG.**

Keywords— **Databases, data warehousing, compact data structures.**

I. Introducción

La forma tradicional de consultar grandes conjuntos de datos es recuperando los datos desde la memoria secundaria a la memoria principal, esto generalmente, con la ayuda de índices para disminuir la transferencia de datos entre ambos tipos de memoria. Luego, las consultas se procesan en la memoria principal. Sin embargo, en los últimos años ha surgido la tendencia de almacenar y consultar los datos directamente en la memoria principal [1], [2] y [3]. Al tener los datos en memoria principal podemos lograr eficiencia en el procesamiento de consultas debido a la alta velocidad de esta memoria. Por ejemplo, la referencia a memoria principal es 100 nanosegundos, mientras que leer y escribir operaciones desde disco puede tomar 5 milisegundos [3]. Sin embargo, la memoria principal sigue siendo más cara que la memoria secundaria, por lo tanto, es relevante usarla de manera eficiente. Para lograr este objetivo presentamos el uso de *estructuras de datos compactas* (EDC) para representar y consultar Data Warehouses (DWs). Las EDC son estructuras de datos que utilizan una pequeña cantidad de espacio pero permiten operaciones de consulta de manera eficiente [4]. Las EDC permiten procesar grandes conjuntos de datos en la memoria principal, evitando parcial o completamente el acceso a la memoria externa, como el disco; se pueden ubicar en los niveles superiores de la jerarquía de memoria (cercana a la CPU). Las EDC se han utilizado para representar conjuntos de datos de diversos tipos. Por ejemplo, en [5], [6] y [7] se usan para representar grafos de la World Wide Web (WEB). En [8], [9], [10] y [11] se usan para representar documentos en el contexto de recuperación de la información. Además, se han utilizado para mejorar la eficiencia del procesamiento de consultas en SIG (Sistemas de Información Geográfica) [12], [13].

En este artículo, proponemos utilizar la estructura de datos compacta k^2-treap para computar consultas de agregación sobre cubos de datos de DWs. Este trabajo extiende el trabajo presentado en [29] donde presentamos la representación de DWs en EDC y mostramos resultados para consultas de agregación considerando la función SUM. En esta nueva versión extendemos los algoritmos para computar consultas de agregación con el resto de las funciones de agregación, es decir, MAX, MIN, COUNT y AVG, sobre DWs compactados. Inicialmente, la EDC k^2-treap se usó para computar consultas *top-k* [14], tales como: "*obtener los cinco vendedores con más ventas*". Además, en esta versión hacemos un análisis más detallado de la nueva estructura de datos compacta CMHD [28], creada para trabajar sobre matrices multidimensionales y responder consultas con función SUM.

A. Data Warehouses

Los DWs integran datos de diferentes fuentes y almacenan datos históricos para análisis y soporte a las decisiones [15]. Los DW organizan los datos de acuerdo a dimensiones y hechos. Las dimensiones reflejan las perspectivas desde las cuales se visualizan los datos, se modelan como jerarquías de elementos (también llamados miembros), donde cada elemento pertenece a un nivel (o categoría) en una jerarquía (llamada esquema jerárquico). Los hechos corresponden a datos cuantitativos (también conocidos como medidas) asociados con las dimensiones. Los datos se pueden agregar (una operación llamada rollup), se pueden filtrar (a través de operaciones de *slide* y *dice*) y hacer referencia a ellos utilizando las dimensiones, un proceso denominado OLAP (procesamiento analítico en línea). Un cubo de datos es una estructura multidimensional para capturar y analizar los hechos de acuerdo a las dimensiones. En otras palabras, un cubo de datos generaliza la forma bidimensional de representar datos utilizando múltiples dimensiones. El Ejemplo 1 ilustra estos conceptos.

C. Vallejos, Universidad de Los Lagos, Chile, cristian.vallejos@ulagos.cl
M. Caniupán, Universidad del Bío-Bío, Chile, mcaniupan@ubiobio.cl
G. Gutiérrez, Universidad del Bío-Bío, Chile, ggutierr@ubiobio.cl

Ejemplo 1. Considere el DW de la Figura 1 con las dimensiones *Tiendas* y *Productos*, cuyos esquemas jerárquicos se muestran en la Figura 1(a) y Figura 1(c), respectivamente. La dimensión *Tiendas* tiene los niveles *Tienda*, *Ciudad*, *Región* y *All*. La última categoría es la categoría superior presente en todas las dimensiones. El nivel *Tienda* alcanza (*rollup*) al nivel *Ciudad* que se asocia a *Región*, que a su vez alcanza el nivel *All*. La Figura 1(b) muestra los elementos y las relaciones rollup para la dimensión *Tiendas*. Los elementos de la categoría *Tienda* son {ST_1, ST_2, ST_3, ST_4, ST_5, ST_6, ST_7, ST_8}. El nivel *Ciudad* tiene los elementos: CHI (Chillán), CON (Concepción), CAU (Cauquenes) y TAL (Talca). Los elementos de la categoría *Región* son VIII y VII. El nivel *All* posee un único elemento denominado all. Como ilustración, el elemento ST_1 se relaciona con CHI en la categoría *Ciudad*, que se relaciona con la región VIII en *Región*. Por otro lado, la dimensión *Productos* está formada por las categorías *Producto*, *Tipo*, *Marca* y *All*. La categoría *Producto* se relaciona con *Tipo*, que alcanza a *Marca*, que a su vez alcanza a *All*. Los elementos en la categoría *Producto* son: {P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8}. El nivel *Tipo* tiene los elementos: {T_1, T_2, T_3, T_4}. Finalmente, los elementos en *Marca* son {B_1, B_2}. El elemento P_1 alcanza el tipo T_1 que se relaciona con la marca B_1.

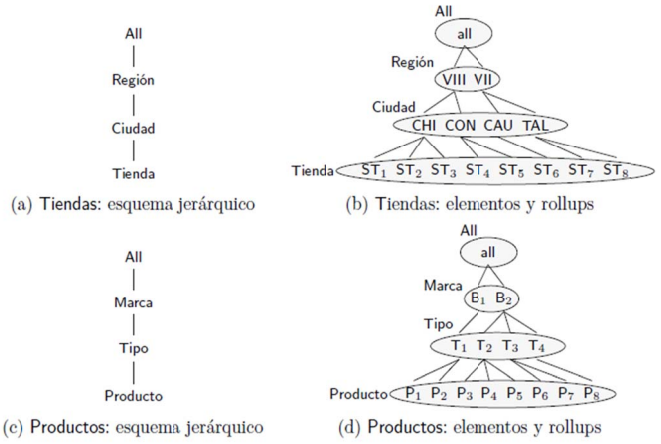

Figura 1. DW con dimensiones Tiendas y Productos.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
ST_1	0	15	7	5	0	0	0	0
ST_2	3	9	10	1	0	0	0	0
ST_3	0	4	0	8	0	0	0	0
ST_4	13	0	0	3	0	0	0	0
ST_5	0	0	4	0	0	0	8	0
ST_6	0	0	7	12	0	0	0	0
ST_7	4	1	0	0	0	0	3	0
ST_8	5	6	0	0	0	0	7	10

(a) Cubo de datos

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
ST_1		15	7	5				
ST_2	3	9	10	1				
ST_3		4		8				
ST_4	13			3				
ST_5			4				8	
ST_6			7	12				
ST_7	4	1					3	
ST_8	5	6					7	10

(b) Cubo de datos sin ceros

Figura 2. Cubos de Datos para el DW del Ejemplo 1.

La Figura 2(a) muestra un cubo de datos bidimensional que almacena cantidades de ventas de *Productos* por cada *Tienda*. Las consultas de agregación agrupan datos de acuerdo a algún atributo (nivel, en este caso) y aplican funciones de agregación sobre cada grupo. Las funciones de agregación son MAX, MIN, COUNT, SUM y AVG (promedio). Por ejemplo, para las dimensiones del DW en el Ejemplo 1 y el cubo de datos en la Figura 2(a), una consulta de agregación puede ser "*obtener la cantidad de ventas agrupadas por Región y Tipo de productos*". Para calcular esta consulta, es necesario conocer las relaciones rollup entre los niveles *Tienda* y *Ciudad* y entre *Ciudad* y *Región* en la dimensión *Tienda* y la relación rollup entre los niveles *Producto* y *Tipo* en la dimensión *Productos.*

B. Trabajo Relacionado

Los DWs pueden almacenar terabytes de datos, por lo que el procesamiento de consultas es un tema clave. Una técnica común para acelerar el procesamiento de consultas es usar resultados pre-calculados, llamadas tablas de resumen, para computar consultas y construir índices sobre las tablas de resumen. Hay trabajos que abordan el problema de cómo seleccionar las tablas de resumen a materializar y cómo mantenerlas actualizadas [16]. En [17] los autores presentan un algoritmo ávido para seleccionar las vistas de cubos de datos a materializar. En la misma dirección, en [18] los autores reportan algoritmos para seleccionar automáticamente tablas de resumen, junto con un método para mantenerlas actualizadas de manera eficiente.

En una dirección diferente, en [19] los autores proponen el operador *shrink* (encoger) para fusionar vistas de cubos similares, reemplazando estas vistas (o rebanadas) con una única vista representativa. La nueva rebanada resultante es una aproximación de las dos rebanadas procesadas. La idea detrás de este trabajo es generar cubos más pequeños para visualización a través de tablas dinámicas. La aplicación de este operador produce pérdida de precisión en la respuesta a consultas. Otros trabajos que reducen el tamaño del cubo calculando valores aproximados se presentan en [20] y [21].

En [22] los autores presentan una forma de condensar un cubo de datos para reducir su tamaño. El cubo condensado corresponde a un cubo totalmente pre-computado sin compresión y, por lo tanto, se puede consultar directamente. En [23], [24] y [25] se presentan algoritmos para calcular consultas agregadas sobre cubos de datos comprimidos, utilizando técnicas de compresión comunes tales como la eliminación de valores nulos, cambiando valores por valores comunes, entre otras. En [26] los autores presentan un algoritmo para realizar particiones en el cubo de datos de acuerdo con la estrategia dividir para conquistar. De esta manera, obtienen diferentes cubos pequeños para responder consultas rápidamente en la memoria principal.

En este artículo optamos por una dirección diferente, la que corresponde a utilizar una representación compacta de los DWs. Para compactar las dimensiones utilizamos bitmaps (secuencias de bits) y para los cubos de datos o *matrices de hechos* utilizamos la EDC k^2-treap presentada en [14] para calcular consultas *top-k*. Esta EDC se basa en dos EDC, el k^2-tree [6] y el treap [27]. Además, implementamos algoritmos para calcular consultas de agregación sobre cubos de datos compactos. Las funciones de agregación implementadas son SUM [29], MAX, MIN, COUNT y AVG.

Una estructura de datos similar a la nuestra se propone en [28] denominada CMHD (Compact representation of

Multidimensional data on Hierarchical Domains). Esta EDC divide recursivamente la matriz de hechos de acuerdo a los agrupamientos definidos en las jerarquías de cada dimensión. En general las submatrices son de diferente tamaño en cada nivel de partición. Esta división recursiva constituye un árbol ordenado, cuya raíz representa la matriz completa; a su vez el segundo nivel del árbol representa cada una de las submatrices obtenidas usando los agrupamientos establecidos en el primer nivel de las jerarquías de cada dimensión. Cada una de estas submatrices es considerada como un nodo de la raíz. El proceso de división se repite por cada submatriz y teniendo en cuenta los siguientes niveles de las jerarquías de las dimensiones. Si para una submatriz no existen valores, dicha submatriz no se vuelve a dividir. Paralelamente a la partición, en cada nodo se almacena el valor pre-calculado de la función de agregación SUM de la submatriz. El árbol definido previamente para CMHD es conceptual, y es representado de manera compacta usando diferentes estructuras de datos, tanto para las dimensiones como para la matriz. Para cada dimensión (usando LOUDS) se utilizan dos bits por cada nodo del árbol permitiendo navegar de manera eficiente a través de los diferentes niveles de la dimensión mediante las operaciones de *Rank* y *Select* (ver Sección II). La topología del árbol y límites de las submatrices se representan mediante dos bitmaps que permiten navegar el árbol. Finalmente, tanto los valores de la matriz original como los valores pre-calculados se generan por medio de un recorrido por nivel del árbol representado con DACs [11].

Con CMHD es posible responder consultas de agregación con la función SUM sobre cualquiera de las submatrices generadas por las jerarquías de las dimensiones. Mediante una serie de experimentos se muestra que CMHD con un poco más de almacenamiento procesa en menor tiempo las consultas de agregación sobre submatrices de tamaño irregular (escenario realista) que la opción que considera una partición regular de la matriz. Los experimentos no consideran el desempeño de CMHD con respecto a Data Warehouses almacenados en memoria secundaria. Tampoco se mide el desempeño considerando funciones de agregación diferentes a SUM, cuya inclusión puede incrementar el almacenamiento requerido por la estructura y el tiempo de las consultas.

II. Representación y Procesamiento de DWs en Estructuras de Datos Compactas

En esta sección se describe cómo representar tanto las dimensiones de DWs como las matrices de hechos (cubos de datos) mediante bitmaps y la estructura compacta k^2-treap, respectivamente. También se presentan los algoritmos para computar consultas de agregación sobre matrices de hechos y dimensiones en las representaciones compactas.

A. Representación de Dimensiones

Para almacenar los elementos de los niveles de las dimensiones se utilizan arreglos y bitmaps para almacenar las relaciones entre los elementos de los niveles de las dimensiones. Se guardan todos los niveles de una dimensión, excepto el nivel inferior, ya que es parte del cubo de datos, y el nivel superior *All*, que tiene un único elemento. La Figura 3 muestra los arreglos para las dimensiones *Tiendas* y *Productos* del DW en el Ejemplo 1 con los esquemas jerárquicos en la Figura 1(a) y Figura 1(c), respectivamente.

Las relaciones rollup almacenan la asociación entre los elementos de dos niveles de una dimensión. Para representar esta asociación se utilizan tablas bitmap. Sean *a* y *b* niveles de una dimensión *D*, tal que, *a* rollup a *b* en el esquema jerárquico de *D*, creamos una tabla bitmap que indica cuantos elementos de *a* se asocian con el primer elemento en el nivel *b*, y así sucesivamente. Cada "1" en el bitmap indica el cambio de elemento en el nivel *b*. No se consideran bitmaps para los niveles que hacen rollup al nivel superior *All*.

Ciudad
CHI
CON
CAU
TAL

Región
VIII
VII

Tipo
T_1
T_2
T_3
T_4

Marca
B_1
B_2

Figura 3. Tablas de una columna para las dimensiones del Ejemplo 1.

Ejemplo 2. La Figura 4(a) muestra los bitmaps para las relaciones rollup entre los niveles *Tienda* y *Ciudad* (R_1) y entre *Ciudad* y *Región* (R_2). La tabla R_1 indica que los elementos ST_1, ST_2 y ST_3 se relacionan con el primer elemento en el nivel *Ciudad* (CHI). Dado que en la entrada para el elemento ST_4 el bit cambia de "0" a "1", esto indica que ST_4 y ST_5 se asocian con el segundo elemento en el nivel *Ciudad* (CON). Luego, ocurre otro cambio en la entrada del elemento ST_6 que indica que ST_6 rollup al tercer elemento en *Ciudad* (CAU). Finalmente, en la entrada para el elemento ST_7 hay otro cambio por lo que los elementos ST_7 y ST_8 se relacionan con el cuarto elemento en *Ciudad* (TAL). El mismo análisis se puede hacer con los bitmaps para la dimensión *Productos* Figura 4(b).

R_1		
Posición	Tienda	Bitmap
0	ST_1	1
1	ST_2	0
2	ST_3	0
3	ST_4	1
4	ST_5	0
5	ST_6	1
6	ST_7	1
7	ST_8	0

R_2		
Posición	Ciudad	Bitmap
0	CHI	1
1	CON	0
2	CAU	1
3	TAL	0

(a) Bitmaps para dimensión Tiendas

R_3		
Posición	Producto	Bitmap
0	P_1	1
1	P_2	0
2	P_3	1
3	P_4	0
4	P_5	0
5	P_6	1
6	P_7	0
7	P_8	1

R_4		
Posición	Tipo	Bitmap
0	T_1	1
1	T_2	1
2	T_3	0
3	T_4	0

(b) Bitmaps para dimensión Productos

Figura 4. Tablas bitmaps para las relaciones rollup de las dimensiones del Ejemplo 1.

B. Representación de Matrices de Hechos

Se consideran cubos de datos de dos dimensiones los que se representan en la EDC k^2-treap [14]. Sean A y B niveles inferiores de dos dimensiones en un cubo C, con n y m elementos, respectivamente. El cubo de datos C conforma una matriz de tamaño $n \times m$. Como un cubo de datos puede contener ceros, no consideramos los ceros al construir el k^2-treap. La Figura 2(b) muestra la representación del cubo de datos de la Figura 2(a) sin ceros.

La construcción del k^2-treap se realiza de la siguiente manera: Dado una matriz de hechos $C[n \times m]$ donde cada celda contiene un valor o puede estar vacía, C se divide en k^2 submatrices y luego se obtiene un árbol de la siguiente manera:

(i) la raíz del árbol almacena las coordenadas de la celda con el valor máximo en la matriz junto con el valor.

(ii) El valor se borra de C. Si muchas celdas comparten el mismo valor, se elige una de ellas.

(iii) Luego, la matriz se descompone en k^2 submatrices de igual tamaño, y k^2 nodos secundarios se agregan a la raíz del árbol, cada uno de ellos representa una de las submatrices. Este proceso se repite recursivamente para cada nodo hijo, hasta que la matriz esté vacía.

La Figura 5 muestra el árbol para el cubo de datos en la Figura 2(b) considerando $k=2$, los nodos vacíos se representan con el símbolo "-" en el árbol. Con el fin de ahorrar almacenamiento, este árbol se codifica previa modificación de las coordenadas originales por las coordenadas de cada submatriz (partiendo cada una en la posición (0,0)) y los valores máximos de cada nodo se codifican diferencialmente con respecto al valor máximo de su nodo padre. La Figura 6 muestra el nuevo árbol codificado para el árbol de la Figura 5. Finalmente, el nuevo árbol se almacena en un k^2-tree [6] y se crean arreglos para almacenar las coordenadas de cada nivel y los valores asociados a ellas.

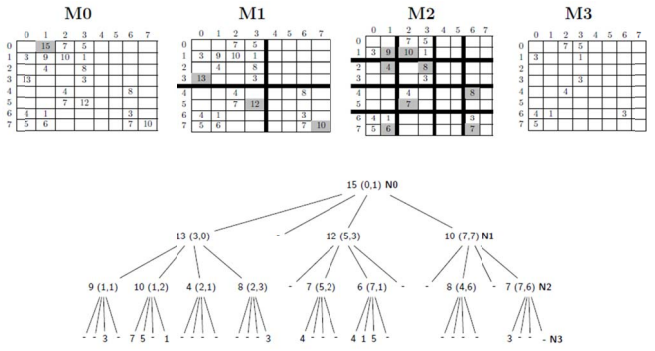

Figura 5. Proceso de construcción del árbol para el cubo de datos de la Figura 2(b).

Básicamente, un k^2-tree almacena 1s y 0s, un "1" representa la presencia de un valor, y un "0" lo contrario. Notar que la topología del k^2-tree se representa mediante un bitmap T (no se usan punteros), el cual es suficiente para navegar sobre el árbol como se explica a continuación. La exploración se realiza utilizando las operaciones *Rank* y *Select*. Sea $B[1, n]$ una secuencia de bits. La operación $Rank_1(B, i)$ devuelve el número de apariciones de 1s en $B[1, i]$. La operación $Select_1(B, j)$ devuelve la posición de la j-*ésima* ocurrencia de 1 en B. Ambas operaciones se pueden resolver en tiempo constante. Para encontrar un hijo en un k^2-tree [6] se aplica la función $child_i(x)=rank_1(T, x)\times k^2+i$, donde i es el i-ésimo hijo del nodo x en T (T parte en la posición 0). Las otras matrices utilizadas para representar la estructura de datos compacta k^2-treap son matrices para almacenar las *coordenadas* del árbol modificado, una matriz llamada *values* para almacenar los valores del árbol, y una matriz llamada *first* que indica la posición inicial de un nuevo nivel en el arreglo *values*. La Figura 7 muestra la representación del k^2-treap para el cubo de datos en la Figura 2(b).

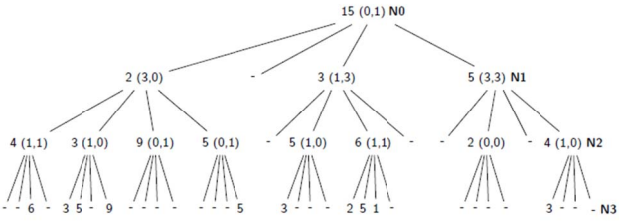

Figura 6. Re-codificación del árbol de la Figura 5.

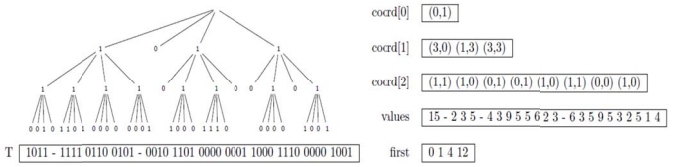

Figura 7. Representación del k^2-treap para el cubo de datos en la Figura 2(b).

C. Navegación en el Árbol

Supongamos que queremos obtener el valor asociado a la coordenada $C(x,y)$ en el k^2-treap de la Figura 7. La navegación comienza por la raíz del k^2-tree, que corresponde al nodo con coordenadas (x_0,y_0) y el valor $v=values[0]$. Si el valor deseado corresponde al valor en la coordenada $C=(x_0,y_0)$, se devuelve el valor v; de lo contrario, necesitamos encontrar el cuadrante en el que se ubica la celda en el k^2-tree. Sea p una posición de un nodo en el bitmap T, si $T[p]=0$ la correspondiente submatriz está vacía y el algoritmo retorna. De lo contrario, necesitamos encontrar las coordenadas y el valor del nodo. Entonces, calculamos $r=Rank_1(T,p)$ que devuelve el número de ocurrencias de "1s" en T hasta la posición p, el valor del nodo es $v_1=values[r]$ y sus coordenadas son $coord[l][r-first[l]]$, donde l es el nivel actual en el árbol. El valor v_1 debe ser reconstruido ya que en la matriz *values* se almacenan diferencialmente con respecto al valor máximo de sus nodos padres. Por lo tanto, $v_1=v-values[r]$. Supongamos que necesitamos encontrar el valor en la coordenada (1,3) del árbol re-codificado, entonces, $p=2$ (T parte en la posición 0), $r=Rank_1(T,2)=2$, el valor del nodo es $values[2]=3$, el cual debe ser reconstruido de acuerdo al valor de su nodo padre que es la raíz, por lo que $v=15$, luego $v_1=v-values[r]=15-3=12$. La búsqueda se realiza de manera recursiva [14].

D. Cómputo de Consultas de Agregación

Las consultas más comunes en DW son consultas de agregación agrupadas por diferentes niveles de las dimensiones. Por ejemplo, "*obtener la suma de ventas totales agrupadas por ciudad y tipo de productos*". En este trabajo implementamos todas las funciones de agregación usadas en DWs y para ello utilizamos la función *top-k* que está implementada en la librería C++ Succinct Data Structure[1], que contiene la EDC k^2-treap con toda su funcionalidad. El proceso para obtener los *top-k* valores sobre una matriz $Q[x_1,x_2]\times[y_1,y_2]$ parte en la raíz del árbol con la implementación de una cola de prioridad (Max Priority) que almacena valores máximos del k^2-treap. El primer elemento de la cola, es el valor de la raíz del árbol, luego, iterativamente, el proceso extrae los primeros elementos de la cola, si las coordenadas de los valores caen dentro del rango de la consulta Q, se genera una nueva respuesta. Todos los hijos del nodo extraído que caen en el rango de Q se insertan en la cola. El proceso continua hasta obtener los k valores máximos. En DWs el rango de la consulta está dado por los niveles de las dimensiones y para obtenerlos necesitamos utilizar los bitmaps descritos en la Sección II.A.

Para calcular la suma de valores en un rango dado, en vez de recuperar los *top-k* valores, todos los valores del rango de la consulta (dados por los niveles en la consulta) se recolectan y suman. El usar la función *top-k* genera un costo mayor, ya que su implementación implica mantener los valores parcialmente ordenados en una cola de prioridad, lo cual no es necesario para sumar, o contar, por ejemplo. Sin embargo, como demostramos en la Sección III, a pesar de esto, nuestros algoritmos requieren mucho menos tiempo de ejecución que el tiempo requerido por un Sistema de Gestión de Base de Datos (SGBD) que aloje al DW.

Ejemplo 3. Considere el cubo de datos en la Figura 8, donde los elementos del nivel *Tienda* se ubican en las filas del cubo y los elementos del nivel *Producto* en las columnas del cubo, junto con los esquemas jerárquicos de la Figura 1(a) y Figura 1(c). Para calcular la consulta "*obtener la cantidad de ventas por Tiendas de la ciudad TAL y Marca B_2*", necesitamos sumar los valores en el rango {(6,2), (7,7)} en el cubo. La respuesta a la consulta es *20*.

Para obtener el rango de una consulta de agregación, necesitamos identificar los niveles involucrados en la consulta, y luego, bajar a través de las jerarquías correspondientes para obtener los niveles inferiores en el cubo (*drill-down* en terminología DW [15]). En el caso del Ejemplo 3 sabemos que TAL es el último elemento del nivel *Ciudad* que se almacena en la tabla *Ciudad* en la Figura 3. Entonces, necesitamos saber cuáles son las tiendas que pertenecen a TAL. Podemos obtener esta información buscando el cuarto 1 en el bitmap R_1 de la Figura 4(a) a través de la operación $Select_1(R_1,4)=$ 6, que entrega ST_7 como la primera tienda que rollup a TAL, dado que TAL es el último elemento en el nivel *Ciudad* el resto de los elementos en R_1 rollup a TAL. Por lo tanto, podemos concluir que el rango de la consulta considera desde las filas 6 hasta la 7 del cubo. Luego, para encontrar las columnas que representan los productos que se agrupan en la marca B_2 buscamos de la misma forma en R_4 de la Figura 4(b), ejecutando $Select_1(R_4,2)=1$. Por lo tanto, el primer tipo de productos que alcanza B_2 es T_2 y el último tipo es T_4. Ahora, necesitamos saber qué productos se asocian (rollup) a los tipos T_2, T_3 y T_4, lo que se obtiene realizando operaciones *Select* sobre R_3 de la Figura 4(b), obteniendo los productos P_3 a P_8, lo que entrega el rango {(6,2), (7, 7)} para la consulta.

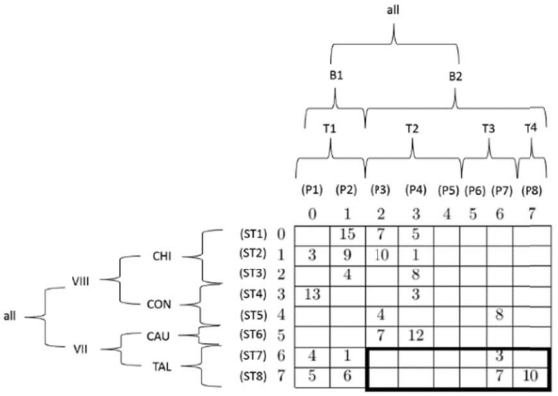

Figura 8. Rango para la consulta de agregación del Ejemplo 3.

La Figura 9 muestra el Algoritmo 1 que es el algoritmo principal, que recibe una consulta Q y la representación compacta del DW. Primero el algoritmo obtiene los niveles de la consulta (Líneas 1-2), y luego llama al Algoritmo 2 de la Figura 10, que obtiene el rango de la consulta (Línea 3). Luego obtiene la respuesta a la consulta. (Línea 4). Si la consulta de agregación corresponde a una consulta con función SUM, se llama al Algoritmo 3 de la Figura 11, que usa la función *TopK* que obtiene todos los valores dentro del rango dado por el rango de la consulta y los suma. Los algoritmos para computar consultas de agregación con el resto de las funciones de agregación son similares al Algoritmo 3.

Algoritmo 1: CÓMPUTO DE CONSULTAS DE AGREGACIÓN SOBRE DWS COMPACTOS

Input: Una consulta Q, un conjunto de bitmaps B, un conjunto de tablas de una columna T, un k^2-treap K
Output: $Res(Q)$

```
1 catDim1 ← ObtenerCategoríaDim1(Q);
2 catDim2 ← ObtenerCategoríaDim2(Q);
3 RQ ← ObtenerRangoConsulta(catDim1, catDim2, B, T);
4 Res(Q) ← ComputarAgregación(RQ, K);
5 return Res(Q)
```

Figura 9. Algoritmo general para computar consultas de agregación.

Algoritmo 2: OBTENER RANGO CONSULTA

Input: Niveles $catDim1$ y $catDim2$, un conjunto de bitmaps B, un conjunto de tablas de una columna T
Output: RQ

```
1 x1 ← ObtenerFilaInicial(catDim1, B, T);
2 x2 ← ObtenerFilaFinal(catDim1, B, T);
3 y1 ← ObtenerColumnaInicial(catDim2, B, T);
4 y2 ← ObtenerColumnaFinal(catDim2, B, T);
5 RQ ← {(x1, y1), (x2, y2)};
6 return RQ
```

Figura 10. Algoritmo para obtener el rango de una consulta.

[1] https://github.com/simongog/sdsl-lite

Algoritmo 3: COMPUTAR AGREGACIÓN (SUM)

Input: Rango RQ, un k^2-treap K
Output: $Res(Q)$

```
1 Res(Q) ← ∅;
2 Valores ← TopK(K, RQ);
3 foreach i ∈ Valores do
4 |  Res(Q) ← Res(Q) + i.valor;
5 return Res(Q)
```

Figura 11. Algoritmo computar consultas de agregación con SUM.

III. EVALUACIÓN EXPERIMENTAL

En esta sección presentamos resultados experimentales de nuestros algoritmos en términos de ahorro de espacio y tiempo de ejecución de consultas. Consideramos el DW presentado en el Ejemplo 1 con las dimensiones mostradas en la Figura 1 y el cubo de datos de la Figura 2(a). El DW se implementó en el SGBD PostgreSQL[2] versión 9.3.12 utilizando un esquema de copo de nieve [15]. Se consideraron diferentes conjuntos de datos sintéticos. Los experimentos se realizaron en una computadora con procesador Intel Core i7 con 2.40 GHz y 12 GB de memoria RAM; con Debian Linux System versión 8.4.0 amd64. Todas las estructuras de datos se implementaron en C ++ y se compilaron con gcc 6.3.

A. Resultados Sobre Espacio en Memoria Principal

La Tabla I muestra los cubos de datos generados con distribución normal, la cual fue elegida por la alta concentración de celdas con ceros que genera, lo que es común en la práctica de DWs. De acuerdo a la Tabla I la representación compacta del cubo de datos ahorra, en la mayoría de los casos, un 80 % de espacio con respecto a PostgreSQL.

TABLA I. ESPACIO OCUPADO POR DIFERENTES CUBOS DE DATOS EN SU FORMA COMPACTA Y EN POSTGRESQL.

#ELEMENTOS	# CEROS	SGBD (KB)	K^2-TREAP (KB)
2.500	99	6.954	5,4
10.000	426	7.130	20,3
250.000	9.563	17.000	500
1.000.000	38.348	49.000	1.995,4

B. Resultados Sobre Tiempo de Ejecución de Consultas

La Tabla II muestra los tiempos de ejecución de todas las consultas de agregación con función MAX ejecutadas sobre cubos generados con distribución normal para el DW en el Ejemplo 1. Como podemos observar nuestros algoritmos obtienen mejores tiempos de ejecución que PostgreSQL. Esto se debe básicamente a que la estructura compacta k^2-treap fue diseñada para obtener los valores máximos. Las Tabla III, IV y V muestran, respectivamente, los tiempos de ejecución para las consultas de agregación con función AVG, COUNT y MIN. En la mayoría de los casos, obtenemos mejores tiempos de ejecución que PostgreSQL, exceptuando algunos casos. Esto se debe principalmente a que al usar la estructura compacta k^2-treap se utiliza una cola de prioridad que no es necesaria para computar las funciones de agregación AVG, COUNT y MIN. Sin embargo, cuando los tiempos son mejores sobre cubos representados en un k^2-treap, las diferencias son considerables. PostgreSQL muestra mejores tiempos de ejecución sobre cubos de datos con más de 10.000 elementos. Esto se puede explicar por el uso de índices sobre los datos y los métodos de optimización de consultas implementados en PostgreSQL.

TABLA II. TIEMPOS DE EJECUCIÓN EN MILISEGUNDOS PARA CONSULTAS DE AGREGACIÓN CON FUNCIÓN MAX

NIVEL DE LA CONSULTA	#ELEMENTOS EN EL CUBO					
	10.000		250.000		1.000.000	
	K^2T	DBMS	K^2T	DBMS	K^2T	DBMS
PRODUCTO – TIENDA	12	335	14	9.100	16	36.686
PRODUCTO – CIUDAD	12	31	41	227	70	696
PRODUCTO – REGIÓN	10	11	36	2.659	67	12.456
PRODUCTO – ALL	8	22	34	172	66	534
TIPO – TIENDA	8	21	18	314	20	1.120
TIPO – CIUDAD	3	12	4	213	4	545
TIPO – REGIÓN	1	11	2	1.923	1	8.801
TIPO – ALL	0	11	1	131	0	514
MARCA – TIENDA	5	12	11	364	14	1.079
MARCA – CIUDAD	1	12	2	142	1	626
MARCA – REGIÓN	0	22	0	1.149	0	5.771
MARCA – ALL	0	12	0	132	0	585
ALL – TIENDA	3	12	7	152	10	525
ALL – CIUDAD	1	11	1	121	1	504
ALL – REGIÓN	0	24	0	131	0	576
ALL – ALL	0	12	0	102	0	444

TABLA III. TIEMPOS DE EJECUCIÓN EN MILISEGUNDOS PARA CONSULTAS DE AGREGACIÓN CON FUNCIÓN AVG

NIVEL DE LA CONSULTA	#ELEMENTOS EN EL CUBO					
	10.000		250.000		1.000.000	
	K^2T	DBMS	K^2T	DBMS	K^2T	DBMS
PRODUCTO – TIENDA	12	529	15	13.806	17	55.546
PRODUCTO – CIUDAD	19	41	86	257	167	859
PRODUCTO – REGIÓN	17	21	83	2.740	167	12.585
PRODUCTO – ALL	16	11	84	152	171	565
TIPO – TIENDA	17	31	79	394	171	1.160
TIPO – CIUDAD	32	11	748	224	3.053	555
TIPO – REGIÓN	31	11	752	2.145	3.054	9.938
TIPO – ALL	29	24	746	132	3.211	626
MARCA – TIENDA	14	32	78	294	172	1.110
MARCA – CIUDAD	28	23	736	182	3.117	575
MARCA – REGIÓN	27	12	724	1.179	3.129	5.800
MARCA – ALL	25	11	748	213	3.259	525
ALL – TIENDA	14	11	79	152	175	545
ALL – CIUDAD	29	22	739	131	3.216	504
ALL – REGIÓN	28	11	730	142	3.224	534
ALL – ALL	26	23	758	111	3.353	405

[2] https://www.postgresql.org/

TABLA IV. TIEMPOS DE EJECUCIÓN EN MILISEGUNDOS PARA CONSULTAS DE AGREGACIÓN CON FUNCIÓN COUNT

NIVEL DE LA CONSULTA	#ELEMENTOS EN EL CUBO					
	10.000		250.000		1.000.000	
	k^2T	DBMS	k^2T	DBMS	k^2T	DBMS
PRODUCTO – TIENDA	6	377	6	8.194	8	33.015
PRODUCTO – CIUDAD	10	32	39	205	71	774
PRODUCTO – REGIÓN	9	24	37	2.760	69	12.556
PRODUCTO – ALL	8	23	36	142	68	535
TIPO – TIENDA	8	21	32	314	74	1.120
TIPO – CIUDAD	15	23	287	162	1.090	625
TIPO – REGIÓN	14	11	279	2.145	1.075	10.281
TIPO – ALL	13	11	274	131	1.062	606
MARCA – TIENDA	7	12	34	283	72	1.141
MARCA – CIUDAD	13	12	389	142	1.069	545
MARCA – REGIÓN	12	12	380	1.189	1.053	5.781
MARCA – ALL	11	11	375	132	1.042	514
ALL – TIENDA	6	12	42	132	70	514
ALL – CIUDAD	13	11	332	131	1.061	494
ALL – REGIÓN	12	12	265	142	1.045	535
ALL – ALL	11	11	262	101	1.032	404

TABLA V. TIEMPOS DE EJECUCIÓN EN MILISEGUNDOS PARA CONSULTAS DE AGREGACIÓN CON FUNCIÓN MIN

NIVEL DE LA CONSULTA	#ELEMENTOS EN EL CUBO					
	10.000		250.000		1.000.000	
	k^2T	DBMS	k^2T	DBMS	k^2T	DBMS
PRODUCTO – TIENDA	18	348	20	9.080	24	36.641
PRODUCTO – CIUDAD	26	22	121	214	236	684
PRODUCTO – REGIÓN	24	12	117	2.730	235	12.527
PRODUCTO – ALL	22	11	119	141	242	524
TIPO – TIENDA	24	21	111	314	242	1.213
TIPO – CIUDAD	46	11	1.037	222	4.259	555
TIPO – REGIÓN	44	11	1.019	2.186	4.291	9.816
TIPO – ALL	41	23	1.037	132	4.439	524
MARCA – TIENDA	20	22	110	273	244	1.111
MARCA – CIUDAD	40	12	1.033	172	4.321	554
MARCA – REGIÓN	39	12	1.015	1.159	4.320	5.811
MARCA – ALL	35	11	1.039	132	4.486	504
ALL – TIENDA	20	11	110	132	248	555
ALL – CIUDAD	20	11	1.035	131	4.447	494
ALL – REGIÓN	41	24	1.019	192	4.435	535
ALL – ALL	40	12	1.052	101	4.588	404

IV. CONCLUSIONES Y TRABAJO FUTURO

En este artículo presentamos la representación de DWs en EDC, en especial, destacamos el uso de la EDC k^2-treap para representar cubos de datos bidimensionales. Recientemente, se ha publicado un trabajo donde se muestra que la EDC k^2-treap se puede modificar para considerar más dimensiones [28]. Los experimentos que presentamos sobre datos sintéticos muestran que, al utilizar EDC, podemos ahorrar espacio de almacenamiento en la memoria principal y realizar consultas de manera más eficiente (en la mayoría de los casos), que utilizando un SGBD tradicional. En este artículo consideramos consultas de agregación con todas las funciones de agregación usadas en la práctica, exceptuando la función SUM que fue presentada en [29]. Como trabajo futuro planeamos construir algoritmos ad hoc sobre la EDC k^2-treap para implementar cada una de las funciones de agregación. Además, de extender las capacidades de las EDC para permitir dimensiones jerárquicas no lineales.

AGRADECIMIENTOS

Mónica Caniupán y Gilberto Gutiérrez fueron financiados por el proyecto DIUBB [171319 4/R], Proyecto exploratorio (Ingeniería 2030) [1638] y el grupo de investigación ALBA [GI 160119/EF]. Agradecemos a Nieves Brisaboa y Carlos Aburto por sus comentarios sobre este trabajo.

REFERENCIAS

[1] H. Garcia-Molina and K. Salem, *Main memory database systems: An overview*, IEEE Trans. on Knowl. & Data Eng., vol. 4, no. 6, pp. 509–516, 1992.

[2] K. A. Ross and K. A. Zaman, *Serving datacube tuples from main memory*, Proc. of the 12th International Conference on Scientific and Statistical Database Management, pp. 182–195, 2000.

[3] H. Plattner and A. Zeier, *In-Memory Data Management: An Inflection Point for Enterprise Applications*, 1st ed. Springer Publishing Company, Incorporated, 2011.

[4] R. Raman, V. Raman, and S. Rao, *Succint dynamic data structures*, Algorithms and Data Structures, LNCS, vol. 2125, pp. 426–437, 2001.

[5] F. Claude and G. Navarro, *A fast and compact web graph representation*, Proc. of the 14th International Symposium on String Processing and Information Retrieval, LNCS, vol. 4726, pp. 105–116, 2007.

[6] N. Brisaboa, S. Ladra, and G. Navarro, *K2-trees for compact web graph representation*, Proc. of the 16th International Symposium on String Processing and Information Retrieval, pp. 18–30, 2009.

[7] N. Brisaboa, S. Ladra, and G. Navarro, *Compact representation of web graphs with extended functionality*, Inf. Syst., vol. 39, pp. 152–174, 2014.

[8] F. Claude and G. Navarro, *Practical rank/select queries over arbitrary sequences*, Proc. of the 15th International Symposium on String Processing and Information Retrieval, pp. 176–187, 2008.

[9] G. Navarro, *Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences*, ACM Computing Surveys, vol. 46, no. 4, pp. 52:1–52:47, 2014.

[10] G. Navarro and K. Sadakane, *Fully functional static and dynamic succinct trees*, ACM Transactions on Algorithms, vol. 10, no. 3, pp. 16:1–16:39, 2014.

[11] N. Brisaboa, S. Ladra, and G. Navarro, *Dacs: Bringing direct access to variable-length codes*, Information Processing and Management, vol. 49, no. 1, pp. 392–404, 2013.

[12] N. Brisaboa, M. Luaces, G. Navarro, and D. Seco, *Space efficient representations of rectangle datasets supporting orthogonal range querying*, Information Systems, vol. 38, no. 5, pp. 635–655, 2013.

[13] G. De Bernardo, S. Álvarez-García, N. Brisaboa, G. Navarro, and O. Pedreira, *Compact querieable representations of raster data*, Proc. of the 20th International Symposium on String Processing and Information Retrieval, pp. 96–108, 2013.

[14] N. Brisaboa, G. De Bernardo, R. Konow, and G. Navarro, *K2-treaps: Range top-k queries in compact space*, Proc. of the 21st International Symposium on String Processing and Information Retrieval, LNCS, vol. 8799, pp. 215–226, 2014.

[15] S. Chaudhuri and U. Dayal, *An overview of data warehousing and olap technology*, SIGMOD Rec., vol. 26, no. 1, pp. 65–74, 1997.

[16] I. Mumick, D. Quass, and B. Mumick, *Maintenance of data cubes and summary tables in a warehouse*, SIGMOD Rec., vol. 26, no. 2, pp. 100–111, 1997.

[17] V. Harinarayan, A. Rajaraman, and J. Ullman, *Implementing data cubes efficiently*, SIGMOD Rec., vol. 25, no. 2, pp. 205–216, 1996.

[18] H. Gupta, V. Harinarayan, A. Rajaraman, and J. Ullman, *Index selection for olap*, Proc. of the Thirteenth International Conference on Data Engineering, pp. 208–219, 1997.
[19] M. Golfarelli and S. Rizzi, *Honey, i shrunk the cube*, Proc. of the 17th East European Conference on Advances in Databases and Information Systems, vol. 8133, pp. 176–189, 2013.
[20] P. Furtado and H. Madeira, *Data cube compression with quanticubes*, Proc. of the Data Warehousing and Knowledge Discovery, pp. 162–167, 2000.
[21] F. Yu and W. Shan, *Compressed data cube for approximate olap query processing*, J. Comput. Sci. Technol., vol. 17, no. 5, pp. 625–635, 2002.
[22] W. Wang, J. Feng, and H. Lu, *Condensed cube: An efficient approach to reducing data cube size*, Proc. of the 18th International Conference on Data Engineering, pp. 155–165, 2002.
[23] J. Li, D. Rotem, and J. Srivastava, *Aggregation algorithms for very large compressed data warehouses*, Proc. of the 25th International Conference on Very Large Data Bases, pp. 651–662, 1999.
[24] J. Li and J. Srivastava, *Efficient aggregation algorithms for compressed data warehouses*, IEEE Trans. Knowl. Data Eng., vol. 14, no. 3, pp. 515–529, 2002.
[25] W. Wu, H. Gao, and J. Li, *New algorithm for computing cube on very large compressed data sets*, IEEE Trans. Knowl. Data Eng., vol. 18, no. 12, pp. 1667–1680, 2006.
[26] K. Ross and D. Srivastava, *Fast computation of sparse datacubes*, Proc. of the 23rd International Conference on Very Large Data Bases, pp. 116–125, 1997.
[27] R. Seidel and C. Aragon, *Randomized search trees*, Algorithmica, vol. 16, no. 4/5, pp. 464–497, 1996.
[28] N. Brisaboa, A. Cerdeira-Pena, N. López-López, G. Navarro, M. Penabad, and F. Silva-Coira, *Efficient representation of multidimensional data over hierarchical domains*, Proc. of the 23rd International Symposium on String Processing and Information Retrieval, pp. 191–203, 2016.
[29] C.Vallejos, M. Caniupán, and G. Gutiérrez, *K^2-treaps to represent and query data warehouses into main memory*, Proc. of the 36th International Conference of the Chilean Computer Society (SCCC 2017), Arica, Chile, 16-20 October, 2017.

Cristian Vallejos se tituló de Ingeniero Civil en Informática en la Universidad del Bío Bío, Concepción, Chile el año 2014. Actualmente es candidato a obtener el grado de Magíster en Ciencias de la Computación en la Universidad del Bío-Bío y se desempeña como Académico en la Universidad de Los Lagos, Puerto Montt, Chile. Sus principales intereses en investigación incluyen Programación de algoritmos y estructuras de datos compactas.

Mónica Caniupán recibió el grado de PhD in Computer Science de Carleton University, Canadá el año 2007. Actualmente, es Académica Asociada del Departamento de Sistemas de Información de la Universidad del Bío-Bío, Concepción, Chile. Sus principales intereses en investigación incluyen teoría de base de datos, integridad de la base de datos, Data Warehouses, limpieza de datos, estructuras de datos compactas, representación del conocimiento y programación lógica.

Gilberto Gutiérrez obtuvo su M.Sc. y Ph.D. en Ciencias de la Computación en la Universidad de Chile, Chile, en los años 1999 y 2007, respectivamente. Actualmente es Académico Asociado del Departamento de Ciencias de la Computación y Tecnologías de la Información de la Universidad del Bío-Bío, Chillán, Chile. Sus áreas de interés comprenden tópicos de bases de datos espaciales, espacio-temporales, algoritmos y estructuras de datos.