REVISTA DE LA FACULTAD DE INGENIERIA
www.revistaingenieria.uda.cl

32 (2016) 69-79

ALGORITMOS POLINOMIALES PARA OBTENER LA REPARACIÓN COMPATIBLE EN DATA WAREHOUSES INCONSISTENTES

Juan Ramírez, Raúl Arredondo

Departamento de Ciencias Exactas. Universidad de Los Lagos. Osorno. Chile. juan.ramirez@ulagos.cl ; Departamento de Gobierno y Empresa. Universidad de Los Lagos. Puerto Montt, Chile. raul.arredondo@ulagos.cl

RESUMEN

Una dimensión en un data warehouse es un concepto abstracto que agrupa datos que comparten un significado semántico común. Las dimensiones se modelan mediante un esquema jerárquico de categorías. Una dimensión es llamada estricta si cada elemento de cada categoría tiene exactamente un ancestro en cada categoría superior, y homogénea si cada elemento de una categoría tiene por lo menos un ancestro en cada categoría superior. Si una dimensión es estricta y homogénea se pueden utilizar consultas pre-computadas en los niveles inferiores para obtener respuestas en los niveles superiores. Sin embargo, cuando las dimensiones no son estrictas/homogéneas debemos conocer sus restricciones para mantener la capacidad de obtener un resultado correcto. En el mundo real, las dimensiones pueden no satisfacer estas restricciones, y en estos casos, es importante encontrar maneras de corregir estas dimensiones o encontrar formas de obtener respuestas correctas a las preguntas planteadas en las dimensiones inconsistentes. Una reparación minimal es una nueva dimensión que satisface las restricciones estrictas y homogéneas, y que se obtiene a partir de la dimensión original a través de un número mínimo de cambios, el cual tiene un costo computacional NP-duro. Para mejorar esto, se define la dimensión extendida y propone la reparación compatible que procesa la dimensión inconsistente para obtener una nueva dimensión consistente, manteniendo la esencia de la dimensión original y puede ser calculada de manera eficiente. En este último punto se centra la experimentación, elaborando y evaluando un programa que genere esta reparación compatible y obtenga resultados en tiempo polinomial.Palabras clave: Data warehouse, inconsistencia, restricción estricta, restricción homogénea, dimensión, experimentos

ABSTRACT

A dimension in a data warehouse is an abstract concept that groups data that share a common semantic meaning. The dimensions are modeled using a hierarchical scheme of categories. A dimension is called strict if every element of each category has exactly one ancestor in each parent category, and covering if each element of a category has an ancestor in each parent category. If a dimension is strict and covering we can use pre-computed results at lower levels to answer queries at higher levels. However, when the dimensions are not strict/covering must know their restrictions to maintain the ability to obtain a correct summarizability, in the real world can not meet these restrictions, and in these cases it is important to find ways of fix these dimensions or find ways to get correct answers to queries posed on inconsistent dimensions. A minimal repair is a new dimension that satisfies the strictness and covering constraints, and that is obtained from the original dimension through a minimum number of changes, which has a NP-hard computational cost. To improve this, the extended dimension id defined and proposed the compatible repair to obtain a new consistent dimension, maintaining the essence of the original dimension and can be calculated efficiently. On this last focuses experimentation, developing and testing a program that generates this compatible repair and get results in polynomial time.

Keywords: Data warehouse, inconsistency, strictness constrains, covering constrains, dimension, experiments.

INTRODUCCIÓN

Los Data Warehouse (DW) son almacenes de datos que se organizan en dimensiones y hechos, reúnen información de múltiples fuentes y la almacenan de forma histórica para el análisis y apoyo a la toma de decisiones. Los DW se representan mediante modelos multidimensionales y se componen de dimensiones y hechos. Las dimensiones se modelan como jerarquías de elementos, entregando el orden jerárquico donde cada elemento pertenece a una categoría (también conocido como niveles) [1] y estas categorías también se organizan a través de los esquemas de jerarquía. Los hechos corresponden a eventos que se asocian generalmente a valores numéricos conocidos como medidas, y hacen referencia a elementos de la dimensión, estos elementos se pueden apreciar en el Ejemplo 1.

Ejemplo 1: La FIFA tiene un DW para administrar la información sobre las selecciones nacionales de futbol. Este DW usa la dimensión Tiempo que está organizada jerárquicamente por las categorías Día, Mes, Año y All, y por la Dimensión Selecciones que se puede ver en la Figura 1a. En la Figura 1b se pueden apreciar los elementos pertenecientes a cada categoría de la dimensión Equipos y las relaciones rollups[1] entre cada una de ellas. Por ejemplo: CH (selección chilena), SP (selección española), AU (selección australiana) son elementos de la categoría Equipos y SA (América del sur), WEU (Europa del este), y AS (Asia) son elementos de la categoría Zona. La relación rollup entre estas categorías son los pares (CH, SA), (SP, WEU) y (AU, AS)[2].

En la Tabla 1 se muestra la tabla de hechos Ingresos(Equipo,Fecha,Total), la cual almacena las ganancias por equipo en una fecha determinada. La estructura multidimensional permite computar consultas en diferentes niveles de granularidad. Por ejemplo, es fácil computar el total de ingresos agrupados por zona y mes, o el total de ingresos por confederación en un año específico, entre otros.

(a) Esquema de jerarquía de la dimensión

(b) Elementos y relaciones rollups de la dimensión

Figura 1: Ejemplo de data warehouse Selecciones [3].

Las dimensiones son consideradas la parte estática de los DW, mientras que los hechos se consideran la parte dinámica, en el sentido de que las operaciones de actualización afectan principalmente a las tablas de hechos [2]. La estructura multidimensional de un DW permite a los usuarios formular consultas en diferentes niveles de granularidad.

Tabla 1: Tabla de Hechos Ingresos para la dimensión Selecciones.

Ingresos		
Equipo	Fecha	Ingreso
CH	01-01-2014	$90.000
SP	01-07-2014	$130.000
CH	01-01-2015	$110.000
SP	01-07-2015	$150.000

[1] Relación hijo-padre entre los elementos de diferentes categorías unidos mediante el esquema de jerarquía

[2] La FIFA considera a Australia como parte de Asia.

Las respuestas consistentes a consultas en DW se basa en el uso de consultas pre-computadas en categorías inferiores para calcular resultados en los niveles más altos de la jerarquía. En una dimensión consistente, cada medida en la tabla de hechos se debe agregar como máximo una vez. Para asegurar que esta calidad se mantiene, cuando se usan consultas pre-computadas la dimensión debe cumplir ciertas restricciones de integridad (RI), las cuales se dividen en estrictas y homogéneas [4-7].

Una dimensión es estricta si todas sus relaciones rollup (directas o indirectas en el esquema jerárquico) entre elementos de las categorías son relaciones muchos a uno. La restricción homogénea indica que los rollups son obligatorios entre elementos de dos categorías que estén conectados en el esquema jerárquico.

Lo ideal sería que todas las relaciones dentro de una categoría satisfagan estas propiedades para asegurarse un cómputo eficiente al usar consultas pre-computadas. Sin embargo, cuando se modela una determinada dimensión, podría darse el caso de que algunas categorías no cumplan alguna de las RI impuestas en ella, como se aprecia en el Ejemplo 2.

Ejemplo 2: En la Figura 1b se puede apreciar que el rollup entre las categorías Equipos y Patrocinador no es estricta, ya que hay elementos en Equipos que tienen más de un Patrocinador como es el caso de CH. Así mismo, esta relación tampoco es homogénea ya que AU no hace rollup a ningún elemento en Patrocinador. Sin embargo, si la dimensión completa es estricta/homogénea dependerá de las RI que se hayan impuesto.

En el mundo real, las dimensiones no siempre son estrictas u homogéneas, por lo tanto, para garantizar la capacidad de respuestas consistentes es importante saber cuáles son las RI impuestas para la dimensión [8-9] y estas restricciones las agruparemos en el conjunto $\sum$.

Una dimensión se dice que es consistente respecto a $\sum$ si satisface cada una de sus restricciones estrictas y homogéneas definidas en ella, de lo contrario, la dimensión es inconsistente [10]. Verificar que una dimensión satisface sus RI se puede hacer en tiempo polinomial [10].

Una dimensión usualmente se vuelve inconsistente luego de operaciones de actualización [10-11] y si se quieren obtener respuestas desde una dimensión inconsistente es necesario repararla o corregirla. Para repararla se han definido algunas técnicas como la reparación minimal, una de las primeras aproximaciones a este tipo de reparación, la cual, realiza cambios de arcos entre las relaciones que no cumplen alguna de las RI impuestas para la dimensión, pero con un mínimo número de cambios de arcos (agregando y eliminando arcos) en la dimensión original [10, 12] y de todas las reparaciones posibles se seleccionan las que posean el menor número de cambios.

El problema de este método es que encontrar las reparaciones minimales de una dimensión inconsistente puede ser exponencial [10], por otro lado, la complejidad computacional para encontrarla es NP-duro y decidir si una dimensión D' es minimal es co-NP-completo [10]. De todas formas en [10] se propuso un programa en Datalog basado en semántica de modelos estables que permite obtener las reparaciones minimales de una dimensión, aunque este programa en lógica no es capaz de obtener todas las posibles reparaciones minimales de una dimensión D respecto a un conjunto de restricciones $\sum$ [10].

En el marco de definir una respuesta aproximada utilizando el conjunto de reparaciones minimales obtenidas de una

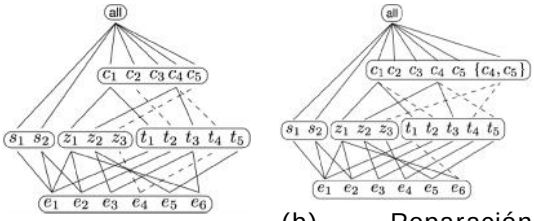
dimensión inconsistente, en [12] se propone la reparación canónica que es una nueva dimensión que agrupa en una única dimensión todas las reparaciones minimales obtenidas para aplicar las consultas pre-computadas.

TRABAJO PREVIO

Para obtener una dimensión consistente respecto a sus RI estrictas y homogéneas, que no necesiten del cómputo de las reparaciones minimales, pero que se obtenga con un mínimo número de cambios, se han desarrollado dos métodos, el primero propuesto en [13], el cual, repara una dimensión que se vuelve inconsistente con respecto a sus restricciones estrictas, después de una reclasificación de elementos, llamada r-reparación. Esta reparación se realiza cuando la dimensión cuenta con una categoría conflictiva llamada Nivel de Conflicto [11].

El segundo método se basa en el modelo de dimensión canónica presentado en [12], para ello, se define la dimensión extendida [3], la cual se crea para poder insertar conjuntos de elementos en sus categorías y poder dar respuesta a consultas de agregación, tal como se muestra en [12], donde se utiliza una dimensión clásica para crear la reparación canónica, sin poder soportar por definición elementos conjunto y así poder obtener una reparación más eficiente por medio de la dimensión extendida.

En la dimensión extendida, se redefine la dimensión D clásica como la tupla $X=(H, E_X, CElem_X, <<_X)$ donde H es el esquema jerárquico de la dimensión original, el cual nunca sufre modificaciones; E_X son los elementos existentes en la dimensión; $CElem_X$ es una función que dada una categoría devuelve una familia de subconjuntos de E_X; y $<<_X$ representa la relación hijo/padre entre los elementos de diferentes categorías. Se denota por $<<^*_X$ la cláusula reflexiva y transitiva de $<<_X$ [3]. La principal necesidad de redefinir la dimensión es para tener la capacidad de poder almacenar conjuntos de elementos en las categorías tal como se aprecia en la Figura 2b, algo que la reparación canónica requería pero no tenía definido.



(a) Dimensión inconsistente D

(b) Reparación compatible X

Figura 2: Ejemplo de dimensión inconsistente y su reparación compatible.

Con la definición de la dimensión extendida, se propone elaborar un método de reparación que se llamó reparación compatible, el cual se elabora para obtener una reparación que no requiera de computar las reparaciones minimales y en especial que funcione cuando no se conoce un estado consistente de la dimensión como para revertirlo, no así, el algoritmo mostrado en [13] ya que para devolver la consistencia a la dimensión es necesario mantener el cambio que originó la inconsistencia. En definitiva, para la reparación compatible es necesario obtener una aproximación que permita responder a consultas de agregación [12].

Dada la dimensión original $D=(H_D, E_D, Elem_D, <_D)$ que es inconsistente respecto a un conjunto de RI $\sum$. La dimensión $X=(H_X, E_X, CElem_X, <<_X)$ es una reparación compatible de D respecto a $\sum$ si se obtiene mediante el Algoritmo 1 propuesto en [3].

El algoritmo de la reparación compatible obtiene todos los elementos que en primera instancia son inconsistentes respecto a $\sum$ para el conjunto de RI homogéneas, posteriormente busca evidencias (línea 6)

para evaluar el mejor cambio de arcos, en caso de haber más de un elemento al que se podría hacer el cambio de arcos, se decide por el primero que se encuentra, en caso de no existir evidencia, realiza un cambio de arcos al primer elemento de la categoría padre con el fin de devolver la consistencia a las RI homogéneas (línea 7), sin embargo, estos últimos casos pueden ocasionar inconsistencias con respecto a sus restricciones estrictas.

Algoritmo 1: Reparación Compatible.

```
Input: Dimensión D y un conjunto de restricciones ∑
Output: Reparación Compatible X
1  C ← conjunto de tuplas de la forma (ic,inc) donde ic es una restricción que viola en ∑ e inc es el conjunto de elementos que participan en esa inconsistencia ic
2  X ← D
3  for (ic, inc) ∈C do
4  if ic es una restricción homogénea then
5  for e ∈ inc do
6  Evidencia ← BuscaEvidencia(D,ic,e)
7  X ← Reparar_C(X,ic,e,Evidencia)
8  for (ic, inc) ∈ C do
9  if ic es una restricción estricta then
10 for e ∈ inc do
11 Evidencia ← BuscaEvidencia(D,ic,e)
12 X ← Reparar_S(X,ic,e,Evidencia)
13 return X
```

Luego, se evalúan todos los elementos que son inconsistentes respecto a las RI estrictas, para buscar alguna evidencia de hacia qué elemento padre deben hacer rollup el elemento conflictivo (línea 11) en este caso, si hay evidencia a un único elemento, se hace el cambio de arco correspondiente, en caso de que no existan evidencias o haya más de una, se genera un elemento conjunto que posea todos los elementos posibles en la categoría padre a los que el elemento conflictivo hace rollup para posteriormente hacer el cambio de arcos a ese nuevo elemento (línea 11).

Este proceso se repite hasta que no existan elementos inconsistentes [3] y cada cambio de arcos no debe generar nuevas inconsistencias [11] La existencia de evidencias ayudará a decidir qué acciones, en términos de reclasificación de arcos, se deben tomar para restaurar la consistencia. En el Ejemplo 3 se puede apreciar el trabajo del Algoritmo 1 en una dimensión D inconsistente.

Ejemplo 3: Considere ahora el esquema de jerarquía de la Figura 1a y la dimensión de la Figura 2a la cual es inconsistente respecto a la RI homogénea Torneo Ç Confederación por el elemento t_5 y también es inconsistente respecto a la RI estricta Equipo ì Confederación por los elementos e_2, e_4 y e_6. El algoritmo presentado en [13] no puede reparar esta dimensión, ya que requiere conocer el cambio de arco que generó la inconsistencia para obtener una solución, además no está definido para reparar inconsistencias con respecto a las RI homogéneas. Entonces, si se computa esta dimensión en el programa en DLV propuesto en [10] para obtener las reparaciones minimales se obtienen 17 reparaciones minimales con 7 cambios para cada una (4 inserciones y 3 eliminaciones de arcos).

Si se computa esta misma dimensión en el Algoritmo 1 se obtiene la reparación compatible X de la figura 2b, la cual es una dimensión extendida que es consistente respecto al conjunto ∑ de RI. Esta dimensión se obtiene con 9 cambios de arcos (5 inserciones y 4 eliminaciones de arcos) y al compararlo con el programa en DLV, es más efectivo ya que solo se procesa una sola reparación con 9 cambios y no 17 reparaciones con 7 cambios como el programa en lógica.

El Algoritmo 1 genera un elemento conjunto llamado {c_4, c_5} insertándolo en la categoría Confederación al cual hace rollup el elemento e_4, por medio de sus categorías superiores z_3 y t_4, estos al no tener otros elementos que les

hagan rollup, no genera nuevas inconsistencias, por lo que no hay problema en hacer el cambio de arcos.

Complejidad del Algoritmo

En [3] se explica el cálculo de la complejidad para el Algoritmo 1, el cual se señala a continuación. Sea n es el número máximo de elementos de una categoría, r es el número máximo de las relaciones del rollup entre dos categorías, m_c es el número de restricciones homogéneas y m_s es el número de restricciones estrictas. El costo de verificar si una restricción homogénea $c_i \hat{G} c_j$ se viola es $O(n^2)$ ya que se debe que comprobar si hay una relación rollup entre cada elemento en c_i y un elemento en c_j. El costo de comprobar si una restricción estricta $c_i \grave{\imath} c_j$ se viola es $O(r^2)$ ya que la relación r_{ci}^{cj} es de tamaño O(r) y se requiere para comparar cada tupla en ella con todas sus tuplas. Por lo tanto, el costo de obtener C, es $O((m_c * n^2)+(m_s * r^2))$. A pesar de que en el peor de los casos r es $O(n^2)$, se espera que r sea O(n) ya que en general el número de violaciones de una restricción es pequeña, y por lo tanto la mayoría de los elementos en una categoría se resumen en un único elemento de una categoría superior.

Dada una restricción ic de la forma $c_i \grave{\imath} c_j$ o $c_i \hat{G} c_j$, BuscaEvidencia calcula las rutas entre un elemento e' que pertenece a la categoría c_a con $c_a \nearrow c_i$, tal que (e',e) con $e \in c_i$, y los elementos en el nivel de conflicto de D. El número de rutas y el costo de computarla, en el peor de los casos es $O(n^{|C|})$, pero en la mayoría de los casos, en que el número de inconsistencias es bajo, es de O(|C|). Por último, las funciones Reparar_C y Reparar_S poseen un costo de O(|C|).

Por lo tanto, en [3] se ha demostrado que computar una dimensión inconsistente respecto a un conjunto ∑ de RI por medio de el algoritmo 1 en el peor de los casos tiene un costo computacional de $O(m_c(n^2+n^{|C|})+m_s(r^2+n^{|C|}))$ y en el caso esperado tiene un orden de $O(n^2(m_c+m_s))$.

EXPERIMENTOS

Para evaluar la factibilidad de obtener una reparación compatible obtenida desde un Data Warehouse inconsistente se implementó el Algoritmo 1 en lenguaje C con estructura de datos basada en listas ligadas por punteros.
El primer ejemplo realizado para probar los resultados fue en la dimensión D de la Figura 2a, donde la reparación compatible de la Figura 2b se obtuvo en 0.061 segundos.

Posteriormente, se propuso la dimensión D_s de la Figura 3, que utiliza el siguiente esquema de jerarquía, la categoría Equipo está conectado con Zona (Zona Geográfica) y ésta a su vez con Confederación. También, la categoría Equipo está conectado a la categoría Torneo y ésta a su vez con Confederación. La categoría más alta es All a la cual llega Confederación.

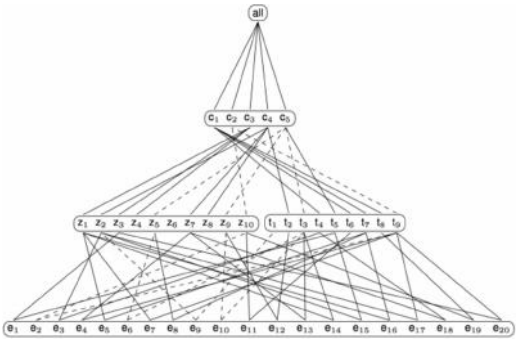


Figura 3: Dimensión Selecciones D_s Inconsistente.

Las restricciones de integridad que debe cumplir la dimensión D_s vienen dadas por el conjunto ∑ de RI que se presentan en la Tabla 2 y como se puede apreciar en la Figura 3 los arcos con líneas segmentadas, representan los los elementos consistentes respecto a ∑. Sin embargo, la dimensión D_s es inconsistente respecto a las restricciones de integridad homogéneas Zona $\hat{G}$ Confederación por el elemento z_6 y Torneo $\hat{G}$ Confederación por el

elemento t_1. También, D_s es inconsistente respecto a la restricción de integridad estricta Equipo ì Confederación por los elementos e_1, e_3, e_4, e_5, e_7, e_8, e_{11}, …, e_{20}. La dimensión D_s posee además, 4 elementos que no participan de la violación de ninguna restricción[3] (e_2, e_6, e_9 y e_{10}).

Tabla 2: Conjunto ∑ de Restricciones de integridad de la dimensión Selecciones D_s.

Restricciones Estrictas	Restricciones Homogéneas
Equipo ì Zona Equipo ì Torneo Equipo ì Confederación Zona ì Confederación Torneo ì Confederación	Equipo Ĝ Zona Equipo Ĝ Torneo Zona Ĝ Confederación Torneo Ĝ Confederación

En el caso de la dimensión D_s cuando se viola la RI estricta Equipo ì Confederación se provoca que un equipo pertenezca a más de una confederación, transformándose en un conteo erróneo al momento de evaluar consultas pre-computadas. En este caso, existe un doble conteo, ósea hay equipos que hacen rollup a dos confederaciones distintas, esto se puede apreciar en la Tabla 3 donde aparecen las confederaciones y todos los equipos que le pertenecen a cada una y en la cual se puede señara, como un ejemplo, que el equipo e_3 pertenece a la confederación c_1 y c_4 cosa que no debe suceder. Al contabilizar de forma simple, se aprecia que en la figura hay 20 equipos o selecciones, y 5 confederaciones, por restricción de la dimensión un equipo solo puede pertenecer a una única confederación, pero en la dimensión D_s hay 36 equipos repartidos entre 5 confederaciones, esto sucede por la inconsistencia en la dimensión y el doble conteo de datos.

[3] Este tipo de dimensiones es un caso muy improbable que suceda en la vida real, lo común es que se obtengan dimensiones inconsistentes con porcentajes inferiores al 50% [3].

Tabla 3: Rollup entre Equipo y Confederación de la Figura 3.

$R_{D\ Equipo}^{Confederación}$	
Confederación	Equipo
c_1	e_3, e_5, e_8, e, e_{11}, e_{16}, e_{18}, e_{19}
c_2	e_2, e_4, e_7, e_{11}, e_{17}, e_{19}, e_{20}
c_3	e_1, e_5, e_7, e_{14}, e_{16}, e_{20}
c_4	e_3, e_4, e_{12}, e_{13}, e_{15}, e_{17}, e_{18}
c_5	e_1, e_6, e_8, e_{10}, e_{12}, e_{13}, e_{14}, e_{15}

Si se computa esta dimensión D_s en el Algoritmo 1 se obtiene la reparación compatible de la Figura 4, la cual es una dimensión extendida X_s que es consistente respecto al conjunto ∑ de RI presentados en la Tabla 2

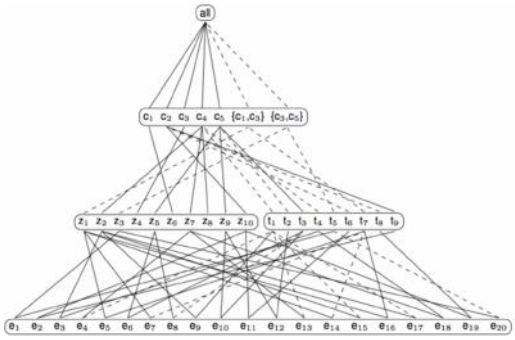


Figura 4: Reparación Compatible X_s obtenida desde la Dimensión D_s Inconsistente y el Algoritmo 1.

Una vez obtenida la reparación compatible X_s se puede apreciar la diferencia con la dimensión D_s. En la reparación compatible los elementos consistentes no sufrieron alteraciones respecto de sus confederaciones originales, los elementos no homogéneos z_6 y t_1 se les asignaron los padres c_1 y c_4 respectivamente, a z_6 se le asignó el elemento c_1 en la categoría Confederación ya que no existía evidencia de rollup, por lo tanto se le asignó el primer elemento de la categoría

padre, en cambio al elemento t_1 se le asignó el elemento c_4 por la evidencia que le daba el elemento e_9 en la categoría Equipo el cual hacia rollup al elemento c_4 por medio de la categoría Zona.

Para reparar la inconsistencia estricta, el algoritmo detectó que 6 de los dieciséis elementos inconsistentes originales no poseían evidencia clara, por lo que la reparación compatible generó el elemento conjunto $\{c_1, c_3\}$, al cual le hacen rollup los elementos e_5, e_7, e_{14} y e_{16}, y el elemento conjunto $\{c_3, c_5\}$ al cual hacen rollup los elementos e_1 y e_{20}. Para el resto de elementos se encontró evidencia, por lo cual solo se hicieron cambios de arcos, eliminando el doble conteo de los datos de la dimensión.

El resultado final es la dimensión X_s de la Figura 4, en la cual se indican que las líneas segmentadas son los arcos que sufrieron cambios a diferencia del original. Cabe señalar que en la Tabla 4 se pueden apreciar a qué confederación finalmente pertenece cada equipo en la dimensión X_s, lo que deja mejor distribuidos los datos y sin repeticiones.

Tabla 4: Rollups entre Equipo y Confederación en la dimensión extendida X_s.

$R_{Xs\ Equipo}^{Confederación}$	
Confederación	Equipo
c_1	
c_2	e_2, e_{11}, e_{19}
c_3	
c_4	e_3, e_4, e_9, e_{13}, e_{15}, e_{17}, e_{18}
c_5	e_6, e_8, e_{10}, e_{12}
$\{c_1, c_3\}$	e_5, e_7, e_{14}, e_{16}
$\{c_3, c_5\}$	e_1, e_{20}

Los nuevos elementos conjunto son elementos que permitirán usarlos para obtener respuestas aproximadas a consultas pre-computadas [3]. Por último, indicar que la dimensión obtenida desde la implementación del Algoritmo 1 obtuvo la reparación compatible X_s (Figura 4) en un tiempo de 0.193 segundos.

En definitiva, el Algoritmo 1 propuesto en [3] no conoce un estado consistente de la dimensión, tampoco los cambios que originaron su inconsistencia, no computa las reparaciones minimales y no busca mantener la semántica de los datos y sus relaciones, solo obtiene una dimensión consistente respecto al conjunto ∑ de RI, sin embargo, en el proceso de buscar evidencia, trata en lo posible de mantener la semántica de los datos al momento de hacer las reparaciones.

EXPERIMENTO EN ESCENARIO REAL

Esta implementación del Algoritmo 1 es una primera versión, para el cual se ha estudiado su comportamiento en una base de datos realista tomando el caso de los Teléfonos de Chile, una dimensión propuesta originalmente en [12], para computar la dimensión canónica. Dada esta misma dimensión, se estudió el tiempo que demora en generar la reparación compatible en distintos porcentajes de inconsistencia para dimensiones que tienen millones de datos.

La dimensión Teléfonos está basada en el país de Chile, el cual está divido políticamente en comunas que en la actualidad ascienden a 346. Cada comuna tiene un cierto número de teléfonos (de acuerdo a su población), los cuales están asociados con un código de área. Ese código de área se relaciona con una única región, lo mismo sucede para una comuna. El esquema jerárquico que modela dicha situación es el que se presenta en la Figura 5. Cabe señalar que se incluye la cantidad de elementos que pertenecen a cada categoría en forma de cardinalidad (# número).

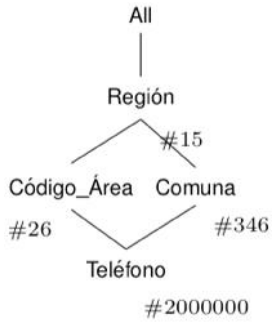


Figura 5: Esquema jerárquico de la dimensión Teléfonos y cantidad de elementos por categoría.

Los resultados obtenidos para el código implementado en la dimensión Teléfonos con distintos porcentajes de inconsistencias, son los que se muestran en la Tabla 5, además, para una mejor interpretación, se proporciona el gráfico de Tiempo (Figura 6) el cual muestra el tiempo de ejecución requerido para obtener la reparación compatible, dados distintos niveles de inconsistencia.

Tabla 5: Resultados de aplicar la reparación compatible a la dimensión Teléfonos en distintos porcentajes de inconsistencia.

Inconsistencia	Elementos Inconsistentes	Tiempo (hh:mm:ss)
0,03%	500	00:00:15,290
0,61%	12.120	00:04:11,687
1,59%	31.720	00:04:10,345
2,35%	46.940	00:18:04,076
5,79%	115.420	02:05:49,978

Analizando el gráfico de la Figura 6 se puede apreciar que en los primeros experimentos el tiempo de ejecución del algoritmo es más o menos estable, realizando cambios en categorías superiores (involucrando pocos cambios de arcos), pero en el último caso se ve un incremento notable respecto a los anteriores casos, debido a la cantidad de elementos que participan de la inconsistencia, y a la gran cantidad de cambios computados para devolver la consistencia a la dimensión y fueron realizados entre las categorías inferiores.

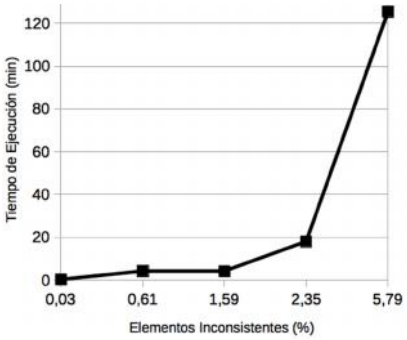


Figura 6: Gráficos obtenidos para computar la reparación compatible en la dimensión Teléfonos.

Basado en los resultados anteriores se puede mencionar que la reparación compatible es obtenida con éxito en todos los casos (eliminando las inconsistencias), con el cual se obtiene una dimensión extendida que es consistente respecto a sus RI.

No es posible presentar como se encontraba la dimensión antes y después por el tamaño de esta y sus cambios, solo se sabe que al procesar la dimensión obtenida nuevamente en el algoritmo, esta no presenta inconsistencias. También el tiempo que tarde el algoritmo dependerá bastante de la maquina donde se esté ejecutando, y es de asumir que una aplicación de reparación de un DW se realizará en un servidor o un equipo con capacidad suficiente para realizar este proceso.

Equipo de pruebas y obtención de datos:

El equipo donde se realizaron las pruebas y computaron los distintos ejemplos posee las siguientes características:
Hardware:

- Procesador Intel® Core™ i5-2430M.
- CPU @ 2.40GHz x 4.
- Memoria 7.7Gb.
- Intercambio 10Gb.

Sistema:

- Debian Versión 8.2 (Jessie) de 64-bit.
- Núcleo Linux 3.16.0-4-amd64.
- GNOME 3.14.0.

Software:

- PostgresSQL versión 9.4.4 para manejo de Bases de Datos.
- gcc versión 4.9.2-10 como compilador.
- Chocolat 3.3 como editor de texto para programación en lenguaje C.
- DLV versión x86-64-linux-elf-unixodbc (2012-12-17).

Para calcular el tiempo de ejecución del programa se ha utilizado el comando time de Linux, este comando permite obtener el tiempo total de ejecución de un programa. Su especificación se puede ver en el manual correspondiente.

CONCLUSIONES Y TRABAJO FUTURO

En [3] se estudia el problema de las dimensiones inconsistentes y proponen una técnica para la reparación de ellas basadas en el uso de la dimensión extendida, que permite la inserción de elementos conjuntos en una categoría. Esta nueva dimensión extendida nos permite expresar imprecisión en los datos. La incertidumbre y la imprecisión se han analizado en el contexto de modelos multidimensionales en [14, 15]. En particular, en [15] la imprecisión está permitida en las dimensiones, y la incertidumbre en los valores de la tabla de hechos.

La reparación compatible es una dimensión ampliada, obtenida a partir de la dimensión incompatible que satisface las restricciones estrictas y homogéneas. Esta reparación permite hacer frente a las ambigüedades que surgen de las inconsistencias y es posible obtenerla en tiempo polinomial, esto porque la reparación canónica propuesta en [12] requiere obligatoriamente obtener el conjunto de reparaciones minimales, y por tanto, no se puede calcular en tiempo polinomial.

En este artículo se presentan los resultados obtenidos por la implementación en lenguaje C del Algoritmo 1 propuesto en [3], donde se busca la consistencia de los elementos inconsistentes en base a la evidencia que estos elementos entregan.

El programa implementado no altera información que es consistente en la dimensión original, principalmente repara en base a evidencias y solo cuando no existen tales evidencias se generan elementos conjuntos.

Mencionado esto, se puede decir que una gran parte de la dimensión original está contenida en la reparación compatible, lo cual permite mantener la semántica de estos elementos a diferencia de métodos como las reparaciones minimales [10] o la misma reparación canónica [12], las cuales buscan obtener una reparación con el mínimo número de cambios pero sin intentar mantener la semántica de los datos, por lo que este método además de reparar, hace un primer intento a mantener la semántica de los datos en la reparación.

En el peor de los casos, si la dimensión es totalmente inconsistente, al principio del proceso se irán generando elementos conjuntos, pero a medida que avance la reparación se encontrarán evidencias, lo que la respuesta aproximada obtenida de la reparación compatible, como se explica en [3] seguirá siendo mejor que la respuesta obtenida de la dimensión inconsistente.

La reparación compatible puede ser considerada en la categoría de limpieza de bases de datos, basado en [16-18]. Así mismo, la reparación compatible podría ser de gran valor para el administrador del DW ya que al ser consistente puede ser utilizada para realizar consultas de agregación.

El algoritmo implementado, permite obtener la reparación compatible en un número finito

de pasos, determinado principalmente por el número de elementos inconsistentes de la dimensión original. Sin embargo, cuando existen millones de elementos pueden surgir miles de evidencias y el algoritmo requiera procesar más opciones, el tiempo para obtener una respuesta es mayor, pero a pesar de este inconveniente, el algoritmo entrega una dimensión consistente y en un tiempo aceptable.

Como trabajo futuro queda mejorar el proceso de generación de la reparación compatible, implementando alternativas de restauración de consistencia, principalmente mejorando la elección de elementos a los cuales insertar un rollup y por último, mejorar el programa utilizando técnicas algorítmicas que permitan optimizar el consumo de memoria temporal y mejorar los tiempo de proceso.
Adicionalmente, incorporar grados de prioridad a la hora de intentar hacer cambios de arcos para evitar realizar cambios que el administrador estime inconvenientes y manteniendo la semántica de los datos.

AGRADECIMIENTOS

Raúl Arredondo agradece a la Dirección de Investigación de la Universidad de Los Lagos, a través del proyecto interno de investigación R12/15.

REFERENCIAS

[1] S. Chaudhuri and U. Dayal, “An Overview of Data Warehousing and OLAP Technology,” SIG-MOD Record, vol. 26, no. 1, pp. 65–74, 1997.
[2] C. Hurtado, A. Mendelzon, and A. Vaisman, “Updating OLAP Dimensions,” in DOLAP’99, 1999, pp. 60–66.
[3] J. Ram´ırez, L. Bravo, and M. Caniup´an, “Ex-tended dimensions for cleaning and querying inconsistent data warehouses,” in Proceedings of the Sixteenth International Workshop on Data Warehousing and OLAP, ser. DOLAP ’13. New York, NY, USA: ACM, 2013, pp. 39–46.
[4] M. Rafanelli and A. Shoshani, “STORM: a Statistical Object Representation Model,” in SSDBM’90, 1990, pp. 14–29.
[5] H.-J. Lenz and A. Shoshani, “Summarizabi-lity in OLAP and Statistical Data Bases,” in SSDBM’97, 1997, pp. 132–143.
[6] C. Hurtado, C. Gutierrez, and A. Mendelzon, “Capturing Summarizability with Integrity Constraints in OLAP,” ACM Transactions on Database Systems, vol. 30, no. 3, pp. 854–886, 2005.
[7] J.-N. Maz´on, J. Lechtenb¨orger, and J. Truji-llo, “A Survey on Summarizability Issues in Multidimensional Modeling,” Data Knowl. Eng., vol. 68, no. 12, pp. 1452–1469, 2009.
[8] C. Hurtado and C. Guti´errez, “Data Warehouses and OLAP: Concepts, Architectures and Solu-tions.” Idea Group, Inc, 2007, ch. Handling Structural Heterogeneity in OLAP.
[9] C. Letz, E. T. Henn, and G. Vossen, “Consistency in Data Warehouse Dimensions,” in Proceedings of the 2002 International Symposium on Database Engineering & Applications, ser. IDEAS ’02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 224–232.
[10] M. Caniup´an, L. Bravo, and C. A. Hurtado, “Repairing inconsistent dimensions in data wa-rehouses,” Data & Knowledge Engineering, vol. 79-80, pp. 17–39, 2012.
[11] M. Caniup´an and A. Vaisman, “Repairing dimension hierarchies under inconsistent reclassifica-tion,” in Proceedings of the 30th international conference on Advances in conceptual modeling: recent developments and new directions, ser. ER’11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 75–85.
[12] L. Bertossi, L. Bravo, and M. Caniup´an, “Con-sistent Query Answering in Data Warehouses,” in AMW, 2009.
[13] M. Caniup´an, A. Vaisman, and R. Arredondo, “Efficient repair of dimension hierarchies under inconsistent reclassification,” Data Knowledge Engineering, vol. 95, pp. 1–22, 2015.
[14] T. Pedersen, C. Jensen, and C. Dyreson, “Sup-porting Imprecision in Multidimensional

Databases Using Granularities," in SSDBM'99, 1999, p. 90.
[15] D. Burdick, P. M. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan, "OLAP over uncertain and imprecise data," The VLDB Journal, vol. 16, no. 1, pp. 123–144, 2007.
[16] L. Bertossi and L. Bravo, "Generic and Declara-tive Approaches to Data Cleaning: Some Recent Developments," in Handbook of Data Quality -Research and Practice, S. Sadiq, Ed. Springer-Verlag Berlin Heidelberg, 2013.
[17] W. Fan, "Extending dependencies with conditions for data cleaning," in The 8th IEEE International Conference on Computer and Information Technology, 2008, pp. 185–190.
[18] W. Fan, F. Fan, X. Jia, and A. Kementsietsi-dis, "Conditional Functional Dependencies for Capturing Data Inconsistencies," ACM Trans. Database Syst., vol. 33, no. 2, pp. 6:1–6:48, 2008.