Universidad del Bío-Bío, Chile

Facultad de Ciencias Empresariales

Departamento de Sistemas de Información

Algoritmos Polinomiales para Reparar Data Warehouses Inconsistentes

Tesis presentada por Raúl Eduardo Arredondo Flores
para obtener el grado de Magíster en Ciencias de la Computación
Dirigida por Dra. Mónica Caniupán Marileo

2013

Agradecimientos

Mis primeras palabras de agradecimiento se dirigen a la persona que más aprecio, amo y admiro en este mundo, al Dios Todopoderoso, el manantial de vida que me ha permitido alcanzar sueños que para mí hace un par de años atrás parecían imposibles. Gracias por darme la fuerza, la vida, una motivación extra cuando la necesité, por acompañarme en la soledad, por mostrarme tu misericordia cada día y por demostrarme que los milagros sí existen. Si algo he logrado ha sido por tu gracia, por tu amor y tu fidelidad.

Agradezco a mis padres, Luis y Esther, y mis hermanos, Christopher y Efraín, porque cuando he requerido su apoyo, su compañía han estado ahí; porque cuando he necesitado un abrazo me lo han dado, porque me incentivaron a seguir adelante en todo momento. Se han constituido un pilar en mi vida y un modelo a seguir, porque siempre han estado presente hasta el día de hoy.

Adicionalmente quiero agradecer a un genial grupo de personas que me han acompañado en esta travesía de 2 años: a mi profesora guía Dra. Mónica Caniupán Marileo, por haberme incentivado a trabajar en sus proyectos de investigación, por apoyarme en el desarrollo de esta tesis, apoyo que ha sido fundamental para poder culminarla, y por guiarme en el proceso de investigación mediante el envío de documentos que han servido como base para este proyecto. A mi colega de investigación y amigo Juan Ramírez Lama por ser un apoyo, un gran compañero de trabajo y por compartir tantas alegrías juntos. A mis colegas de Magíster y amigos Mauricio Elgueta Lizarrague, Esteban Salazar Santis y Luis Cabrera Crot por toda la ayuda que me han prestado en estos años, por ser un apoyo a mi desarrollo del conocimiento en las ciencias de la computación. Agradecer a todos los académicos del Magíster que participaron en mi desarrollo curricular, a los administrativos del Departamento de Sistemas de Información, a mis familiares, a la congregación eclesiástica en la que participo y a un grupo anónimo de personas, que en cada momento demostraron con un gesto, con una palabra, su aprecio y cariño.

Muchas gracias a todos los que me rodearon en este tiempo. Nunca se olviden que todo sueño es posible alcanzarlo, porque hay un versículo del libro más vendido de la historia que dice: Si puedes creer, al que cree todo le es posible.

Abstract

A Data Warehouse (DW) is a data repository that is modeled according to the multidimensional model, which structures the information according to dimensions and facts. A dimension is an abstract concept that groups data that share a common semantic meaning. The dimensions are modeled using hierarchies of categories, that contain elements. A dimension is *strict* if every element of every category has a unique ancestor in each higher category, and is *covering* if every element has at least one ancestor in each higher category. If a dimension does not satisfy the integrity constraints that impose these conditions, we can get wrong answers when computing queries. A dimension may become inconsistent with respect to its strictness and covering constraints after update operations, for example, after a reclassification of elements. When this happens the dimension needs to be repaired (corrected). A repair is a new dimension that satisfies the set of strictness and covering constrains, and that is obtained by inserting and deleting edges between elements of categories. If a repair is *minimal*, then, it is obtained by a minimum number of changes.

It has been shown that, in general, computing minimal repairs regarding strictness and covering constraints, is an NP-complete problem. However, it has been proved that if a dimension becomes inconsistent after a single reclassify operation of elements, it is possible to compute a repair, that contains the update, in polynomial time.

In this thesis we implement algorithms to compute repairs that keep the reclassify operation that produces inconsistencies. The algorithms work for a particular class of dimensions. Also, we consider others restrictions that may be imposed by the Data Warehouse Administrator, such as priority and security constrains.

Keywords — Data Warehouse, inconsistency, strictness constraints, covering constraints.

Resumen

Un Data Warehouse (DW) es un almacén de datos que se modela utilizando el modelo multidimensional, el cual estructura la información de acuerdo a dimensiones y hechos. Una dimensión es un concepto abstracto que agrupa datos que comparten un significado semántico común. Las dimensiones se modelan mediante jerarquías de categorías, las que contienen elementos. Una dimensión es *estricta* si cada elemento de toda categoría tiene un único ancestro en cada categoría superior y *homogénea* si cada elemento tiene al menos un ancestro en cada categoría superior. Si una dimensión no satisface las restricciones de integridad que imponen estas condiciones, al utilizar vistas precomputadas para responder consultas, se pueden obtener respuestas incorrectas.

Una dimensión se puede volver inconsistente con respecto a sus restricciones de integridad *estrictas* y *homogéneas* luego de efectuar una actualización, por ejemplo, luego de una reclasificación de elementos. Cuando esto sucede es necesario reparar (corregir) la dimensión. Una reparación es una nueva dimensión que satisface el conjunto de restricciones estrictas y homogéneas y se obtiene mediante inserción y eliminación de arcos entre elementos de las categorías. Una reparación es *minimal* si se obtiene mediante un número mínimo de cambios.

Se ha demostrado que en general computar reparaciones minimales con respecto a restricciones de integridad estrictas y homogéneas es un problema NP-completo. Sin embargo, se ha mostrado que si la dimensión se vuelve inconsistente luego de una única operación de reclasificación de elementos es posible computar una reparación, que contiene la actualización, en tiempo polinomial.

En esta tesis se implementan algoritmos para computar reparaciones que mantienen las operaciones de reclasificación que producen las inconsistencias. Esto para un caso particular de dimensiones. Además, se consideran otras restricciones que pueden ser impuestas por el administrador del Data Warehouse, tales como restricciones de prioridad y seguridad.

Palabras Clave — Data Warehouse, inconsistencia, restricciones estrictas, restricciones homogéneas

Índice general

Índice de figuras

Índice de tablas

Índice de Algoritmos

Capítulo 1

Introducción

Un Data Warehouse (DW) es un conjunto de datos orientados a temas específicos, integrados, que se acumulan en el tiempo, para apoyar la toma de decisiones (**?**), los cuales son integrados de distintas fuentes.

Los Data Warehouses se modelan bajo un enfoque multidimensiónal, a través dimensiones y hechos. Las dimensiones reflejan la forma en que se organizan los datos. Algunas dimensiones típicas, son el tiempo, ubicación y clientes.

Las dimensiones se modelan como jerarquías de elementos, llamadas ***esquemas jerárquicos*** donde cada elemento pertenece a una categoría. Los hechos corresponden a eventos que se asocian generalmente a valores numéricos llamados ***medidas***, y hacen referencia a elementos de una dimensión. Por ejemplo, hechos de venta pueden estar asociados a la dimensión Tiempo y dimensión Ubicación, y deben ser entendidos como las ventas realizadas en ciertos lugares, en determinados períodos de tiempo. El siguiente ejemplo muestra un Data Warehouse.

Ejemplo 1.1 *La F.I.F.A.*[1] *cuenta con un Data Warehouse y el administrador de éste ha determinado el esquema de la dimensión Profesional, junto a la dimensión Tiempo (ver la Figura 1.1(a)).*

El esquema de la dimensión Profesional está compuesto de las categorías: Jugador, Equipo, Liga, País Residencia, Confederación *y* All. *La categoría* Jugador *se relaciona con* Equipo, Equipo *se relaciona con* Liga, Liga *con* Confederación, Jugador *con* País Residencia, País Residencia *se relaciona con* Confederación *y* Confederación *con* All. *Cabe señalar que la categoría All es una categoría obligatoria que aparece en cada esquema jerárquico (ver detalles en la sección 2.1).*

También en la Figura 1.1(a) se muestra el esquema jerárquico de la dimensión Tiempo el cual está compuesto de las categorías Día, Mes, Año *y* All. *La categoría* Día *se relaciona con* Mes, Mes *con* Año *y* Año *con* All.

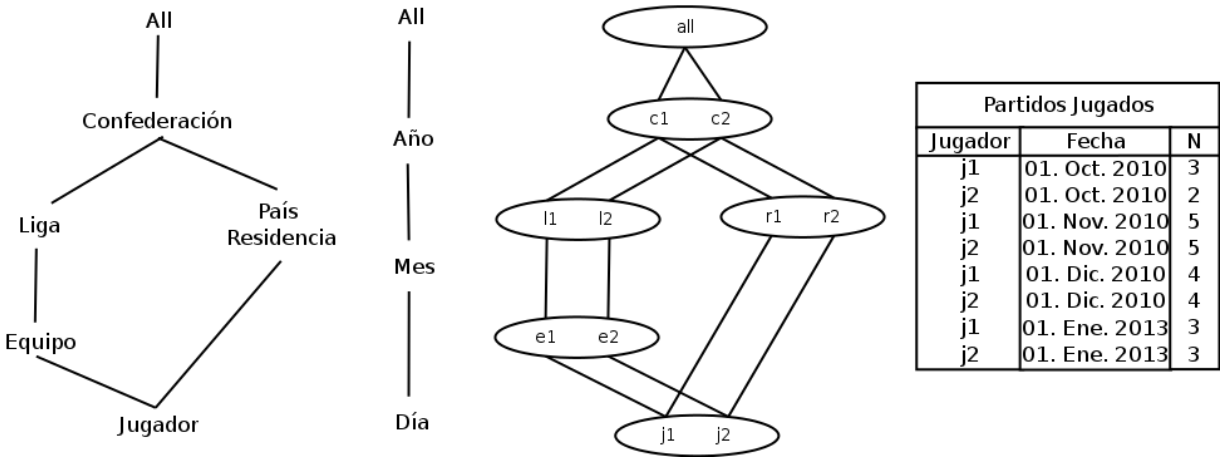
La Figura 1.1(b) muestra los elementos y relaciones entre ellos (denominadas relaciones rollup) de la dimensión Profesional donde los elementos de la categoría Jugador *son:* j_1, j_2, *los que pertenecen a* Equipo *son:* e_1, e_2, *los elementos de* Liga *son:* l_1, l_2, *los que reúne* País Residencia *son:* r_1, r_2, Confederación *cuenta con:* c_1, c_2 *y el perteneciente a* All *es:* all. *Algunas*

[1] Fédération Internationale de Football Association.

de las relaciones rollup entre elementos de las categorías de la dimensión Profesional son: $\{(j_1, e_1), (j_2, e_2), (e_1, l_1), (e_2, l_2), (j_1, r_1), (j_2, r_2), (r_1, c_1), (r_2, c_2), (l_1, c_1), (l_2, c_2), (c_1, all), (c_2, all)\}$.

La Figura 1.1(c) muestra una tabla de hechos que almacena los partidos jugados por un jugador en una fecha determinada.

Se espera que un jugador esté asociado a una única confederación, es decir que la relación entre los elementos de Jugador *y* Confederación *sea estricta. Además, un* Jugador *debe vincularse con al menos un elemento en* Equipo*, en* Liga*, en* Confederación *y en* País Residencia*. Es decir que las correspondientes relaciones rollup entre las categorías anteriormente mencionadas deben ser homogéneas u obligatorias.* □



(a) Esquema Jeráquico Profesional y Tiempo (b) Dimensión Profesional: Elementos y Relaciones Rollup (c) Tabla de Hechos

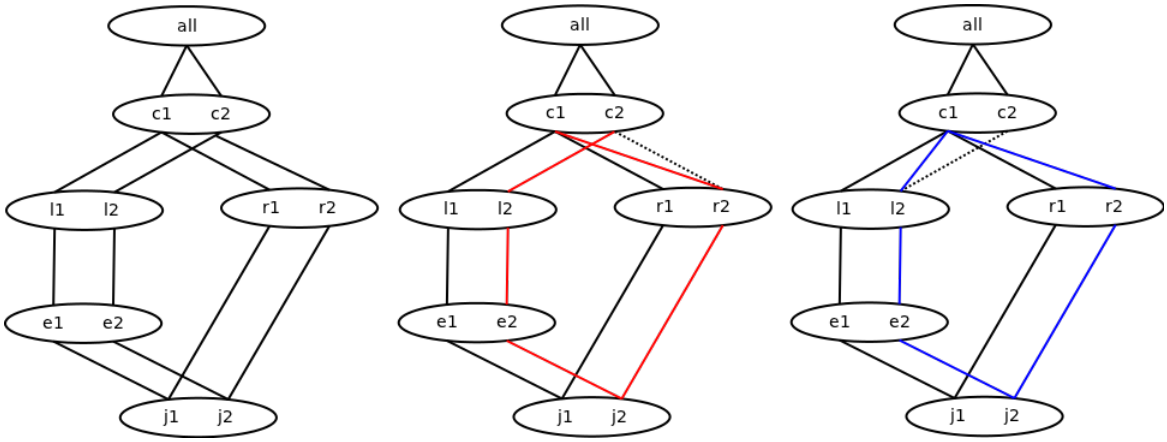
Figura 1.1: Esquemas Jerárquicos, Dimensión Profesional y Tabla de Hechos Partidos Jugados

Las relaciones rollup, pueden ser estrictas, es decir que cada elemento de una categoría inferior esté relacionado con un único elemento en la categoría superior, y homogéneas, es decir que cada elemento de una categoría inferior esté relacionado por lo menos con un elemento en las categorías superiores. Estas relaciones rollup se ven condicionadas mediante las ***restricciones de integridad*** que facilitan la navegación y cómputo de consultas agregadas (**?**), las cuales son impuestas por el administrador del Data Warehouse. Una dimensión es consistente si satisface todas sus restricciones de integridad, de lo contrario, es inconsistente (**?**).

Inicialmente, las dimensiones eran consideradas la parte estática de los DWs, mientras que los hechos se consideraban la parte dinámica, en el sentido de que las operaciones de actualización afectan principalmente a las tablas de hechos (**?**). Sin embargo, una dimensión puede volverse inconsistente con respecto a sus restricciones de integridad después de una *reclasificación* de elementos, que es una modificación a la relación rollup entre dos elementos, sobre todo para dimensiones cuyos esquemas jerárquicos cuentan con un nivel de conflicto (**?**), es decir, una categoría que es alcanzada desde la categoría inferior por caminos alternativos, como el esquema

jerárquico de la Figura 1.1(a), donde la categoría Confederación es un nivel de conflicto (detalles en capítulo 2.2). De acuerdo a (**?**) una dimensión inconsistente debe ser reparada (corregida).

Una reparación es una nueva dimensión que satisface su conjunto de restricciones de integridad (**?**) la cual se obtiene desde la dimensión original aplicando eliminación e inserción de arcos entre elementos. El ejemplo 1.2 muestra un ejemplo de reclasificación y reparación.



(a) Dimensión Profesional (b) Dimensión Reclasificada (c) Dimensión Reparada

Figura 1.2: Ejemplo de dimensión original, reclasificada y reparada

Ejemplo 1.2 *El administrador del Data Warehouse de la F.I.F.A. cuenta con la dimensión Profesional (detallada en Figura 1.2(a)). Se ha realizado una Reclasificación*($Profesional$,País Residencia,Confederación, r_2, c_1)*, es decir un cambio de arco en la dimensión Profesional, en el que* r_2 *que pertenece a la categoría* País Residencia *se relaciona ahora con* c_1 *en* Confederación *(ver Figura 1.2(b)), originando una inconsistencia con respecto a la restricción de integridad estricta* Jugador $\rightarrow$ Confederación*, que exige que cada jugador esté conectado con una única confederación. En este caso, el elemento* j_2 *no satisface esta restricción, ya que está actualmente relacionado con los elementos* c_1 *y* c_2 *en la categoría* Confederación*.*

Como consecuencia de esta inconsistencia el cómputo de consultas sobre la dimensión arroja resultados erróneos. Como ilustración se computa una consulta que obtiene la cantidad de partidos jugados agrupado por confederación. Para llevar a cabo este procesamiento las relaciones rollup serán vistas como tablas, tal como se indica a continuación:

$\mathcal{R}_{\mathcal{D}}$(Jugador,Equipo)	
Jugador	Equipo
j_1	e_1
j_2	e_2

$\mathcal{R}_{\mathcal{D}}$(Equipo,Liga)	
Equipo	Liga
e_1	l_1
e_2	l_2

$\mathcal{R}_{\mathcal{D}}$(Liga,Confederación)	
Liga	Confederación
l_1	c_1
l_2	c_2

$\mathcal{R}_{\mathcal{D}}$(Jugador,País Residencia)	
Jugador	País Residencia
j_1	r_1
j_2	r_2

$\mathcal{R}_{\mathcal{D}}$(País Residencia,Confederación)	
País Residencia	Confederación
r_1	c_1
r_2	c_1

Posteriormente se considera la tabla de hechos Partidos Jugados mostrada en la Figura 1.1(c). Una de las posibilidades de consulta es:

SQL 1.1: Cantidad de partidos jugados por Confederación

```
SELECT R3.Confederacion, SUM(PJ.N)
FROM R_D(Jugador,Equipo) R1, R_D(Equipo,Liga) R2, R_D(Liga,Confederacion) R3, ↩
    PartidosJugados PJ
WHERE R1.Jugador = PJ.Jugador AND R1.Equipo = R2.Equipo AND R2.Liga=R3.Liga
GROUP BY R3.Confederacion
```

Esta consulta de agregación es computada utlizando las relaciones rollups entre las categorías Jugador, Equipo, Liga *y* Confederación. *El resultado que entrega esta consulta en la dimensión de la Figura 1.2(b) es* $(c_1, 15)$ *y* $(c_2, 14)$.

Ahora, otra posibilidad de consulta SQL se define a continuación:

SQL 1.2: Cantidad de partidos jugados por Confederación

```
SELECT R2.Confederacion, SUM(PJ.N)
FROM R_D(Jugador,PaisResidencia) R1, R_D(PaisResidencia, Confederacion) R2, ↩
    PartidosJugados PJ
WHERE R1.Jugador = PJ.Jugador AND R1.PaisResidencia = R2.PaisResidencia
GROUP BY R2.Confederacion
```

En este caso el cómputo de la misma consulta utiliza las relaciones rollups entre las categorías Jugador, País Residencia *y* Confederación. *El resultado de aplicar esta consulta a la dimensión de la Figura 1.2(b) es* $(c_1, 29)$.

En un Data Warehouse consistente ambas consultas deben retornar el mismo resultado, sin embargo dada la inconsistencia presentada en este ejemplo los resultados de ambas consultas difieren.

Una posible reparación es la que se indica en la Figura 1.2(c). Para obtenerla se ha eliminado el arco $(l2, c2)$ *de la dimensión en la Figura 1.2(b) y se ha añadido el arco* $(l2, c1)$, *devolviendo la consistencia a la dimensión con respecto a sus restricciones de integridad.* □

La reparación obtenida en el ejemplo 1.2 es minimal, ya que fue obtenida mediante un mínimo número de cambios en las relaciones rollup, además esta reparación mantiene la reclasificación que dio origen a la inconsistencia, sin embargo, no siempre una reparación minimal podría mantener las actualizaciones realizadas en una dimensión. Por ejemplo, la consistencia en la dimensión de la Figura 1.2(b) puede además restaurarse deshaciendo la reclasificación. Es interesante en esta tesis el cómputo de reparaciones que mantengan los cambios realizados por los usuarios.

En (**?**) se ha demostrado que el número de reparaciones minimales para una dimensión con respecto a un conjunto de restricciones estrictas y homogéneas puede ser exponencial y la complejidad de computar las reparaciones minimal para una dimensión es un problema NP[1]-Completo (**?**).

En (**?**) se muestra que para un tipo particular de dimensiones que se vuelven inconsistentes después de una reclasificación, obtener una reparación sin deshacer la reclasificación de elementos puede computarse en tiempo polinomial. En esta tesis se presentan algoritmos que permiten computar una reparación para una dimensión inconsistente de forma procedural e iterativa, sin deshacer la actualización, de tal manera que satisfaga un conjunto de restricciones de integridad estrictas y homogéneas. Esto mediante la implementación de las heurísticas detalladas en (**?**). Además se consideran e implementan *restricciones de aplicación*, las cuales son impuestas por el administrador del Data Warehouse, las que imponen grados de *prioridad* entre las relaciones rollup y restricciones *seguras*. Esta clase de restricciones permite guiar el proceso de reparación, de tal forma de obtener reparaciones que tengan más sentido para el administrador del DW.

Para lograr una mayor comprensión del tema, este documento continúa con un detalle conceptual del concepto de dimensiones, restricciones de integridad, y reparaciones (Capítulo 2), definiendo la hipótesis, objetivos, metodología de trabajo, estado del arte y alcance de la tesis (Capítulo 3), exponiendo conceptos de grados de prioridad y restricciones seguras entre relaciones e implementación de los algoritmos de reparación (Capítulo 4), detallando los resultados de experimentos ejecutados para demostrar la eficacia y eficiencia de los algoritmos, además de ver el funcionamiento del algoritmo en una situación real (Capítulo 5), concluyendo en base a los experimentos y objetivos de la tesis (Capítulo 6).

[1] non-deterministic polynomial time

Capítulo 2

Preliminares

En este capítulo se formalizan los conceptos relevantes de la terminología de data warehouses tales como esquema de jerarquía, dimensión, relaciones rollup, restricciones de integridad, reclasificación, entre otros.

2.1. Dimensiones y Consistencia

Definición 2.1 (Esquema de Jerarquía (?)) Un ***Esquema de Jerarquía*** $\mathcal{H}$ se define como un par $(\mathcal{C}_{\mathcal{H}}, \nearrow_{\mathcal{H}})$, donde $(\mathcal{C}_{\mathcal{H}}, \nearrow_{\mathcal{H}})$ es un grafo acíclico dirigido. Los vértices en $\mathcal{C}_{\mathcal{H}}$ son categorías y los arcos $\nearrow_{\mathcal{H}}$ son las relaciones hijo/padre entre las categorías. El símbolo $\nearrow^*_{\mathcal{H}}$ representa la clausura transitiva y reflexiva de la relación $\nearrow_{\mathcal{H}}$. Cabe señalar que $C_{\mathcal{H}}$ incluye una categoría suprema definida $\mathsf{All}_{\mathcal{H}}$. □

Ejemplo 2.1 *El esquema de jerarquía de la dimensión Profesional, descrito en la Figura 1.1(a), es:*

$\mathcal{C}_{\mathcal{H}}$=*{Jugador, Equipo, Liga, País Residencia, Confederación, All}*,

$\nearrow$=*{(Jugador,Equipo), (Equipo,Liga), (Liga,Confederación), (Jugador,País Residencia), (País Residencia,Confederación), (Confederación,All)}*,

$\nearrow^*$=$\nearrow \cup$ *{(Jugador,Jugador), (Liga,Liga), (Equipo,Equipo), (Confederación,Confederación), (All,All), (País Residencia,País Residencia), ...}* □

Definición 2.2 (Dimensión (?)) La ***Dimensión*** $\mathcal{D}$ es una tupla $(\mathcal{H}_{\mathcal{D}}, \mathcal{E}_{\mathcal{D}}, \mathsf{Cat}_{\mathcal{D}}, <_{\mathcal{D}})$, donde $\mathcal{H}_{\mathcal{D}} = (\mathcal{C}_{\mathcal{H}_{\mathcal{D}}}, \nearrow_{\mathcal{H}_{\mathcal{D}}})$ es un esquema de jerarquía, $\mathcal{E}_{\mathcal{D}}$ son los elementos de la dimensión; $\mathsf{Cat}_{\mathcal{D}} : \mathcal{E}_{\mathcal{D}} \rightarrow \mathcal{C}_{\mathcal{H}_{\mathcal{D}}}$ es una función que determina la pertenencia de un elemento a su categoría, y la relación $<_{\mathcal{D}} \subseteq \mathcal{E}_{\mathcal{D}} \times \mathcal{E}_{\mathcal{D}}$ representan a las relaciones hijo/padre entre los elementos de diferentes categorías. $<^*_{\mathcal{D}}$ indica la clausura transitiva y reflexiva de $<_{\mathcal{D}}$. □

Ejemplo 2.2 *La definición de la dimensión Profesional (ver Figura 1.1(a)) es* $\mathcal{D} = (\mathcal{H}_{\mathcal{D}}, \mathcal{E}_{\mathcal{D}}, \mathsf{Cat}_{\mathcal{D}}, <_{\mathcal{D}})$*, donde:*

$\mathcal{H}_{\mathcal{D}}$ *se definió en el ejemplo 2.1,*

$\mathcal{E}_{\mathcal{D}}$*:* $\{j_1, j_2, e_1, e_2, l_1, l_2, r_1, r_2, c_1, c_2, all\}$,

$\mathsf{Cat}_{\mathcal{D}}$: $\{j_1 \rightarrow$*Jugador*, $j_2 \rightarrow$*Jugador*, $e_1 \rightarrow$*Equipo*, $e_2 \rightarrow$*Equipo*, $l_1 \rightarrow$*Liga*, $l_2 \rightarrow$*Liga*, $r_1 \rightarrow$*País Residencia*, $r_2 \rightarrow$*País Residencia*, $c_1 \rightarrow$*Confederación*, $c_2 \rightarrow$*Confederación*, $all \rightarrow$*All*$\}$,

$<_{\mathcal{D}}$: $\{(j_1, e_1), (j_2, e_2), (e_1, l_1), (e_2, l_2), (l_1, c_1), (l_2, c_2), (j_1, r_1), (j_2, r_2), (r_1, c_1), (r_2, c_2), (c_1, all), (c_2, all)\}$,

$<^*_{\mathcal{D}}$: $<_{\mathcal{D}} \cup \{(j_1, j_1), (j_1, l_1), (j_1, c_1), (j_1, all)...\}$ □

A continuación, se define el conjunto de relaciones rollup de la dimensión $\mathcal{D}$.

Definición 2.3 (Relaciones Rollup (?)) Una ***Relación Rollup*** entre dos categorías c_i y c_j denotada por $\mathcal{R}_{\mathcal{D}}(\mathsf{c}_i, \mathsf{c}_j)$ contiene el conjunto de pares $\{(a, b) | \mathsf{Cat}_{\mathcal{D}}(a) = \mathsf{c}_i, \mathsf{Cat}_{\mathcal{D}}(b) = \mathsf{c}_j$ y $a <^*_{\mathcal{D}} b\}$.

Dada una relación rollup $\mathcal{R}_{\mathcal{D}}(\mathsf{c}_i, \mathsf{c}_j)$:

- $\mathcal{R}_{\mathcal{D}}(\mathsf{c}_i, \mathsf{c}_j)$ es *estricta* si para todos los elementos x, y, z, si $(x, y) \in \mathcal{R}_{\mathcal{D}}(\mathsf{c}_i, \mathsf{c}_j)$ y $(x, z) \in \mathcal{R}_{\mathcal{D}}(\mathsf{c}_i, \mathsf{c}_j)$ entonces $y = z$.
- $\mathcal{R}_{\mathcal{D}}(\mathsf{c}_i, \mathsf{c}_j)$ es *homogénea* si para todos los elementos e tal que $\mathsf{Cat}(e) = \mathsf{c}_i$, existe un elemento e' tal que $\mathsf{Cat}(e') = \mathsf{c}_j$ y $(e, e') \in \mathcal{R}_{\mathcal{D}}(\mathsf{c}_i, \mathsf{c}_j)$. □

Verificar si las relaciones rollup de una dimensión $\mathcal{D}$ son estrictas y homogéneas se puede realizar computacionalmente en tiempo polinomial (**?**).

Las condiciones sobre las relaciones rollups se expresan en términos de las *restricciones de integridad* que se definen a continuación.

Definición 2.4 (Restricciones Estrictas y Homogéneas (?)) Sea $\mathcal{H} = (\mathcal{C}_{\mathcal{H}}, \nearrow_{\mathcal{H}})$ un esquema de jerarquía y $\mathcal{D} = (\mathcal{H}_{\mathcal{D}}, \mathcal{E}_{\mathcal{D}}, \mathsf{Cat}_{\mathcal{D}}, <_{\mathcal{D}})$ una dimensión, tal que: $\mathcal{H}_{\mathcal{D}} = \mathcal{H}$.

- Una *restricción estricta* sobre $\mathcal{H}$ es una expresión de la forma $c_i \rightarrow c_j$ donde c_i, $c_j \in \mathcal{C}_{\mathcal{H}}$ y $c_i \nearrow^*_{\mathcal{H}} c_j$. La dimensión $\mathcal{D}$ satisface la restricción $c_i \rightarrow c_j$ sí y solo sí la relación rollup $\mathcal{R}_{\mathcal{D}}(c_i, c_j)$ es estricta. El conjunto de restricciones estrictas se denota con $\Sigma_e(\mathcal{H})$.
- Una *restricción homogénea* sobre $\mathcal{H}$ es una expresión de la forma $c_i \Rightarrow c_j$ donde c_i, $c_j \in \mathcal{C}_{\mathcal{H}}$ y $c_i \nearrow^*_{\mathcal{H}} c_j$. La dimensión $\mathcal{D}$ satisface la restricción $c_i \Rightarrow c_j$ sí y solo sí la relación rollup $\mathcal{R}_{\mathcal{D}}(c_i, c_j)$ es homogénea. El conjunto de restricciones homogéneas se denota con $\Sigma_h(\mathcal{H})$. □

Una dimensión $\mathcal{D}$ es **consistente** si satisface el conjunto definido de restricciones de integridad $\Sigma = \Sigma_e(\mathcal{H}) \cup \Sigma_h(\mathcal{H})$, en caso contrario es una dimensión **inconsistente** (**?**).

Las restricciones de integridad se usan para comprobar si la agregación de datos es correcta (**?**) y consistente, es decir, si cada hecho se agrega una sola vez.

Ejemplo 2.3 *Para la dimensión de la Figura 1.1(a) el conjunto de restricciones de integridad* Σ *es:* φ_1 : *Jugador* $\Rightarrow$ *All*, φ_2 : *Equipo* $\Rightarrow$ *All*, φ_3 : *Liga* $\Rightarrow$ *All*, φ_4 : *Confederación* $\Rightarrow$ *All*, φ_5 : *País Residencia* $\Rightarrow$ *All*, *y* φ_6 : *Jugador* $\rightarrow$ *Confederación*.

La dimensión Profesional de la Figura 1.1(b) es consistente con respecto a Σ *ya que todo elemento que pertenece a Jugador está relacionado con all que es el único elemento de All* (φ_1)*, al igual que Equipo* (φ_2)*, Liga* (φ_3)*, Confederación* (φ_4) *y País Residencia* (φ_6)*. Adicionalmente, cada elemento de Jugador está relacionado con un único elemento en Confederación.* □

A continuación se define el operador de actualización reclasificación, que puede volver una dimensión inconsistente.

Definición 2.5 (Reclasificación (?)) Sea una dimensión estricta y homogénea $\mathcal{D} = (\mathcal{H}_\mathcal{D}, \mathcal{E}_\mathcal{D}, \mathsf{Cat}_\mathcal{D}, <_\mathcal{D})$, un par de categorías c_i y c_j tal que c_i se relacione con c_j, un par de elementos a, b con $\mathsf{Cat}(\mathsf{a}) = \mathsf{c}_i$ y $\mathsf{Cat}(\mathsf{b}) = \mathsf{c}_j$. La *Reclasificación*$(\mathcal{D}, \mathsf{c}_i, \mathsf{c}_j, \mathsf{a}, \mathsf{b})$ retorna una nueva dimensión $\mathcal{D}_u = (\mathcal{H}_\mathcal{D}, \mathcal{E}_\mathcal{D}, \mathsf{Cat}_\mathcal{D}, <'_\mathcal{D})$ y $<'_\mathcal{D} = (<_\mathcal{D} \setminus \{(\mathsf{a}, \mathsf{c}) \mid (\mathsf{a}, \mathsf{c}) \in <_\mathcal{D} \text{ y } \mathsf{Cat}(c) = \mathsf{c}_j\}) \cup \{(\mathsf{a},\mathsf{b})\}$. □

En otras palabras, una operación de reclasificación permite un cambio en la relación rollup entre dos categorías, tal que para un elemento en una categoría c_i, su padre en la categoría c_j es cambiado por otro elemento en la misma categoría. Esto mediante la inserción y eliminación de arcos.

La reclasificación mostrada en el ejemplo 1.2 aplicada a la dimensión de la Figura 1.2(a) deja a la dimensión inconsistente con respecto a Σ, ya que el elemento j_2 (que pertenece a la categoría Jugador) está relacionado con dos elementos en Confederación (c_1 y c_2). La restricción específica que se viola es la restricción estricta φ_6.

Cuando una dimensión contiene un nivel de conflicto, una reclasificación puede dejar la dimensión inconsistente con respecto a Σ.

Definición 2.6 (Nivel de Conflicto (?)) *Sea una dimensión* $\mathcal{D} = (\mathcal{H}_\mathcal{D}, \mathcal{E}_\mathcal{D}, \mathsf{Cat}_\mathcal{D}, <_\mathcal{D})$*, un par de categorías* c_i *y* c_j *tal que* c_i *se relaciona con* c_j*, un par de elementos* a*,* b *con* $\mathsf{Cat}(\mathsf{a}) = \mathsf{c}_i$ *y* $\mathsf{Cat}(\mathsf{b}) = \mathsf{c}_j$*. Una categoría* c_k *tal que* c_j *se relacione con* c_k*, es* un nivel de conflicto *respecto a la reclasificación, si hay una categoría* c_m *tal que* c_i *se relacione con* c_m*, habiendo una ruta alternativa entre* c_k *y* c_m *que no incluye el arco* $(\mathsf{c}_i,\mathsf{c}_j)$*, y si* a *alcanza un elemento en* c_m*.* □

En el esquema jerárquico de profesional mostrado en la Figura 1.1(a), el nivel de conflicto es la categoría Confederación.

2.2. Reparación de Dimensiones Inconsistentes

Cuando una dimensión es inconsistente, ésta debe ser reparada (**?**). Para definir el concepto de reparación es necesario formalizar el concepto de distancia entre dos dimensiones:

Definición 2.7 (Distancia (?)) *Sean* $\mathcal{D} = (\mathcal{H}_\mathcal{D}, \mathcal{E}_\mathcal{D}, \mathsf{Cat}_\mathcal{D}, <_\mathcal{D})$ *y* $\mathcal{D}' = (\mathcal{H}_{\mathcal{D}'}, \mathcal{E}_{\mathcal{D}'}, \mathsf{Cat}_{\mathcal{D}'}, <_{\mathcal{D}'})$ *dos dimensiones sobre un mismo esquema de jerarquía* $\mathcal{H}_\mathcal{D}$*.*

La distancia entre $\mathcal{D}$ *y* $\mathcal{D}'$ *es definida como* $dist(\mathcal{D}, \mathcal{D}') = |(<_{\mathcal{D}'} \setminus <_\mathcal{D}) \cup (<_\mathcal{D} \setminus <_{\mathcal{D}'})|$. □

Es decir, la distancia $dist(D, D')$ es el tamaño de la diferencia simétrica entre las relaciones hijo/padre, entre elementos de caegorías. Este concepto se establece para poder determinar como se diferencian dos dimensiones.

Definición 2.8 (Reparación y Reparación Minimal (?)) *Sea* $\mathcal{D} = (\mathcal{H}_\mathcal{D}, \mathcal{E}_\mathcal{D}, \mathsf{Cat}_\mathcal{D}, <_\mathcal{D})$ *una dimensión y* Σ *un conjunto de restricciones de integridad, estrictas y homogéneas, en* $\mathcal{H}_\mathcal{D}$*.*

Dimensión de Destino	Arcos Eliminados	Arcos Insertados	Distancia a D
$\mathcal{D}_1$	$(l2, c2)$	$(l2, c1)$	2
$\mathcal{D}_2$	$(e2, l2)$	$(e2, l1)$	2
$\mathcal{D}_3$	$(j2, e2)$	$(j2, e1)$	2
$\mathcal{D}_4$	$(r2, c1)$	$(r2, c2)$	2
$\mathcal{D}_5$	$(l2, c2), (e1, l1), (e2, l2)$	$(l2, c1), (e1, l2), (e2, l1)$	6

Tabla 2.1: Inserción y eliminación de arcos de $\mathcal{D}$

- *Una* **reparación** *de $\mathcal{D}$ respecto a Σ es una dimensión $\mathcal{D}' = (\mathcal{H}_{\mathcal{D}'}, \mathcal{E}_{\mathcal{D}'}, \mathsf{Cat}_{\mathcal{D}'}, <_{\mathcal{D}'})$ tal que $\mathcal{H}_{\mathcal{D}'} = \mathcal{H}_{\mathcal{D}}$, $\mathcal{E}_{\mathcal{D}'} = \mathcal{E}_{\mathcal{D}}$, $\mathsf{Cat}_{\mathcal{D}'} = \mathsf{Cat}_{\mathcal{D}}$, y $\mathcal{D}'$ satisface Σ.*
- *Una* **reparación minimal** *de $\mathcal{D}$ con respecto a Σ es una dimensión $\mathcal{D}'$, tal que la dist$(\mathcal{D}, \mathcal{D}')$ es mínima entre todas las reparaciones de $\mathcal{D}$ respecto a Σ.* □

Dicho de otra forma, una reparación es una nueva Dimensión $\mathcal{D}'$ con los mismos elementos y categorías de la dimensión original, pero que satisface por completo el conjunto de restricciones de integridad Σ. Cabe señalar que las restricciones prevalecen sobre los datos y la dimensión debe ser actualizada para satisfacer las restricciones.

Ejemplo 2.4 *La dimensión Profesional de la Figura 1.2(b) no satisface el conjunto de restricciones de integridad Σ, ya que se viola la restricción φ_6, debido a que j_2 va a dos elementos diferentes en* Confederación*. Algunas reparaciones de esta dimensión se muestran en la Figura 2.1. La tabla 2.1 muestra el detalle de la inserción y eliminación de arcos realizados sobre la dimensión original y la distancia de las reparaciones a ésta.*

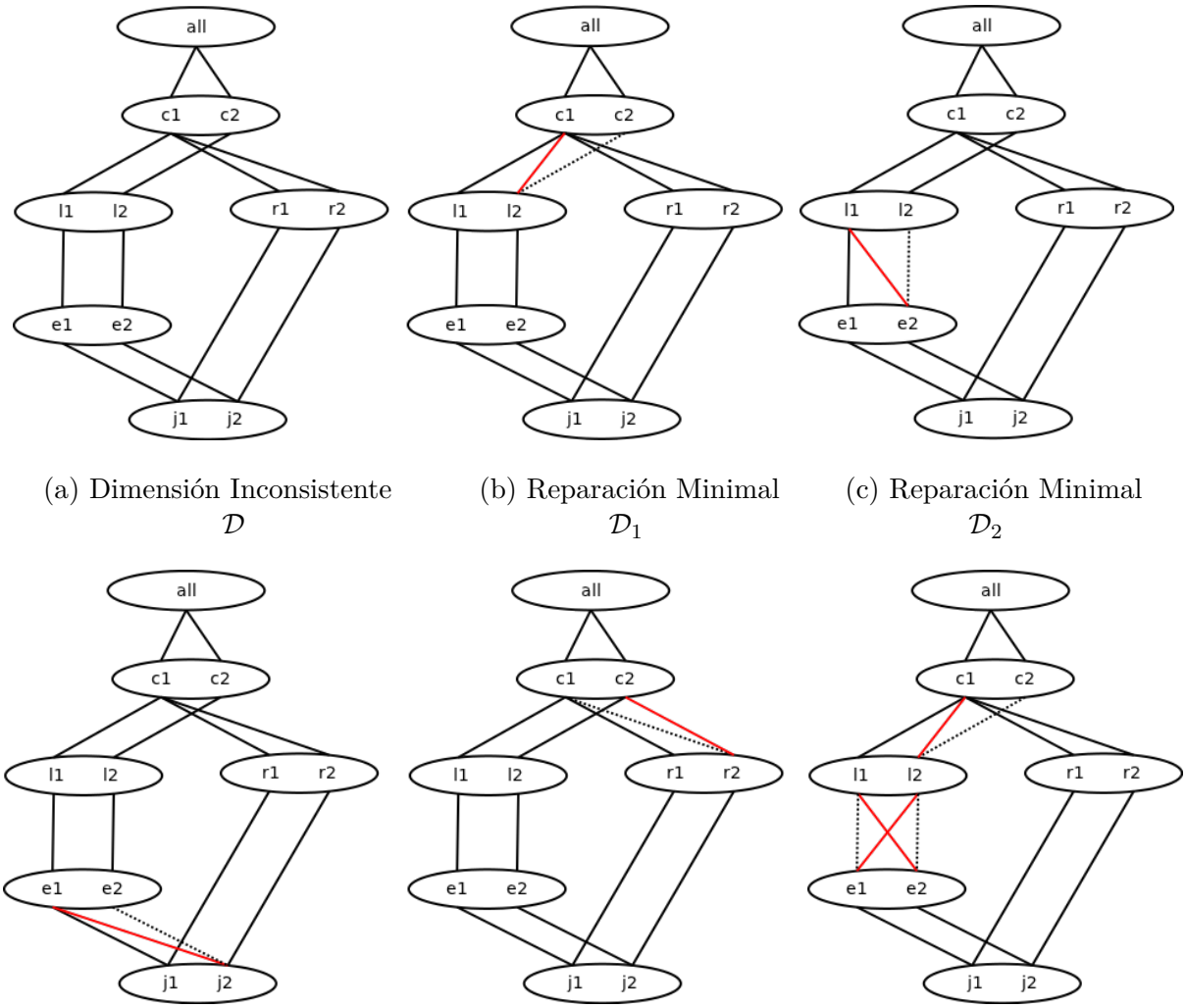
Existen cuatro **reparaciones minimales** *para $\mathcal{D}$ respecto a Σ, las cuales son $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$ y $\mathcal{D}_4$, ya que estas nuevas dimensiones presentan la menor distancia respecto a la dimensión original.* □

En general, computar reparaciones minimales de una dimensión $\mathcal{D}$ con respecto a Σ, es NP-completo (**?**). Sin embargo en (**?**) se ha mostrado que computar una r-reparación para dimensiones que tienen un nivel de conflicto se puede hacer en tiempo polinomial. Una r-reparación es una reparación que contiene las operaciones de reclasificación que producen la inconsistencia.

Definición 2.9 (R-reparación (?)) *Sea $\mathcal{D} = (\mathcal{H}_{\mathcal{D}}, \mathcal{E}_{\mathcal{D}}, \mathsf{Cat}_{\mathcal{D}}, <_{\mathcal{D}})$ una dimensión, Σ un conjunto de restricciones de integridad en $\mathcal{H}_{\mathcal{D}}$ y un conjunto de reclasificaciones R. Una r-reparación para $\mathcal{D}$ sobre R es una dimensión $\mathcal{D}' = (\mathcal{H}_{\mathcal{D}'}, \mathcal{E}_{\mathcal{D}'}, \mathsf{Cat}_{\mathcal{D}'}, <_{\mathcal{D}'})$ tal que:*

- *$\mathcal{D}'$ satisface Σ*
- *$\mathcal{H}_{\mathcal{D}'} = \mathcal{H}_{\mathcal{D}}$*
- *Para cada Reclasificación$(\mathcal{D}, c_i, c_j, \mathsf{a}, \mathsf{b}) \in R$, el par $(\mathsf{a},\mathsf{b}) \in <^*_{\mathcal{D}'}$*
- *Una* ***r-reparación*** *$\mathcal{D}'$ es minimal si la distancia dist$(\mathcal{D}, \mathcal{D}')$ es mínima entre todas las r-reparaciones para $\mathcal{D}$.*

R-reparación$(\mathcal{D}, R)$ denota el conjunto de r-reparaciones para $\mathcal{D}$ sobre R. □



(a) Dimensión Inconsistente $\mathcal{D}$ (b) Reparación Minimal $\mathcal{D}_1$ (c) Reparación Minimal $\mathcal{D}_2$

(a) Reparación Minimal $\mathcal{D}_3$ (b) Reparación Minimal $\mathcal{D}_4$ (c) Reparación No Minimal $\mathcal{D}_5$

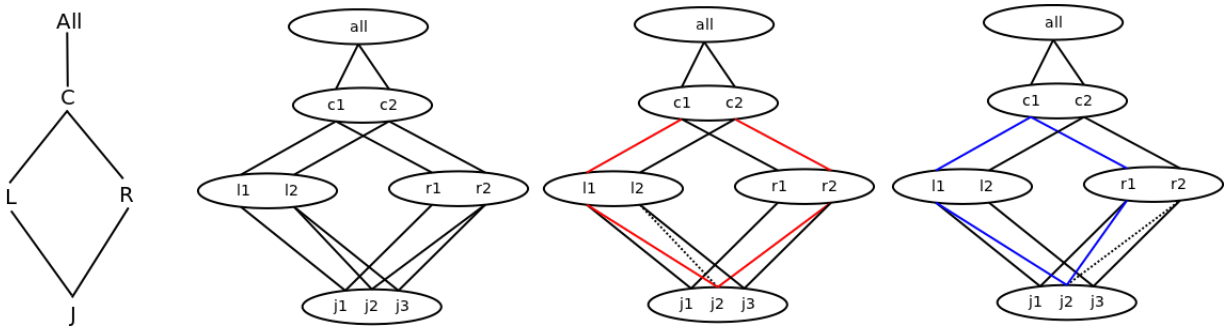
Figura 2.1: Reparaciones para la dimensión de la Figura 1.2(b)

Ejemplo 2.5 *Según la dimensión mostrada en la Figura 2.1 el conjunto r-reparación=*$(\mathcal{D}, R)$ *está formado por las dimensiones* $\mathcal{D}_1$*,* $\mathcal{D}_2$*,* $\mathcal{D}_3$ *y* $\mathcal{D}_5$*, ya que fueron obtenidas sin deshacer la Reclasificación*$(\mathcal{D}, \textsf{País Residencia}, \textsf{Confederación}, r2, c1)$*. Solo las r-reparaciones* $\mathcal{D}_1$*,* $\mathcal{D}_2$ *y* $\mathcal{D}_3$ *son minimales.* □

En esta tesis se considera el caso de computar una r-reparación para una dimensión que fue actualizada mediante una única reclasificación, es decir, solo la relación rollup de un elemento de una categoría es modificado. Para poder computar una r-reparación es necesario analizar dos heurísticas planteadas en (**?**), las cuales serán revisadas a continuación mediante ejemplos.

- ***No generar nuevas inconsistencias en el proceso de reparación.***

  Ejemplo 2.6 *Sea el esquema jerárquico de la Figura 2.2(a), la dimensión de la Figura 2.2(b), la dimensión reclasificada $\mathcal{D}$ de la Figura 2.2(c) y el conjunto de restricciones de integridad Σ compuesto por $\varphi_1 : J \Rightarrow All$, $\varphi_2 : L \Rightarrow All$, $\varphi_3 : R \Rightarrow All$, $\varphi_4 : C \Rightarrow All$ y la restricción estricta $\varphi_5 : J \rightarrow C$.*



(a) Esquema Jerárquico (b) Dimensión Original (c) Reclasificación(D, J, L, j_2, l_1) (d) r-reparación

Figura 2.2: R-reparación: Heurística 1

En primer lugar se necesita obtener los elementos que participan en violaciones de la restricción de integridad estricta φ_5. En este caso el elemento de la categoría inferior que participa en una violación es j_2 y conforma los siguientes caminos desde el nivel inferior del esquema jerárquico:

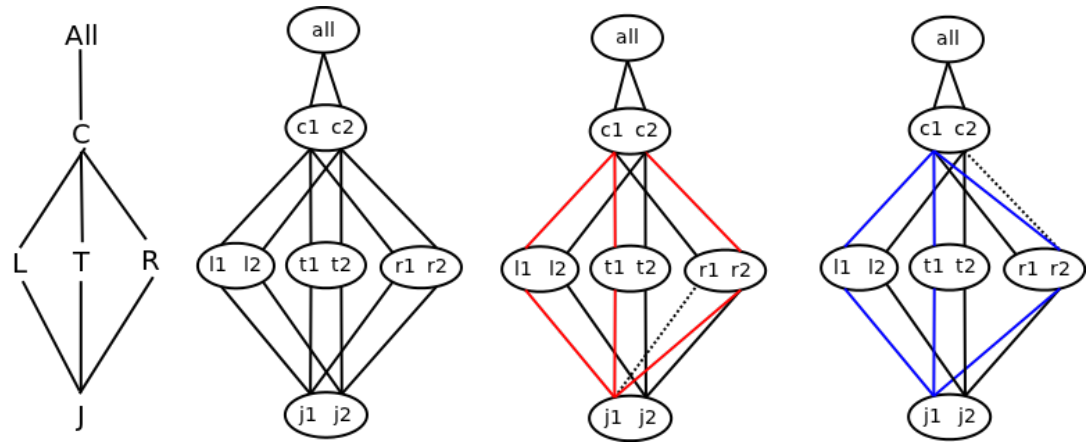
- j_2 - l_1 - c_1
- j_2 - r_2 - c_2

Luego, una operación de reparación válida es eliminar el arco (j_2, r_2) e insertar el arco (j_2, r_1) (ver Figura 2.2(d)), ya que este cambio no origina nuevas inconsistencias. Eliminar el arco (l_1, c_1) e insertar el arco (l_1, c_2), soluciona la inconsistencia respecto al elemento j_2, pero produce una nueva inconsistencia a través del elemento j_1, por lo tanto este cambio no es una operación válida de reparación. La dimensión en la Figura 2.2(d) es una r-reparación de $\mathcal{D}$. □

- ***En lo posible, cuando se elije una operación de reparación elegir aquella que produzca menos cambios en la dimensión.***

  Ejemplo 2.7 *Sea la dimensión de la Figura 2.3(b) y el siguiente conjunto de restricciones de integridad Σ: $\varphi_1 : J \Rightarrow All$, $\varphi_2 : L \Rightarrow All$, $\varphi_3 : T \Rightarrow All$, $\varphi_4 : R \Rightarrow All$, $\varphi_5 : C \Rightarrow All$, y $\varphi_6 : J \rightarrow C$.*
  El resultado de aplicar la operación Reclasificación(D, J, R, j_1, r_2) en la dimensión de la Figura 2.3(b) deja la dimensión inconsistente con respecto a la restricción φ_6 a través del elemento j_1, tal como se aprecia en la Figura 2.3(c). El esquema jerárquico de esta dimensión contiene el nivel de conflicto C.

Para poder elegir una operación válida de reparación que involucre pocos cambios es necesario saber cuantos caminos desde j_1 participan en la violación de φ_6 y cuáles son los elementos en C a los cuales estos caminos llegan (c_1 y c_2). Luego, como el número de caminos que alcanzan a c_1 es dos y el número de caminos que llegan a c_2 es uno, entonces, la operación válida de reparación de acuerdo a esta heurística es aquella que modifica la relación rollup de los elementos que alcanzan c_2. En este sentido se obtiene la r-reparación en la Figura 2.3(d). Note que esta operación de cambio es válida porque no viola la heurística anterior.



(a) Esquema Jerárquico (b) Dimensión Original (c) Reclasificación (D, J, R, j_1, r_2) (d) R-reparación

Figura 2.3: R-reparación: Heurística 2

□

Capítulo 3

Propuesta de Tesis

3.1. Hipótesis

Es posible desarrollar algoritmos que se resuelvan en tiempo polinomial para obtener reparaciones de dimensiones que se vuelven inconsistentes con respecto a un conjunto de restricciones de integridad estrictas y homogéneas posterior a una actualización.

3.2. Objetivos

3.2.1. Objetivo General

El Objetivo de esta Tesis es desarrollar una solución algorítmica que permita obtener reparaciones de dimensiones que se vuelven inconsistentes con respecto a un conjunto de restricciones de integridad estrictas y homogéneas posterior a una actualización. Además es un objetivo implementar un proceso de reparación que permita al administrador del DW obtener reparaciones con sentido semántico, a través de la incorporación grados de prioridad en las relaciones rollup y restricciones de seguridad (que evitan cambios en relaciones rollup).

3.2.2. Objetivos Específicos

- Definir el tipo de esquemas jerárquicos a considerar.
- Analizar distintos tipos de heurísticas para reparar DWs con respecto a restricciones de integridad estrictas y homogéneas.
- Redefinir el procedimiento de reparación, de tal forma que considere restricciones de rollup seguros y prioridades entre rollups.
- Analizar restricciones del DW que guíen el proceso de reparación (rollups con prioridades y rollups seguros).
- Estudiar y mejorar las heurísticas presentadas en estudios anteriores que permiten reparar dimensiones.
- Estudiar la complejidad de los algoritmos que permiten obtener reparaciones.
- Mostrar la eficacia y eficiencia de los algoritmos mediante experimentos.

3.3. Alcance de la investigación

Los algoritmos implementados en esta tesis consideran dimensiones de DWs que cuenten con un nivel de conflicto y que no satisfacen un conjunto de restricciones de integridad estrictas y homogéneas después de una reclasificación de elementos. Además, se considera la definición de grados de prioridad en las relaciones rollup y restricciones de seguridad.

3.4. Estado del Arte

3.4.1. Inconsistencias en DWs

Los DWs se originan como colecciones de vistas materializadas de los datos extraídos desde bases de datos operacionales. Existen trabajos que han resuelto problemas de inconsistencia entre las fuentes de datos operacionales y los DWs (**??????**).

Existen artículos que han estudiado los datos imprecisos (**??**) en las dimensiones. En (**?**), el modelo de datos multidimensional busca soportar la ambigüedad de los datos, la imprecisión y la incertidumbre.

Hay pocos trabajos que buscan resolver las inconsistencias que se generan al no satisfacer las restricciones de integridad sobre las dimensiones de un DW. En las primeras investigaciones sobre DWs, se consideró que las dimensiones eran la parte estática de los DWs y los hechos eran la parte dinámica. Sin embargo, en (**??**) se muestra que las dimensiones deben adaptarse a cambios en las fuentes de datos. Cuando los DWs se actualizan, las dimensiones pueden volverse inconsistentes con respecto a sus restricciones de integridad estrictas y homogéneas. Además, las inconsistencias de datos pueden ser causada por la disparidad entre las diferentes fuentes de datos que alimentan el DW, o por datos erróneos.

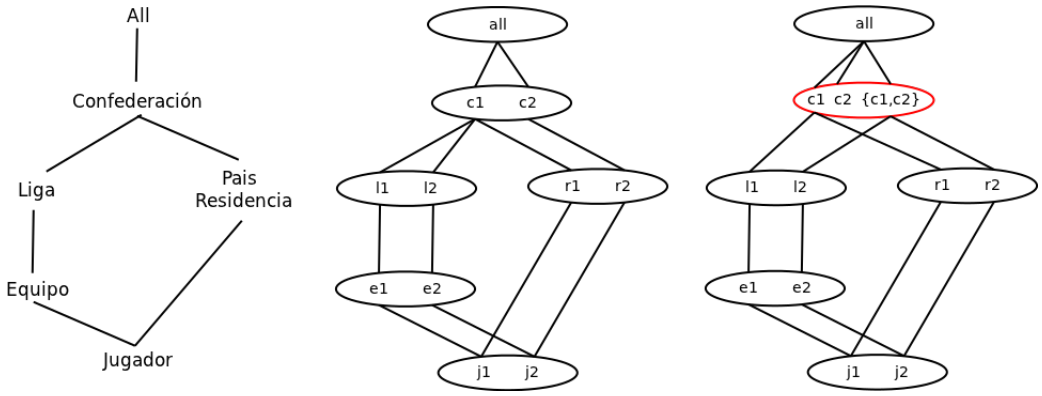
Letz (**?**) muestra que es importante utilizar las restricciones de integridad para guiar las operaciones de actualización, y de esta forma no realizar cambios que puedan producir inconsistencia.

Pedersen (**?**) fue uno de los primeros en presentar un método para transformar las dimensiones no estrictas en dimensiones estrictas y para ello propone un método que inserta nuevos elementos artificiales en las categorías. Ilustramos el método propuesto en el siguiente ejemplo.

Ejemplo 3.1 *Sea la dimensión Profesional definida en el ejemplo 1.1 (ver Figura 3.1(a)).* j_2 *es el elemento inconsistente ya que está relacionado transitivamente en* Confederacion *con* c_1 *y* c_2 *(ver Figura 3.1(b)), violando la restricción* φ_6 : Jugador $\rightarrow$ Confederación. *Según Pedersen (**?**) esta violación se corrige agregando el elemento artificial* $\{c_1, c_2\}$ *en la categoría* Confederación *y luego asociando* l_2 *y* r_2 *con este elemento, tal como se muestra en la dimensión de la Figura 3.1(c).*

□

Sin embargo, este método es recomendado cuando los datos son correctos pero no se acomodan a la restricción estricta, por ejemplo un río que está en el límite de dos países podría estar asociado a dos elementos en una categoría país, en este caso la solución no es reparar en el sentido propuesto en (**?**), sino transformar los datos para cumplir con la restricción de integridad. En el ejemplo



(a) Esquema Jerárquico (b) Dimensión Inconsistente (c) Dimensión Reparada

Figura 3.1: Reparación utilizando lo propuesto por Pedersen

sería agregar un elemento compuesto con el nombre de los dos países a los que el río está asociado en la categoría país. El método de reparación seguido en esta tesis ayuda a limpiar los errores de los datos de tal manera de devolver la consistentencia.

3.4.2. Inconsistencias en bases de datos relacionales

El problema de reparación en bases de datos relacionales con respecto a un conjunto de restricciones de integridad (por ejemplo, dependencias funcionales y dependencias de inclusión) ha sido ampliamente estudiado en (**?????**).

Si bien existen muchas formas de poder representar un DW usando modelos relacionales (esquema en estrella o esquema copo de nieve (**??**)), no es posible usar las técnicas de reparación para bases de datos relacionales para calcular las reparaciones de DWs. Esto se debe a que en el enfoque relacional, la inconsistencia de la base de datos se soluciona mediante la inserción, eliminación y actualización de tuplas y esto no coincide con el enfoque utilizado en DWs que modifica arcos entre elementos. En (**?**) se muestran varios ejemplos donde se puede concluir que es imposible codificar una dimensión multidimensional dentro de un esquema relacional de DW y obtener las mismas reparaciones minimales mostradas en esta tesis utilizando técnicas relacionales.

Una diferencia importante con respecto a las reparaciones relacionales es que en los DWs se utiliza una semántica de reparación basada en cardinalidad (**??**), es decir, reducir al mínimo el número de cambios realizados en una dimensión, mientras que en el modelo relacional además de basarse en la cardinalidad, se puede reparar reduciendo al mínimo el conjunto de tuplas insertadas y eliminadas.

3.4.3. Métodos para computar reparaciones

Se han propuestos varios métodos para computar reparaciones en bases de Datos relacionales ((**?**) realiza un buen análisis de estos métodos). Los métodos más generales son aquellos basa-

dos en programas lógicos con semántica de modelos estables ((**??**)), que son representaciones compactas de las reparaciones de bases de datos relacionales con respecto a sus restricciones de integridad (**???**). También se han implementado los programas en lógica para reparar sistemas de integración de datos ((**??**)).

En (**?**) se presentan programas en lógica para computar reparaciones minimales de dimensiones con respecto a restricciones estrictas y homogéneas. En este artículo se demuestra además que el número de reparaciones para una dimensión inconsistente puede ser exponencial y poder computar las reparaciones minimales para una dimensión es, en general NP-completo. Sin embargo para un particular tipo de dimensiones, dimensiones lineales, este proceso puede hacerse en tiempo polinomial. Cabe señalar que el desempeño de programas en lógica resulta ineficiente, en términos de tiempo. Por lo tanto, es relevante buscar casos particulares que puedan ser resueltos con algoritmos polinomiales como los presentados en esta tesis.

En (**?**) se muestra que computar una *r-reparación* para un determinado tipo de dimensiones puede realizarse en tiempo polinomial, siempre y cuando sea una dimensión con un nivel de conflicto, y que además se vuelve inconsistente luego de una reclasificación de arcos. Cabe señalar que las *r-reparaciones* son la base de este proyecto.

En la mayoría de las investigaciones y aplicaciones industriales OLAP, las dimensiones se diseñan para satisfacer todas las restricciones de integridad estrictas y homogéneas definidas en el esquema jerárquico. Sin embargo las dimensiones podrían no satisfacer estas restricciones (**??????**). Cuando esto ocurre es necesario especificar las restricciones de integridad para identificar los rollup que son estrictos u homogéneos, y de esta manera poder mantener la capacidad de responder correctamente a consultas de agregación (**?**). En los principales sistemas de gestión de bases de datos (SGBD) existentes en el mercado, las dimensiones se modelan en base de datos relacionales, principalmente optando por el esquema estrella o copo de nieve, el principal problema que posee este tipo de técnicas es que es muy difícil mantener y evaluar la consistencia de la dimensión, puesto que requiere de muchas restricciones en las tablas. Además, es muy probable que solucionar una inconsistencia requiera realizar un cambio que elimine muchos más arcos de la dimensión que hacerlo directamente en ella, puesto que por ejemplo en el esquema estrella, requiere eliminar una tupla de la dimensión.

3.5. Metodología

Las siguientes son las actividades a realizar en esta investigación en función de los objetivos específicos planteados anteriormente:

- Analizar los tipos de dimensiones que pueden ser reparadas en tiempo polinomial junto con sus restricciones de integridad.
- Analizar restricciones impuestas por el administrador del DW que permitan guiar el proceso de reparación.
- Analizar los distintos tipos de heurísticas para la reparación de dimensiones planteadas en la literatura.
- Implementar los algoritmos en lenguaje C.
- Analizar la complejidad de los algoritmos.
- Finalmente, realizar distintos experimentos que demuestren la efectividad de los algoritmos planteados.

Capítulo 4

Algoritmos de Reparación

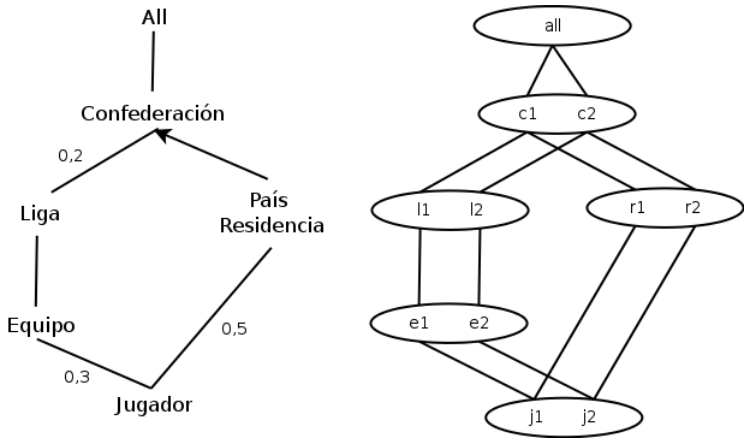
En esta sección se presentan los algoritmos implementados para computar r-reparaciones para dimensiones que poseen un nivel de conflicto con respecto a restricciones de integridad estrictas. Además, estos algoritmos implementan el proceso de reparación guiados por restricciones impuestas por el administrador del DW, que son las siguientes:

- ***Grados de prioridad:*** Un grado de prioridad es una constante real que oscila entre 0 y 1, la cual indica el grado de confianza que existe entre los vínculos directos entre dos categorías. Mientras la prioridad sea más cercana a 0 es menos confiable es la relación que existe entre las categorías involucradas, y por lo tanto, se preferirá realizar cambios entre los elementos de estas categorías para restaurar consistencia; contrariamente es el caso en que la prioridad sea más cercana al valor 1.

  Ejemplo 4.1 *Sea el esquema jerárquico de la Figura 4.1(a) y la dimensión mostrada en la Figura 4.1(b). El administrador del Data Warehouse ha definido un grado de prioridad a las relaciones rollup entre categorías, tal que la prioridad entre* Jugador *y* Equipo *es de* $0,3$*, entre* Equipo *y* Liga *es de* $0,7$*, luego entre* Liga *y* Confederación *es de* $0,2$*, y por último, entre* Jugador *y* País Residencia *es de* $0,5$*.* □*0*

  Se asume que los grados de prioridad son entregados por el administrador del DW previo a ejecutar un proceso de reparación y no son parte del esquema jerárquico de una dimensión.
- ***Restricciones Seguras:*** Una restricción segura señala que el vínculo directo entre dos categorías no se debe modificar, es decir, que ninguna relación rollup que involucre a elementos de estas categorías puede alterarse aunque sea necesario hacer cambios para lograr la consistencia de una dimensión.

  Ejemplo 4.2 *Sea el esquema jerárquico (Figura 4.1(a)) y la dimensión mostrada en la Figura 4.1(b). El administrador del Data Warehouse ha definido algunas restricciones seguras indicadas en el diseño de la dimensión, las cuales se modelan mediante una flecha unidireccional entre la categoría inferior y la superior.*
  En este caso, el Administrador del Data Warehouse ha indicado que el rollup entre País Residencia *y* Confederación *no se puede modificar.* □



(a) Esquema Jerárquico (b) Dimensión Profesional

Figura 4.1: Dimensión Profesional con restricciones impuestas por el administrador

4.1. Algoritmos de Reparación con Restricciones de Integridad

4.1.1. Descripción

Los algoritmos presentados en esta sección producen una r-reparación luego de que el administrador del Data Warehouse ha hecho una reclasificación que ha dejado la dimensión inconsistente con respecto a sus restricciones de integridad estrictas. Se asume en esta tesis, que la dimensión nunca viola una restricción homogénea.

Los algoritmos implementan las heurísticas detalladas en (**?**) que son:

(a) No generar nuevas inconsistencias en el proceso de reparación.

(b) En lo posible, cuando se elije una operación de reparación elegir aquella que produzca menos cambios en la dimensión.

El algoritmo 1 primero obtiene los elementos que participan en inconsistencias de restricciones de integridad estrictas y los almacena en una lista dinámica bidimensional enlazada por punteros, para optimizar el uso de memoria. En la lista se almacena para cada elemento inconsistente, los caminos desde la categoría inferior de la restricción estricta a la categoría nivel de conflicto . Los campos de esta estructura son:

- valor: es un elemento de una categoría.
- cat: es el nombre de una categoría.

A continuación se indica cómo se definió dicha estructura:

```
struct Lista{
    char* valor;
    char* cat;
    Lista *ant, *sgte;
}

struct Nodo{
    Lista raiz;
    Lista *sgte;
}
```

Almacenamiento de Caminos Inconsistentes

Para almacenar los caminos de los elementos inconsistentes, es necesario identificar las siguientes dos categorías:

- **cat_bottom**: Esta categoría es la categoría conflictiva inferior involucrada en la restricción de integridad (línea 2 del algoritmo 1). En el caso del esquema de la Figura 4.2 cat_bottom es A.

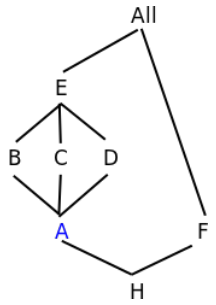


Figura 4.2: Ejemplo para obtener categoría cat_bottom y cat_cl

- **cat_cl**: Esta categoría es la categoría conflictiva superior de la restricción de integridad (línea 3 del algoritmo 1). En el caso de la Figura 4.2 cat_cl es E.

Para optimizar la estructura de almacenamiento en términos de memoria, se debe conocer cuántos elementos inconsistentes hay (línea 7 del algoritmo 1) y luego se debe llenar la lista de elementos inconsistentes (ver líneas 8 a la 23 del algoritmo 1).

Para una mayor comprensión de este algoritmo se presenta el siguiente ejemplo:

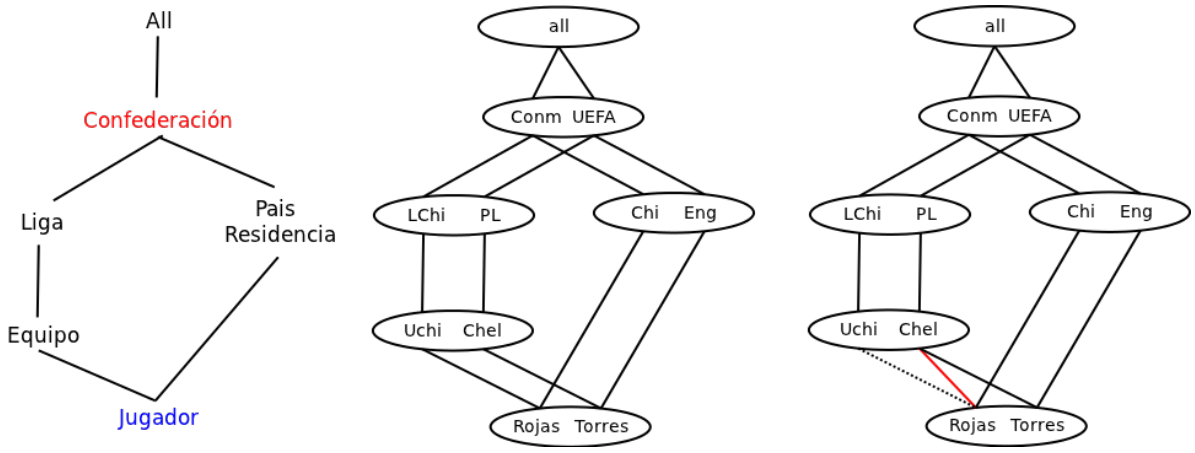
Ejemplo 4.3 *Consideremos la dimensión profesional con esquema jerárquico detallado en la Figura 4.3(a) y la restricción de integridad estricta φ: Jugador$\rightarrow$ Confederacion. Ha llegado el mercado de fichajes de invierno en Europa, y el Chelsea de Inglaterra ha decidido fichar a un jugador que actualmente milita en las filas de la Universidad de Chile llamado José Rojas. Dicho traspaso ha generado un cambio en la dimensión, y esa actualización ha quedado mostrada en Figura 4.3(c).*

La categoría cat_bottom es Jugador (línea 2 del algoritmo 1), y la categoría nivel de conflicto (cat_cl) que se obtiene al ejecutarse la línea 3 del algoritmo 1 es Confederación. Luego, en la línea

Algoritmo 1: GENERACIÓN DE LISTA DE CAMINOS INCONSISTENTES

Input: Nodo Lista de Categorías
Output: Nodo Lista de Caminos Inconsistentes

```
1  elementos, Caminos_Inconsistentes, *cat_bottom, *cat_cl, *query, *aux, *aux2, *elem, num_hijos, n, i = 1, j;
2  cat_bottom = cat_bottom();
3  cat_cl = cat_cl();
4  query = consulta_inconsistente();
5  num_hijos = obtener_num_hijos(cat_cl);
6  elementos = Ejecutar(query);
7  n = contar(elementos);
8  while i ≤ n do
9      j = 1;
10     elem = obtiene(elementos);
11     Insertar_Caminos_Inconsistentes(elem);
12     aux = cat_bottom;
13     while j ≤ num_hijos do
14         aux = obtiene(Lista_Categorias);
15         while aux ≠ cat_cl do
16             aux2 = obtiene_sgte(Lista_Categorias, aux);
17             elem = Ejecutar_Consulta_Padre(aux, aux2, elem);
18             Insertar_Caminos_Inconsistentes(elem);
19             aux = aux2;
20         j++;
21     elementos = elementos → sgte;
22     Lista_Categorias = Lista_Categorias → sgte;
23     i++;
```



(a) Dimensión Profesional Esquema Jerárquico

(b) Dimensión Profesional Original

(c)Dimensión Profesional Inconsistente respecto φ

Figura 4.3: Ilustración del Ejemplo

6 del algoritmo 1 se ejecuta la función Ejecutar(query), la que verifica si existe inconsistencia respecto de la restricción de integridad y de ser así obtiene los elementos involucrados en la inconsistencia. En este caso, se obtiene el elemento **Rojas** *de la categoría* Jugador *(ver Figura*

4.3(c)), luego entre las líneas 8 y 23 del algoritomo 1 se realiza el proceso de llenado de la lista, lo que consiste en la inserción de elementos y categorías en la **Lista de Caminos Inconsistentes**. *Es importante aquí conocer el número de caminos que existe entre la categoría bottom y la categoría nivel de conflicto, esta información se obtiene en la lína 5 del algoritmo. En este caso existen dos caminos para alcanzar* Confederación *y por lo tanto, la rutina entre la línea 13 y 20 del algoritmo 1 se efectúa 2 veces (1 por camino).*

En el ejemplo, en primer lugar se inserta el elemento **Rojas** *y la categoría* Jugador*, luego* **Che** *y la categoría* Equipo*, posteriormente* **PL** *y la categoría* Liga *y finalmente* **UEFA** *y la categoría* Confederación*. Luego se insertan los elementos* **Rojas** *y la categoría* Jugador*,* **Chi** *y la categoría* **País Residencia** *y* **Conm** *y la categoría* Confederación*. El resultado final del proceso de inserción, se detalla en la lista de la Figura 4.4.*

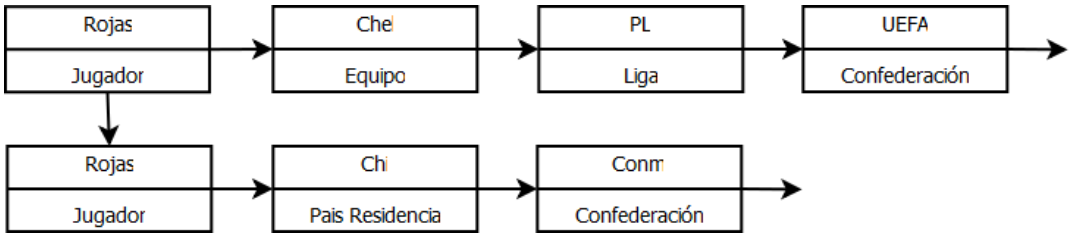


Figura 4.4: Implementación de Algoritmo 1 según Ejemplo 4.3

□

Una vez generado los caminos de los elementos inconsistentes, se procede a ejecutar el algoritmo 2.

Reparación

El algoritmo 2 para cada elemento involucrado en inconsistencia escoge una vía de reparación, es decir, elige la ruta desde cat_bottom hacia cat_cl que será modificada para restaurar consistencia. Para ello es necesario obtener algunos datos que sirven como base para el análisis de las heurísticas que permiten encontrar una *r-reparación*:

- npadre: es el elemento padre en la categoría cat_cl del elemento involucrado en la inconsistencia luego de la reclasificación. Se obtiene analizando la reclasificación realizada previamente y que dejo al DW inconsistente y la lista de elementos inconsistentes (ver línea 2 del algoritmo 2).
- opadre: es el padre original en la categoría cat_cl del elemento involucrado en la inconsistencia, es decir el que tenía el elemento inconsistente previo a la reclasificación. Se obtiene analizando la lista de elementos inconsitentes y la varialbe npadre (ver línea 3 del algoritmo 2).
- camino_opadre: contiene la cantidad de caminos que llegan desde un elemento inconsistente al elemento padre original en la categoría cat_cl (ver línea 4 del algoritmo 2).
- camino_npadre: contiene la cantidad de caminos que llegan desde un elemento inconsistente al elemento padre afectado por la actualización (ver línea 5 del algoritmo 2).

Luego, por cada elemento inconsistente con el propósito de cumplir con la heurística de "elegir en lo posible el menor número de cambios" (heurística 2) se analiza lo siguiente:

(a) Si el número de caminos que llegan al valor de la variable *npadre* es igual al número de caminos que llegan al valor de la variable *opadre*, el algoritmo trata de reparar la dimensión otorgando como padre en la categoría cat_cl al valor de la variable *npadre* (línea 8 y 9 del algoritmo 2), invocando a la función *repair()*. Si no se puede otorgar como padre en la categoría cat_cl al contenido en la variable *npadre*, se repara la dimensión estableciendo como padre en la categoría conflictiva al elemento almacenado en la variable *opadre* (ancestro original en la categoría conflictiva), para esto se utiliza la variable bandera *repair* de la línea 10 del algoritmo 2.

(b) Si el número de caminos que llegan al valor de la variable *opadre* es mayor que el número de caminos que llegan al valor de la variable *npadre*, el algoritmo trata de reparar la dimensión estableciendo como padre en la categoría cat_cl al valor de la variable *opadre*, ya que de esta forma se van a realizar menos cambios en la dimensión, objetivo de la heurística 2 (línea 10 y 11 del algoritmo 2). Para implementar los cambios se llama a la función *repair()*. En caso que no se pueda otorgar como padre en la categoría cat_cl al valor de la variable en *opadre*, se procede a establecer como padre en la categoría conflictiva al elemento en la variable *npadre*. En este caso, el número de cambios no será el menor (línea 9 del algoritmo 2).

La función *repair()* es la encargada de buscar y materializar los cambios de arcos. Es posible demostrar que siempre se implementará un cambio para un camino inconsistente.

Algoritmo 2: MENÚ REPARACIÓN

Input: Listas de Caminos inconsistentes y su elemento
Output: Conjunto de Reclasificaciones para su posterior reparación

1 $camino\_opadre, camino\_npadre, repair = 0, *npadre, *opadre, *elementoinconsistente;$
2 $npadre = Encontrarnpadre(ListadeElementosInconsistentes);$
3 $opadre = Encontraropadre(ListadeElementosInconsistentes, npadre);$
4 $camino\_opadre = contar\_opadre(ListadeElementosInconsistentes, opadre);$
5 $camino\_npadre = contar\_npadre(ListadeElementosInconsistentes, npadre);$
6 $elementoinconsistente = ListadeCaminosInconsistentes \rightarrow valor;$
7 **while** $ListadeCaminosInconsistentes \neq NULL$ **do**
8 **if** $camino\_opadre = camino\_npadre\ OR\ repair = 0$ **then**
9 $repair = repair(ListadeCaminosInconsistentes, npadre, opadre);$
10 **if** $camino\_opadre > camino\_npadre\ OR\ repair = 0$ **then**
11 $repair = repair(ListadeCaminosInconsistentes, opadre, npadre);$
12 **if** $repair \neq 0$ **then**
13 $repair = 0;$
14 **while** $elementoinconsistente = ListadeCaminosInconsistentes \rightarrow valor$ **do**
15 $ListadeCaminosInconsistentes = ListadeCaminosInconsistentes \rightarrow sgte;$
16 $elementoinconsistente = ListadeCaminosInconsistentes \rightarrow valor;$

El algoritmo 3 corresponde a la función *repair()*. Esta función recibe los siguientes parámetros: (i) los caminos inconsistentes, para un elemento involucrado en la inconsistencia (almacenados en la lista *ListadeCaminosInconsistentes*, (ii) el elemento a preservar en la categoría conflicto (almacenado en la variable *padrea*) y (iii) el elemento a modificar en la categoría conflicto (guardado en la variable *padreb*). El ciclo *for*, (línea 2) realiza tantas iteraciones como caminos inconsistentes hay para el elemento involucrado en la inconsistencia. Lo primero a realizar es

encontrar el camino que se desea modificar, es decir, aquel que contiene el valor de la variable *padreb* en la categoría cat_cl. Esto se verifica en la línea 3. Una vez, encontrado el camino a modificar, se procede a computar la reparación. Para ello, recordemos que se necesita en primera instancia ubicar al elemento, digamos e, en la categoría inferior a cat_cl que rollups a *padreb*, para asignarle en lo posible el nuevo padre en la categoría conflictiva (valor en *padrea*). Esto se realiza entre las líneas 4 y la 20 del algoritmo. En las líneas 9 y 10 se verifica que el elemento e no correponda al elemento involucardo en la reclasificación realizada previamente. Luego en las líneas 14 a la 19 se verifica que si se elige modificar el arco que involucra a e no se generen nuevas inconsistencias (heurística 1). Si se puede realizar el cambio de arco para este e, entonces se actualiza su padre en la categoría conflictiva al valor dado por la variable *padrea* (líneas 22 a 28).

Si por el contrario, no se puede modificar el elemento e en la categoría inferior a la categoría conflicto, el algoritmo iterará buscando un elemento a modificar hacia abajo, hasta llegar a la categoría cat_bottom, y realizar una actualización que retorne la consistencia. Si el algoritmo no encuentra un elemento a modificar, entonces, esta función retorna el valor 0 que indica que no se llevo a cabo una operación (actualización) de reparación, y por lo tanto, se deben buscar otras alternativas (lo que realiza el algoritmo 2).

Para ilustrar el procesamiento que realiza este algoritmo utilizaremos el ejemplo que se detalla a continuación.

Ejemplo 4.4 *Sea la lista llenada en el ejemplo 4.3, los valores para las variables del algoritmo 2:*

- npadre*(nuevo padre en la categoría conflicto):* **UEFA** *(ver Figura 4.4)*
- opadre *(padre original en la categoría conflicto):***Conm**

En la línea 4 se obtiene el número de caminos que alcanzan **Conm** *que en este caso es uno, y en la línea 5 se obtiene el número de caminos que llegan a* **UEFA***, que también es uno. Como el número de caminos que llegan al* **UEFA** *es igual al número de caminos que llegan a* **Conm***, se pretende mantener como padre en cat_cl a* **UEFA** *(líneas 8 y 9), para lo cual se llama a la función repair() que recibe la lista de caminos inconsistentes de la Figura 4.5 (algoritmo 3).*

El ciclo detallado entre las líneas 2 y 35 del algoritmo 3 se realiza dos veces ya que hay dos caminos involucrados para el único elemento inconsistente. Al comenzar con la primera iteración en la línea 3 se pregunta si el valor del elemento en la categoría Confederación *es* **Conm***. En este caso no corresponde, por lo tanto, se busca en el siguiente camino donde si se encuentra el valor* **Conm** *(ver Figura 4.5).*

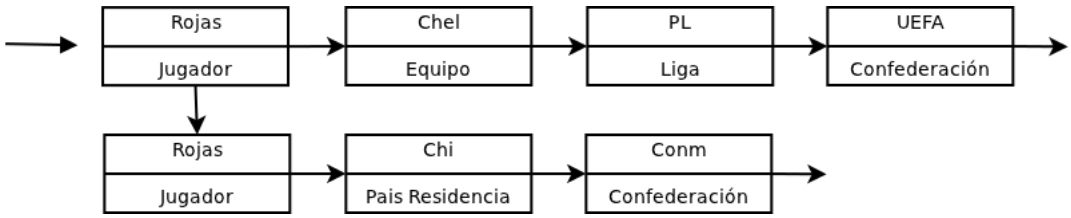


Figura 4.5: Iteración 1 algoritmo 3 según Ejemplo 4.4

Algoritmo 3: FUNCIÓN *repair()*

Input: 1 Camino inconsistente, padrea, padreb
Output: Reclasificación r_i

```
1  i, reparar = 0, deshacer = 0, autorizo = 0, *ehijo, *cat_h, *epadre, *cat_p, *aux, *aux2, caminopadre;
2  for i = 1; i <= num_hijos; i++ do
3      if esta_elemento(ListadeCaminosInconsistentes, padreb) = 1 then
4          ehijo = obtienepenultimoelemento(ListadeCaminosInconsistentes);
5          aux = ehijo;
6          cat_h = obtienepenultimacategoria(ListadeCaminosInonsistentes);
7          aux2 = cat_h;
8          while aux ≠ elementoinconsistente do
9              if ehijo = Reclassify1.hijo then
10                 deshacer = 1;
11             if deshacer = 0 then
12                 if hijosenlista(aux, aux2) = 1 then
13                     autorizo = 1;
14                 else
15                     autorizo = 0;
16                     ehijo = obtieneanterior(ListadeCaminosInconsistentes, ehijo);
17                     cat_h = obtienecategoriaanterior(ListadeCaminosInconsistentes, cat_h);
18             aux = obtieneanterior(ListadeCaminosInconsistentes, aux);
19             aux2 = obtienecategoriaanterior(ListadeCaminosInconsistentes, aux2);
20         if autorizo = 1 then
21             caminopadre = ArmarCaminos(padrea);
22             if caminopadre ≠ NULL then
23                 epadre = caminopadre → valor;
24                 c_padre = caminopadre → cat;
25                 InsertarActualizacionenR(cat_h, cat_p, ehijo, epadre);
26                 reparar = 1;
27                 ActualizarBD();
28             else
29                 reparar = 0;
30         else
31             reparar = 0;
32     ListadeCaminosInconsistentes = ListadeCaminosInconsistentes → sgte;
33 return reparar;
```

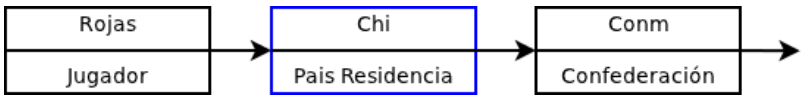
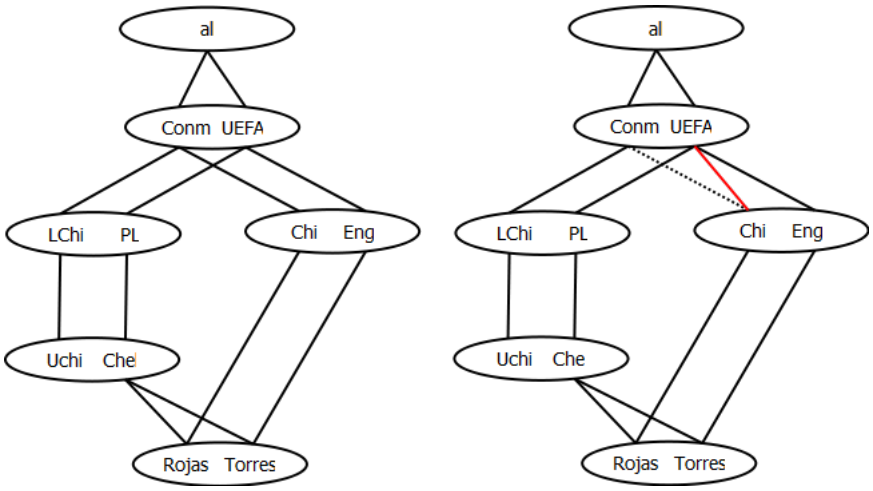


Figura 4.6: Primer elemento analizado por el algoritmo 3 según Ejemplo 4.4

En la línea 4 y 6 de este algoritmo se toma el penúltimo elemento **Chi** *y la categoría* **País Residencia** *(ver Figura 4.6). Luego, según el análisis realizado en la línea 9 el elemento* **Chi** *no está involucrado en la reclasificación original que causó la inconsistencia, y además el cambiar su padre en la categoría* Confederación *no genera nuevas inconsistencias (función* hijosenlista() *de la línea 12). Por lo tanto, se ejecuta la actualización (línea 28) y el algoritmo 3 retornará el valor de* reparar *igual a 1. Como en este ejemplo no existen más inconsistencias, el algoritmo 2 finaliza entregando la r-reparación de la Figura 4.7(b) la cual se obtiene con un número de cambios igual a 2.*



(a) Dimensión Profesional Actualizada (b) Reparación de Profesional

Figura 4.7: Solución de Ejemplo 4.4

□

4.1.2. Complejidad algorítmica

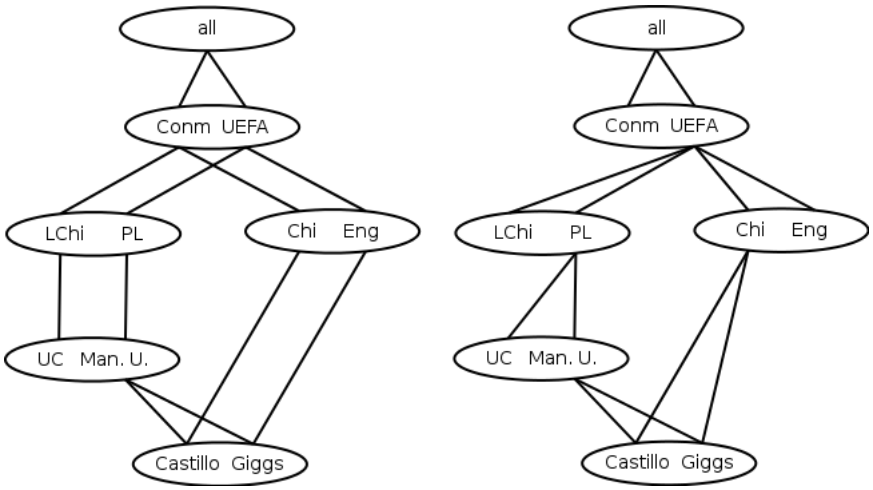
El algoritmo que obtiene una r-reparación, se compone de dos grandes procesos:

- El proceso que se encarga de obtener los elementos que son inconsistentes con respecto a restricciones estrictas (algoritmo 1).
- El proceso que se encarga de computar los cambios aplicando las heurísticas definidas en la sección 2.9 (algoritmos 2 y 3)

Sea n el número máximo de elementos de una categoría, r el número máximo de relaciones del rollup entre dos categorías, y m_s el número de restricciones estrictas. El costo de comprobar si una restricción estricta $c_i \rightarrow c_j$ es violada por una dimensión $\mathcal{D}$ es $O(r^2)$ ya que la relación $r_{c_i}^{c_j}$ es de tamaño $O(r)$ y se necesita comparar cada tupla en ella con todas sus tuplas. 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 relacionan con un único elemento en una categoría superior. Luego, para realizar los cambios que restauran la consistencia es necesario obtener el o los elementos que hacen rollup a un elemento inconsistente $e \in c_i$, sea ese elemento e' que pertenece a la categoría $c_{inferior}$ con $c_{inferior} \nearrow^* c_i$, tal que (e', e) con $e \in c_i$. El número de rutas y el costo de computarlas, en el peor de los casos es $O(n^{|C|})$, pero en la mayoría de los casos, en que el número de elementos inconsistentes es bajo, es de $O(|C|)$. Por lo tanto, el Algoritmo para obtener una r-reparación se ejecuta en el peor de los casos en $O(m_s(r^2 + n^{|C|}))$ y en el caso esperado en $O(n^2(m_s))$.

Notese que podrían existir otras alternativas más eficientes para restaurar la consistencia de una dimensión con respecto a restricciones estrictas. Por ejemplo, una opción podría ser elegir un elemento en cada categoría distinta a la inferior y hacer que cada elemento en una categoría inferior se relacione con el respectivo elemento elegido en la categoría superior. En la Figura 4.8(a)

se muestra una dimensión inconsistente con respecto a la restricción Jugador → Confederacion, en esta dimensión el elemento Castillo se relaciona con dos elementos en Confederacion. Una forma rápida de reparar se muestra en la Figura 4.8(b) donde por cada categoría (distinta a la inferior) se elige un elemento y luego se actualizan las relaciones rollup de tal forma que todo elemento en la categoría inferior sigue la misma ruta hacia el elemento en la categoría nivel de conflicto.



(a) Dimensión Profesional (b) R-reparación de Profesional

Figura 4.8: Otra forma de encontrar una r-reparación

Esta forma de encontrar una r-reparación es mucho más expedita en términos de tiempo, sin embargo, en lo que respecta a la semántica es menos adecuada, ya que se realiza un número excesivo de cambios y por lo tanto, la r-reparación se aleja de la dimensión original.

4.2. Algoritmos de Reparación con Restricciones de Integridad e impuestas por el administrador

4.2.1. Descripción

Los algoritmos presentados en esta sección permiten obtener una r-repación para una dimensión bajo las mismas condiciones descritas en la sección 4.1 más la incorporación de:

- Grados de prioridad entre las relaciones de las categorías hijo/padre.
- Restricciones seguras, es decir, indicar qué relaciones entre categorías hijo/padre no deben ser modificadas.

Los algoritmos que se detallan en la sección 4.1 se han modificado en los siguientes puntos:

(a) Adición de un atributo más en la lista mencionada en la sección 4.1.1, denominado prioridad.

(b) Modificación al algoritmo 1, añadiendo la inserción de los grados de prioridad en las relaciones rollup.

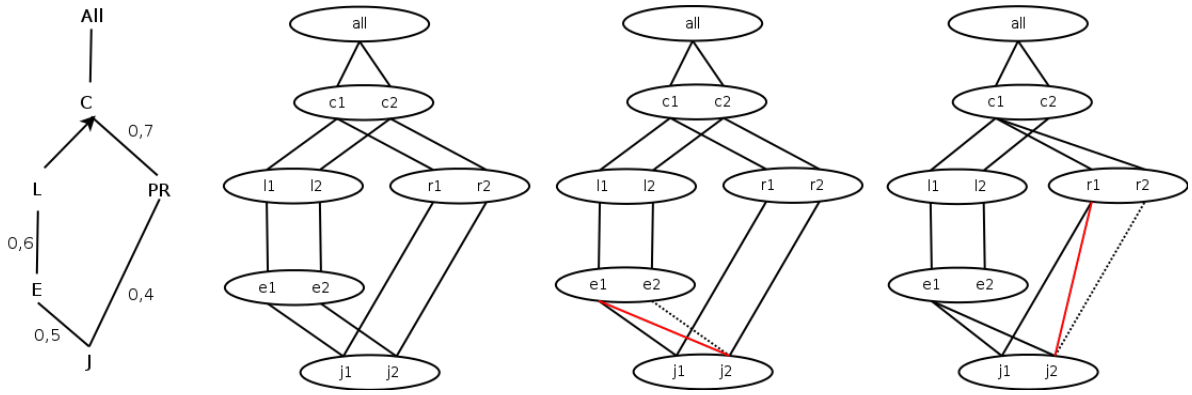
(c) Adaptabilidad del proceso de r-reparación considerando restricciones impuestas por el administrador (grados de prioridad en los rollups y las restricciones seguras).

Para cumplir el tercer punto, se han definido las siguientes heurísticas para computar una r-reparación con restricciones impuestas por el administrador:

- **No generar nuevas inconsistencias en el proceso de reparación (explicada en la sección 2.9).**

- **Elegir en lo posible, los cambios donde el rollup tenga menor grado de prioridad.**

  Ejemplo 4.5 *Sea el esquema jerárquico mostrado en la Figura 4.9(a) (los números representan los grados de prioridad en los rollups, mientras que las flechas simbolizan las restricciones seguras), la dimensión $\mathcal{D}$ mostrada en la Figura 4.9(b), la dimensión reclasificada $\mathcal{D}'$ mostrada en la Figura 4.9(c) y la restricción estricta φ $\mathsf{J}\rightarrow\mathsf{C}$. El esquema jerárquico, tiene como categoría nivel de conflicto a C. El administrador del DW ha definido grados de prioridad en los rollups (entre J y E es $0,5$, entre E y L es $0,6$, entre J y PR es $0,4$, y por último, entre PR y C es $0,7$) y una restricción segura (entre L y C). Para poder elegir una operación válida de reparación que involucre hacer los cambios donde exista el menor grado de prioridad posible, se necesita analizar cuál es el mínimo grado de prioridad existente en el camino desde j_2 hasta los elementos c_1 (llamado p_1) y c_2 (llamado p_2). Cómo p_2 es menor que p_1, una operación válida de reparación de acuerdo a esta heurística es aquella que modifica la relación rollup de los elementos que alcanzan c_2. En este sentido se obtiene la r-reparación en la Figura 4.9(d). Note que esta operación de cambio es válida porque no viola la heurística anterior.*



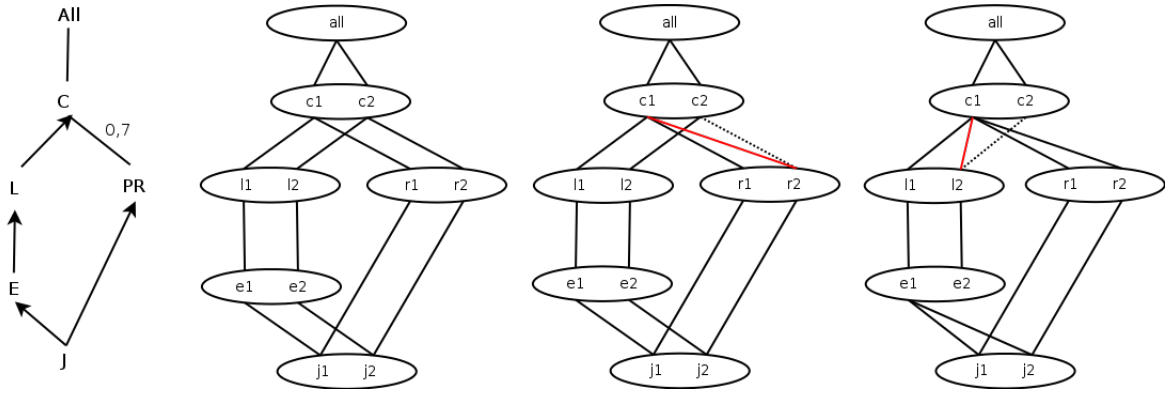
(a) Esquema Jerárquico (b) Dimensión Original (c) Reclasificación (D, J, E, j_2, e_1) (d) R-reparación

Figura 4.9: R-reparación: Heurística 2 (al existir restricciones impuestas por el administrador)

□

- **No modificar rollups en la restricción segura.**

  Ejemplo 4.6 *Sea el esquema jerárquico mostrado en la Figura 4.10(a) (los números representan los grados de prioridad en los rollups, mientras que las flechas simbolizan las restricciones seguras), la dimensión $\mathcal{D}$ mostrada en la Figura 4.10(b), la dimensión reclasificada $\mathcal{D}'$ mostrada en la Figura 4.10(c) y la restricción estricta φ J→C. El esquema jerárquico, tiene como categoría nivel de conflicto a C. El administrador del DW ha definido grados de prioridad en los rollups (entre PR y C es $0,7$), y restricciones seguras (entre J y E, entre E y L, entre L y C, y por último entre J y PR). El cambio mostrado en la Figura 4.10(d) no es válido porque al imponer restricciones seguras, esas relacinoes rollups no deben ser modificados.* □



(a) Esquema Jerárquico (b) Dimensión Original (c) Reclasificación (D, R, C, r_2, c_1) (d) Mal Cambio

Figura 4.10: R-reparación: Heurística 3 (al existir restricciones impuestas por el administrador)

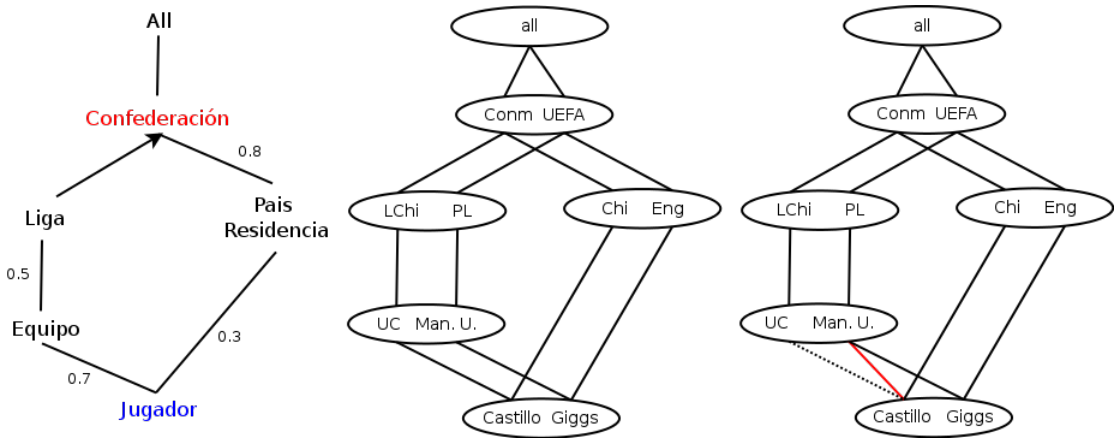
Almacenamiento de Caminos Inconsistentes

En este caso es necario almacenar los elementos inconsistentes, sus categorías y los grados de prioridad. Para mostrar su funcionamiento se presenta el siguiente ejemplo:

Ejemplo 4.7 *Sea el ejemplo 4.3. Adicionalmente el administrador del Data Warehouse ha determinado que la relación entre las categorías* Liga *y* Confederación *es fija (indicada con una flecha en Figura 4.11(a)), transformándola en una restricción segura.*

Otro dato importante es que cada relación entre categorías, posee un grado de prioridad (ver números en la Figura 4.11(a)), definiendo el grado de importancia de la relación rollup, es decir, si tiene un valor más cercano a cero menos importante será, mientras que si tiene un valor más cercano a uno, más importante será.

Ha llegado el mercado de fichajes de invierno en Europa, y el Manchester United ha decidido fichar a un jugador que actualmente milita en Universidad Católica llamado Nicolás Castillo. Dicho traspaso ha involucrado un cambio en la dimensión, y esa actualización ha quedado mostrada en Figura 4.11(c).



(a) Esquema Jerárquico Dimensión Profesional (b) Dimensión Profesional Original (c) Dimensión Profesional Actualizada

Figura 4.11: Ilustración de Ejemplo 4.5

La categoría cat_bottom es Jugador *(línea 2 del algoritmo 1), y la categoría nivel de conflicto (cat_cl) que se obtiene al ejecutarse la línea 3 del algoritmo 1 es* Confederación*. Luego, en la línea 6 del algoritmo 1 se ejecuta la función Ejecutar(query), la que verifica si existe inconsistencia respecto de la restricción de integridad y de ser así obtiene los elementos involucrados en la inconsistencia. En este caso, se obtiene el elemento* **Castillo** *de la categoría* Jugador *(ver Figura 4.3(c)), luego entre las líneas 8 y 23 del algoritomo 1 se realiza el proceso de llenado de la lista, lo que consiste en la inserción de elementos y categorías en la Lista de Caminos Inconsistentes. Es importante aquí conocer el número de caminos que existe entre la categoría bottom y la categoría nivel de conflicto, esta información se obtiene en la lína 5 del algoritmo. En este caso existen dos caminos para alcanzar Confederación y por lo tanto, la rutina entre la línea 13 y 20 del algoritmo 1 se efectúa 2 veces (1 por camino). En primer lugar se inserta el elemento* **Castillo***, la categoría* Jugador *y la prioridad* **0.7***; luego* **Man. U.***, la categoría* Equipo *y la prioridad* **0.3***; posteriormente* **PL***, la categoría* Liga *y la prioridad* **2.0** *(ya que Liga es la categoría hijo de una restricción segura, ver Figura 4.11(a)); y finalmente para el camino 1* **UEFA***, la categoría* Confederación *con un valor constante* **1.0***. Ya al terminar el primer camino, de la misma forma se insertan los elementos* **Castillo** *y la categoría* Jugador*,* **Chi** *y la categoría* **País Residencia** *y para finalizar* **Conm** *y la categoría* Confederación*. El resultado final del proceso de inserción, se detalla en la Figura 4.12.* □

Reparación

El algoritmo 4 para cada elemento involucrado en inconsistencia escoge una vía de reparación, es decir, elige la ruta desde cat_bottom hacia cat_cl que será modificada para restaurar consistencia. Para ello es necesario obtener nuevos parámetros que sirven como base para el análisis de las heurísticas que permiten encontrar una *r-reparación*:

- p_opadre: el contenido de esta variable es el menor grado de prioridad existente en una

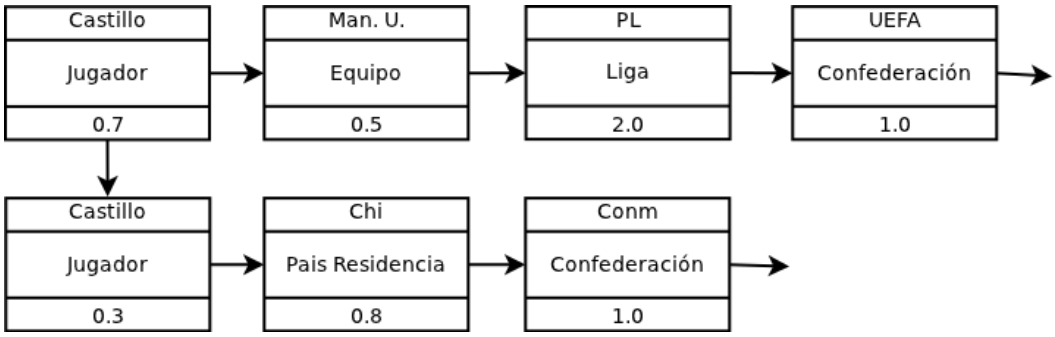


Figura 4.12: Implementación de Algoritmo 1 según Ejemplo 4.5

ruta que alcanza al valor de la variable opadre (ver línea 4 del algoritmo 4).

- p_npadre: el contenido de esta variable es el menor grado de prioridad existente en una ruta que alcanza al valor de la variable npadre (ver línea 5 del algoritmo 4).

Algoritmo 4: Menú Reparación (con restricciones impuestas por el administrador)

Input: Listas de Caminos inconsistentes y su elemento
Output: Conjunto de Reclasificaciones para su posterior reparación

```
1  p_opadre, p_npadre, repair = 0, *npadre, *opadre, *elementoinconsistente, intento = 0;
2  npadre = Encontrarnpadre(ListadeElementosInconsistentes);
3  opadre = Encontraropadre(ListadeElementosInconsistentes, npadre);
4  p_opadre = Minprioridad(ListadeElementosInconsistentes, opadre);
5  p_npadre = Minprioridad(ListadeElementosInconsistentes, npadre);
6  elementoinconsistente = ListadeCaminosInconsistentes → valor;
7  while ListadeCaminosInconsistentes ≠ NULL do
8      if p_npadre < p_opadre OR repair = 0 then
9          repair = repair(ListadeCaminosInconsistentes, npadre, opadre, p_opadre);
10         if repair = 0 then intento++;
11         ;
12     if p_opadre ≤ p_npadre OR repair = 0 then
13         repair = repair(ListadeCaminosInconsistentes, opadre, npadre, p_npadre);
14         if repair = 0 then intento++;
15         ;
16     if repair ≠ 0 then
17         repair = 0;
18         while elementoinconsistente = ListadeCaminosInconsistentes → valor do
19             ListadeCaminosInconsistentes = ListadeCaminosInconsistentes → sgte;
20         elementoinconsistente = ListadeCaminosInconsistentes → valor;
21     if intento ≥ 2 then exit();
22     ;
```

Por cada elemento inconsistente, para cumplir con la heurística que dice "elegir en lo posible, los cambios donde el rollup tenga menor grado de prioridad" (heurística 2) se analiza lo siguiente:

(a) Si la mínima prioridad existente en la ruta que alcanza al valor de la variable *npadre* es menor a la mínima prioridad existente en la ruta que alcanza al valor de la variable *opadre*, el algoritmo trata de reparar la dimensión eligiendo como padre en la categoría cat_cl al contenido

de la variable npadre (líneas 8 y 9 del algoritmo 4), llamando a la función repair(). Si no se pueden obtener los cambios para mantener al valor de npadre como elemento padre en la categoría cat_cl, se trata de reparar manteniendo al valor de opadre como elemento padre en la categoría cat_cl, para esto se utiliza la bandera repair de la linea 9 del algoritmo 4.

(b) Si la mínima prioridad existente en la ruta que alcanza al valor de la variable opadre es menor o igual a la mínima prioridad existente en la ruta que alcanza al valor de la variable npadre, el algoritmo trata de reparar la dimensión eligiendo como padre en la categoría cat_cl al contenido de la variable opadre (líneas 11 y 12 del algoritmo 4), llamando a la función *repair*(). Si no se pueden obtener los cambios para mantener al valor de opadre como elemento padre en la categoría cat_cl, se trata de reparar manteniendo al valor de npadre como elemento padre en la categoría cat_cl, para esto se utiliza la bandera repair de la linea 12 del algoritmo 4.

La función repair() es la encargada de buscar y materializar los cambios de arcos, sin embargo, es posible que no se pueda obtener una r-reparación al existir restricciones seguras impuestas por el administrador.

El algoritmo 5 corresponde a la función repair(). Esta función recibe los siguientes parámetros: (i) los caminos inconsistentes, para un elemento involucrado en la inconsistencia (almacenados en la lista ListadeCaminosInconsistentes, (ii) el elemento a preservar en la categoría conflicto (almacenado en la variable *padrea*), (iii) el elemento a modificar en la categoría conflicto (guardado en la variable *padreb*) y la mínima prioridad existente en la ruta que alcanza al elemento a modificar en la categoría conflicto. El ciclo for, (línea 2) realiza tantas iteraciones como caminos inconsistentes hay para el elemento involucrado en la inconsistencia. Lo primero a realizar es encontrar el camino que se desea modificar, es decir, aquel que contiene el valor de la variable *padreb* en la categoría cat_cl. Esto se verifica en la línea 3. Una vez, encontrado el camino a modificar, se procede a tratar de computar la reparación. Para ello, recordemos que se necesita en primera instancia ubicar al elemento, digamos e, en la categoría inferior a cat_cl que rollups a *padreb*, para asignarle en lo posible el nuevo padre en la categoría conflictiva (valor en *padrea*). Esto se realiza entre las líneas 4 y la 19 del algoritmo. En las líneas 10 se verifica que el elemento e no correponda al elemento involucardo en la reclasificación realizada previamente, que al efectuar un cambio no origine nuevas inconsistencias, ni que se viole la restricción segura impuesta por el administrador. Si se puede realizar el cambio de arco para este e, entonces se actualiza su padre en la categoría conflictiva al valor dado por la variable *padrea* (líneas 20 a 26).

Si por el contrario, no se puede modificar el elemento e en la categoría inferior a la categoría conflicto, el algoritmo iterará buscando un elemento a modificar hacia abajo, hasta llegar a la categoría cat_bottom, y tratar de realizar una actualización que retorne la consistencia. Si el algoritmo no encuentra un elemento a modificar, entonces, esta función retorna el valor 0 que indica que no se llevo a cabo una operación (actualización) de reparación, y por lo tanto, se deben buscar otras alternativas (lo que realiza el algoritmo 2).

Para ilustrar el procesamiento que realiza este algoritmo utilizaremos el ejemplo que se detalla a continuación.

Ejemplo 4.8 *Sea el llenado de la lista de elementos no estrictos obtenida en el ejemplo 4.7, los valores para las variables del algoritmo 4:*

- *npadre (nuevo padre en la categoría conflictiva):* ***UEFA***

Algoritmo 5: FUNCIÓN *repair()* (CON RESTRICCIONES IMPUESTAS POR EL ADMINISTRADOR)

```
   Input: 1 Camino inconsistente, padrea, padreb, prioridadb
   Output: Reclasificación r_i
 1 i, reparar = 0, minp, e_hijo, c_hijo, e_padre, c_padre, aux1, aux2, caminopadre;
 2 for i = 1; i ≤ num_hijos; i++ do
 3     if esta_elemento(ListadeCaminosInconsistentes, padreb) = 1 then
 4         minp = Minprio(ListadeCaminosInconsistentes);
 5         e_hijo = Get_elemento(ListadeCaminosInconsistentes, minp);
 6         c_hijo = Get_categoria(ListadeCaminosInconsistentes, elemento);
 7         aux1 = e_hijo;
 8         aux2 = c_hijo;
 9         while aux1 ≠ NULL do
10             if hijosenlista(aux2, aux1) = 1 AND (e_hijo ≠ Reclassify1.hijo OR minp ≠ 2,0) then
11                 autorizo = 1;
12             else
13                 minp = PrioAnterior(ListadeCaminosInconsistentes, minp);
14                 if minp ≤ prioridadb then
15                     e_hijo = Get_elemento(ListadeCaminosInconsistentes, minp);
16                     c_hijo = Get_cat(ListadeCaminosInconsistentes, elemento);
17                 else autorizo = 0;
18                 ;
19             aux1 = obtieneanterior(ListadeCaminosInconsistentes, aux1);
20             aux2 = obtienecategoriaanterior(ListadeCaminosInconsistentes, aux2);
21         if autorizo = 1 then
22             caminopadre = ArmarCaminos(padrea);
23             if caminopadre ≠ NULL then
24                 e_padre = caminopadre → valor;
25                 c_padre = caminopadre → cat;
26                 InsertarActualizacionenR(c_hijo, c_padre, e_hijo, e_padre);
27                 ActualizarBD();
28             else return reparar = 0;
29             ;
30         else return reparar = 0;
31         ;
32     ListadeCaminosInconsistentes = ListadeCaminosInconsistentes → sgte;
33 if reparar = 0 then
34     return reparar = 0;
35 else
36     return reparar = 1;
```

- *opadre (padre original en la categoría conflictiva):* ***Conm***
- *p_opadre (mínima prioridad en la ruta que alcanza al valor de opadre):* $0,3$
- *p_npadre (mínima prioridad en la ruta que se encuentra la variable npadre):* $0,5$

Como el valor de p_opadre es menor o igual que el valor de p_npadre, se pretende mantener como padre en cat_cl a ***UEFA*** *(líneas 11 y 12), para lo cual se llama a la función repair() que recibe la lista de caminos inconsistentes de la Figura 4.12 (algoritmo 5).*

El ciclo detallado entre las líneas 2 y 30, se realiza 2 veces ya que hay dos caminos involucrados para el único elemento inconsistente. Al comenzar con la primera iteración en la línea 3 se pregunta si el valor del elemento en la categoría Confederación *es* ***Conm****. En este caso no corresponde, por lo tanto, se busca en el siguiente camino donde si se encuentra el valor* ***Conm***

(ver Figura 4.12).

En la línea 4, se obtiene la mínima prioridad existente en la ruta a reparar, en este caso es $0,3$*, para posteriormente ejecutarse la línea 5, que almacena el elemento asociado a la mínima prioridad (Castillo) y finalmente en la línea 6 se obtiene la categoría a la que pertenece el elemento asociado a la mínima prioridad (Jugador). Como este elemento no está involucrado en la reclasificación (línea 10 del algoritmo 5), se procede a obtener un elemento y su categoría que permitan llegar a* ***UEFA*** *(líneas 23 y 24), para efectuar la actualización de la línea 26 y el algoritmo 5 retornará el valor de* reparar *igual a 1. Como en este ejemplo no existen más inconsistencias, el algoritmo 2 finaliza entregando la r-reparación de la Figura 4.14(b) la cual se obtiene con un número de cambios igual a 2.*

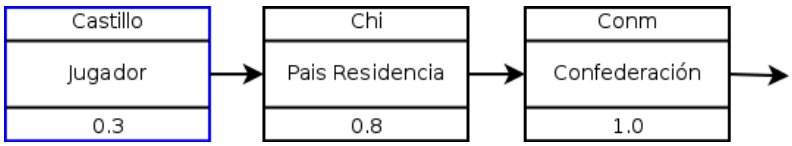
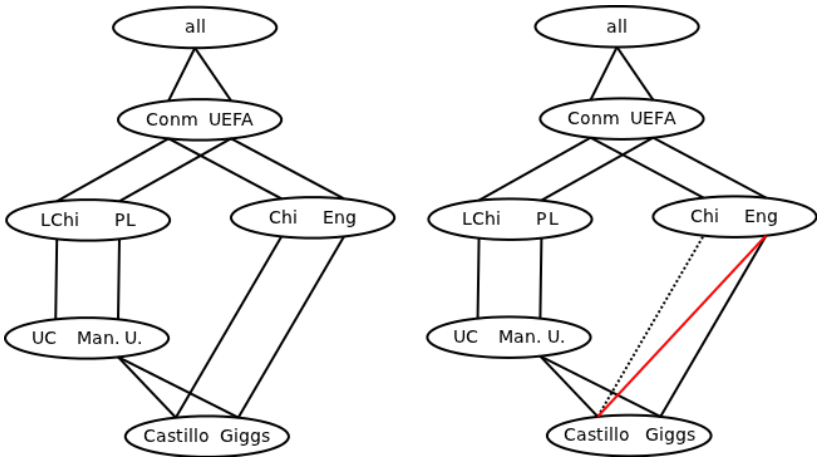


Figura 4.13: Primer elemento analizado por el algoritmo 5 según Ejemplo 4.6



(a) Dimensión Profesional Actualizada (b) Dimensión Profesional Reparada

Figura 4.14: Solución de Ejemplo 4.2

□

La complejidad algoritmica de los algoritmos que resuelven inconsistencia con respecto a restricciones estrictas considerando grados de prioridad y restricciones de aplicación seguras no difiere de la complejidad mostrada en la sección 4.1.2.

Capítulo 5

Experimentos

En esta sección se presentan experimentos para poder validar los algoritmos detallados en el capítulo 4.

5.1. Antecedentes preliminares

La máquina de prueba tiene las siguientes especificaciones:

Hardware

- Procesador Intel® Core™ i5-2430M CPU @ 2.40GHz × 4.
- Memoria 7,7 GiB.
- Intercambio 2 GiB.

Sistema

- Debian Versión 7.1 (wheezy) de 64-bits.
- Núcleo Linux 3.2.0-4-amd64.
- GNOME 3.4.2.

Software

- **PostgresSQL** versión 9.1.9-1 y **pgadminIII** versión 1.14.2-2 para manejo de bases de datos.
- **gcc** versión 4.7.2-5 y **Geany** 1.22 para programación en lenguaje *C*.
- **DLV:** Sistema deductivo de base de datos, basado en programación lógica disyuntiva (**?**).

Para calcular el tiempo de ejecución de los algoritmos se utiliza el comando `time` de Linux, el cual permite obtener el tiempo total de ejecución de un programa. La sintaxis es: `time` seguido del comando cuya ejecución se quiere medir (con los parámetros correspondientes). El resultado de `time` se escribe en la salida de error estándar (en C `stderr`, en C++ `cerr`, y en línea de comandos). Para la medición de memoria, en teoría, se debería utilizar el comando `time` de Linux, el cual debería permitir ver información sobre la memoria usada por un proceso, tal y como lo fija el estándar *GNU 1.7*, sin embargo, la mayoría de las versiones actuales de Linux no lo hacen. Por esta razón, se ha ejecutado un programa denominado *memory* que permite obtener una estimación (aproximada) de la máxima cantidad de memoria usada por cualquier otro programa.

5.2. Experimentos con Restricciones de Integridad

En esta sección, se presentan en detalle los resultados obtenidos al aplicar los algoritmos implementados en la sección 4.1.

Experimento I Se cuenta con el esquema jerárquico mostrado en la Figura 5.1 y con la restricción estricta $\varphi : A \rightarrow D$:

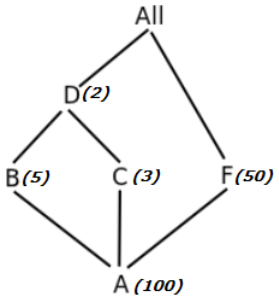


Figura 5.1: Esquema de Prueba I

En (**?**) se propone utilizar DLV para computar reparaciones minimales, es por esa razón que la primera prueba de esta sección apunta a verificar la calidad de la r-reparación (con respecto al número de cambios). Sin embargo, DLV no es capaz de computar las reparaciones minimales de las dimensiones que posean muchos elementos en sus categorías, es por esa razón que esta prueba se ejecutará en una dimensión donde la categoría inferior tenga 100 elementos. El resultado de la ejecución del algoritmo se muestra en la Figura 5.2 y su detalle en la Tabla 5.1.

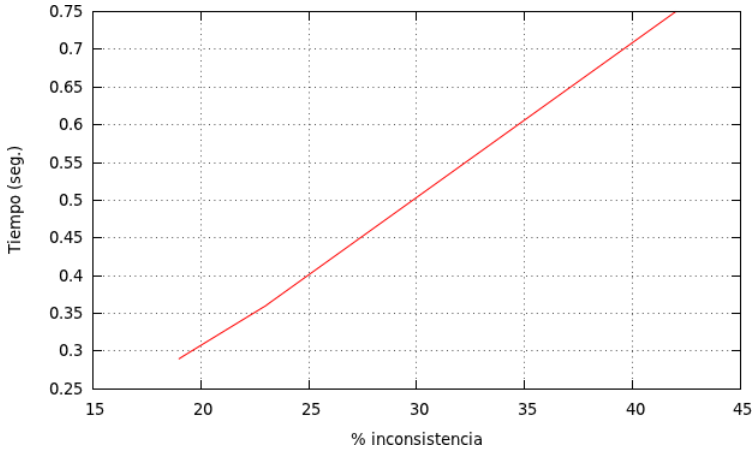


Figura 5.2: Resultado Experimento I

Reclasificación	% Incon.	T(s)	Cambios	DLV	Mem. (Kb)
Reclasificación(D, B, D, b_4, d_1)	19 % (19)	0,29	38	4	2420
Reclasificación(C, B, D, b_1, d_1)	23 % (23)	0,36	46	4	2548
Reclasificación(D, C, D, c_3, d_1)	42 % (42)	0,75	84	7	4660

Tabla 5.1: Detalles Experimento I

El comportamiento del algoritmo en la dimensión con 100 elementos en la categoría inferior, indica que a mayor grado de inconsistencia en la dimensión, es decir, a mayor cantidad de elementos que violan la restricción estricta φ, se requiere de más tiempo y más memoria temporal para poder computar la r-reparación. La calidad de la r-reparación (en términos de números de cambios) es mala, porque el número de cambios obtenidos por el algoritmo es proporcional al número de elementos inconsistentes (lo duplica), mientras que DLV obtiene un número de cambios menor al de los elementos inconsistentes.

Experimento II Se cuenta con el siguiente esquema jerárquico mostrado en la Figura 5.3 y la restricción estricta $\varphi : A \rightarrow D$:

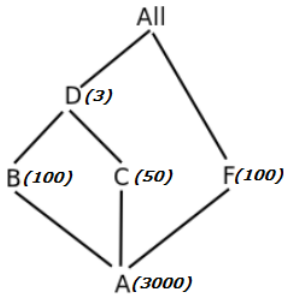


Figura 5.3: Esquema de Prueba Experimento II

El resultado de la ejecución del algoritmo se muestra en la Figura 5.4 y su detalle en la Tabla 5.2.

Reclasificación	% Incon.	T(s)	Cambios	DLV	Mem. (Kb)
Reclasificación(D, B, D, b_1, d_7)	0,70 % (21)	1,49	42	s/r	12196
Reclasificación(D, B, D, b_{73}, d_3)	1,17 % (35)	1,95	70	s/r	3352
Reclasificación(C, C, D, c_{26}, d_4)	1,83 % (55)	6,49	110	s/r	59184

Tabla 5.2: Detalles Experimento II

El comportamiento del algoritmo en la dimensión con 3000 elementos en la categoría inferior, indica que a mayor grado de inconsistencia en la dimensión, es decir, a mayor cantidad de elementos que violan la restricción estricta φ, se requiere de más tiempo para poder computar la r-reparación. La cantidad de cambios obtenidos por el algoritmo es el doble a la cantidad de elementos inconsistentes, mientras que DLV no entregó respuesta.

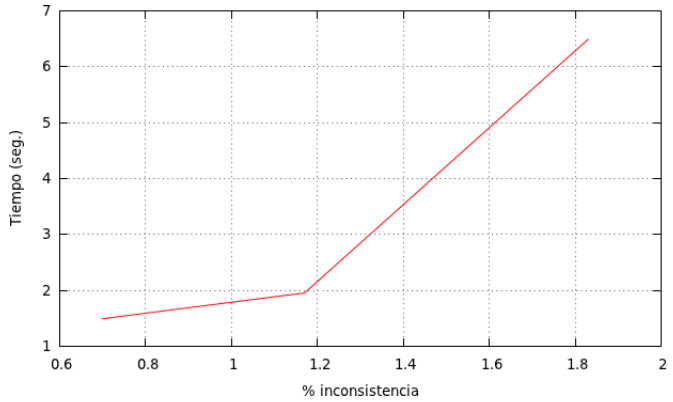


Figura 5.4: Resultado Experimento II

Experimento III Se cuenta con el siguiente esquema jerárquico mostrado en la Figura 5.5 y la restricción estricta $\varphi : A \rightarrow H$:

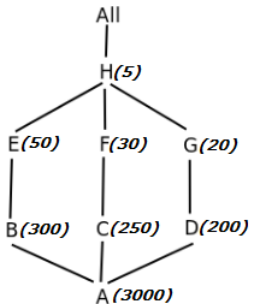


Figura 5.5: Esquema de Prueba Experimento III

El resultado de la ejecución del algoritmo se presenta en la Figura 5.6 y su detalle en la Tabla 5.3.

Reclasificación	% Incon.	T(s)	Cambios	DLV	Mem. (Kb)
Reclasificación(D, G, H, g_{20}, h_3)	0,10 % (3)	0,12	6	s/r	1352
Reclasificación(D, E, H, e_4, h_5)	5,03 % (151)	0,54	22	s/r	1508
Reclasificación(D, F, H, f_{10}, h_4)	7,50 % (225)	1,21	30	s/r	1636

Tabla 5.3: Detalle Experimento III

El comportamiento del algoritmo en la dimensión con 3000 elementos en la categoría inferior, indica que a mayor grado de inconsistencia en la dimensión, es decir, a mayor cantidad de elementos que violan la restricción estricta φ, se requiere de más tiempo y más memoria temporal para poder computar la r-reparación. La cantidad de cambios obtenidos por el algoritmo es relevante, porque en las pruebas B y C, se obtiene una cantidad de cambios mucho menor a la cantidad de elementos inconsistentes. DLV, no entrega respuesta en estas pruebas.

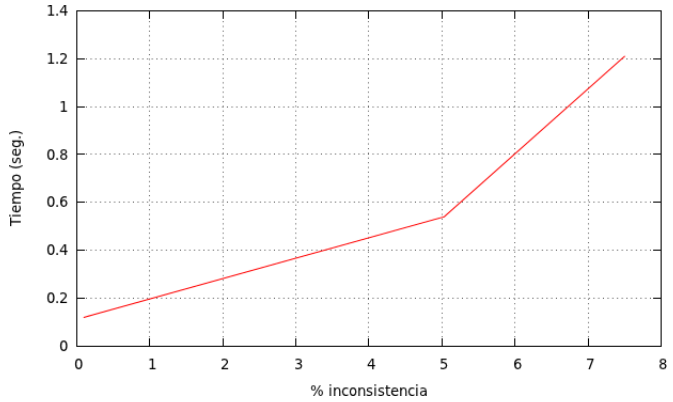


Figura 5.6: Resultado Experimento III

Experimento IV Se cuenta con el siguiente esquema jerárquico mostrado en la Figura 5.7 y la restricción estricta $\varphi : A \rightarrow E$:

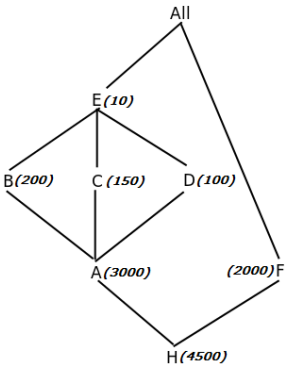


Figura 5.7: Esquema de Prueba Experimento IV

El resultado de la ejecución del algoritmo se señala en la Figura 5.8 y su detalle en la Tabla 5.4.

Reclasificación	% Incon.	T(s)	Cambios	DLV	Mem. (Kb)
Reclasificación(D, E, G, e_{45}, g_2)	1,23 % (37)	2,92	74	s/r	32264
Reclasificación(D, F, G, f_{48}, g_3)	1,57 % (47)	3,12	94	s/r	14180
Reclasificación(D, F, G, f_{40}, g_4)	6,83 % (205)	5,75	410	s/r	9420

Tabla 5.4: Detalle Experimento IV

El comportamiento del algoritmo en la dimensión con 3000 elementos en la categoría inferior, indica que a mayor grado de inconsistencia en la dimensión, es decir, a mayor cantidad de elementos que violan la restricción estricta φ, se requiere de más tiempo para poder computar la r-reparación. La cantidad de cambios obtenidos por el algoritmo es el doble a la cantidad de elementos inconsistentes, mientras que DLV no entregó respuesta.

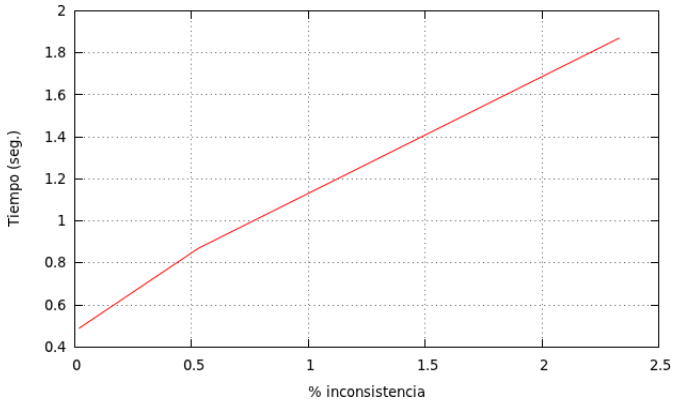


Figura 5.8: Resultado Experimento IV

Caso Real Chile es un país divido políticamente en comunas que en la actualidad ascienden a 346. Cada comuna tiene un cierto número de teléfonos proporcionales a su población (de un total de 4.985.600 números telefónicos), los cuales están asociados con un código de área (26 en total). Ese código de área se relaciona con una única región (de un universo de 15), lo mismo sucede para una comuna. El esquema jerárquico que modela dicha situación es el que se presenta en la Figura 5.9. Cabe señalar que se incluye la cantidad de elementos que pertenecen a cada categoría.

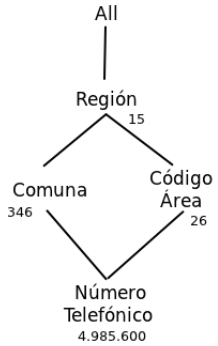


Figura 5.9: Esquema jerárquico dimensión Teléfono

Adicionalmente se cuentan con la siguiente restricción estricta φ: Número Telefónico $\rightarrow$ Región. Se ha realizado el cambio respecto de la relación rollup entre el código 39 que ahora pasa de la quinta región a la tercera. Este cambio produce un total de 1.250 números inconsistentes. El código de área 39 solo involucra a la comuna de Isla de Pascua, por lo que sólo hay una operación de inserción/eliminación de arcos. El detalle de la r-reparación se encuentra en la Tabla A.1. Por lo tanto, el algoritmo es bastante eficiente en obtener la r-reparación ya que la obtuvo en 00:00:11,77 segundos, y la memoria requerida fue de 1,68 Mb.

Luego se llevaron a cabo las siguientes reclasificaciones

- Actualización: Reclasificación(D,CA,R,67,XII)

- Elementos inconsistentes (número): 30.300 elementos inconsistentes.
- **Tiempo de Reparación:** 00:01:45,81.
- **Memoria Requerida:** 14,34 Mb. En este caso el código de área 67 involucra a 10 comunas de la Región de Aysén, por lo que hay 10 operaciones de inserción/eliminación de arcos. El detalle de la r-reparación se encuentra en la Tabla A.2.

- Actualización: Reclasificación(D,CA,R,58,II)
  - Elementos inconsistentes (número): 79.300 elementos inconsistentes.
  - **Tiempo de Reparación:** 00:06:45,30.
  - **Memoria Requerida:** 35,67 Mb. En este caso el código de área 58 involucra a 7 comunas de la Región de Tarapacá, por lo que hay 7 operaciones de inserción/eliminación de arcos. El detalle de la r-reparación se encuentra en la Tabla A.3.
- Actualización: Reclasificación(D,CA,R,43,IX)
  - Elementos inconsistentes (número): 117.350 elementos inconsistentes.
  - **Tiempo de Reparación:** 00:13:41,09.
  - **Memoria Requerida:** 52,02 Mb. En este caso el código de área 43 involucra a 14 comunas de la Región del Bío-Bío, por lo que hay 14 operaciones de inserción/eliminación de arcos. El detalle de la r-reparación se encuentra en la Tabla A.4.

Resumen de los Experimentos en el Caso Real:

El gráfico 5.10 muestra los resultados obtenidos, como se puede apreciar a mayor nivel de inconsistencia se requiere mayor tiempo para obtener una r-reparación.

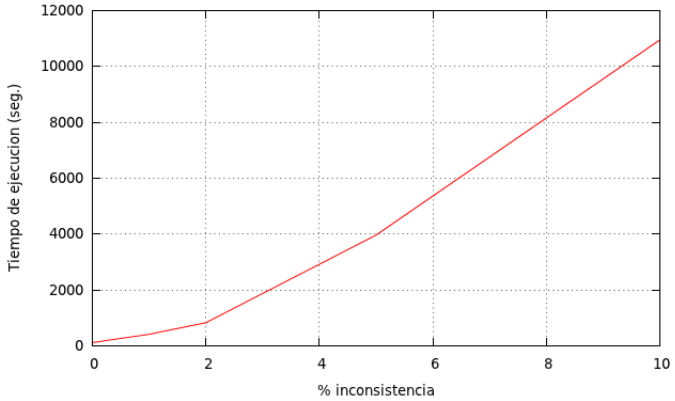


Figura 5.10: Rendimiento del algoritmo en caso real con restricciones de integridad

5.3. Experimentos con Restricciones de Integridad e impuestas por el administrador

En esta sección, se presentan en detalle los resultados obtenidos al aplicar los algoritmos implementados en 4.2.

Experimento I Se cuenta con el Esquema Jerárquico mostrado en la Figura 5.12 con una restricción segura entre las categorias C y D y con grados de prioridad. La restricción estricta es $\varphi : A \rightarrow D$.

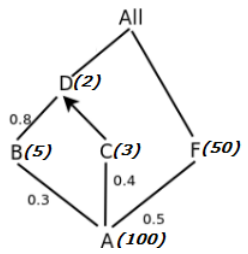


Figura 5.11: Esquema de Prueba I (con restricciones impuestas por el administrador)

El resultado de la ejecución del algoritmo se muestra en la Figura 5.12 y su detalle en la Tabla 5.5

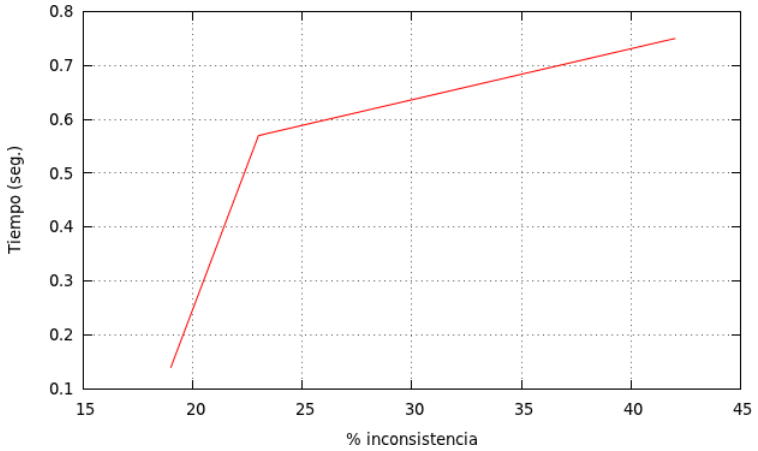


Figura 5.12: Resultado Experimento I (con restricciones impuestas por el administrador)

Reclasificación	% Incon.	T(s)	Cambios (P)	DLV (P)	Mem.(Kb)
Reclas.(D, B, D, b_4, d_1)	19 % (19)	0,14	38 (0.3)	4 (0.8(2)-0.3(2))	2004
Reclas.(C, B, D, b_1, d_1)	23 % (23)	0,57	46 (0.3)	4 (0.8(2)-0.3(2))	2136
Reclas.(D, C, D, c_3, d_1)	42 % (42)	0,75	84 (0.4)	38 (0.8(10)-0.3(2))	5304

Tabla 5.5: Detalles Experimento I (con restricciones impuestas por el administrador)

El comportamiento del algoritmo en la dimensión con 100 elementos en la categoría inferior, indica que a mayor grado de inconsistencia en la dimensión, es decir, a mayor cantidad de elementos que violan la restricción estricta φ, se requiere de más tiempo y más memoria temporal para poder computar la r-reparación. La calidad de la r-reparación (dónde se hacen los cambios) es buena, porque el algoritmo los hace en la menor prioridad, no así DLV.

Cabe recordar, que la heurística 2 de la r-reparación al existir restricciones impuestas por el administrador es "elegir en lo posible, los cambios donde exista la menor prioridad".

Caso Real Para los experimentos se muestra el esquema de los teléfonos con grados de prioridad y restricciones de seguridad, además se considera la restricción estricta φ: Número Telefónico $\rightarrow$ Región.

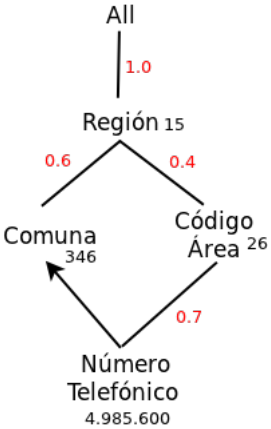


Figura 5.13: Esquema jerárquico dimensión Teléfono con restricciones impuestas por el administrador

El administrador del Data Warehouse ha decidido que la prioridad entre: Número Telefónico y Código Área sea 0,7, entre Código Área y Región sea 0,4, entre Comuna y Región sea 0,6 y entre Región y All sea 1, además de definir que el rollup entre Número Telefónico y Comuna es fija, transformándola en una restricción segura. Se realizaron varias actualizaciones y tal como muestra el gráfico 5.14, se da la tendencia que a mayor cantidad de elementos inconsistentes se requiere un mayor tiempo para obtener una r-reparación.

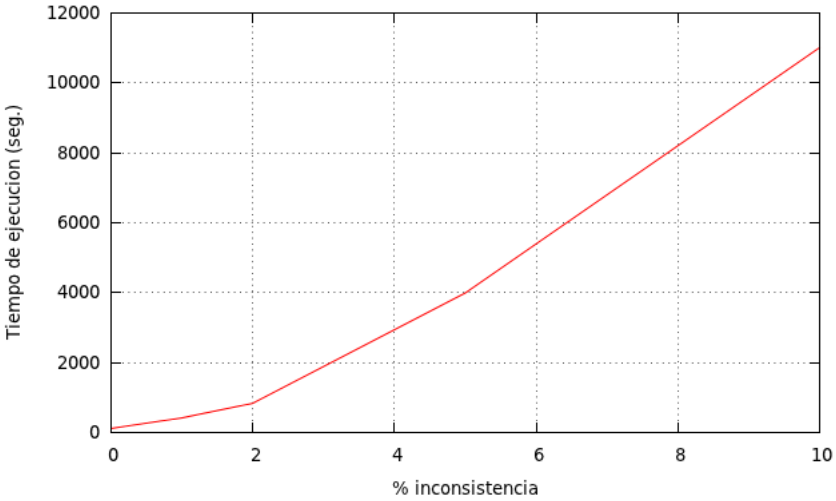


Figura 5.14: Rendimiento del algoritmo en caso real con restricciones de integridad e impuestas por el administrador

Capítulo 6

Conclusión

El origen de esta tesis es crear un algoritmo iterativo y procedural que permita computar una r-reparación (**?**) para dimensiones que cuenten con un nivel de conflicto (Sección 4.1). Una r-reparación se obtiene mediante la aplicación de las siguientes heurísticas:

- No generar nuevas inconsistencias en el proceso de reparación.
- En lo posible elegir el mínimo número de cambios.

Adicionalmente, se ha ampliado la usabilidad de este algoritmo a dimensiones que además de contar con restricciones de integridad, cuenten con:

- Grados de Prioridad, que indican la afinidad de los elementos del rollup.
- Restricciones Seguras, que indican qué relaciones rollup no pueden modificarse (Sección 4.2).

La motivación de este trabajo se basó en desarrollar una solución polinomial que permita encontrar una r-reparación de una dimensión inconsistente con respecto a sus restricciones de integridad, sin tener que calcular todas sus posibles r-reparaciones. Al existir restricciones impuestas por el administrador, se han establecido nuevas heurísticas que permiten guiar el proceso de encontrar una r-reparación:

- No generar nuevas inconsistencias en el proceso de reparación.
- En lo posible elegir reparar donde exista la menor prioridad.
- No computar cambios donde exista una restricción segura.

Sería relevante conocer el historial de los datos que se almacenan en el warehouse, para definir, por ejemplo los grados de prioridad en los rollups, ya que si los datos no han sido modificados en el transcurso del tiempo, tienen una mayor confiabilidad. Adicionalmente, sería interesante saber la fuente de donde se obtienen los datos, porque puede darse el caso, que la información almacenada en el data warehouse, pueda venir de otras bases de datos que no sean de confianza para una organización, por la que los datos obtenidos de ésta, tienen una menor prioridad. Cabe señalar que al existir restricciones seguras impuestas por el administrador, es posible que el algoritmo no compute una r-reparación.

Los algoritmos fueron implementados en el lenguaje C con los cuales se desarrollaron diferentes pruebas en determinadas dimensiones con tamaños en orden de los miles de datos (Sección 5.2 y 5.3), además de probarlos considerando data warehouses reales.

El rendimiento de los algoritmos depende de los siguientes factores, de los cuales ninguno es excluyente:

- Cantidad de elementos inconsistentes: en el peor de los casos todos los elementos de la categoría inferior son inconsistente.
- Restricciones impuestas por el administrador: puede darse el caso que no se encuentre una *r-reparación* por la existencia de restricciones seguras.
- Número de cambios de rollup entre elementos: mientras menos actualizaciones se tenían que realizar, más rápido fue la ejecución del algoritmo.

Un rol clave en el rendimiento del algoritmo es el índice-B, implementado en Postgres, el cual optimizó en gran manera los tiempos que en primera instancia se obtuvieron.

Queda como trabajo futuro el analizar nuevos esquemas que puedan resolverse en tiempo polinomial. Probar nuevas estructuras de datos que entreguen un mejor rendimiento de este algoritmo. Desarrollar técnicas de paralelismo para obtener una r-reparación.

Apéndice A

Resultados Experimentos Caso Real

A.1. Algoritmo con Restricciones de Integridad

Operador	Consulta SQL
Reclasificación(D,C,R,IPC,III)	Update “RCR” Set “Son”=‘IPC’, “Father”=‘III’ Where “Son” = ‘IPC’

Tabla A.1: Detalle de reparación código 39

Operador	Consulta SQL
Reclasificación(D,C,R,WPA,XII)	Update “RCR” Set “Son”=‘WPA’, “Father”=‘XII’ Where “Son” = ‘WPA’
Reclasificación(D,C,R,CSN,XII)	Update “RCR” Set “Son”=‘CSN’, “Father”=‘XII’ Where “Son” = ‘CSN’
Reclasificación(D,C,R,GTC,XII)	Update “RCR” Set “Son”=‘GTC’, “Father”=‘XII’ Where “Son” = ‘GTC’
Reclasificación(D,C,R,LGR,XII)	Update “RCR” Set “Son”=‘LGR’, “Father”=‘XII’ Where “Son” = ‘LGR’
Reclasificación(D,C,R,OHI,XII)	Update “RCR” Set “Son”=‘OHI’, “Father”=‘XII’ Where “Son” = ‘OHI’
Reclasificación(D,C,R,TRT,XII)	Update “RCR” Set “Son”=‘TRT’, “Father”=‘XII’ Where “Son” = ‘TRT’
Reclasificación(D,C,R,GXQ,XII)	Update “RCR” Set “Son”=‘GXQ’, “Father”=‘XII’ Where “Son” = ‘GXQ’
Reclasificación(D,C,R,CCH,XII)	Update “RCR” Set “Son”=‘CCH’, “Father”=‘XII’ Where “Son” = ‘CCH’
Reclasificación(D,C,R,RIB,XII)	Update “RCR” Set “Son”=‘RIB’, “Father”=‘XII’ Where “Son” = ‘RIB’
Reclasificación(D,C,R,LVE,XII)	Update “RCR” Set “Son”=‘LVE’, “Father”=‘XII’ Where “Son” = ‘LVE’

Tabla A.2: Detalle de reparación código 67

Operador	Consulta SQL
Reclasificación(D,C,R,AHO,II)	Update “RCR” Set “Son”=‘AHO’, “Father”=‘II’ Where “Son” = ‘AHO’
Reclasificación(D,C,R,IQQ,II)	Update “RCR” Set “Son”=‘IQQ’, “Father”=‘II’ Where “Son” = ‘IQQ’
Reclasificación(D,C,R,CMN,II)	Update “RCR” Set “Son”=‘CMN’, “Father”=‘II’ Where “Son” = ‘CMN’
Reclasificación(D,C,R,COL,II)	Update “RCR” Set “Son”=‘COL’, “Father”=‘II’ Where “Son” = ‘COL’
Reclasificación(D,C,R,HUR,II)	Update “RCR” Set “Son”=‘HUR’, “Father”=‘II’ Where “Son” = ‘HUR’
Reclasificación(D,C,R,PIC,II)	Update “RCR” Set “Son”=‘PIC’, “Father”=‘II’ Where “Son” = ‘PIC’
Reclasificación(D,C,R,PAL,II)	Update “RCR” Set “Son”=‘PAL’, “Father”=‘II’ Where “Son” = ‘PAL’

Tabla A.3: Detalle de reparación código 58

Operador	Consulta SQL
Reclasificación(D,C,R,ABB,IX)	Update “RCR” Set “Son”=‘ABB’, “Father”=‘IX’ Where “Son” = ‘ABB’
Reclasificación(D,C,R,ANT,IX)	Update “RCR” Set “Son”=‘ANT’, “Father”=‘IX’ Where “Son” = ‘ANT’
Reclasificación(D,C,R,CBR,IX)	Update “RCR” Set “Son”=‘CBR’, “Father”=‘IX’ Where “Son” = ‘CBR’
Reclasificación(D,C,R,LJA,IX)	Update “RCR” Set “Son”=‘LJA’, “Father”=‘IX’ Where “Son” = ‘LJA’
Reclasificación(D,C,R,LSQ,IX)	Update “RCR” Set “Son”=‘LSQ’, “Father”=‘IX’ Where “Son” = ‘LSQ’
Reclasificación(D,C,R,MUL,IX)	Update “RCR” Set “Son”=‘MUL’, “Father”=‘IX’ Where “Son” = ‘MUL’
Reclasificación(D,C,R,NAC,IX)	Update “RCR” Set “Son”=‘NAC’, “Father”=‘IX’ Where “Son” = ‘NAC’
Reclasificación(D,C,R,NEG,IX)	Update “RCR” Set “Son”=‘NEG’, “Father”=‘IX’ Where “Son” = ‘NEG’
Reclasificación(D,C,R,QLC,IX)	Update “RCR” Set “Son”=‘QLC’, “Father”=‘IX’ Where “Son” = ‘QLC’
Reclasificación(D,C,R,QLL,IX)	Update “RCR” Set “Son”=‘QLL’, “Father”=‘IX’ Where “Son” = ‘QLL’
Reclasificación(D,C,R,SRS,IX)	Update “RCR” Set “Son”=‘SRS’, “Father”=‘IX’ Where “Son” = ‘SRS’

Reclasificación(D,C,R,SBB,IX)	Update “RCR” Set “Son”=‘SBB’, “Father”=‘IX’ Where “Son” = ‘SBB’
Reclasificación(D,C,R,TUC,IX)	Update “RCR” Set “Son”=‘TUC’, “Father”=‘IX’ Where “Son” = ‘TUC’
Reclasificación(D,C,R,YUM,IX)	Update “RCR” Set “Son”=‘YUM’, “Father”=‘IX’ Where “Son” = ‘YUM’

Tabla A.4: Detalle de reparación código 43

A.2. Algoritmo con Restricciones de Integridad e impuestas por el administrador

Operador	Consulta SQL
Reclasificación(D,C,R,IPC,III)	Update “RCR” Set “Son”=‘IPC’, “Father”=‘III’ Where “Son” = ‘IPC’

Tabla A.5: Detalle de reparación código 39 (con restricciones impuestas por el administrador)

Operador	Consulta SQL
Reclasificación(D,C,R,WPA,XII)	Update “RCR” Set “Son”=‘WPA’, “Father”=‘XII’ Where “Son” = ‘WPA’
Reclasificación(D,C,R,CSN,XII)	Update “RCR” Set “Son”=‘CSN’, “Father”=‘XII’ Where “Son” = ‘CSN’
Reclasificación(D,C,R,GTC,XII)	Update “RCR” Set “Son”=‘GTC’, “Father”=‘XII’ Where “Son” = ‘GTC’
Reclasificación(D,C,R,LGR,XII)	Update “RCR” Set “Son”=‘LGR’, “Father”=‘XII’ Where “Son” = ‘LGR’
Reclasificación(D,C,R,OHI,XII)	Update “RCR” Set “Son”=‘OHI’, “Father”=‘XII’ Where “Son” = ‘OHI’
Reclasificación(D,C,R,TRT,XII)	Update “RCR” Set “Son”=‘TRT’, “Father”=‘XII’ Where “Son” = ‘TRT’
Reclasificación(D,C,R,GXQ,XII)	Update “RCR” Set “Son”=‘GXQ’, “Father”=‘XII’ Where “Son” = ‘GXQ’
Reclasificación(D,C,R,CCH,XII)	Update “RCR” Set “Son”=‘CCH’, “Father”=‘XII’ Where “Son” = ‘CCH’
Reclasificación(D,C,R,RIB,XII)	Update “RCR” Set “Son”=‘RIB’, “Father”=‘XII’ Where “Son” = ‘RIB’
Reclasificación(D,C,R,LVE,XII)	Update “RCR” Set “Son”=‘LVE’, “Father”=‘XII’ Where “Son” = ‘LVE’

Tabla A.6: Detalle de reparación código 67 (con restricciones impuestas por el administrador)

Operador	Consulta SQL
Reclasificación(D,C,R,AHO,II)	Update “RCR” Set “Son”=‘AHO’, “Father”=‘II’ Where “Son” = ‘AHO’
Reclasificación(D,C,R,IQQ,II)	Update “RCR” Set “Son”=‘IQQ’, “Father”=‘II’ Where “Son” = ‘IQQ’
Reclasificación(D,C,R,CMN,II)	Update “RCR” Set “Son”=‘CMN’, “Father”=‘II’ Where “Son” = ‘CMN’
Reclasificación(D,C,R,COL,II)	Update “RCR” Set “Son”=‘COL’, “Father”=‘II’ Where “Son” = ‘COL’
Reclasificación(D,C,R,HUR,II)	Update “RCR” Set “Son”=‘HUR’, “Father”=‘II’ Where “Son” = ‘HUR’
Reclasificación(D,C,R,PIC,II)	Update “RCR” Set “Son”=‘PIC’, “Father”=‘II’ Where “Son” = ‘PIC’
Reclasificación(D,C,R,PAL,II)	Update “RCR” Set “Son”=‘PAL’, “Father”=‘II’ Where “Son” = ‘PAL’

Tabla A.7: Detalle de reparación código 58 (con restricciones impuestas por el administrador)

Operador	Consulta SQL
Reclasificación(D,C,R,ABB,IX)	Update “RCR” Set “Son”=‘ABB’, “Father”=‘IX’ Where “Son” = ‘ABB’
Reclasificación(D,C,R,ANT,IX)	Update “RCR” Set “Son”=‘ANT’, “Father”=‘IX’ Where “Son” = ‘ANT’
Reclasificación(D,C,R,CBR,IX)	Update “RCR” Set “Son”=‘CBR’, “Father”=‘IX’ Where “Son” = ‘CBR’
Reclasificación(D,C,R,LJA,IX)	Update “RCR” Set “Son”=‘LJA’, “Father”=‘IX’ Where “Son” = ‘LJA’
Reclasificación(D,C,R,LSQ,IX)	Update “RCR” Set “Son”=‘LSQ’, “Father”=‘IX’ Where “Son” = ‘LSQ’
Reclasificación(D,C,R,MUL,IX)	Update “RCR” Set “Son”=‘MUL’, “Father”=‘IX’ Where “Son” = ‘MUL’
Reclasificación(D,C,R,NAC,IX)	Update “RCR” Set “Son”=‘NAC’, “Father”=‘IX’ Where “Son” = ‘NAC’
Reclasificación(D,C,R,NEG,IX)	Update “RCR” Set “Son”=‘NEG’, “Father”=‘IX’ Where “Son” = ‘NEG’
Reclasificación(D,C,R,QLC,IX)	Update “RCR” Set “Son”=‘QLC’, “Father”=‘IX’ Where “Son” = ‘QLC’
Reclasificación(D,C,R,QLL,IX)	Update “RCR” Set “Son”=‘QLL’, “Father”=‘IX’ Where “Son” = ‘QLL’
Reclasificación(D,C,R,SRS,IX)	Update “RCR” Set “Son”=‘SRS’, “Father”=‘IX’ Where “Son” = ‘SRS’
Reclasificación(D,C,R,SBB,IX)	Update “RCR” Set “Son”=‘SBB’, “Father”=‘IX’ Where “Son” = ‘SBB’
Reclasificación(D,C,R,TUC,IX)	Update “RCR” Set “Son”=‘TUC’, “Father”=‘IX’ Where “Son” = ‘TUC’

Reclasificación(D,C,R,YUM,IX)	Update “RCR” Set “Son”=‘YUM’, “Father”=‘IX’ Where “Son” = ‘YUM’

Tabla A.8: Detalle de reparación código 43 (con restricciones impuestas por el administrador)

Apéndice B

Códigos Adicionales

B.1. Código en C++ para el Consumo de Memoria

Compilar con: $ g++ memory.cpp -o memory

Código B.1: memory.cpp

```
// Por Gines G.M., basado en el codigo de la funcion SafeExec, por Jose Paulo ↩
    Leal.        25/1/2008

#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include <stdio.h>
#include <sys/stat.h>
#include <ctype.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>

#define SIZE          8192  /* tamanio de buffer para leer /proc/<pid>/status */
#define INTERVAL        61  /* muestrear unas 16 veces por segundo */

pid_t pid;
int max_data= 0;
int max_stack= 0;

void error (char *pt){
  if (pt)
    fprintf(stderr, pt);
  abort();
}

int max (int a, int b){
  return (a > b ? a : b);
```

```
29  }
30
31  void terminate (pid_t pid){
32    int v;
33    v = kill (-1, SIGKILL);
34    if (v < 0)
35      if (errno != ESRCH)
36        error("error en terminate\n");
37  }
38
39  void msleep (int ms){
40    struct timeval tv;
41    int v;
42    do{
43        tv.tv_sec = ms / 1000;
44        tv.tv_usec = (ms %1000) * 1000;
45        v = select (0, NULL, NULL, NULL, &tv);
46      }
47    while ((v < 0) && (errno == EINTR));
48    if (v < 0)
49      error("error en msleep\n");
50  }
51
52  int memusage (pid_t pid){
53    char a[SIZE], *p, *q;
54    int data, stack;
55    int n, v, fd;
56
57    p = a;
58    sprintf (p, "/proc/%d/status", pid);
59    fd = open (p, O_RDONLY);
60    if (fd < 0)
61      error ("error despues de fd = open\n");
62    do
63      n = read (fd, p, SIZE);
64    while ((n < 0) && (errno == EINTR));
65    if (n < 0)
66      error ("error despues de n = read\n");
67    do
68      v = close (fd);
69    while ((v < 0) && (errno == EINTR));
70    if (v < 0)
71      error ("error despues de v = close\n");
72
73    data = stack = 0;
74    q = strstr (p, "VmData:");
75    if (q != NULL){
76        sscanf (q, "%*s %d", &data);
77        q = strstr (q, "VmStk:");
78        if (q != NULL)
79      sscanf (q, "%*s %d\n", &stack);
80      }
```

```
81    max_data= max(data, max_data);
82    max_stack= max(stack, max_stack);
83    return (data + stack);
84  }
85
86  void printusage (char **p){
87    fprintf (stderr, "Uso: %s <comando>\n", *p);
88    fprintf (stderr, "\t<comando>  Programa ejecutable y parametros, en su caso.\n↩
        ");
89  }
90
91  int main (int argc, char **argv, char **envp){
92    int status, mem;
93    int v;
94
95    if (argc <= 1 || (argc==2 && !strcmp(argv[1], "--help"))) {
96      printusage (argv);
97      return (EXIT_FAILURE);
98    }
99    else {
100     pid = fork ();
101     if (pid < 0)
102        error ("error despues del fork\n");
103     if (pid == 0) {
104       if (execve (argv[1], argv+1, envp) < 0) {
105         kill (getpid (), SIGPIPE);
106         error ("error despues de execve\n");
107       }
108     }
109     else {
110       mem= 16;
111       do {
112         msleep(INTERVAL);
113         mem= max(mem, memusage(pid));
114         do
115           v= waitpid (pid, NULL, WNOHANG | WUNTRACED);
116         while ((v < 0) && (errno != EINTR));
117         if (v < 0)
118           error("error despues de waitpid\n");
119       } while (v == 0);
120       fprintf(stderr, "\nMemoria total usada: %d Kbytes\n", mem);
121       fprintf(stderr, "  \"   de datos: %d Kbytes\n", max_data);
122       fprintf(stderr, "  \"    de pila: %d Kbytes\n", max_stack);
123     }
124   }
125   return (EXIT_SUCCESS);
126 }
```