CAPÍTULO 3. MINERÍA DE DATOS NO ESPACIALES

En este capítulo se describen algoritmos de minería de datos no espaciales y también se presenta el sistema de descubrimiento de subestructuras SUBDUE (desarrollado en la Universidad de Texas en Arlington). Los conceptos, definiciones y ejemplos de SUBDUE fueron extraídos de los documentos disponibles en la página Web del sistema [39], y de los comentarios y puntos de vista realizados por el Dr. Jesús Antonio González Bernal, investigador del Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE) en Tonanzintla, Puebla. En primera instancia se presentan los conceptos generales del sistema y posteriormente se abordan los temas de descubrimiento de subestructuras, codificación de la longitud de descripción mínima de grafos, comparación inexacta de grafos, descubrimiento de conceptos jerárquicos, y conocimiento del dominio.

3.1 Conceptos generales.

La enorme cantidad de datos que se generan hoy en día está superando rápidamente las posibilidades de los investigadores para interpretarlos y analizarlos. En respuesta a este problema se han desarrollado técnicas para descubrir conceptos y/o patrones en bases de datos. Estas técnicas trabajan adecuadamente en datos expresados en representación atributo-valor, no estructurado, y se enfocan a temas de relevancia en los datos, datos perdidos, inconsistencia, y utilización de conocimiento del dominio. Sin embargo, los proyectos actuales de recopilación de datos están incluyendo datos estructurados

describiendo las relaciones entre sus objetos. De tal forma, existe la necesidad de desarrollar técnicas para analizar y descubrir patrones en bases de datos estructuradas.

Un método para el descubrimiento de conocimiento en datos estructurados es la identificación de subestructuras comunes en los propios datos. La motivación de este proceso no es meramente encontrar subestructuras capaces de representar a los datos a través de la abstracción de instancias de la subestructura, sino identificar conceptualmente subestructuras que realcen la interpretación de los datos. El descubrimiento de subestructuras es el proceso de identificar conceptos y/o patrones describiendo subestructuras repetitivas dentro de los datos estructurados. Una vez descubierto, el concepto puede ser usado para simplificar los datos a través de reemplazar las instancias de la subestructura con un apuntador al nuevo concepto descubierto. Los conceptos descubiertos permiten realizar abstracciones, y ofrecen nuevos y relevantes atributos para la interpretación de los datos. La iteración del descubrimiento de subestructuras y el proceso de reemplazo construyen una descripción jerárquica de los datos estructurados en términos de las subestructuras descubiertas. Esta jerarquía ofrece diferentes niveles de interpretación que pueden ser utilizados en base a las metas del análisis de datos.

A continuación se comentan dos sistemas de minería de datos y posteriormente se explicará el sistema SUBDUE, empleado en este proyecto para aplicar minería de datos no espaciales a la base de datos del volcán Popocatépetl.

3.2 WEKA.

El sistema WEKA [41] (*Waikato Environment for Knowledge Analysis*) fue desarrollado en la Universidad de Waikato en Nueva Zelanda. Está implementado en el lenguaje de programación Java y ha sido probado en los ambientes operativos Windows, Linux y Macintosh. Implementa algoritmos de minería de datos que pueden aplicarse a bases de datos desde su línea de comando o bien desde su interfaz gráfica.

Este sistema incluye una variedad de herramientas para transformar conjuntos de datos. Permite realizar preprocesamientos de datos para transformarlos en un esquema de aprendizaje, a fin de que sus resultados puedan ser analizados. Una manera de usar WEKA es aplicar un método de aprendizaje a conjuntos de datos y analizar los resultados para extraer información. Otra es aplicar varios métodos de aprendizaje y comparar sus resultados en orden de escoger una predicción. Estos métodos son llamados clasificadores.

La implementación de los esquemas de aprendizaje son los recursos más valiosos del sistema. Las herramientas para el preprocesamiento de datos, llamados filtros, son los segundos más importantes. La atención de WEKA se centra en los algoritmos de clasificación y filtro, sin embargo, también incluye la implementación de algoritmos para el aprendizaje de reglas de asociación y el agrupamiento de datos (*clustering*).

3.3 DBMiner.

DBMiner [6] es un sistema de minería de datos desarrollado por el Grupo de Investigación de Minería de Datos del Laboratorio de Investigaciones de Sistemas de Base de Datos

Inteligentes en la Universidad Simon Fraser en Canadá. Entre sus funcionalidades más importantes podemos mencionar:

1. La habilidad para manejar grandes volúmenes de datos.
2. Graficación de resultados.
3. Características de seguridad integradas.
4. Interfaz gráfica para el usuario, herramientas de visualización para los datos e información.
5. Un lenguaje de consultas para minería de datos (DMQL).
6. Habilidad para realizar asociaciones, clasificaciones y agrupamientos de datos.

Al inicio del presente proyecto, uno de los primeros sistemas de minería de datos que se investigó fue DBMiner. Originalmente este sistema era conocido como DBLEARN, su constante mejoramiento e integración de nuevas funcionalidades lo han convertido en una herramienta muy valiosa en el campo de la minería de datos.

3.4 SUBDUE.

El sistema SUBDUE [39] descubre subestructuras que comprimen la base de datos original y representan conceptos estructurados en los datos. Como resultado de reemplazar subestructuras descubiertas previamente en los datos, múltiples ciclos de procesamiento del SUBDUE generan una descripción jerárquica de las regularidades estructurales de los datos. SUBDUE emplea una comparación de grafos inexacta que identifica instancias similares, pero no idénticas, de una subestructura, y encuentra una medida aproximada de

similitud entre dos subestructuras. Adicional al principio de longitud de descripción mínima, otras técnicas son empleadas en SUBDUE para guiar la búsqueda hacia subestructuras más apropiadas.

3.4.1 Descubrimiento de subestructuras.

El sistema de descubrimiento de subestructuras representa datos estructurados como un grafo dirigido. Los objetos en los datos (conceptos) se convierten en vértices o subgrafos pequeños en el grafo, y las relaciones entre los objetos se convierten en arcos dirigidos o no dirigidos. Una subestructura es un grafo conectado dentro de la representación gráfica. Esta representación gráfica sirve como datos de entrada al sistema de descubrimiento de subestructuras. La figura 3.1 muestra un ejemplo de una base de datos de entrada y su representación en grafo. El ejemplo es presentado en términos del dominio de casas, donde una casa se define como triángulo sobre cuadrado. *T* representa un triángulo, *C* un cuadrado, *E* una estrella y *R* un rectángulo. Los objetos en la figura (ej. *T1*, *C1*, *R1*) se convierten en vértices etiquetados en el grafo, y las relaciones (ej. *sobre* (*T1*, *C1*), *forma* (*E1*, estrella)) se convierten en arcos etiquetados. La representación gráfica de la subestructura descubierta por SUBDUE en estos datos es mostrada en la figura 3.2.

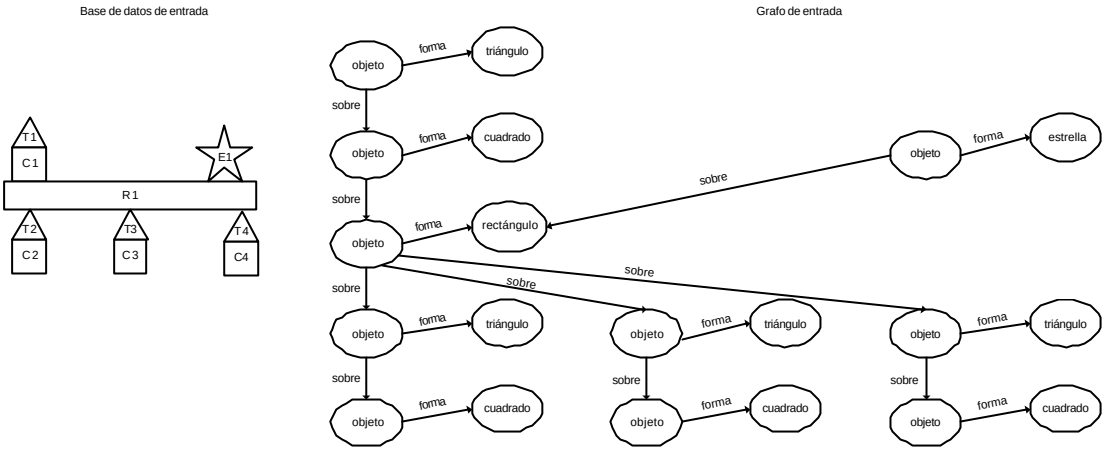

Figura 3.1 Ejemplo de una base de datos representada en grafo.

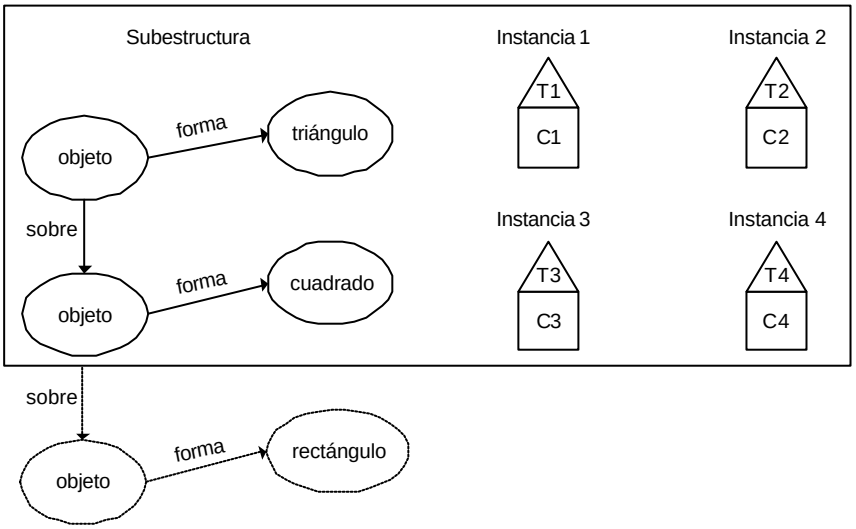

Figura 3.2 Instancias de una subestructura.

Una instancia de una subestructura en un grafo de entrada, es un subconjunto de vértices y arcos del grafo de entrada que corresponde a la definición gráfica de la subestructura. Por ejemplo, la subestructura descubierta en la figura 3.1 y sus instancias se muestran en la figura 3.2. Un arco vecino de una instancia es un arco en el grafo de entrada que no está

contenido en la instancia, pero está conectado al menos a un vértice de la instancia. Por ejemplo, la primera instancia tiene un arco vecino: *sobre* (*C1*, *R1*). Una conexión externa de una instancia de una subestructura es un arco vecino de la instancia que está conectada al menos a un vértice no contenido en la instancia. De tal forma, *sobre* (*C1*, *R1*) es también la única conexión externa de la primera instancia en la figura 3.2.

El algoritmo de descubrimiento de subestructuras usado por SUBDUE es una búsqueda dirigida con restricciones de cálculo. El algoritmo empieza con una lista de subestructuras de distintos vértices en el grafo. Cada iteración permite al algoritmo seleccionar la mejor subestructura y expande las instancias de esas subestructuras con un arco vecino ó un arco y un vértice en todas las direcciones posibles. Las nuevas subestructuras generadas se convierten en candidatos para otras expansiones. El algoritmo continúa la búsqueda de la mejor subestructura hasta que todas las posibles subestructuras han sido consideradas o la cantidad de cálculo excede a un límite establecido.

Típicamente, una vez que la longitud de descripción de una subestructura expandida comienza a incrementarse, otras expansiones de la subestructura no tendrán un tamaño de descripción menor. Como resultado, SUBDUE hace uso de un mecanismo de poda que elimina expansiones de subestructuras cuando la longitud de descripción para esas expansiones se incrementa.

El empleo de heurísticas para evaluar cada subestructura afecta significativamente los resultados del sistema. Si se emplean heurísticas independientes del dominio, el proceso de descubrimiento podría generar resultados sorprendentes. Sin embargo, las subestructuras

descubiertas pueden no ser siempre útiles para el usuario. Por el otro lado, el conocimiento específico del dominio puede ayudar al proceso de descubrimiento dirigiendo la búsqueda y también puede ayudar a que las estructuras descubiertas sean más útiles y productivas al usuario. En la versión actual del sistema SUBDUE se emplea el principio de longitud de descripción mínima (*MDL* por las siglas en ingles de *minimum description length*) que se describe en la siguiente sección.

3.4.2 Codificación de la longitud de descripción mínima de grafos.

Empleando *MDL* como modelo de evaluación en SUBDUE, conocimiento dependiente del dominio puede ser agregado a los procesos de descubrimiento para dirigir al sistema hacia subestructuras de particular interés en un dominio dado.

El principio de longitud de descripción mínima introducido por Rissanen [37] establece que la mejor teoría para describir un conjunto de datos es aquella que minimiza la longitud de descripción del conjunto de datos. El principio *MDL* ha sido empleado para la inducción en árboles de decisión [31], procesamiento de imágenes [21] y modelos de aprendizaje en dominios ingenieriles no homogéneos [34]. En esta misma sección se presenta un ejemplo del cálculo *MDL* en SUBDUE.

Una subestructura es evaluada en base a su capacidad para comprimir el conjunto total de datos usando la longitud de su descripción mínima. Se define el principio de longitud de descripción mínima de un grafo como el número de *bits* necesarios para describir completamente al grafo.

De acuerdo a este principio, la teoría que mejor se ajusta para una colección de datos es aquella que minimiza $I(S) + I(G|S)$, donde S es la subestructura descubierta, G es el grafo de entrada, $I(S)$ es el número de *bits* requeridos para codificar la subestructura, y $I(G|S)$ es el número de *bits* requeridos para codificar el grafo de entrada G con respecto a S. Se mostrarán detalles más adelante.

La conectividad del grafo puede ser representada por una matriz de adyacencia. Considere un grafo que tiene n vértices, los cuales son numerados $0,1,\ldots,n-1$. Una matriz de adyacencia n x n A puede ser formada con datos de entrada $A[i,j]$ con valor 0 o 1. Si $A[i,j] = 0$, entonces no hay conexión desde el vértice i al vértice j. Si $A[i,j] = 1$ entonces hay al menos una conexión desde el vértice i al vértice j. Los arcos no dirigidos son registrados una única vez en la matriz. La matriz de adyacencia para el grafo de la figura 3.3 es mostrada en la figura 3.4.

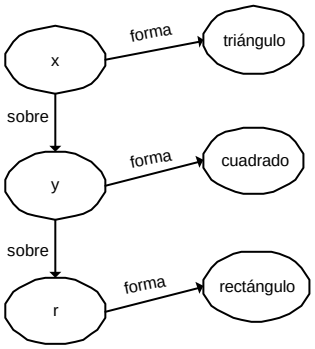

Figura 3.3 Ejemplo MDL.

$$
\begin{array}{r}
\textbf{x} \\
\textbf{triangulo} \\
\textbf{y} \\
\textbf{cuadrado} \\
\textbf{r} \\
\textbf{rectángulo}
\end{array}
\begin{pmatrix}
0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

Figura 3.4 Ejemplo de una matriz de adyacencia para un grafo.

La codificación del grafo consiste en los siguientes pasos. Se asume que el decodificador tiene una tabla de las l_u etiquetas únicas del grafo original G.

1. Determinar el número de *bits* *vbits* necesarios para codificar las etiquetas de los vértices del grafo. Primero, se necesitan (lg v) *bits* para codificar el número de vértices v del grafo. La notación lg representa en el documento $\lg_2$ (logaritmo base 2). Después, para codificar las etiquetas de todos los v vértices se requieren (v lg l_u) *bits*. Se asume que los vértices son especificados en el mismo orden en el que aparecen en la matriz de adyacencia. El número total de *bits* para codificar las etiquetas de los vértices es:

   $$vbits = \lg v + v \lg l_u$$

   Por ejemplo en la figura 3.3, v = 6, y se asume que existen l_u = 8 etiquetas únicas en el grafo original. El número de *bits* necesarios para codificar los vértices es lg 6 + 6 lg 8 = 20.58 *bits*, lo que se redondea a 21 *bits*.

2. Determinar el número de *bits rbits* necesarios para codificar los renglones de la matriz de adyacencia *A*. Típicamente, en grafos de gran tamaño, un vértice tiene arcos hacia un pequeño porcentaje de los vértices del grafo. De tal forma, un renglón típico en la matriz de adyacencia tendrá menos uno's que v, donde v es el número total de vértices en el grafo. Se aplica una variante del esquema de codificación usado por [31] para codificar cadenas de *bits* de longitud n consistiendo de k uno's y $(n - k)$ cero's, donde $k << (n - k)$. En nuestro caso, el renglón i $(1 ? i ? v)$ puede ser representado como una cadena de *bits* de longitud v conteniendo k_i uno's. Si asumimos $b = \max_i k_i$, entonces el i^{th} renglón de la matriz de adyacencia puede ser codificado de la manera siguiente:

   a) Codificar el valor de k_i requiere lg $(b + 1)$ *bits*.

   b) Asumiendo que solo k_i uno's ocurren en el renglón de cadenas de *bits* de longitud v, solo $\binom{v}{k_i}$ cadenas de cero's y uno's son posibles. Dado que todas estas cadenas tiene probabilidades iguales de ocurrencia, lg $\binom{v}{k_i}$ *bits* son necesarios para codificar las posiciones de uno's en el renglón i. El valor de v es conocido a partir de la codificación del vértice.

Finalmente, se necesitan lg $(b + 1)$ *bits* adicionales para codificar el número de *bits* necesarios para especificar el valor k_i en cada renglón. La longitud total de codificación en *bits* para la matriz de adyacencia es:

$$rbits = \lg(b+1) + \sum_{i=1}^{v} \lg(b+1) + \lg \binom{v}{k_i}$$

$$= (v+1)\lg(b+1) \sum_{i=1}^{v} \lg \binom{v}{k_i}$$

Para el ejemplo de la figura 3.3, $b = 2$, y el número de *bits* necesarios para codificar la matriz de adyacencia es $(7 \lg 3) + \lg \binom{6}{2} + \lg \binom{6}{0} + \lg \binom{6}{2} + \lg \binom{6}{0} + \lg \binom{6}{1} + \lg \binom{6}{0} = 21.49$ *bits*, lo que se redondea a 22 *bits*.

3. Determinar el número de *bits* *ebits* necesarios para codificar los arcos representados por los datos de entrada $A[i,j] = 1$ de la matriz de adyacencia A. El número de *bits* necesarios para codificar el dato de entrada $A[i,j]$ es $(\lg m) + e(i,j)[1 + \lg l_u]$, donde $e(i,j)$ es el número de arcos entre el vértice i y j en el grafo y $m = \max_{i,j} e(i,j)$. Los $(\lg m)$ *bits* son los necesarios para codificar el número de arcos entre el vértice i y j, y los $[1 + \lg l_u]$ *bits* son los necesarios para codificar por cada arco la etiqueta del arco, sea este un arco dirigido o no dirigido. Adicionalmente para codificar los arcos, se necesita codificar el número de *bits* $(\lg m)$ necesarios para especificar el número de arcos por dato de entrada. La codificación total de los arcos es:

$$ebits = \lg m \sum_{i=1}^{v} \sum_{j=1}^{v} \lg m + e(i,j)[1 + \lg l_u]$$

$$= \lg m + e(1 + \lg l_u) + \sum_{i=1}^{v} \sum_{j=1}^{v} A[i,j] \lg m$$

$$= e(1 + \lg l_u) + (K + 1) \lg m$$

donde e es el número de arcos en el grafo, y K el número de uno's en la matriz de adyacencia A. Para el ejemplo en la figura 3.3, $e = 5$; $m = 1$, $l_u = 8$, y el número de *bits* necesarios para codificar los arcos es 5 (1 + lg 8) + 6 lg 1 = 20 *bits*.

La codificación total del grafo toma (*vbits* + *rbits* + *ebits*) *bits*. Para el ejemplo de la figura 3.3, el valor es 62.07 *bits*, lo que se redondea a 63 *bits*.

Tanto el grafo de entrada como la subestructura descubierta pueden codificarse usando el esquema presentado. Después de que una subestructura es descubierta, cada instancia de la subestructura en el grafo de entrada es reemplazada por un único vértice representativo de toda la estructura. La estructura descubierta es representada en $I(S)$ *bits*, y el grafo después del reemplazo de la subestructura es representado en $I(G|S)$ *bits*. SUBDUE busca para la subestructura S en el grafo G minimizando $I(S) + I(G|S)$. Todas las subestructuras se evalúan con este método y se elige la que mejor comprima el grafo.

3.4.3 Comparación inexacta de grafos.

Aunque las comparaciones exactas de estructuras pueden usarse para encontrar subestructuras, la mayoría de las subestructuras aparecen con formas diferentes en los datos. Estas diferencias pueden ser debido a ruido y/o distorsiones, o simplemente pueden ilustrar diferencias entre las instancias de la misma clase general de estructuras.

Dado un grafo de entrada y un conjunto de estructuras definidas, se quiere encontrar aquellos subgrafos del grafo de entrada que de manera muy cercana reconstruyan dichas estructuras, así mismo se quiere asociar una medida de similitud con un par de grafos formados de una subestructura definida y un subgrafo del grafo de entrada. Para ello se adopta el enfoque de comparación inexacta de grafos de Bunke y Allermann[4].

En este enfoque de comparación inexacta, a cada distorsión de un grafo se le asigna un costo. Una distorsión se describe en términos de transformaciones básicas tales como borrado, inserción y sustitución de vértices y arcos. El costo de borrar (o insertar) un vértice con etiqueta *A* es denotado por *DELVERTICE*(*A*) ó *INSVERTICE*(*A*). El costo de sustituir un vértice con etiqueta *A* por un vértice con etiqueta *B* es denotado por *SUBVERTICE*(*A,B*). Similarmente, *SUBARCO*(*a,b*) es el costo de la sustitución de un arco. La definición de costos para el borrado e inserción de arcos es un poco más difícil dado que estas transformaciones son dependientes de sus vértices bordes. Cuando un vértice es borrado, todos sus arcos entrantes y salientes también desaparecen. Cuando se inserta un vértice, generalmente hay arcos que deben ser insertados para acoplar adecuadamente el nuevo vértice al grafo. Como consecuencia, se define *DELARCO*(*a*) e *INSARCO*(*a*) como los costos de borrado e inserción de un arco con etiqueta *a* mientras ninguno de sus vértices bordes sea borrado o insertado. Adicionalmente, *DELARCO'*(*a*) e *INSARCO'*(*a*) son los costos correspondientes para el borrado e inserción de arcos cuando al menos uno de sus vértices bordes ha sido borrado o insertado. Costos de sustitución específicos ofrecen a los usuarios un medio para expresar un conocimiento acerca del dominio del problema. La figura 3.5 muestra un ejemplo: DELVERTICE(*cuadrado*) indica la operación de borrar el vértice *cuadrado*, INSVERTICE(*cuadrado*) indica la operación de insertar el vértice

cuadrado, SUBVERTICE(*rectángulo*, *triángulo*) indica la operación de sustituir el vértice *rectángulo* por el vértice *triángulo*, SUBARCO(*forma*, *figura*) indica la operación de sustituir el arco *forma* por el arco *figura*, INSARCO(*sobre*) indica la operación de insertar el arco *sobre* siempre y cuando ninguno de sus vértices sea insertado, DELARCO(*forma*) indica la operación de borrar el arco *forma* siempre y cuando ninguno de sus vértices sea borrado, INSARCO'(*forma*) indica la operación de insertar el arco *forma* cuando al menos uno de sus vértices ha sido insertado y por último DELARCO'(*sobre*) indica la operación de borrar el arco *sobre* cuando al menos uno de sus vértices ha sido borrado.

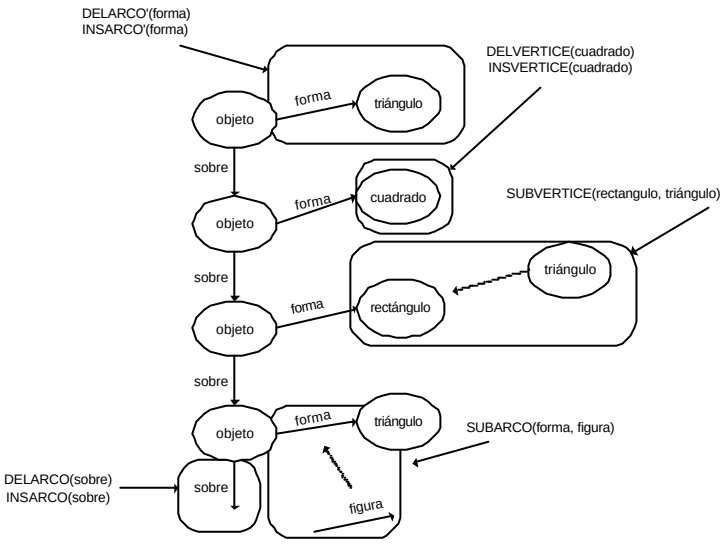

Figura 3.5 Ejemplos de distorsiones en un arco.

3.4.4 Descubrimiento de conceptos jerárquicos.

Después de que una subestructura es descubierta, cada instancia de la subestructura en el grafo de entrada es reemplazada por un vértice único que representa a toda la subestructura. El proceso de descubrimiento puede entonces repetirse sobre el conjunto de datos

comprimidos, resultando en nuevas estructuras. Si las nuevas subestructuras descubiertas son definidas en términos de conceptos existentes, las definiciones de las subestructuras forman una jerarquía de conceptos. La figura 3.6 muestra un ejemplo, la subestructura Sub_4 es definida por las subestructuras Sub_3 y Sub_2, y esta última por Sub_1.

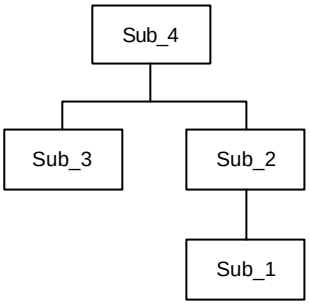

Figura 3.6 Jerarquía de conceptos.

Una vez que SUBDUE selecciona una subestructura, todos los vértices que conforman las instancias de la subestructura son reemplazadas en el grafo por un vértice único representante de la subestructura descubierta. Los arcos que conectan vértices fuera de la instancia a vértices dentro de la instancia ahora se conectan al nuevo vértice (información sobre los puntos terminales de cada arco dentro de la subestructura debe ser registrada y actualizada). Los arcos internos de la subestructura son removidos.

Mientras que el reemplazo de las porciones del grafo de entrada con las subestructuras descubiertas comprime los datos, y ofrecen una base para el descubrimiento de conceptos jerárquicos en los datos, el procedimiento de reemplazo de la subestructura se convierte más complicado cuando conceptos no tan claros son encontrados. Cuando instancias inexactas de un concepto descubierto son reemplazadas por un vértice único en la base de

datos, todas las distorsiones del grafo (diferencias entre el grafo instanciado y la definición de la subestructura) deben ser registradas como anotaciones a la etiqueta vértice. Esto añade complejidad a la rutina de comparación del grafo aplicado en ciclos de procesamiento posteriores en la base de datos.

3.4.5 Conocimiento previo del dominio.

Dado que SUBDUE es apropiado para encontrar subestructuras que son irrelevantes al dominio, el conocimiento del dominio puede ser agregado para guiar el proceso de descubrimiento y para separar las subestructuras importantes de las irrelevantes. La compresión usando conocimiento del dominio puede incrementar las posibilidades de encontrar subestructuras interesantes y generar compresiones mayores.

El conocimiento del dominio se representa usando jerarquías. Los nodos de la jerarquía pueden ser clasificados como primitivos y no primitivos. Los nodos primitivos residen en los niveles inferiores y todos los nodos no primitivos residen en los niveles más altos de la jerarquía. Los nodos primitivos representan elementos básicos del dominio del conocimiento y los nodos no primitivos representan subestructuras que consisten de una conglomeración de nodos primitivos y/o nodos no primitivos de nivel más bajo. Las ramas de la jerarquía representan una relación jerárquica entre las subestructuras y sus componentes. La jerarquía para un dominio en particular es definida por un experto en el dominio. Las subestructuras en la jerarquía y sus funcionalidades son conocidas en el contexto del dominio. Con este enfoque SUBDUE realiza el proceso de descubrimiento usando ambas heurísticas, independiente del dominio y conocimiento del dominio.

SUBDUE será utilizado para aplicar minería de datos no espaciales. La búsqueda de subestructuras representativas permitirá conocer patrones existentes en la base de datos del volcán. Por ejemplo, podríamos conocer cuántas carreteras en una zona específica (correspondiente a una población aledaña al volcán) son de tipo terracería, lo cual podría generar un cuello de botella en caso de una evacuación. En el capítulo siguiente se explica la base de datos y la arquitectura del sistema desarrollado para aplicar minería de datos espaciales y no espaciales.