BÚSQUEDA SEMÁNTICA EN UN REPOSITORIO DE PROCESOS DE NEGOCIO

DANIEL FELIPE RIVAS BURBANO
DAVID SANTIAGO CORCHUELO CASTRO

ANEXO No 3.

ALGORITMOS EMPLEADOS EN LOS PROCESOS DE TRASFORMACIÓN E INDEXACIÓN DE MODELOS DE PROCESOS DE NEGOCIO

Director: Dr. JUAN CARLOS CORRALES

UNIVERSIDAD DEL CAUCA

FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES

DEPARTAMENTO DE TELEMÁTICA

POPAYÁN

2010

ANEXO No 3. ALGORITMOS EMPLEADOS EN LOS PROCESOS DE TRASFORMACIÓN E INDEXACIÓN DE MODELOS DE PROCESOS DE NEGOCIO

Algoritmo 1. TDA: Algoritmo de detección de trazas

INPUT: Graph G; **OUTPUT**: Traces Set

```
1. GET Start Node i
2. ADD i to SetVisitedNodes
3. APPLY the AdjacencyFunction (i) and ADD n to SetNeighbors
4. IF SetNeighbors > 1 THEN
5.     ADD SetVisitedNodes to BackTrace
6. IF SetNeighbors = 0 THEN END
7. WHILE NodesToVisit > 0 THEN
8.             GET (j) the last node ADDED to NodesToVisit
9.             IF (j) is the Last node of G THEN
10.                ADD (j) to SetVisitedNodes
11.                ADD SetVisitedNodes to TracesSet
12.                IF BackTrace > 0 THEN
13.                   GET Last Set of BackTrace
14.                      REPLACE SetVisitedNodes FOR BackTrace
15.                ELSE   RETURN TracesSet
16.            ELSE
17.            ADD (j) to SetVisitedNodes
18.            APPLY the AdjacencyFunction (j) ADD m to SetNeighbors
19.            IF m > 1 THEN
20.                ADD SetVisitedNodes to BackTrace
21. END WHILE
```

El algoritmo TDA está basado en 4 conjuntos: conjunto de nodos visitados, nodos vecinos, nodos recorridos (BackTrace) en la traza y conjunto de trazas. El conjunto de nodos contiene todos los nodos visitados por el algoritmo; el conjunto de nodos vecinos comprenden los nodos que el algoritmo debería visitar. El BackTrace contiene el conjunto de nodos conectores que tienen la condición para evaluación. Y el conjunto de trazas contiene las trazas de nodos.

Primero el algoritmo obtiene el nodo de inicio y lo adiciona en el conjunto de nodos visitados. Entonces, este aplica una función de adyacencia y adiciona los nodos a visitar en el conjunto de nodos vecinos (Línea 3). Si hay más de un nodo en el conjunto de nodos vecinos, entonces este adiciona el conjunto de nodos visitados en el Backtrace (Línea 5). Si no hay nodos en el conjunto de nodos vecinos entonces finaliza (el primer nodo no tiene nodos vecinos) (Línea 6). Mientras haya nodos en el conjunto de nodos vecinos, el algoritmo obtiene el último nodo adicionado (j) y analiza si este es el último nodo del grafo. Si no es el ultimo (existe una traza), el algoritmo adiciona (j) en el conjunto

de nodos visitados y coloca los nodos visitados en el TracesSet (Línea 11). Por otro lado, si el BackTrace contiene nodos, el algoritmo obtiene el último nodo coloca y reemplaza el conjunto de nodos visitados con el último conjunto de nodos del BackTrace (Línea 14).

Si el BackTrace está vacío se retorna el conjunto de trazas (el algoritmo retorna el resultado: trazas de grafos). Si (j) no es el último nodo del grafo, el algoritmo adiciona (j) en el conjunto de nodos visitados y aplica la función de adyacencia sobre (j) y adiciona los nodos vecinos (m) en el conjunto de nodos vecinos (Línea 18). Si (m) es mayor que uno, el algoritmo adiciona el conjunto de nodos visitados en el BackTrace (Línea 20).

Ejemplo:

La Figura 12 muestra un grafo con dos conectores *F* de tipo Split e *I* de tipo Join. Primero el algoritmo obtiene el nodo de inicio *A* y lo adiciona en el conjunto de nodos visitados. Entonces, es aplicada la función de adyacencia y se adiciona los nodos *B* y *F* en el conjunto de nodos vecinos. Como hay más de un nodo en el conjunto de nodos vecinos, entonces el algoritmo adiciona *A* en el BackTrace. Mientras haya más nodos en el conjunto de nodos vecinos, entonces el algoritmo obtiene el último nodo adicionado *(F)* y analiza si *F* es el último nodo del grafo. En este caso *F* no es el último nodo del grafo, entonces el algoritmo adiciona a *(F)* en el conjunto de nodos visitados y aplica la función de adyacencia sobre *(F)* y adiciona los nodos vecinos *(G y H)* en el conjunto de nodos vecinos. Como hay más de un nodo adyacente, el algoritmo adiciona el conjunto de nodos visitados *(A,F)* en el BackTrace (Fila 0 y 1 de la Tabla 1).

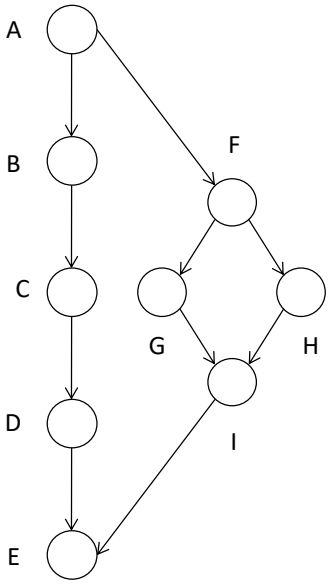

Figura 1. Grafo de prueba

Como hay nodos en el conjunto de nodos vecinos, el algoritmo obtiene el último nodo adicionado *(H)* y analiza si *H* es el último nodo del grafo. En este caso *H* no es el último nodo del grafo, entonces el algoritmo adiciona *(H)* en el conjunto de nodos visitados y aplica la función de adyacencia sobre *(H)* y adiciona el nodo vecino *(I)* en el conjunto de

nodos vecinos. Como hay un solo nodo de adyacencia, el algoritmo no adiciona el conjunto de nodos visitados en el BackTrace (Fila 2 de la Tabla 1).

Como hay nodos en el conjunto de nodos vecinos, el algoritmo obtiene el último nodo adicionado *(I)* y analiza si *I* es el último nodo del grafo. En este caso *I* no es el último nodo del grafo, entonces el algoritmo adiciona *(I)* en el conjunto de nodos visitados y aplica la función de adyacencia sobre *(I)* y adiciona el nodo vecino *(E)* en el conjunto de nodos vecinos. Como hay un solo nodo de adyacencia, el algoritmo no adiciona el conjunto de nodos visitados en el BackTrace (Fila 3 de la Tabla 1).

Como hay nodos en el conjunto de nodos vecinos, el algoritmo obtiene el último nodo adicionado *(E)* y analiza si *E* es el último nodo del grafo. En este caso *E* es el último (aquí se dice que existe una traza), el algoritmo adiciona *(E)* en el conjunto de nodos visitados y coloca los nodos visitados en el conjunto de trazas *(A,F,H,I,E)* (Fila 4 de la tabla). Por otro lado, como el BackTrace contiene nodos, el algoritmo obtiene el último conjunto de nodos *(A,F)* y reemplaza el conjunto de nodos visitados con el último conjunto de nodos del BackTrace *(A,F)* (Fila 5 de la Tabla 1). En este caso el BackTrace no está vacío y hay nodos en el conjunto de nodos vecinos, entonces el algoritmo obtiene el último nodo adicionado *(G)* y analiza si *G* es el último nodo de grafo. En este caso *G* no es el último nodo de grafo, entonces el algoritmo adiciona *G* en el conjunto de nodos visitados y aplica la función de adyacencia sobre *(G)* y adiciona el nodo vecino *(I)* en el conjunto de nodos vecinos. Como hay un nodo adyacente, el algoritmo no adiciona el conjunto de nodos visitados en el BackTrace (Fila 6 de la Tabla 1).

Como hay nodos en el conjunto de nodos vecinos, el algoritmo obtiene el último nodo adicionado *(I)* y analiza si *I* es el último nodo del grafo. En este caso *I* no es el último nodo del grafo, entonces el algoritmo adiciona *(I)* en el conjunto de nodos visitados y aplica la función de adyacencia sobre *(I)* y adiciona el nodo vecino *(E)* en el conjunto de nodos vecinos. Como hay un solo nodo de adyacencia, el algoritmo no adiciona el conjunto de nodos visitados en el BackTrace (Fila 7 de la Tabla 1).

Como hay nodos en el conjunto de nodos vecinos, el algoritmo obtiene el último nodo adicionado *(E)* y analiza si *E* es el último nodo del grafo. En este caso *E* es el último (aquí se dice que existe una traza), el algoritmo adiciona *(E)* en el conjunto de nodos visitados y coloca los nodos visitados en el conjunto de trazas *(A,F,H,I,E)* (Fila 8 de la Tabla 3). Por otro lado, como el BackTrace contiene nodos, el algoritmo obtiene el último conjunto de nodos *(A)* y reemplaza el conjunto de nodos visitados con el último conjunto de nodos del BackTrace *(A)* (Fila 9 de la Tabla 1). En este caso el BackTrace no está vacío y hay nodos en el conjunto de nodos vecinos, entonces el algoritmo obtiene el último nodo adicionado *(B)* y analiza si *B* es el último nodo de grafo. En este caso *B* no es el último nodo de grafo, entonces el algoritmo adiciona *B* en el conjunto de nodos visitados y aplica la función de adyacencia sobre *(B)* y adiciona el nodo vecino *(C)* en el conjunto de nodos vecinos. Como hay un nodo adyacente, el algoritmo no adiciona el conjunto de nodos visitados en el BackTrace (Fila 10 de la Tabla 1).

Como hay nodos en el conjunto de nodos vecinos, el algoritmo obtiene el último nodo adicionado *(C)* y analiza si *C* es el último nodo del grafo. En este caso *C* no es el último nodo del grafo, entonces el algoritmo adiciona *(C)* en el conjunto de nodos visitados y aplica la función de adyacencia sobre *(D)* y adiciona el nodo vecino *(D)* en el conjunto de nodos vecinos. Como hay un solo nodo de adyacencia, el algoritmo no adiciona el conjunto de nodos visitados en el BackTrace (Fila 11 de la Tabla 1).

Como hay nodos en el conjunto de nodos vecinos, el algoritmo obtiene el último nodo adicionado *(D)* y analiza si *D* es el último nodo del grafo. En este caso *D* no es el último nodo del grafo, entonces el algoritmo adiciona *(D)* en el conjunto de nodos visitados y aplica la función de adyacencia sobre *(E)* y adiciona el nodo vecino *(E)* en el conjunto de nodos vecinos. Como hay un solo nodo de adyacencia, el algoritmo no adiciona el conjunto de nodos visitados en el BackTrace (Fila 12 de la Tabla 1).

Como hay nodos en el conjunto de nodos vecinos, el algoritmo obtiene el último nodo adicionado *(E)* y analiza si *E* es el último nodo del grafo. En este caso *E* es el último (aquí se dice que existe una traza), el algoritmo adiciona *(E)* en el conjunto de nodos visitados y coloca los nodos visitados en el conjunto de trazas *(A,B,C,D,E)* (Fila 8 de la tabla). Por otro lado el BackTrace está vacío, entonces el algoritmo retorna el conjunto de trazas *(A,F,H,I,E)* y *(A,B,C,D,E)* (Fila 13 de la Tabla 1).

	Conjunto de nodos visitados	**Conjunto de nodos vecinos**	**Conjunto de nodos BackTrace**	**Conjunto de Trazas**
0	A	B,F	(A)	-
1	A,F	B,G,H	(A), (A,F)	
2	A,F,H	B,G,I	(A), (A,F)	
3	A,F,H,I	B,G,E	(A), (A,F)	
4	A,F,H,I,E*	B,G	(A), (A,F)	(A,F,H,I,E)
5	A,F	B,G	(A)	(A,F,H,I,E)
6	A,F,G	B,I	(A)	(A,F,H,I,E)
7	A,F,G,I	B,E	(A)	(A,F,H,I,E)
8	A,F,G,I,E**	B	(A)	(A,F,H,I,E) (A,F,G,I,E)
9	A	B	-	(A,F,H,I,E) (A,F,G,I,E)
10	A,B	C	-	(A,F,H,I,E) (A,F,G,I,E)
11	A,B,C	D	-	(A,F,H,I,E) (A,F,G,I,E)
12	A,B,C,D	E	-	(A,F,H,I,E) (A,F,G,I,E)
13	A,B,C,D,E***	-	-	(A,F,H,I,E) (A,F,G,I,E) (A,B,C,D,E)

Tabla 1. Ejecución del algoritmo.

Descripción del algoritmo VF2

VF2 es un algoritmo de isomorfismo de grafos, el cual es un método de correspondencia determinístico que permite verificar correspondencias entre grafos. A través de este algoritmo es posible realizar las comparaciones entre diferentes grafos y determinar cual grafo es exactamente igual a otro en cuanto a su estructura de nodos y aristas (isomorfismo), ó cual grafo está contenido dentro de otro (subgrafo). El algoritmo es válido mientras que no haya restricciones impuestas a la topología de los grafos.

Algoritmo 2. VF2: Algoritmo de isomorfismo de grafos

PROCEDURE Match (s)
INPUT: an intermediate state s; the initial state; the initial state s_0 has $M(s_0) = \emptyset$
OUTPUT: the mappings between two graphs

IF $M(s)$ covers all the nodes of G_2 **THEN**
OUTPUT $M(s)$
ELSE
Compute the set $P(s)$ of the pairs candidate for inclusion in $M(s)$
FOREACH p in $P(s)$
IF the feasibility rules succeed for the inclusion of p in $M(s)$ **THEN**
Compute the state's obtained by adding p to $M(s)$
CALL $Match(s')$
END IF
END FOREACH
Restore data structures
END IF
END PROCURE MATCH

Un proceso de correspondencia entre 2 grafos $G_1 = (N_1, B_1)$ y $G_2 = (N_2, B_2)$ consiste en la determinación de un mapeo M en donde se asocia los nodos de G_1 con los nodos de G_2 y viceversa, de acuerdo a algunas restricciones predefinidas. Generalmente, el mapeo M es expresado como el conjunto de pares (n, m) (con $n \in G_1$ y $m \in G_2$) cada uno representando el mapeo de un nodo n de G_1 con un nodo m de G_2. Un mapeo $M \subset N_1 \times N_2$ se dice ser isomorfo si y solo sí M es una función biyectiva que preserva la estructura de ramas de 2 grafos. Un mapeo $M \subset N_1 \times N_2$ se dice ser un isomorfismo de grafo y subgrafo si y solo sí M es un isomorfismo entre G_2 y un subgrafo de G_1. El proceso de encontrar la función de mapeo puede ser apropiadamente descrita por medio de una representación estado-espacial (SSR)[1]. Cada estado s del proceso de correspondencia puede ser asociado a una solución de un mapeo parcial $M(s)$, que contiene solo un subconjunto de M. $M(s)$ unívocamente identifica 2 subgrafos de G_1 y G_2, dicho como $G_1(s)$ y $G_2(s)$, obtenidos por la selección de G_1 y G_2 solamente los nodos incluidos en $M(s)$, y las ramas que los conectan. Ahora, se denotará a $M_1(s)$, $M_2(s)$, $B_1(s)$ y $B_2(s)$ el conjunto de nodos de $G_1(s)$ y $G_2(s)$ y las correspondientes aristas. De acuerdo a esta definición, una

[1] SSR (State Space Representation) es un modelo matemático de un sistema físico como un conjunto de entradas, salidas y variables de estado relacionadas por ecuaciones diferencias de primer orden.

transición de un estado genérico s a un sucesor s' representa la adición de grafos parciales asociados a s en el SSR, de un par de (n, m) nodos concurrentes. Entre todos los posibles estados SSR, solamente un pequeño subconjunto es consistente con el tipo de estructura requerida, en el sentido que no hay condiciones que imposibiliten la posibilidad de alcanzar una solución completa. La consistencia de la condición puede ser verificada, en caso de isomorfimo o isomorfismo de subgrafos, cuando los grafos parciales $G_1(s)$ y $G_2(s)$ asociados a $M(s)$ son isomórficos.