Tesis de Licenciatura:
“Implementación de algoritmos de Reconocimiento de Grafos Arco-Circulares Unitarios”

Tesista: Javier F. Burgos
Director: Dr. Min Chih Lin - UBA
Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Diciembre de 2007

Resumen

Los grafos arco circulares unitarios son los grafos de intersección de arcos alrededor de un círculo, en donde todos los arcos tienen la misma longitud y no hay dos arcos cuyos extremos coincidan en el mismo punto. En esta tesis se describen las caracterizaciones de la clase de grafos arco circulares unitarios enunciadas por distintos autores. Se implementan algoritmos con certificados para determinar si un grafo pertenece o no a esta clase; tanto el reconocimiento como la construcción de los respectivos certificados tienen complejidad temporal de orden lineal. Luego se desarrollan e implementan algoritmos para autenticar los certificados emitidos.

Palabras clave: grafos arco circulares unitarios, algoritmos de complejidad temporal de orden lineal, algoritmos de reconocimiento con certificados, modelos arco circulares unitarios.

Abstract

Unit circular-arc graphs are the intersection graphs of arcs surrounding a circle, where all arcs have the same length and no pair of arcs share the same endpoints in the circle. This thesis describe characterizations for the class of unit circular-arc graphs given by several authors. Then algorithms with certificate for determining whether a graph is a unit circular-arc graph are implemented. Both the recognition problem and the certificate building have linear time complexity. Also authentication algorithms for the certificates are presented, described and implemented.

Keywords: unit circular-arc graphs, linear-time complexity algorithms, certified recognition algorithms, unit circular-arc models.

INDICE

Capítulo 1

Introducción

El problema de reconocer si un grafo es un grafo de intervalos jugó un rol central para determinar la topología lineal del ADN en los años 50's [26]. El problema de encontrar un conjunto independiente máximo, que muchas veces también se lo conoce como el problema de scheduling de tareas, y el de mínimo coloreo propio en un grafo de intervalos, son muy importantes en la Teoría de scheduling, donde los intervalos representan intervalos de tiempo ocupados por tareas.

Un grafo G es un grafo de intervalos si existe una representación o modelo de intervalos, que consista en una línea recta real y una familia de intervalos sobre la recta tal que G sea el grafo intersección de este modelo.

Un grafo es llamado arco-circular si admite un modelo arco-circular que está compuesto por un círculo y una familia de arcos alrededor del círculo, y el grafo sea el grafo de intersección de este modelo. Claramente, todos los grafos de intervalos son grafos arco-circulares pues cualquier modelo de intervalos se puede transformar trivialmente a un modelo de arco-circular uniendo los dos extremos de la recta.

Los grafos arco circulares han sido estudiados desde 1964. Pero a partir de la caracterización que hizo Alan Tucker en los '70, de estos y de algunas de sus subclases, han recibido considerable atención. Hay aplicaciones de grafos arco circulares en áreas tales como genética y control de tráfico, entre otras. Entre las subclases de grafos arco circulares se encuentran los grafos arco circulares propios (PCA por sus siglas en inglés) y los grafos arco circulares unitarios (UCA).

Llamamos reconocimiento de una clase de grafos al problema de decidir si un grafo *G* pertenece, o no, a esa clase. Certificar ese reconocimiento es devolver una estructura asociada al grafo *G*, llamada certificado, que avala el reconocimiento de pertenencia / no pertenencia de *G* a la clase en cuestión. Por ejemplo para certificar que un grafo *G* dado no es bipartito, serviría un subgrafo de *G* que sea un ciclo de longitud impar (certificado negativo) y para certificar que *G* es bipartito, una bipartición de los vértices de *G* que carece de aristas intraparticiones sería suficiente (certificado positivo). Para los grafos arco circulares y sus subclases, se han publicado varias caracterizaciones y algoritmos de reconocimiento. En particular se han descrito algoritmos de tiempo lineal de reconocimiento con certificados para las clases PCA y UCA mencionadas. Cabe destacar que el certificado positivo usual para el reconocimiento de los grafos arco circulares y sus subclases, es precisamente un modelo arco circular que cumple con la propiedad de la definición.

En cambio, el certificado negativo dependerá de las caracterizaciones particulares de cada subclase. Certificar algoritmos de reconocimiento es en la práctica muy recomendable, si un algoritmo fue probado teóricamente su implementación pudo contener bugs, tener ambos certificados nos sirve de evidencia de que la respuesta obtenida no fue comprometida por ningún bug en la implementación.

La historia de reconocer y construir certificados para los grafos arco circulares unitarios comienza en 1974, cuando Tucker [21] dio una caracterización por subgrafos inducidos prohibidos que permite identificar cuáles de los grafos arco-circulares propios son también unitarios. En ese trabajo propuso un algoritmo que dado un modelo arco-circular propio del grafo en cuestión, genera otro modelo unitario equivalente del grafo, siempre y cuando el grafo sea arco-circular unitario. Este algoritmo propuesto finalmente no resulta ser de tiempo polinomial, ya que manipula números enteros muy grandes en relación al tamaño del input.

El primer algoritmo de reconocimiento de tiempo polinomial para grafos arco-circulares unitarios fue dado en 2006 por Durán, Gravano, McConnell, Spinrad y Tucker [5]. Ellos propusieron en este trabajo calcular dos parámetros en tiempo polinomial a partir de un modelo arco-circular propio dado y, de acuerdo al resultado de comparar entre sí estos dos valores, determinar si el grafo en cuestión es unitario o no. La principal deficiencia de este algoritmo es que no puede brindar un modelo unitario cuando el grafo es arco-circular unitario ni un certificado negativo en el caso contrario.

En el mismo año, Lin y Szwarcfiter presentaron un algoritmo de reconocimiento lineal para esta clase de grafos [11] que permite además construir un modelo arco circular unitario también en tiempo lineal con los extremos de su arcos de tamaño $O(n)$, donde n es la cantidad de vértices. Este algoritmo plantea un sistema de ecuaciones sobre las longitudes de los arcos. Para resolver este sistema de ecuaciones en forma eficiente, dicho sistema es reducido a un caso especial del problema de flujo llamado weakly eulerian segment digraph, también conocida como circulation graph.

También en el mismo año, otro algoritmo de reconocimiento lineal fue propuesto por Kaplan y Nussbaum [10]. Básicamente en este trabajo, se siguió con el mismo esquema de [5], pero consiguiendo calcular los dos parámetros en tiempo lineal. Además, se pudo armar una especie de certificado negativo, también en tiempo lineal cuando el grafo no es unitario. Pero si el grafo es unitario se debe utilizar el algoritmo de [11] para conseguir el modelo unitario.

Finalmente en este año, Lin y Szwarcfiter [12] extendieron el algoritmo de [11]. Esta extensión permite generar en tiempo lineal, a partir de las mismas estructuras utilizadas para obtener el modelo unitario, un certificado negativo que es en realidad un subgrafo inducido prohibido descrito por Tucker en [21].

Cabe destacar que todos los algoritmos de reconocimiento de grafos arco-circulares unitarios utilizan el algoritmo lineal de Deng, Hell y Huang [3] para reconocer si el grafo en cuestión es arco-circular propio y, en el caso afirmativo, obtener el modelo propio también en tiempo lineal que luego será tomado como input. En el caso en que el grafo no sea arco circular propio, se utiliza el algoritmo de Kaplan y Nussbaum [10] que construye un certificado negativo.

Al ser los algoritmos de complejidad temporal lineal, los certificados pueden construirse cuando el input corresponde a grafos muy grandes. Por ello, es importante contar con una forma de poder autenticar estos certificados automáticamente, para poder verificar que la respuesta sea correcta, ya que es casi imposible corroborar la autenticidad visual o manualmente. De esta forma, se puede verificar que el certificado se condice con el input recibido y que no hubo bugs en la implementación que interfirieran en su construcción.

El objetivo de esta tesis es implementar los algoritmos, descriptos por Lin y Szwarcfiter [11,12], que llevan a reconocer y certificar un grafo arco circular unitario en tiempo lineal, y medir los tiempos de las ejecuciones reales para compararlos con los resultados teóricos. Cabe destacar que al día de hoy, no existían implementaciones de los algoritmos presentados. También es objetivo de esta tesis describir e implementar algoritmos de autenticación para verificar la validez de los certificados construidos.

La estructura de esta tesis es el siguiente: el Capítulo 1 es introductorio y describe las definiciones básicas y preliminares necesarias para poder seguir este informe, mostrando también aplicaciones prácticas de esta familia de grafos. El Capítulo 2 presenta las caracterizaciones y resultados teóricos de los grafos UCA realizadas por Tucker y por Lin y Szwarcfiter, esta última más en detalle ya que es la utilizada en la implementación de los algoritmos. El Capítulo 3 hace una breve reseña desde los primeros algoritmos de reconocimiento de grafos arco circulares, hasta los algoritmos de reconocimiento con certificado UCA en tiempo lineal. Estos últimos son descriptos con un nivel de detalle suficiente como para poder visualizar su codificación y analizar su complejidad. Por último, se muestran los resultados obtenidos en las mediciones efectuadas para corroborar que los tiempos de ejecución reales se corresponden con los esperados en el análisis teórico. El Capítulo 4 describe los algoritmos para autenticar los certificados obtenidos mediante los algoritmos descriptos en el capítulo anterior, su complejidad y su implementación. Luego se analizan las mediciones efectuadas para corroborar la similitud de los tiempos de ejecución prácticos con los esperados del cálculo de la complejidad teórica. Y finalmente las conclusiones obtenidas del trabajo son presentadas en un capítulo aparte.

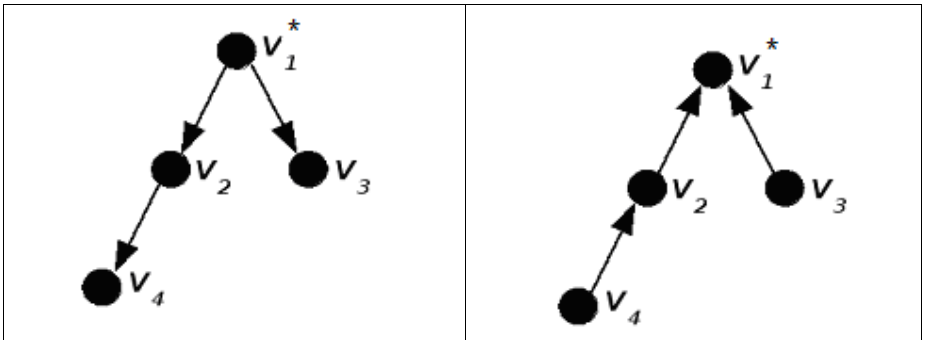
Definiciones básicas y clases de grafos, preliminares

Sea G un grafo, con $E(G)$ su conjunto de aristas y $V(G)$ su conjunto de vértices, $|V(G)| = n$ y $|E(G)| = m$. Dados $v, w \in V(G)$ notamos a la arista $e \in E(G)$ incidente a v y w como $e = (v,w)$. Si G es un digrafo se denota con la letra D y notamos la arista orientada de v a w como $v \rightarrow w$.

Denotamos $d_G(v)$ el grado de v en G, o simplemente $d(v)$ cuando G quede claro por contexto. En los digrafos usamos $d^-_D(v)$ y $d^+_D(v)$ para los *grados de entrada* y los *grados de salida* (*indegree* y *outdegree*) de v, simplificados como $d^-(v)$ y $d^+(v)$ respectivamente, cuando D quede claro por contexto. En general $E^-(v), E^+(v) \subseteq E(D)$, representan el conjunto de aristas de D saliendo y entrando de v, respectivamente.

Un digrafo D se dice que es *euleriano* cuando $d^-(v) = d^+(v)$, para todos los vértices $v \in V(D)$. Sea w una función de pesos sobre las aristas de D. Se llama $w^-(v)$ a la suma de los pesos de las aristas $e \in E(D)$ incidentes al vértice v, y $w^+(v)$ a la suma de los pesos de los ejes salientes de v. Llamaremos también *flujo entrante* a $w^-(v)$ y *flujo saliente* $w^+(v)$. Entonces se dice que D es *débilmente euleriano* (*weakly eulerian*) cuando admite un *weakly eulerian weighting,* se dice que D tiene un weakly eulerian weighting si admite una función w de peso sobre las aristas, tal que $w^-(v_i) = w^+(v_i)$ para todo $v_i \in V(D)$. Observar que todo D euleriano es débilmente euleriano utilizando $w_e = 1, \forall e \in E(G)$.

Un o*ut-arborescence* (*in-arborescence*) T de D es un árbol generador dirigido (en inglés spanning tree) de D tal que existe un vértice distinguido $v^* \in V(T)$, llamado *raíz* de T, satisfaciendo $d^-_T(v^*) = 0$ ($d^+_T(v^*) = 0$), mientras que $d^-_T(v) = 1$ ($d^+_T(v) = 1$) para todos los vértices remanentes $v \neq v^*T$. Cuando $d^+_T(v) = 0$ ($d^-_T v) = 0$), v se dice *hoja* de T. Para $u, v \in V(T)$, si T contiene un camino dirigido desde u a v, entonces u es un *ancestro* de v, y v un *descendiente* de u. En la Fig. 1.1 se muestra un ejemplo de un in-arborescence y de un out-arborescence.



out-arborescence	in-arborescence

Fig 1.1: Ejemplos de out-arborescence e in-arborescence con raíz v_1.

Un *ordenamiento hoja-raíz* de una arborescencia T es un secuencia $v_1, \ldots, v_n$ de sus vértices, tal que para todo $1 \leq i < j \leq n$ implica que v_j no es un ancestro (descendiente) de v_j en T. Por ejemplo, v_4, v_2, v_3, v_1 y v_3, v_4, v_2, v_1 son dos ordenamientos hoja-raíz del out-arborescence de la Figura 1.1.

Un grafo de intersección es un grafo que representa las intersecciones de una familia de conjuntos. Más formalmente, para una familia ***F*** de conjuntos $C_1, ..., C_n$, se define el grafo de intersección de ***F*** como el grafo que tiene un vértice v_i por cada conjunto C_i, donde v_i y v_j son adyacentes cuando C_i y C_j tienen intersección no vacía.

Un grafo G es *arco circular* (CA) si es el grafo de intersección de un conjunto de arcos alrededor de un círculo. En otras palabras, si se lo puede representar como una familia de arcos ***A*** alrededor de un círculo C, de manera tal que todo vértice es modelado por un arco y dos vértices de G son adyacentes si sólo si sus correspondientes arcos se intersecan. Al par *(C, **A**)* se lo llama *representación o modelo* arco circular de G. Se asume, sin pérdida de generalidad, que todos los arcos de C son abiertos, que ningún par de extremos de arcos distintos de ***A*** coinciden y que ningún arco cubre todo C. Cuando recorremos el círculo, vamos a suponer que lo hacemos en sentido horario a no ser que se aclare lo contrario.

Escribimos cada arco A_i como (s_i, t_i), donde s_i y t_i son los extremos de A_i; s_i es el *extremo inicial* y t_i el *extremo final* del arco. Los *extremos de **A*** son los extremos de todos los arcos de ***A***. Definimos *ordenamiento circular* de los extremos de ***A*** como la secuencia de extremos que obtenemos de recorrer C en sentido horario.

Además, consideraremos el *ordenamiento circular* de los arcos $A_1, \ldots, A_n$ que resulta del ordenamiento circular de sus extremos iniciales $s_1, \ldots, s_n$. Las sumas y restas de índices en estos ordenamientos deben entenderse módulo n.

Para *p, q puntos* de A_i, si $(s_i, p) \subseteq (s_i, q)$ entonces *p precede* a *q*, y *q* es *sucesor* de *p* en A_i. Para *p, q, t* de C, notamos $max\{p, q\}_t$ para representar el punto, *p o q*, que esté mas lejos de *t*, en el ordenamiento circular de C.

Un arco de C definido por dos extremos consecutivos de ***A*** es un *segmento* de *(C, **A**)*. Por lo tanto C es la unión de *2n* segmentos de *(C, **A**)* y de 2*n* extremos, ya que cada extremo es el inicio y el fin de un segmento (ver Fig. 1.2).

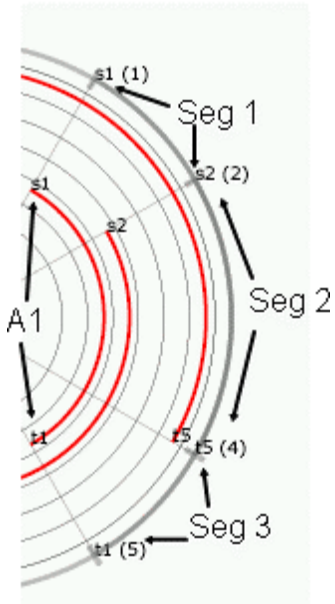


Fig. 1.2: el arco A_1 comprende a la unión de los segmentos 1, 2 y 3, contenidos entre sus extremos s_1 y t_1.

Un grafo *G* es *arco circular propio (PCA)* si existe una representación arco circular de *G* tal que ningún arco esté contenido en forma propia en otro. Estos grafos fueron caracterizados por Tucker [22] y es una de las subclase más estudiada de los grafos arco circulares.

Un grafo *G* es *arco circular unitario (UCA)* cuando se puede representar con un modelo donde todos los arcos son de igual longitud. Claramente todos los grafos UCA son grafos PCA [21]. Un *modelo normal* es un modelo PCA donde ningún par de arcos cubre el círculo. Como veremos más adelante todo grafo PCA admite un modelo normal.

Un digrafo *D* es *fuertemente conexo* si entre cada par de vértices v_i y v_j de *D*, existen dos caminos, uno de v_i a v_j y otro de v_j a v_i. Una *componente fuertemente conexa* de un digrafo es un subgrafo fuertemente conexo maximal. Los vértices de todo digrafo pueden ser particionados de forma unívoca en componentes fuertemente conexas. Dado un digrafo *D*, llamamos *fuente* a un componente fuertemente conexa *F* de *D* si no existen aristas de ($D \setminus F$) a *F* y *sumidero* a un componente fuertemente conexa *S* de *D* si no existen aristas de *S* a ($D \setminus S$).

Se llama *problema de reconocimiento*, a determinar si un grafo pertenece o no a una determinada clase de grafos. Un *reconocimiento con certificado* es un reconocimiento donde se obtiene como output un *certificado*, que es una evidencia de que el grafo pertenece a la clase en el caso positivo, o bien una evidencia de que el grafo no pertenece a la clase en el caso negativo. Para la clase de grafos arco circulares unitarios, si el grafo es reconocido positivamente, el certificado es una representación arco circular del grafo; en caso contrario el certificado consta de un conjunto de arcos, llamados *selected arcs*, que representan a un subgrafo inducido prohibido llamado *CI(n, k)*.

Una *descripción por rangos* de un conjunto finito de números enteros *S* es una familia de rangos disjuntos cuya unión es *S*. ejemplo, una descripción por rangos de $S = \{2, 6, 7, 8, 12, 13\}$ podría ser {[2..2],[6..8],[12..13]} y otra podría ser {[2..2],[6..7],[8..8],[12..12],[13..13]}. Ambas son descripciones por rangos *de S,* la cantidad de rangos de la primera es 3 y la de la segunda es 5. Una descripción con la menor cantidad de rangos posible se denomina *descripción mínima*. En este caso, la primera descripción es una descripción mínima de *S*. Claramente, cualquier conjunto finito de números enteros tiene al menos una descripción por rangos, llamada *descripción trivial,* que consiste en a cada número miembro del conjunto como un rango unitario. La descripción trivial para *S* ejemplo anterior es {[2..2],[6..6], [7..7], [8..8], [12..12],[13..13]}.

Aplicaciones de grafos UCA.

Aplicaciones de grafos arco-circulares se dan en genética [17], problemas de scheduling circular tales como el de scheduling de los semáforos para el control de tráfico [19] (ver Fig. 1.3), asignaciones de variables a registros en loops para diseño de compiladores [25], estadística [8], y problemas de almacenamiento [1]. Más recientemente, podemos citar aplicaciones en modelado de redes de fibra óptica que utilizan WDM (Wave Division Multiplexing) [18] e inteligencia artificial [2]. En [7,15,19] se puede encontrar más información sobre aplicaciones.

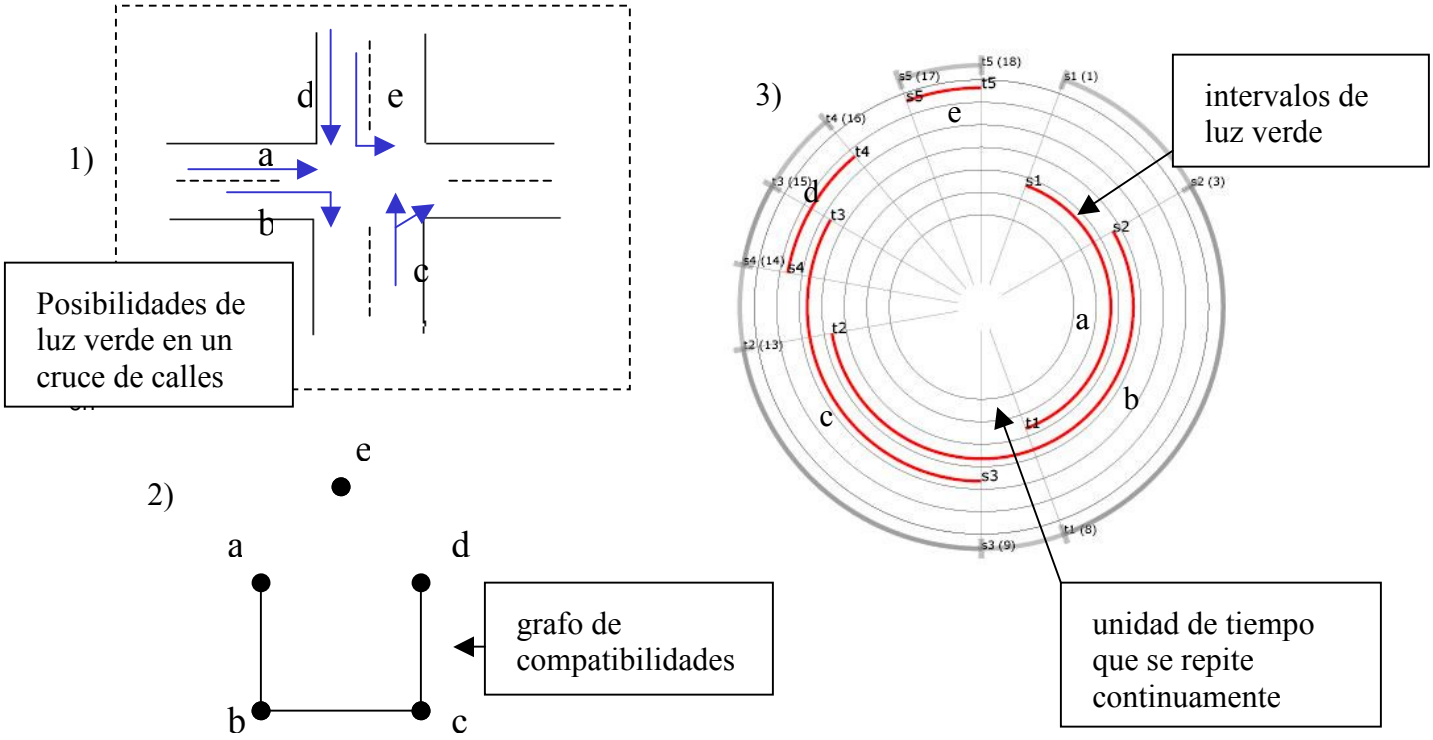


Fig. 1.3 : Ejemplo de aplicación de grafos arco circulares para control de tráfico. En 1) se muestra un cruce de calles y las posibles alternativas de cruce, en 2) el grafo que mapea las compatibilidades de cruce y en 3) el modelo arco circular que resuelve el problema. Si utilizáramos un modelo arco circular unitario todas las luces verdes deberían tener la misma duración de tiempo.

En el caso de los grafos arco circulares unitarios, estos pueden utilizarse en aplicaciones en donde se requiera distribuir algún recurso periódico como el tiempo, longitudes circulares, etc., en partes de igual magnitud, donde puede haber restricciones sobre su uso, como ser incompatibilidades de dos eventos simultáneos.

Como ejemplo de una aplicación se podría pensar en la distribución de banners publicitarios para la tv, las publicidades en el campo en los estadios de fútbol o para un sitio de internet, en donde las publicidades deban tener la misma cantidad de tiempo en el aire, ya se por cantidad de tiempo o

por cantidad de banners, se podría utilizar el algoritmo presentado en esta tesis para determinar la secuencia en que deben aparecer cada una de las publicidades ya que no todas las publicidades son compatibles entre si y no podrían salir en simultaneo (por ejemplo no se puede poner juntos una publicidad de banner1='vote al partido A' y banner2='vote al partido B' o de marcas rivales). Las restricciones se mapean en un grafo de la siguiente manera: cada publicidad representa un vértice y las publicidad que no tienen problemas de incompatibilidad entre si son representados por un eje. De esta forma se puede construir el grafo de publicidades luego se construye un modelo UCA donde cada arco indica los periodos de tiempo asignados que le corresponde a cada uno, o bien se determina que no se puede asignar a cada uno de ellos una cantidad de tiempo igual. Una opción a esto último sería utilizar el certificado negativo para dar una solución alternativa.

Capítulo 2

Base teórica de los algoritmos de reconocimiento y certificación.

En este capítulo presentamos las caracterizaciones y resultados teóricos de los grafos UCA realizadas por Tucker y por Lin y Szwarcfiter. Estos resultados motivan los algoritmos de reconocimiento, a la vez que dan una demostración de su correctitud. Como síntesis de este obtendremos una visión global del algoritmo de reconocimiento y certificación, para estar en condiciones de mostrar una implementación de complejidad temporal lineal en el capítulo siguiente.

Caracterización de Tucker

La primera caracterización de grafos arco circular unitarios (UCA) es de Tucker [21], donde se caracteriza a los grafos UCA como una subclase de los PCA mediante una familia subgrafos inducidos prohibidos. Es decir, se demuestra que un grafo PCA es UCA si y sólo si no contiene como subgrafo inducido a ningún grafo de la familia prohibida. A continuación describimos los grafos pertenecientes a la familia prohibida.

Un *(n, k)-circuito* en un modelo *(C, **A**)* es un conjunto $\{A_1,A_2,...,A_n\}$ de arcos ($n \geq 1$), donde el arco A_{i+1} comienza después del extremo inicial de A_i y antes del extremo final de A_i, y el extremo final de A_n se encuentra contenido en el arco A_1, donde además el círculo *C* es recorrido *k* veces por los arcos del (*n*, *k*)-circuito. Para contar el número de vueltas alrededor de *C*, comenzamos en el extremo final de A_1, saltando de A_1 a A_2, de A_2 a A_3, y así sucesivamente, y contamos un nuevo giro cada vez que pasamos por el punto de partida. Por ejemplo, en Fig. 2.1, los arcos A_1, A_3, A_5, y A_7 forman un (4,1)-circuito.

Un *(m, l)-conjunto-independiente* en un modelo (*C, **A***) es un conjunto $\{A_1,A_2,...,A_m\}$ de arcos *(m ≥ 1)*, donde el arco A_{i+1}comienza después del extremo final de A_i (por lo tanto no se intersecan), el arco A_n finaliza antes del extremo inicial de A_1, donde además *C* es recorrido *l* veces. Para contar el número de vueltas alrededor de C de la misma manera que antes, solamente que ahora consideramos el último giro como completo (es decir, sumamos 1 al número de giros). Por ejemplo, en Fig. 2.1, los arcos A_2, A_4, A_6 y A_8 forman un (4,1)-conjunto-independiente.

Si un modelo UCA contiene un (*n, k*)-circuito y cada arco tiene longitud 1, entonces el círculo debe tener longitud *L* a lo sumo *n/k* ya que los *n* arcos cubren una longitud superior a *k*L*. Análogamente, si un modelo UCA contiene un (*m, l*)-conjunto-independiente entonces el círculo debe tener longitud como mínimo *m/l* ya que los n arcos se encuentran contenidos en un perímetro de a lo sumo *k*L*.

Consecuentemente, un modelo UCA no puede contener a la vez un (*n*, *k*)-circuito y un (*n*, *k*)-conjunto-independiente. Se define el grafo *CI(n, k)* como el grafo que tiene un modelo PCA en donde se encuentran a la vez un (*n*, *k*)-circuito y un (*n*, *k*)-conjunto-independiente. En Fig. 2.1 se muestra el grafo *CI*(4, 1) y su modelo PCA. Tucker [21] demostró que la familia de grafos prohibidos es precisamente la familia de grafos *CI*(*n*, *k*) donde 2*n* > *k* y *n*, *k* son coprimos.

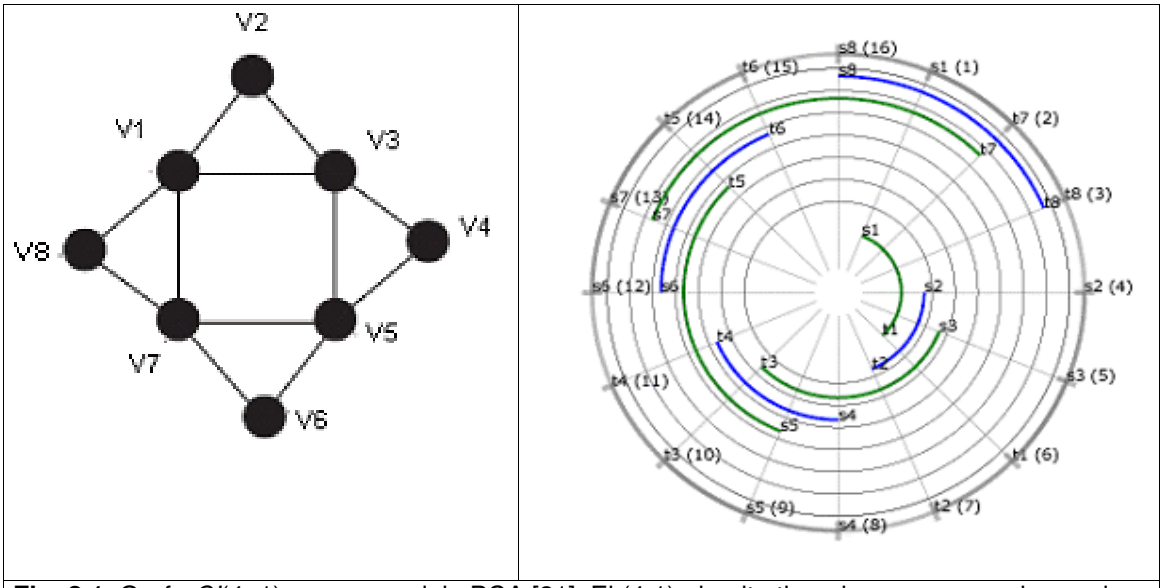


Fig. 2.1: Grafo *CI*(4, 1) con su modelo PCA [21]. El (4,1)-circuito tiene los arcos en color verde y el (4,1)-conjunto independiente en azul.

Teorema 2.1 *[5, 21]: Sea G un grafo PCA. Entonces G es UCA si y sólo si G no contiene un subgrafo inducido CI(n, k), para n > 2k y n, k coprimos.*

Este teorema es importante para la construcción del certificado negativo en el algoritmo que dan Lin y Szwarcfiter [12], en donde se identifica los arcos que pertenecen al (*n*, *k*)-circuito y al (*n*, *k*)-conjunto-independiente entre los del modelo PCA del input, y que conforman el *CI*(*n*, *k*) prohibido. El siguiente teorema, permite eliminar los (2,1)-circuitos, ya que si un modelo contiene un (2,1)-circuito, entonces tiene dos arcos cubriendo el círculo.

Teorema 2.2 *[7, 21]: Todo grafo PCA, admite un modelo normal.*

Más aún, el Teorema 2.2 es utilizado por Lin y Szwarcfiter [11], ya que se requiere normalizar el modelo al inicio del algoritmo. En [11] se presenta un nuevo algoritmo que normaliza un modelo

PCA en tiempo lineal. Este resultado es muy interesante independientemente de su rol en el reconocimiento. Previamente, Durán, Gravano, McConnell, Spinrad y Tucker [5] habían diseñado un algoritmo polinomial (cuadrático) para este problema.

Teorema 2.3 *[21]: sea G un grafo UCA y (C, **A**) un modelo normal de este. Entonces G admite un modelo UCA, tal que sus extremos están en el mismo ordenamiento circular que tenían en su modelo normal (C, **A**).*

Este último teorema (2.3) es muy importante, para el trabajo de Lin y Szwarcfiter [11] donde se presenta su algoritmo para reconocer grafos UCA con un certificado positivo, ya que da la base teórica (y buena parte de la motivación) que permite al algoritmo recalcular las posiciones relativas de los extremos de los arcos para alcanzar el modelo UCA, a partir de un modelo normal.

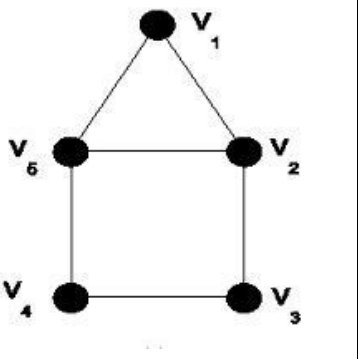
Caracterización de Lin y Szwarcfiter

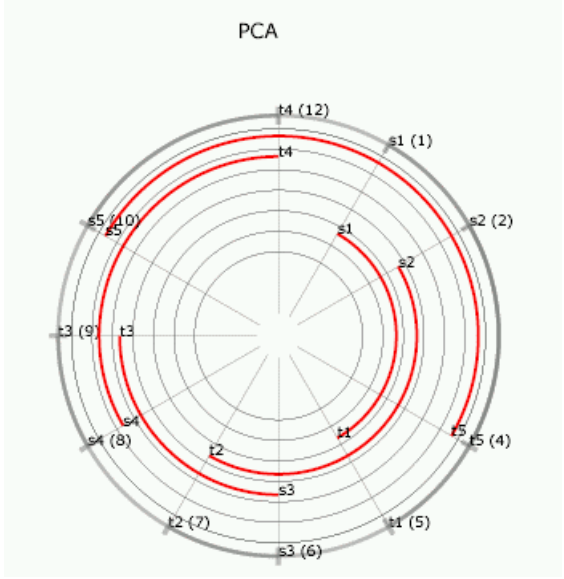
La caracterización descripta por Lin y Szwarcfiter [11] de grafos UCA, es la que utilizan los algoritmos que se implementan en esta tesis. Esta caracterización esta dada en términos de la clase de digrafos débilmente eulerianos.

La idea general es asignar a variables las longitudes de los segmentos de un modelo normal de un grafo PCA. Luego se expresa que la longitudes de los arcos deben ser iguales mediante un sistema de ecuaciones. Con la solución a este sistema, se ajusta el modelo para reflejar las longitudes de los segmentos, manteniendo el ordenamiento circular original de los extremos de los arcos, obteniendo así un modelo UCA. Los Teoremas 2.2 y 2.3 aseguran que se puede obtener el modelo normal de un modelo PCA, y que un grafo es UCA si solo si hay una solución al sistema de ecuaciones.

Más detalladamente, sea *G* un grafo arco circular y (*C*, ***A***) su modelo arco circular. Se denota $S_1, \ldots, S_{2n}$ a los segmentos de (*C*, ***A***) en un ordenamiento circular, cuando el inicio de S_1 y el de A_1 coinciden. Denotamos también con l_j al largo de S_j. Entonces, cualquier arco $A_i \in$ A puede ser descompuesto en los segmentos que lo conforman como ya habíamos mostrado en Fig. 1.2. La longitud de A_i es igual a la suma de las longitudes de estos segmentos. La descomposición permite representar relaciones entre el largo de los arcos mediante relaciones entre el largo de los correspondientes segmentos. Por ejemplo, en Fig. 2.2, la ecuación q_1 expresa que el largo del arco A_1 (conformado por los segmentos S_1, S_2 y S_3) debe ser igual al largo del arco A_2 (conformado por los segmentos S_2, S_3, S_4 y S_5). Consecuentemente la condición de igualdad entre el largo de cualquier par de arcos requerida por el modelo UCA puede ser expresada por un

sistema de n - 1 ecuaciones q_i, junto a 2n desigualdades que establecen que cada segmento debe tener longitud mayor a 0. Este sistema es llamado el *full system* de (C, $\boldsymbol{A}$).





$$\begin{cases} q_i : \sum_{S_j \subseteq A_i} l_j = \sum_{S_j \subseteq A_{i+1}} l_j, \ 1 \leq i \leq n-1 \\ l_j > 0, \ 1 \leq j \leq 2n \end{cases}$$

q_1: $l_1 + l_2 + l_3 = l_2 + l_3 + l_4 + l_5$
q_2: $l_2 + l_3 + l_4 + l_5 = l_5 + l_6 + l_7$
q_3: $l_5 + l_6 + l_7 = l_7 + l_8 + l_9$
q_4: $l_7 + l_8 + l_9 = l_9 + l_{10} + l_1 + l_2$

Fig. 2.2 : *grafo, modelo PCA, ecuación para q_i y ejemplo de full system*

La ecuación q_i corresponde a la condición de que el largo del arco A_i es igual al de A_{i+1}. El largo de los segmentos l_j son las variables de este sistema de ecuaciones. El objetivo es encontrar una solución para el full system si es que esta existe, ya que G es UCA si y sólo si el full system tiene solución, por Teorema 2.3. Más aún los valores de l_j obtenidos de la resolución del sistema definirán un modelo UCA para G.

En la ecuación q_i, la parte de la igualdad correspondiente a la suma de los segmentos de A_j se llama *lado izquierdo*, mientras que el *lado derecho* es la correspondiente a A_{i+1}. Cada ecuación q_i podría ser simplificada si el largo l_j de un mismo segmento aparece a ambos de lados de q_i. Las ecuaciones obtenidas forman entonces el *sistema reducido* de (C, $\boldsymbol{A}$) (ver Fig. 2.3). Obsérvese que para representar el full system y el sistema reducido se necesitan solo $O(n)$ índices. Esto se debe a que en cada ecuación los índices de las variables de cada lado forman un rango y por lo

tanto se necesitan a lo sumo cuatro índices para describir cada ecuación. Por ejemplo, en Fig. 2.2, la ecuación q_1 se puede representar como [1,3] = [2,5], y en el sistema reducido de la Fig. 2.3, dicha ecuación se representa como [1] = [4,5].

q_1: $l_1 = l_4 + l_5$
q_2: $l_2 + l_3 + l_4 = l_6 + l_7$
q_3: $l_5 + l_6 = l_8 + l_9$
q_4: $l_7 + l_8 = l_{10} + l_1 + l_2$

Fig. 2.3 : Ejemplo de sistema reducido para el mismo grafo y modelo (ver Fig. 2.2).

El siguiente lema describe una útil propiedad de estos sistemas.

Lema 2.2 *[11]: Sea G un grafo PCA, (C,* ***A****) su modelo normal y R el sistema reducido de (C,* A*). Entonces cada variable l_j aparece al menos un vez en R, a menos que el segmento correspondiente S_j sea un segmento contenido en todos los arcos de* ***A*** *o en ninguno de ellos. Además, l_j aparece a lo sumo dos veces en R, en tal situación, las dos instancias de l_j en R ocurren en diferentes lados de sus ecuaciones.*

A partir del sistema reducido se construye el *digrafo de segmento D*, colocando un vértice v_i por cada ecuación q_i de *R* y una arista $e_k = v_i \rightarrow v_j$ si la variable l_k se encuentra al lado izquierdo de q_i y al derecho de q_j. Además se agrega un vértice v_0 a donde van y vienen aristas por cada l_j que aparece solo una vez. Siguiendo el mismo ejemplo el digrafo de segmento se muestra en Fig. 2.4.

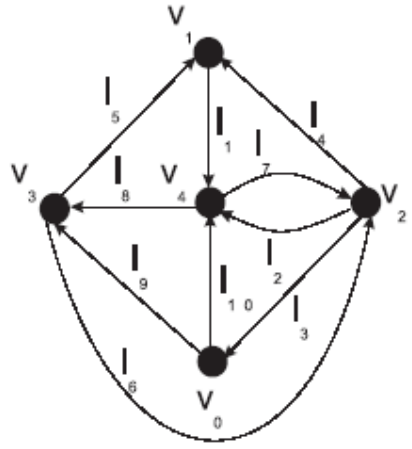


Fig. 2.4: Digrafo de segmentos correspondiente al sistema reducido de la Fig. 2.3.

Los dos siguientes teoremas describen completamente los grafos UCA.

Teorema 2.4 *[11]: Sea G un grafo PCA, (C, **A**) un modelo normal y D su digrafo de segmento. Entonces G es un grafo UCA si y sólo si D es un digrafo débilmente euleriano.*

Teorema 2.5 *Lin Szwarcfiter [11] : Un digrafo D es débilmente euleriano si y sólo si cada una de sus componentes conexas es fuertemente conexa.*

Entonces el Teorema 2.4 dice cuándo *G* es un grafo UCA en relación a un digrafo de segmento *D* y el Teorema 2.5 dice cómo determinarlo a partir de *D*. Por lo tanto hay que tomar *D* y averiguar si es fuertemente conexo, si lo es, entonces *G* es UCA, y hay que construir el modelo UCA correspondiente, si no, hay que brindar un certificado negativo que lo avale. A continuación describimos cómo construir los certificados.

Construcción del certificado positivo

En el caso en que determinamos que el grafo es UCA a partir de su digrafo de segmentos, sabemos que es porque el mismo es fuertemente conexo. El objetivo para construir el certificado es encontrar valores para las variables que satisfagan el sistema reducido. Consideremos un weakly eulerian weighting *w* del digrafo de segmentos. Recordemos que cada arista e_k del digrafo de segmentos se corresponde con una variable l_k. Un aspecto clave que surge en la demostración del Teorema 2.5 es que si a cada variable l_k se le asigna el peso $w(e_k)$ entonces se obtiene una solución al sistema reducido. Esto se debe a que cada vértice v_i del digrafo de segmentos satisface la ley de conservación de flujo, i. e. $w^+(v_i) = w^-(v_i)$. Ahora, $w^+(v_i)$ es la sumatoria de los pesos de las aristas saliendo de v_i, que es precisamente la sumatoria de las longitudes del lado izquierdo. De la misma forma ocurre para el lado derecho. Luego, por la conservación de flujo, la ecuación q_i queda satisfecha.

Veremos ahora brevemente como encontrar un weakly eulerian weighting. Para ello, se empieza con un weighting inicial donde a cada arista se le asigna peso 1 y trataremos de equilibrar el flujo entrante y saliente de cada vértice. Luego se construye un out-arborescence y un in-arborescence ambos con la misma raíz v^*. A partir del out-arborescence modifica el flujo de cada vértice, de forma tal que su flujo saliente no sea mayor al flujo entrante, recorriendo los vértices en el orden hoja-raíz. Por último, con el in-arborescence se modifica el flujo de cada vértice para que el flujo entrante no sea mayor al saliente, también con un orden hoja-raíz. Lin y Szwarcfiter [11] demuestran que este procedimiento efectivamente nivela el flujo de todos los vértices, lo cual es posible por el Teorema 2.5.

El certificado positivo se consigue luego construyendo un nuevo modelo (*C*, ***A***) que contiene el mismo conjunto de segmentos que los del modelo normalizado que le dio origen y en el mismo orden. La diferencia es que cada segmento tendrá la longitud definida por el weakly eulerian weighting, por lo tanto solo hay que redibujar los segmentos con su nueva longitud y luego ubicar

los arcos.

Construcción certificado negativo

Para el caso en que *D* no es fuertemente conexo, se elige una componente fuertemente conexa secundaria (*CFC*) que no contenga al vértice v_0 de las calculadas previamente, y que sea fuente o sumidero (siempre existe una de estas [12]). Luego se arma una descripción por rangos mínima formada por rangos de vértices contiguos de *D* que estén en la *CFC*, y a partir del full system, también previamente calculado, se eligen los arcos que pertenecen a CFC conformando un *selected full system*.

Ejemplo del sistema de ecuaciones (full system) del grafo CI(4, 1) (ver Fig. 2.1), aquí no es necesario utilizar las longitudes de los segmentos de los arcos sino que usamos las longitudes de los arcos.

q_1: $|A_1| = |A_2|$
q_2: $|A_2| = |A_3|$
q_3: $|A_3| = |A_4|$
q_4: $|A_4| = |A_5|$
q_5: $|A_5| = |A_6|$
q_6: $|A_6| = |A_7|$
q_7: $|A_7| = |A_8|$

Selected full system se obtiene seleccionando los arcos que están en la CFC elegida para ello utilizamos su descripción mínima {[A_1, A_1], [A_3, A_3], [A_5, A_5], [A_7, A_7]} .

q_1: $|A_1| = |A_2|$
q_3: $|A_3| = |A_4|$
q_5: $|A_5| = |A_6|$
q_7: $|A_7| = |A_8|$

Fig. 2.5 Full system, selected full system y descripción mínima para el *grafo de la Fig. 2.1*

Luego con estos rangos se arma una nueva secuencia de arcos llamados *selected arcs* donde los arcos que forman los rangos van a la primera mitad y los arcos que están fuera del rango a la segunda, los arcos inter-rango no pertenecen a los selected arcs.

Por lo tanto siguiendo el ejemplo los selected arcs son:
[A_1, A_3, A_5, A_7, A_2, A_4, A_6,A_8]

Fig. 2.6: selected arcs para el grafo de la Fig. 2.1

Según sea CFC fuente o sumidero cada mitad corresponderá a los arcos del (*n, k*)-circuito del (*n, k*)-conjunto independiente, los arcos se encuentran en el ordenamiento horario, y representan al grafo prohibido *CI*(*n*, *k*) . Para construir el certificado se muestran que arcos del modelo (*C*, ***A***) PCA original conforman el (*n, k*)-circuito y el (*n, k*)-conjunto independiente.

El valor de *k* se obtiene como la mitad de tamaño de selected arcs, o bien como el tamaño de (*n,*

k)-circuito. El valor de *t*, que es la cantidad de vueltas alrededor de *C*, se determina, a partir del extremo final (inicial) de A_1 notado t_{i1} (s_{i1}) si *C* es fuente (sumidero), como la posición de su extremo inmediatamente anterior en el sentido horario entre los selected arcs, que es $s_{iNext(1)}$ ($t_{i\ Next(1)}$). Con lo cual se halla el valor de *Next(*1*)* y podemos calcular *t = (Next(*1*)*-1*) mod k.* La función *Next(p) = q* y se puede determinar a partir de *p* de manera muy eficiente en tiempo constante ya que conocemos el ordenamiento de los extremos de los arcos.

Concluyendo el ejemplo (de la Fig. 2.1)
Entonces k es 4, y t se calcula a partir del punto inmediato anterior a t_{i1} *entre los arcos de selected arcs, que es* s_3 *y la posición de* A_3 *es 2, por lo tanto Next(1) = 2 y t = (Next(1)-1) mod k = (2-1)mod 4 = 1.*
y tomando una CFC fuente el
(4, 1)circuito es [A_1, A_3, A_5, A_7*] y el*
(4, 1)conjunto independiente es [A_2, A_4, A_6,A_8*],*
por el lema 2.1 forman un CI(4,1) que es un grafo prohibido.

Fig. 2.7: Se puede seguir la corrida de este ejemplo en el anexo II.

En el capítulo siguiente mostraremos cómo implementar el algoritmo obtenido en tiempo lineal, calcularemos su complejidad y mostraremos los resultados empíricos obtenidos en la ejecución del programa.

Capítulo 3

Implementación de Reconocimiento y Certificación de Grafos UCA

En este capítulo se muestra cómo implementar el algoritmo de Lin y Szwarcfiter [11,12] para reconocer un grafo UCA y construir el certificado correspondiente en tiempo lineal, describimos su complejidad y mostramos los resultados empíricos obtenidos en la ejecución del programa. Previamente se hace una reseña de otros algoritmos de reconocimiento (alguno de ellos con certificado) para grafos arco circulares unitarios, con especial énfasis en la reducción de la complejidad temporal de éstos y en la construcción de certificados, desde un primer algoritmo que no pudo probarse que sea polinomial hasta los algoritmos de Lin y Szwarcfiter de reconocimiento con ambos certificados en tiempo lineal.

El primer algoritmo de reconocimiento de grafos arco-circulares unitarios es de Tucker [23, 24] y utiliza la caracterización por subgrafos prohibidos resumida en el capitulo anterior (Tucker [21]). La demostración que hace Tucker de esta caracterización de grafos UCA, de hecho construye un modelo UCA, cuando el grafo no contiene ninguno de los subgrafos prohibidos. Sin embargo el proceso involucra encoger y alargar los arcos, implicando la manipulación de enteros grandes en relación al tamaño del input, por lo que no puede asumirse que operar con ellos insuma tiempo constante, por lo tanto no se puede comprobar si este proceso termina o no en tiempo polinomial (Durán et Al. [5]).

Deng, Hell y Huang [3] describen un algoritmo para reconocer grafos PCA en tiempo lineal que también construye el modelo PCA correspondiente. Kaplan y Nussbaum [10] describieron otro algoritmo para reconocer grafos PCA en tiempo lineal, que analiza dos casos, si es Co-Bipartito o si no; en el primer caso utiliza el algoritmo de Deng, Hell y Huang [3] y en el segundo de McConnell [14]. El algoritmo genera ambos certificados en tiempo lineal, un modelo PCA en caso positivo y una estructura que muestra el grafo prohibido en caso contrario.

Todos los algoritmos de reconocimiento para grafos UCA utilizan el algoritmo de Deng et al. [3] (y la modificación de Kaplan y Nussbaum [10]), para reconocer si el grafo es PCA y construir el modelo arco circular propio en tiempo lineal, en caso afirmativo. Este modelo es tomado como input por cada algoritmo de reconocimiento de grafos UCA. El primer algoritmo de reconocimiento polinomial de grafos UCA es de Durán, Gravano, McConnell, Spinrad y Tucker [5]. Se basa en el Teorema 2.1 y la idea es determinar límites superiores e inferiores para la longitud del círculo. La

cota superior para la longitud de la circunferencia es el valor más chico de *n/k* entre todos los (*n, k*)-circuitos, y la cota inferior es el valor más grande de *m/l* entre todos los (*m, l*)-conjuntos-independientes. El rango entre la cota inferior y superior es vacío, si y sólo si, no existe modelo UCA para *G*. Esto quiere decir que en un modelo UCA de *G* la circunferencia debe medir menos que *n/k* y más que *m/l*. Si esto es imposible, *G* no es UCA. El algoritmo construye un (*n, k*)-circuito por cada arco, eligiendo como próximo arco del circuito al arco adyacente que se extiende más lejos en el modelo. Para los (*m, l*)-conjuntos-independientes también comienza en cada posible arco, pero elige el próximo arco como el siguiente no adyacente. La principal desventaja es que este algoritmo no puede generar un modelo arco circular unitario a modo de certificado positivo cuando el grafo es unitario. Es más, en la conclución se preguntaban si es posible conseguir un algoritmo que construya tal modelo y además los extremos de los arcos de este modelo puedan ser números enteros de tamaño polinomial.

En [11] Lin y Szwarcfiter describen un algoritmo de reconocimiento de grafos UCA que genera un certificado en caso positivo en tiempo lineal. El algoritmo recibe como input un modelo PCA y utiliza la caracterización de grafos UCA basada en grafos débilmente eulerianos [11], descripta en detalle en el capitulo anterior. La implementación de estos algoritmos se explicará detalladamente más adelante en este capítulo.

Kaplan y Nussbaum [10] describieron un algoritmo para reconocer un grafo UCA basado en la caracterización de Tucker [21] y que emplea la misma estrategia que en [5] computa los límites superior e inferior para el largo del círculo de un posible modelo UCA. Sin embargo, en vez de considerar (*n, k*)-circuitos y (*m, l*)-conjuntos-independientes empezando en cada arco del modelo, el algoritmo [10] requiere sólo empezar desde un arco particular que puede ser encontrado en tiempo lineal. Además, en [10] se realiza una generalización de (*n, k*)-circuitos y (*m, l*)-conjuntos-independientes, permitiendo repeticiones de vértices y usando caminos en lugar de ciclos, y así reducen la complejidad del algoritmo de [23] de $O(n^2)$ a $O(n + m)$. Para el caso positivo, se construye un certificado positivo usando [11] y por último con [10] se construye un certificado negativo. Este certificado se da en términos de (*n, k*)-circuitos mínimos y (*m, l*)-conjuntos-independientes máximos y requiere un tiempo de ejecución de $O(n + m)$. Como desventaja se puede ver que se requiere utilizar dos estrategias distintas: la de Lin y Szwarcfiter [11] para construir el certificado positivo y la de Tucker [21] para construir el certificado negativo.

Recientemente Lin y Szwarcfiter [12] describieron un algoritmo lineal para construir un certificado negativo, esto es cuando el grafo es PCA y no es UCA basándose el la caracterización de [11]. De esta forma se puede determinar si un grafo es UCA o no y, con lo ya calculado, se construye el certificado positivo o negativo según corresponda. A diferencia del algoritmo de Kaplan y Nussbaum [10] en este caso se utiliza una única estrategia y por lo tanto se simplifica la implementación ya que se pueden seguir usando los datos calculados en el reconocimiento para

ambos certificados.

La siguiente tabla resume esta reseña haciendo hincapié en los algoritmos para grafos UCA con certificado de complejidad lineal.

Clase	Referencia	Caracterización UCA utilizada	Complejidad	Construye modelo UCA (Cert. Positivo)	Certificado negativo
UCA	Tucker [23,24]	-	¿polinomial?	si	no
PCA	Deng et al. [3]	-	O(n+m)	si (construye PCA)	no
PCA	Kapplan y Nussbaum [10]	-	O(n+m)	si *2	si
UCA	Duran et al. [5]	Tucker [21]	$O(n^2)$	no	no
UCA	Lin y Szwarcfiter [11]	Lin y Szwarcfiter [11]	O(n)*1	**si**	no
UCA	Kapplan y Nussbaum [10]	Lin y Szwarcfiter [11] y Tucker [21]	O(n)*1	si *2	**si**
UCA	Lin y Szwarcfiter [12]	Lin y Szwarcfiter [11]	O(n)*1	si *2	**si**

Tabla 3.1: Resumen de los resultados más importantes en relación al reconocimiento de grafos UCA y la construcción de los modelos. Para los casos de UCA se considera que el input es un modelo PCA, en otro caso vale *1.
*1 utilizan Deng et al.[3] y por lo tanto el orden total es $O(n+m)$
*2 utiliza [11] para generar modelo UCA

A continuación se describen los algoritmos [11,12] para reconocer y construir ambos certificados según la caracterización de Lin y Szwarcfiter [11]. El modelo PCA puede obtenerse utilizando el algoritmo de Deng, Hell y Huang [3].

Descripción de los algoritmos

En esta sección se describen los algoritmos de [11,12] para reconocer y certificar un grafo arco circular unitario. Primero se describe el algoritmo general y luego los distintos pasos de éste más detalladamente. Cada algoritmo se presenta con su pseudocódigo y su explicación, luego se describe su implementación tanto de manera conceptual como de las decisiones que se tomaron en cada caso para conseguir una complejidad temporal lineal.

Algoritmo general

El algoritmo general consta de dos partes, la primera es reconocer si un grafo es arco circular

unitario. Luego, si es UCA se construye un modelo (*C,* ***A***) que lo represente (certificado positivo), y si no es UCA se devuelve un subgrafo prohibido *CI*(*n, k*) (certificado negativo).

Descripción del algoritmo general

Algoritmo 3.1: algoritmo general [11]

1. *Dado un grafo G, primero verificar si G es un grafo PCA, utilizando el algoritmo de Deng et al. [3]. En caso afirmativo (C,* ***A****) es el modelo PCA obtenido, caso contrario G no es UCA.*
2. *Transformar (C,* ***A****) en un modelo normal (C',* ***A'****).*
3. *Construir un digrafo de segmento D a partir de (C',* ***A'****). Verificar si D es fuertemente conexo.*
    a. *En caso afirmativo G es UCA*
        i. *Construir modelo (C,* ***A****) que represente a G*
    b. *en otro caso no es UCA.*
        i. *Obtener grafo prohibido CI(n, k)*

Descripción de la implementación del procedimiento general

El input del algoritmo implementado es un modelo PCA idéntico al que seria obtenido por el paso 1. Cabe aclarar que el modelo generado por el algoritmo de Deng et al. contiene los extremos de los arcos ordenado. El modelo es ingresado desde un archivo de texto que contiene una lista de extremos ordenados (ver Anexo I). Éstos se guardan en la estructura de datos principal formada por un arreglo de *n* arcos, un arreglo de 2*n* extremos, un arreglo de 2*n* segmentos y un arreglo de *n* ecuaciones. Los arcos y sus extremos se completan al levantar el archivo de entrada, los segmentos y ecuaciones al calcular el sistema reducido. Cada arco tiene conocimiento de sus dos extremos y de los segmentos que lo componen. Los extremos se pueden pensar como un número entero entre 1 y 2*n* que representa su posición en el círculo.

El primer paso del algoritmo es normalizar el modelo PCA del input, porque el modelo de Deng et al. puede no estar normalizado. Las ecuaciones *q* están formadas por una estructura de 4 enteros que representan los rangos de los índices de los segmentos para cada arco A_i y A_{i+1} en q_i. Entonces, en total, se usan 4*n* índices en las ecuaciones del full system y del sistema reducido, aunque en la práctica se construye directamente este último. Luego el digrafo de segmento *D* se construye recorriendo una vez estos últimos arreglos. La estructura de datos de *D* está formada por una arreglo de *n* vértices y un arreglo de *m* aristas; por la forma de construir *D* que mapea uno a uno a los segmentos, $m = 2n$. Se recorre el digrafo con el procedimiento de depth first forest, utilizando Strongly Connected Components [16] cara calcular las componentes conexas de éste. Si tiene una única componente fuertemente conexa se construye el modelo UCA equilibrando los ejes del digrafo mediante out e in-arborescences, si no, no es UCA y se construye un (*n, k*)-circuito y un

(*n, k*)-conjunto-independiente a partir de las descripción por rangos mínima que se obtiene de la componente conexa fuente o sumidero elegida. La Fig. 3.1 muestra el flujo de los procedimientos de esta implementación.

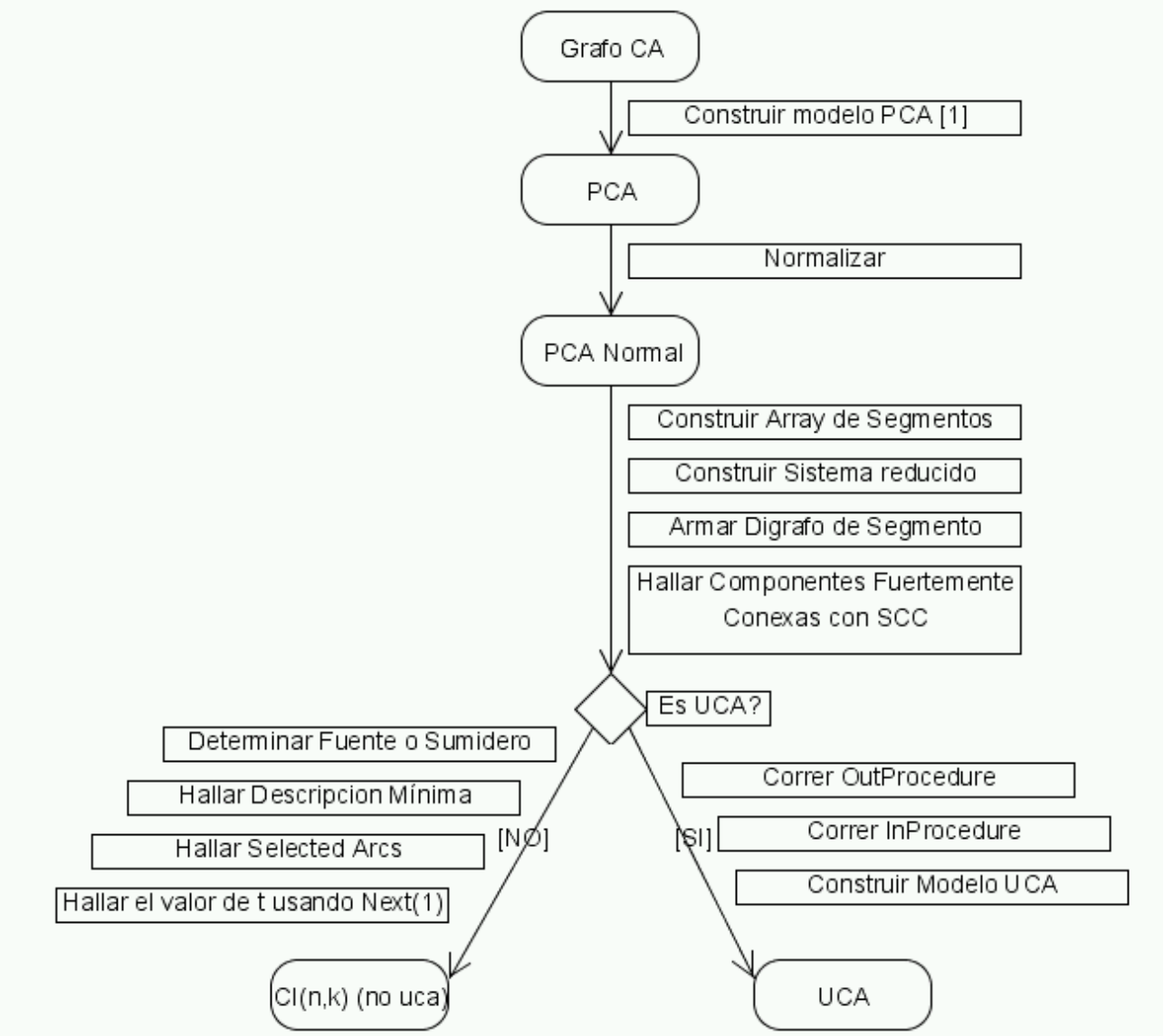


Fig. 3.1: El gráfico muestra la secuencia de procedimientos necesarios para reconocer el grafo UCA y obtener los certificados correspondiente, en esta tesis se implementan los algoritmos desde el modelo PCA hacia abajo.

Algoritmos de reconocimiento

En esta sección se describen los algoritmos que llevan a reconocer un grafo arco-circular unitario.

Normalización

Este algoritmo transforma un modelo (*C,* ***A***) de un grafo PCA, en un modelo normal. Se sabe por el Teorema 2.2 que el modelo PCA admite un modelo normal. En la práctica su función es eliminar los *Cl*(2, 1) del modelo propio, ya que los modelo con *Cl*(2, 1) no se pueden transformar a modelos UCA manteniendo el orden de los extremos, un (2, 1)-circuito se ve en un modelo arco circular como dos arcos cubriendo el círculo.

Descripción del algoritmo normal

Algoritmo 3.2: algoritmo normal [11]

1. *para cada punto i (extremo de un arco A_i) consecutivo en orden circular*

   si i = 1 entonces p := s_1
   si no p := max{p, s_i}s_{i-1}

   repetir
   si p es un extremo final t_j entonces
   si $A_i \cup A_j = C$ entonces $t_j \leftarrow POINT(PRED(s_i), s_i)$
   sino terminar
   p ← SUC(p)

En donde *PRED(p)* indica el punto anterior a *p* (en el sentido de las agujas del reloj) y *POINT(p, q)* denota un punto arbitrario en el arco que va de los puntos *p* a *q*. La idea es recorrer *C* buscando un arco A_j de ***A*** que junto a A_i cubran *C*. Si dicho arco es hallado entonces el arco A_j es convenientemente comprimido para que A_i y A_j ya no cubran *C*, el nuevo modelo obtenido continúa siendo un modelo PCA .

Este método considera los arcos A_i en orden circular $A_1....A_n$. Para cada *i* sólo una fracción de los arcos de *A* son examinados en general, si el punto siendo examinado *p* es un extremo inicial s_j se saltea y se pasa a examinar el siguiente. Si *p* es un extremo final t_j y $A_j \cup A_i$ cubren *C*, entonces se achica A_j de tal forma que ya no cubran *C*. Esto se lleva a cabo moviendo t_j en el sentido contrario a las agujas del reloj para que quede ubicado inmediatamente antes de s_i, es decir, después del predecesor original de s_i y antes de s_i. Cuando $A_j \cup A_i$ no cubren *C* entonces no se analizan los puntos posteriores a *p* en A_i, ya que no van a cubrir el *C* con A_i. Al empezar la iteración que analiza s_{i+1}, p representa al primer extremo final que aparece después de s_i. Los puntos finales entre *p* y s_{i+1} (si *p* aparece antes de s_{i+1}) corresponden a arcos que no van a cubrir el círculo con A_i y por ello se descartan (con la instrucción *p := max{p, s_i}s_{i-1}*).

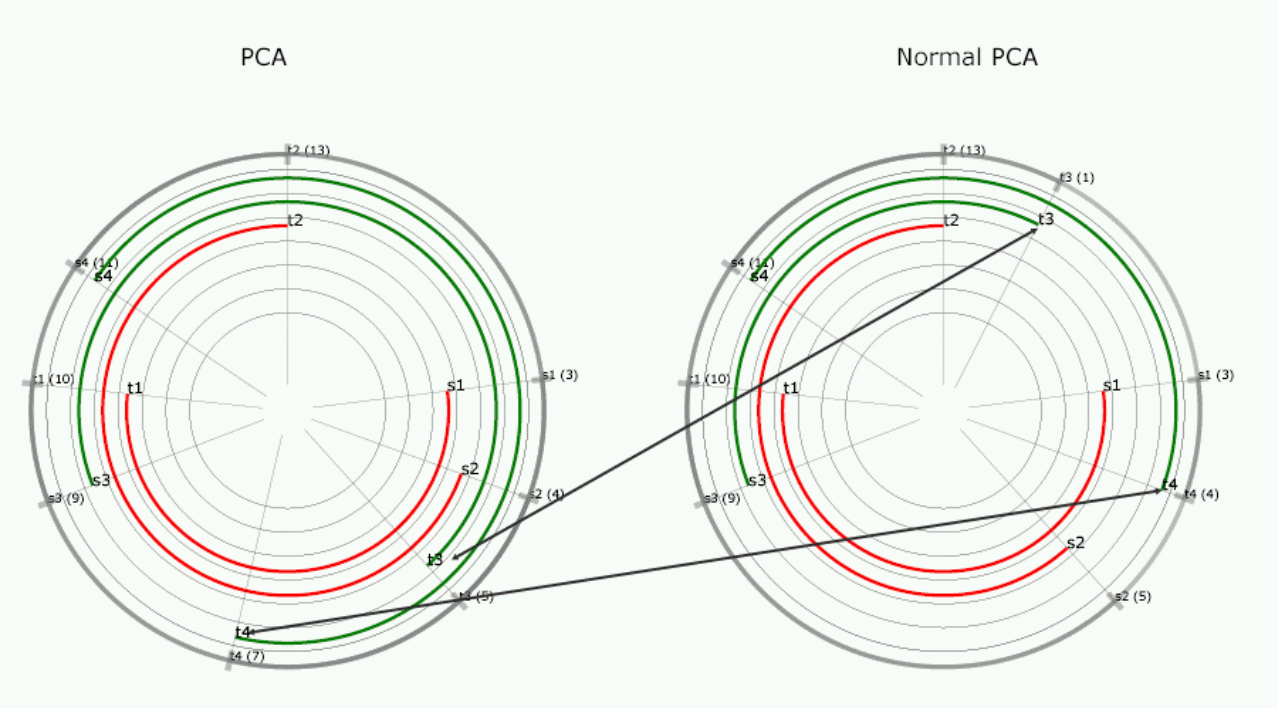


Fig. 3.2: Un modelo PCA y su correspondiente luego de ejecutar el procedimiento normal, los pares de arcos A_1-A_3 y A_2-A_4 cubren C por lo tanto se encogen A_3 y A_4 para ello se corre hacia atrás (en sentido anti-horario) sus extremos finales t_3 y t_4 respectivamente.

Descripción de la implementación del procedimiento normal

Los procedimientos generados implementan eficientemente los algoritmos para la normalización del modelo arco circular propio. Antes de pasar a la normalización se importa el modelo PCA desde un archivo de texto, el procedimiento que efectúa esta tarea también tiene una complejidad temporal lineal, ya que recorre un vez todos los extremos, para completar los arreglos de arcos y extremos.

Según los resultados teóricos el ciclo interno *Repetir* debe tener menos de 6*n* iteraciones, durante todo el proceso [11]. Luego, para obtener una complejidad de O(n) para todo el procedimiento, debemos poder insertar en tiempo constante el extremo t_j, entre el extremo inicial de A_i y su predecesor en orden horario. Se implementó para los extremos una estructura de lista doblemente encadenada (con punteros al extremo anterior y al siguiente), donde cada extremo conoce su posición actual, mientras mantiene su posición constante en el arreglo. De esta manera se mantuvo las posiciones relativas entre los extremos sin tener que desplazarlos realmente, permitiéndonos reubicar los arcos en *O(1)*. Luego, una vez terminado el procedimiento, se hace un posprocesamiento que efectúa un corrimiento de los extremos permitiendo el ordenamiento del

arreglo en *O(n)*. El siguiente procedimiento resuelve si dos arcos cubren *C* en tiempo constante.

Procedimiento 3.1: Dos arcos cubren C

Resuelve si dos arcos cubren toda la circunferencia en tiempo constante.

Dado un modelo arco circular(C, A), ai y aj ∈ A de, con s el extremo inicial y t el extremo final
Si se cumple (si < tj < sj < ti)
o (tj < sj < ti < si)
o (sj < ti < si < tj)
o (ti < si < tj < sj)
entonces ai y aj cubren C

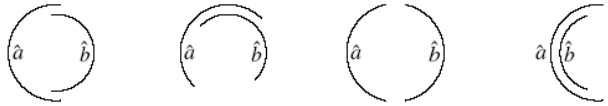


Fig. 3.3 [10]: Sólo en el primer caso hay dos arcos cubren que el círculo *C*. Se necesitan 4 comparaciones porque el punto 0 de *C* puede estar en cualquiera de los cuatro segmentos.

Construcción del digrafo de segmento

Aquí se describen los algoritmos para construir el digrafo de segmento y luego los detalles de su implementación. Una vez obtenido el modelo normal se construye un sistema de ecuaciones a partir de las longitudes de los segmentos, utilizando el hecho de que las longitudes de los arcos deben ser iguales entre sí. Sean $S_1, \ldots, S_{2n}$ los segmentos de (*C,* ***A***) en ordenamiento circular, y l_j el largo S_j. Entonces el largo de A_i es $\Sigma_{Sj \subseteq Ai}\ l_j$, es decir $|A_i|$ es igual a la suma de los $|S_j|$ que lo componen.

$$
\begin{array}{lll}
q_1: & |A_1| = |A_2| \\
q_2: & |A_2| = |A_3| \\
\vdots & \vdots \quad \vdots \\
q_n: & |A_n| = |A_n|
\end{array}
$$

Fig. 3.4: Sistema de ecuaciones llamado full system

De este sistema de ecuaciones se eliminan los l_j que aparecen a ambos lados de la igualdad y se arma el sistema reducido *R.* Para resolver eficientemente *R*, se construye el siguiente digrafo *D*, llamado digrafo de segmentos.

Algoritmo 3.3: construcción del digrafo de segmento

Poner un vértice $v_i \in V(D)$ por cada ecuación q_i de R, un vértice distinguido v_0 y un eje e_j por cada segmento S_j de (C,A). Cada eje $e_j \in E(D)$ se orienta de acuerdo a lo siguiente:

1. *si l_j aparece a la izquierda de alguna ecuación q_i de R entonces e_j comienza en v_i, si no*

comienza en v_0. (ver Fig. 3.4)

2. *si l_j aparece a la derecha de alguna ecuación q_k entonces e_j termina en v_k, si no termina en v_0.*

La descripción de D es completa, D tiene n vértices y 2n ejes no tiene loops con excepción posiblemente de v_0. [11]

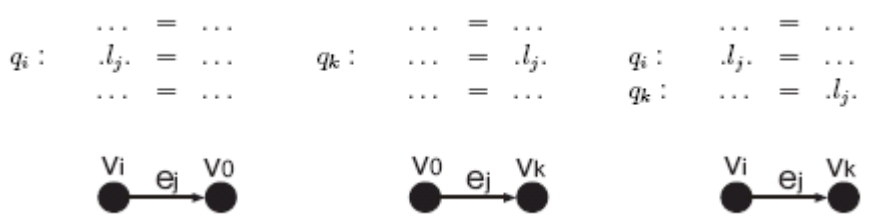


Fig. 3.5 [13]: Mapeo de las ecuaciones del sistema reducido a los vértices y ejes del Digrafo de Segmentos

Descripción de la implementación de la construcción del digrafo de segmento

Los siguientes procedimientos implementan eficientemente los algoritmos para la construcción del digrafo de segmento. Cada segmento se implementa como un punto en *C* que concuerda con su inicio y su longitud, ambos valores enteros.

Procedimiento 3.2: Construir un arreglo de segmentos

1. *Crear un arreglo de tipo segmento*
2. *Recorrer el modelo en sentido horario iterando sobre los extremos*
3. *Por cada par de extremos consecutivos*
4. *agregar un segmento al arreglo*
5. *agregar a los arcos los segmentos en que comienzan y terminan*
6. *Agregar un segmento entre el último y el primer extremo*

El procedimiento recorre los extremos una sola vez realizando 2*n* iteraciones, es *O(n)*. Se agregan a los arcos el par de índices de los segmentos de donde empieza y termina el arco como precálculo para el sistema reducido. Se decidió asignarlos de esta manera para evitar poner un lista de segmentos a cada arco, ya que esto nos daría una complejidad temporal polinomial. Poder completar estas estructuras en tiempo lineal es posible debido al ordenamiento circular relativo.

Procedimiento 3.3: Construir el Sistema Reducido

1. *Crear un arreglo de ecuaciones Q*
2. *Para cada arco a_i del modelo*
3. *Comparar a_i con el próximo arco en sentido horario a_j*
4. *Crear una ecuación q_i donde:*
   a la izquierda van los segmentos inicial de a_i y último de a_i que no esté en a_j
   y a la derecha van los segmentos primero de a_j que no este en a_i y final de a_j.
5. *Agregar la ecuación q_i a Q.*

El procedimiento itera *n* arcos del modelo y agrega una ecuación por cada uno de los arcos. Cada ecuación está implementada por cuatro números enteros, que representan a los índices de los

segmentos. Estos cuatro índices, delimitan el rango de los segmentos, a cada lado de la igualdad. En total se necesitan 4*n* índices. Por lo tanto la complejidad del procedimiento es *O(n)*.

Procedimiento 3.4: Construir el digrafo de segmento D

Dado un modelo normal con su sistema reducido, con ecuaciones Q y segmentos S

1. *Crear dos arreglos del tamaño de la cantidad de segmentos del modelo:*
   uno para la parte izquierda de los ejes
   y otro para la parte derecha.
2. *Crear un digrafo con n vértices, siendo n la cantidad de ecuaciones* q_i
3. *Para cada ecuación* q_i *del lado izquierdo de Q*
   i. *Recorrer el arreglo de segmentos S del modelo*
   ii. *mientras el [índice del segmento* sg_k *]< [extremo izquierdo de* q_i*]*
      1. *No hacer nada*
   iii. *mientras el [índice de* sg_k *] => [extremo izquierdo de* q_i*] y <= [extremo derecho de* q_i*]*
      1. *agregar al arreglo de ejes izquierdo en la posición k,el valor i*
4. *Para cada ecuación* q_j *del lado derecho de Q*
   i. *Elegir el segmento* sg_k *de la posición[extremo izquierdo de* q_j*]*
   ii. *Mientras la diferencia entre el [extremo izquierdo de* q_j*] y el [extremo derecho de* q_j*] >= k*
      1. *agregar al arreglo de ejes derecho en la posición k, el valor j*
      2. *elegir* sg_k *de la posición[extremo izquierdo de* $q_j + k$*]*
5. *Recorrer los arreglos de ejes izquierdo y derecho en paralelo*
   i. *agregar el par eje izquierdo* ei_i *y eje derecho* ed_i *a ejes de D*

El procedimiento para construir del digrafo de segmento *D* toma como entrada el modelo normal desde el cual fueron calculadas las ecuaciones del sistema reducido y devuelve un digrafo de segmento. En este procedimiento se crea una estructura temporal de 2 arreglos, de 2*n* enteros cada uno, tal que la arista e_i que va del vértice *u* al vértice *v* se conforma a partir de la posición *i*-ésima de ambos arreglos. El primero de estos dos arreglos contiene a los vértices de origen y el segundo a los vértices de destino. Esta estructura temporal es necesaria para poder obtener una complejidad temporal lineal, ya que las ecuaciones del sistema reducido mantienen un ordenamiento circular relativo distinto a cada lado de la igualdad, es decir, el lado izquierdo tiene un ordenamiento y el derecho otro.

Primero se inicializan todos los vértices de *D* en *n* operaciones. Luego las ecuaciones y los segmentos se recorren juntos completando la información de origen de las aristas. El recurso de recorrerlos juntos es para evitar que se visiten todos los segmentos para cada ecuación (esto es posible debido a que están ordenados). Por cada ecuación se camina por sobre los segmentos comprendidos en el lado izquierdo de ésta, para poder completar parte del arreglo de vértices de origen. Además, por cada segmento sg_i visitado se crea una arista en *D* a la cual se le asigna la longitud de sg_i, recorriendo, en total, una vez cada una de las *n* ecuaciones y una vez cada uno de los 2*n* segmentos. Luego hay que repetir el procedimiento usando para el otro lado de la igualdad (de las ecuaciones) para completar el arreglo de vértices destino.

Por último se recorren ambos arreglos en paralelo y se guardan para las aristas del digrafo *D* los

vértices de origen y destino. En resumen el procedimiento tiene un costo temporal lineal.

Determinar si el modelo es UCA o no

En esta sección se describen los algoritmos que llevan a tomar la decisión de si el es grafo arco-circular unitario o no.

Determinar si D es un digrafo de segmento débilmente euleriano

Aquí veremos si *D* es fuertemente conexo y luego los detalles de su implementación.

Algoritmo 3.4: Strongly-connected components SCC es una modificación del algoritmo de Kosaraju, Sharir [16]

1. *Llamar a DFF(D)*
2. *computar* D^T
3. *Llamar a DFF*(D^T)*, pero en el ciclo principal de DFS, considerar los vértices en orden de f[u] decreciente*
4. *devolver los vértices de cada árbol en el DFF formados en el punto 3 como una SCC aparte.*

En 1 se computan los tiempos de terminación *f[u]* para cada vértice *u*, en 2 se computa D^t que es la transposición de *D*, es decir se cambia el sentido a las aristas de *D*. Utiliza Depth First Search (*DFS)* para ver si una componente es fuertemente conexa o Depth First Forest (*DFF*) para hallar las componentes.

Algoritmo 3.5 : Decisión

1. *Si SCC tiene una única componente conexa el modelo es UCA*
2. *en otro caso no lo es.*

Basándose en los Teoremas 2.4 y 2.5, si el digrafo de segmento *D* es fuertemente conexo entonces *D* es débilmente euleriano y el modelo es UCA, si *D* tiene más de una componente el modelo no es fuertemente conexo por lo tanto no es débilmente euleriano y el modelo no es UCA.

Descripción de la implementación de SCC

Se eligió implementar el algoritmo de Kosaraju, Sharir [16] por que es muy claro y aunque consume tiempo extra en transponer el grafo en este caso no es tan relevante ya que *D* transpuesto también es utilizado más adelante en otros procedimientos (en in-procedure), por lo tanto se calcula previamente y sólo una vez. El otro algoritmo lineal clásico es el de Tarjan que utiliza mucho espacio en la pila. Depth first forest (*DFF*) se implementó de manera recursiva y no recursiva optándose finalmente por la no recursiva debido a que se producía una excepción de overflow en la pila cuando los modelos superaban los 3000 arcos. En cambio con esta versión no

recursiva se pudo correr modelos con 50000 arcos sin problemas. El procedimiento aquí implementado toma un digrafo y su transpuesto como input y retorna la cantidad de componentes conexas. Con este resultado el problema de reconocimiento está completo. Si hay una sola componente fuertemente conexa el modelo es UCA, si hay más de una no lo es.

Este procedimiento además le dice a cada vértice de *D* a qué componente fuertemente conexa pertenece. Esto es un preprocesamiento en caso de tener que construir un certificado negativo. La complejidad de SCC es $O(n + m)$, sin embargo, como mencionamos previamente la cantidad de aristas de *D* es $2n$ y por lo tanto el procedimiento cuesta $O(n)$. Para calcular D^t se recorren las $2n$ aristas una vez y para DFF dos veces, luego esta implementación es de orden lineal con respecto a la cantidad de vértices de *D*.

Certificado Positivo

En esta sección se describen los algoritmos que están involucrados en la construcción del certificado positivo. En este caso el certificado positivo es un modelo arco circular unitario.

Construir el modelo UCA

Dado el digrafo de segmento *D* fuertemente conexo veremos cómo resolver el sistema reducido *R* equilibrando las variables de longitud a ambos lados de la igualdad. Para resolver eficientemente *R* se utiliza *D* y se encuentra un weakly eulerian weighting de sus aristas que, como ya habíamos marcado, se mapean a la longitud de los segmentos que forman las variables de *R*. Luego veremos los detalles de su implementación.

Algoritmo 3.6 : construcción del modelo UCA. [11]

*Sea G un grafo UCA y (C, **A**) un modelo PCA normal de G. Este algoritmo construye un modelo (C',**A'**) a partir del digrafo de segmento D de (C, **A**).*
*Sea $p_1, \ldots, p_{2n}$ los extremos de (C, **A**) en ordenamiento circular, con p_1 correspondiendo al punto inicial s_1 de A_1.*

1. *Para cada $v \in V(D)$, asignar a las variables $w^-(v)$ y $w^+(v)$, valores iniciales iguales al grado de entrada (in-degree) y al grado de salida (out-degree) de v, respectivamente.*
   Para cada arista $e_j \in E(D)$, inicializar su peso $w_j := 1$.

2. *Para D, correr el OUT-PROCEDURE y luego el IN-PROCEDURE. Luego construir (C',**A'**) como sigue. El largo de C' es $\sum e_{j \text{ en } E(D)} w_j$. Los extremos $p'_1, \ldots, p'_{2n}$ de (C',A') se definen según las siguientes condiciones: p'_1 es un punto arbitrario de C', y el segmento (p'_j, p'_{j+1}) tiene un largo w_j, $1 \leq j < 2n$.*

Al principio se asigna una función de peso *w* sobre las aristas *e* de *D* que representan las variables del sistema reducido. El valor inicial elegido es 1 por cada arista, lo cual no necesariamente es un

weakly eulerian weighting. Por lo tanto la idea de este algoritmo es utilizar dos procedimientos *OUT-PROCEDURE* y luego el *IN-PROCEDURE* para equilibrar el peso *w* de las aristas de *D.* Una vez equilibrados, es decir con el mismo peso de salida que de entrada para cada vértice, se le asigna al largo de los segmentos sg_i el peso de las aristas $w(e_i)$ de *D*. Por último se recalcula la posición de los extremos de los arcos utilizando la nueva longitud de los segmentos. Cabe aclarar que los extremos continúan quedando ordenados. Con estas modificaciones sobre el modelo normal, el modelo resultante es un modelo UCA donde todos sus arcos tienen la misma longitud, ya que las longitudes de los segmentos que los componen fueron equilibradas.

En síntesis el procedimiento general para equilibrar las ecuaciones es el siguiente primero se asigna a $w(e_j)_D$:= 1 al construir *D*, luego aquí se ejecutan el *OUT-PROCEDURE* y el *IN-PROCEDURE* en ese orden, para equilibrar los $w(e_j)_D$. Por último se crea un modelo UCA donde $|sg_j|_{UCA} := w(e_j)_D$.

Algoritmo 3.7: out-procedure[11]

Elegir un vértice cualquiera v de V(D) y encontrar un out-arborescence T de D, con raíz v*.*
Repetir las siguientes operaciones,
para cada vértice v de V(T), $v \neq v^*$, en orden hoja-raíz:
si $w^-(v) < w^+(v)$ entonces poner
$w^+(u) := w^+(u) + w^+(v) - w^-(v)$
$w_j := w_j + w^+(v) - w^-(v)$ y
$w^-(v) := w^+(v)$
donde $e_j = uv$ es el eje de T que termina en v.

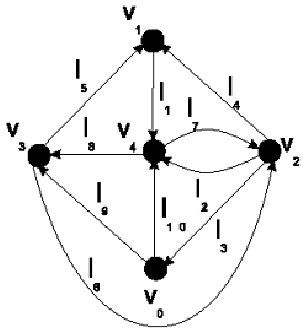
Algoritmo 3.8: in-procedure[11]:

Construir un in-arborescence T de D, con la misma raíz v que la elegida para la computación de OUT PROCEDURE para D.*
Repetir la siguientes operaciones,
para cada vértice v de V(T), $v \neq v^*$, en orden hoja-raíz:
si $w^+(v) < w^-(v)$ entonces poner
$w^-(u) := w^+(u) + w^-(v) - w^+(v)$
$w_j := w_j + w^-(v) - w^+(v)$
$w^+(v) := w^-(v)$
donde $e_j = uv$ es la arista de T que termina en v.

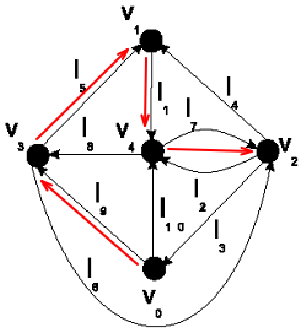
La idea es partir del vértice hoja *v* e ir equilibrando todos los vértices hacia la raíz. Cuando *v* tiene $w^-(v) < w^+(v)$ (o sea cuando tiene menos flujo de entrada que de salida) se dice es *in-deficiente.* Al terminar out-procedure no hay ningún *v* in-deficiente. Luego con el in-procedure se equilibran los vértices donde $w^+(v) < w^-(v)$ es decir se eliminan los out-deficientes. Una vez terminados ambos procesos podemos estar seguros que todos los vértices están balanceados, a excepción tal vez de *v**. Pero como *D* es fuertemente conexo y no existe ninguna arista desde o hacia el exterior de la componente fuertemente conexa, entonces *v** queda automáticamente balanceado.

El algoritmo para generar un out-arborescence (in-arborescence) *T* de un grafo dirigido *D* (i. e. *T* es un árbol generador dirigido de *D*) simplemente efectúa un recorrido DFS y genera un árbol

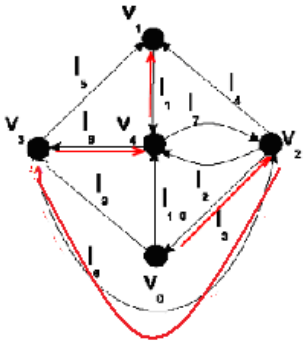
con todos los vértices de *D* y los ejes visitados. Donde el punto de partida de DFS es la raíz de la arborescencia. Entonces todas las aristas de *T* se dirigen hacia las hojas. Para el caso de in-arborescence todos las aristas se dirigen hacia la raíz. El algoritmo para generar un in-arborescence es el mismo pero se usa DFS con las aristas de *D* transpuestas.



Digrafo de segmento D



out-arborescence de D, los ejes de D usados se dirigen a las hojas



in-arborescence, es una out-arborescence de D^t, los ejes de D usados se dirigen a la raíz.

Fig. 3.6 : Ejemplo de arborescencia out e in generadas por el algoritmo a partir de un digrafo de segmento y un ordenamiento dado por DFS sobre *D* y D^t respectivamente. Ambas con vértice distinguido en v_0.

Descripción de la implementación de la construcción del modelo UCA

Los procedimientos aquí descriptos implementan eficientemente los algoritmos para la construcción del modelo UCA, a partir de un digrafo de segmento fuertemente conexo *D*. La misión principal de éstos es equilibrar el peso de los vértices, modificando a su vez el peso de las aristas de *D*. Por último se construye un modelo arco circular unitario.

Primero se construyen las out e in arborescencias, su función principal es establecer una forma de recorrer *D* para que out e in-procedure puedan equilibrar las aristas. Se implementó un procedimiento que genera arborecencias out o in según los parámetros que recibe. El input de este procedimiento es un digrafo de segmento que conoce un ordenamiento procesado por DFS, en pasos previos. Si el input es *D* genera una out-arborescence, donde *T* es un árbol con las aristas que de una raíz con dirección a las hojas, la raíz es el vértice de donde partió DFS. Si el input es D^t genera una in-arborescence, donde al volver a poner las aristas como en *D* todos las aristas tienen dirección hacia la raíz. La cantidad de iteraciones de este procedimiento es *n,* pero DFS tiene $m + n$ pasos, donde *m* es la cantidad de aristas de *D*, como $m = 2n$ el orden es lineal.

Las implementaciones de OUT-PROCEDURE e IN-PROCEDURE, reciben como input a *D* y lo primero que hacen es generar un out-arborescence e in-arborescence *T,* respectivamente. Utilizan un arreglo auxiliar de tamaño *n*, que contiene una secuencia de ordenamiento de los vértices calculado previamente cuando se realizó SCC. Esta estructura sumado al arreglo de *n*-1 aristas

que nos provee *T*, nos permiten recorrer los vértices de *D* en el ordenamiento hoja-raíz simplemente iterando sobre los arreglos desde la última posición de estos hacia la primera. El arreglo de vértices nos da la forma de recorrerlo el de aristas nos dice que arista de *D* hay que modificar. Durante el recorrido se asignan los nuevos valores de los pesos de las aristas en tiempo constante. En total se realizan *n* iteraciones, el procedimiento consume pues un tiempo lineal.

Finalmente el procedimiento que construye el certificado positivo recibe de entrada un digrafo de segmento *D* y devuelve un modelo de representación arco circular, en este caso como *D* es débilmente euleriano el modelo generado es unitario.

Se recorren las 2*n* aristas del digrafo de segmentos, y a partir de allí se arma un arreglo de segmentos, en 2*n* iteraciones. A cada segmento sg_i se le asigna una longitud, que sale del peso de la arista $w(e_i)_D$. Luego se recorren los segmentos en paralelo con los extremos del modelo, a cada extremo e_i se le asigna una nueva posición, ésta posición es el punto de inicio del segmento sg_i, que es un valor entero. En total se efectúan 4*n* iteraciones, el orden es lineal. Como output se obtiene un archivo con el modelo, con el mismo formato que el de input y un gráfico que lo representa con los arcos alrededor de un círculo.

Cuando la cantidad de aristas es menor a 2*n* se tomó la decisión de completar con valor 1 las longitudes de los segmentos que coinciden con una arista $v_0 \rightarrow v_0$. Este tipo de aristas mapean a segmentos que no se encuentran en el sistema reducido. Estos segmentos no están contenidos en ningún arco del modelo normal, o bien están contenidos en todos. El primero de los casos puede ocurrir cuando en el modelo hay un segmento sin arcos, y el segundo cuando el modelo representa a un grafo completo. Si la intención fuera reducir la longitud del círculo *C* del modelo se podría asignar la una longitud tan chica como uno quiera como una optimización, aunque siempre mayor a cero, para evitar que haya extremos del circulo que compartan puntos.

Certificado Negativo

En esta sección se describen los algoritmos que están involucrados en la construcción del certificado negativo. En este caso el certificado negativo es una secuencia de arcos que conforman el grafo prohibido *CI*(*n, k*). En la implementación, el programa devuelve como output un dibujo donde se presentan estos arcos resaltados sobre el modelo arco circular propio del input.

Construir el certificado negativo

Algoritmo 3.9: Construcción Certificado no UCA [12]:

Sea G un grafo PCA y no-UCA, y sea D su digrafo de segmento entonces:

1. *Como D no es fuertemente conexo, entonces ubicamos en D una componente fuertemente conexa CFC secundaria C, que sea fuente o sumidero.*

2. *Determinar la <descripción mínima> de C y obtener los <selected arcs> que representan al subgrafo prohibido CI(k, t).*
3. *Construir las secuencias de (k, t)-circuito y (k, t)-conjunto-independiente, despejando el valor t que utiliza la función Next.*

Como *D* no es fuertemente conexo, se sabe que no es UCA por los Teoremas 2.4 y 2.5. Luego se elige una componente fuertemente conexa secundaria *CFC* de *D*, que no contenga al vértice v_0. La componente debe ser fuente o sumidero, el siguiente teorema nos asegura que ésta siempre existe.

***Teorema 3.4**. [12]: Si un digrafo de segmento D no es fuertemente conexo entonces existe una componente fuertemente conexa secundario CFC en D que es fuente o es sumidero.*

La CFC debe ser fuente (sumidero) ya que de esta forma el lado derecho (izquierdo) de la *selección del sistema reducido*, es decir de las ecuaciones del sistema reducido que pertenecen a la CFC, se hace nulo [12]. Esto sucede por la definición de fuente ya que no puede haber aristas entrantes desde afuera de la componente fuente. Cabe aclarar que los sistemas de ecuaciones mencionados son equivalentes [12]. Los arcos de estas componentes conforman un grupo de ecuaciones que no se pueden equilibrar con los arcos que le siguen. Por ejemplo si de A_i , y A_{i+1} y A_{i+2} pertenecen a la seleccionada CFC, pero A_{i+3} no, entonces $|A_i| \neq |A_{i+3}|$.

Se determina la descripción por rangos mínima, a partir de los índices de los vértices de *CFC.* Esta descripción mínima nos permite seleccionar del sistema reducido las ecuaciones de la *CFC*, formando la selección del sistema reducido. Luego con estas ecuaciones se construye una secuencia de arcos denominados selected arcs de la siguiente forma: primero se incluyen los arcos del lado derecho de la selección del sistema reducido y luego los del lado izquierdo. Este conjunto de selected arcs contiene los arcos que conforman el grafo prohibido *CI(k,t).*

El (*k*, *t*)-circuito esta formado por la primera mitad de los selected arcs y el (*k*, *t*)-conjunto independiente por el resto. También puede haber otros arcos que no estén en los selected arcs (aquellos que no pertenecen a *CI*(*k, t*)). A partir de aquí solo resta averiguar el valor de *t* ya que *k* es la mitad del tamaño de selected arcs.

El valor de *t* se puede determinar a partir de t_{i1} (s_{i1}), donde el subíndice *1* identifica que pertenece al primer rango de la descripción mínima, y la el subíndice *i* indica el inicio de éste cuando *CFC* es fuente (sumidero). Para ello, se determina la posición de su extremo inmediatamente anterior en el sentido horario entre los selected arcs, que es $s_{iNext(1)}$ ($t_{iNext(1)}$), en tiempo constante. Esto se debe a que por ser un modelo normal los extremos mantienen un ordenamiento circular relativo constante entre ellos [12].

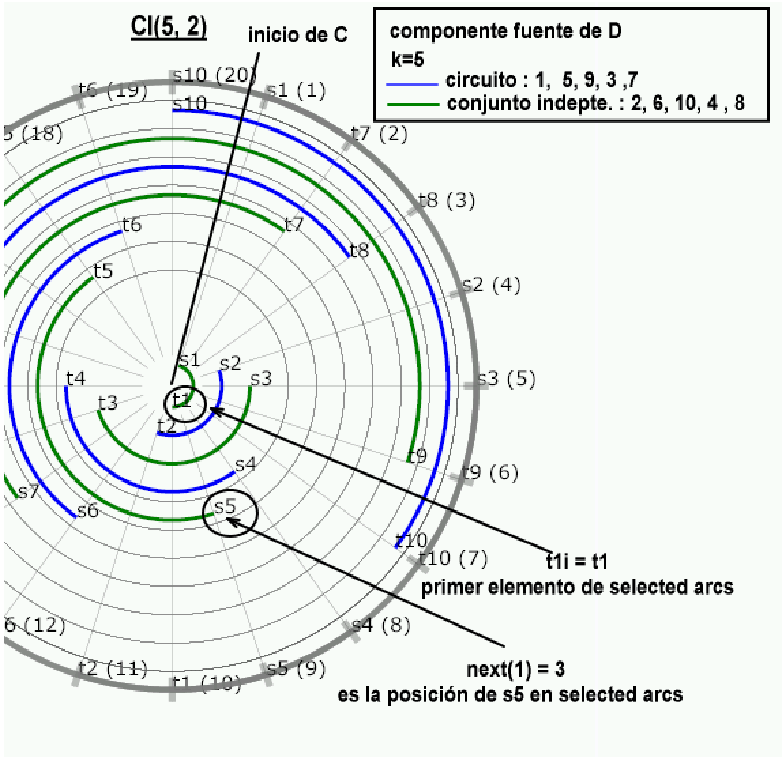


Fig. 3.7 : Ejemplo cálculo de *t* en grafo prohibido *CI*(5,2)

Con lo cual se puede hallar el valor de *Next(1)* y podemos calcular *t = (Next(1)-1) mod k* , en *O*(1). Next(1) nos da el índice del rango en el que está el extremo anterior. Si el extremo anterior está en el rango siguiente quiere decir que dio solo una vuelta, si está dos rangos más adelante es por que dio 2 vueltas. Por ejemplo si t es 2 los arcos se presentan en orden horario pero intercalados de a 2, en el grafo CI(5, 2) selected arcs es [1 3 5 7 9 2 4 6 8 10] y el orden de (*5*, *2*)-circuito es [1 5 9 3 7] y de (*5*, *2*)-conjunto independiente es [2 6 10 4 8].

Descripción de la implementación de la construcción del certificado negativo.

A continuación de describe una implementación eficiente de los algoritmos mencionados en esta sección.

Procedimiento 3.5: de Construcción del certificado negativo UCA

1. *Obtener una componente secundaria fuertemente conexa CFC de D*
2. *Calcular la DescripcionMinima dm de CFC*
3. *Averiguar si CFC es fuente o sumidero*
4. *Hallar los SelectedArcs sa de D*
5. *Calcular k como el tamaño de sa/2*
6. *Si CFC es fuente → calcular Next(1) como $s_{inext(1)}$ (modelo PCA, sa)*
7. *Sino → Next (1) es $t_{inext(1)}$(modelo PCA, sa)*
8. *Calcular t como (Next (1)-1) mod k*

9. *Devolver (k, t)-circuito y (k, t)-conjunto-independientet*

Las componentes fuertemente conexas de *D* ya fueron calculadas previamente en tiempo lineal, mediante al algoritmo de Kosaraju, cuando se determinó si el modelo era o no UCA. Para elegir una componente fuente o sumidero, se toma al vértice v_0, y se saltea la componente a la que pertenece, entonces recorriendo una vez cada uno de los *n* vértices se evalúan los grados de entrada y salida totales de cada componente y se elige la primera que cumpla con lo pedido.

Las descripciones mínimas *dm* se implementan como un par de números enteros (dm_{INICIO}, dm_{FIN}) que indican los índices de inicio y fin de los segmentos consecutivos de la componente *CFC* secundaria. La descripción mínima *DM* de *D* es un arreglo de *descripciones mínimas* que en ningún caso puede tener un tamaño mayor a *n*. En este punto es sencillo hallar *DM*, se recorren los vértices v_i del digrafo de segmentos *D* que pertenecen a *CFC* seleccionada, cada v_i conoce a que componente pertenece desde SCC. Cuando se encuentra un v_i que pertenece a *CFC* se coloca el índice *i* en dm_{INICIO} y se va avanzando sobre los vértices hasta hallar un vértice v_j que no esté en *CFC* y coloca el indice *j*-1 en dm_{FIN}. Luego se busca el siguiente vértice en *CFC*. El recorrido del arreglo de vértices se realiza en tiempo lineal.

Procedimiento 3.6: Calcular los Selected Arcs

Sea DM una Descripción Mínima de D, sea k el tamaño de DM, Sea Selected Arcs sa una lista de enteros que representa a los arcos del modelo PCA de tamaño 2k.

1. *Recorrer DM, con i de 0 a k*
2. *asignar en sa[i] ← el valor izquierdo de DM[i]*
3. *asignar en sa[i+k] ← el valor derecho + 1 de DM[i]*

A partir de las descripciones mínimas se arma un arreglo de enteros que representan a los arcos del modelo PCA de *G*, en donde los arcos de la primera mitad (los primeros *k*), pertenecen al conjunto de arcos que forman el (*k, t*)-circuito y los arcos de la otra mitad (los últimos *k*), pertenecen a (*k*, *t*)-conjunto-independiente. Todo esto contando que la componente elegida es fuente, ya que en caso de ser sumidero, los primeros *k* arcos corresponden al (*k, t*)-conjunto-independiente y el resto al (*k, t*)-circuito.
El arreglo de descripciones mínimas se recorre una vez por lo tanto el algoritmo tiene una complejidad temporal lineal.

Procedimiento 3.7: calcular next(1) para una componente fuente

Dado el modelo PCA y sa los Selected Arcs.

1. pos_t_{i1} ← *posición del extremo final del arco de la posición sa[0]-1*
2. $pos\_s_{inext(1)}$ ← $post_{i1} - 1$
3. *Si* ($pos\_s_{inext(1)} < 0$) *entonces*
4. $pos\_s_{inext(1)}$ ← *la cantidad de extremos de r*
5. $label\_s_{inext(1)}$ ← *etiqueta del extremo de la posición* $pos\_s_{inext(1)}$
6. *devolver* $pos\_s_{inext(1)}SA$ ← *posición en sa del arco* $label\_s_{inext(1)}$

El algoritmo obtiene el valor de t correspondiente al grafo *CI*(*k, t*). Para calcular next(1) en el caso en que la componente es fuente, se recorre la primera mitad de los selected arcs; en el caso de ser sumidero se recorre la segunda mitad. Se calcula el valor de *t* como *next*(1)-1 *mod k* esto es la distancia (medida en cantidad de extremos) al punto previo inmediato módulo la cantidad de elementos del circuito/conjunto-independiente. Este valor *t* nos indica el número de vueltas alrededor de *C*, para el grafo prohibido *CI(t, k)*. Por lo antes visto el valor de t se obtiene en tiempo lineal.

Detalles de la Implementación en la máquina

La Implementación fue realizada en el lenguaje de programación *Java*[1], para la salida gráfica de los modelos se utilizó el formato *SVG*[2] de gráficos vectoriales. La elección de esta tecnología permitió realizar tres interfaces: una en consola para medir tiempos de ejecución; otra stand alone utilizando *javax.swing.**; y por último una interfaz web para poder dejar la aplicación para el uso público. Para las estructuras de datos se utilizaron las bibliotecas estándar *java.util.** . No se utilizaron bibliotecas específicas de grafos.

La aplicación esta organizada en un proyecto compuesto de 5 paquetes: *representation*, *uca*, *ui*, *graph* y *test*. El paquete *representation* agrupa las clases referentes a la operación y composición de los modelos de representación de los grafos, *uca* contiene clases referentes a la determinación de la pertenencias y construcción y verificación de un modelo UCA, *graph* contiene a las clases que componen a un grafo orientado, *ui* a la que interactúan con el usuario y por último *test* que agrupa el manejo de los contadores de operaciones y timers para la evaluación del tiempo de ejecución.

A continuación se explican algunos detalles de la Implementación de los algoritmos como así también de decisiones tomadas.

La aplicación recibe como entrada un archivo de texto conteniendo un modelo PCA cuyos extremos están ordenados en sentido horario.

---

[1] Entorno de desarrollo Java 2 SDK, Standard Edition, Version 1.6 Entorno de ejecución Java(TM) Runtime Environment, Standard Edition Version 1.6. La plataforma de desarrollo utilizada fue Eclipse Version: 3.1.2.

[2] SVG Scalable Vector Graphics (http://www.w3.org/Graphics/SVG).

Muchos de los arreglos utilizados están implementados utilizando la clase *ArrayList*, donde las operaciones *size*, *isEmpty*, *get* y *set* se ejecutan en tiempo constante. La operación *add* corre en tiempo constante amortizado, esto es, al agregar *n* elementos requiere $O(n)$. Cada instancia de *ArrayList* tiene una capacidad que es el tamaño del arreglo que es utilizado para guardar los elementos. Mientras los elementos son agregados con la operación *add* al *ArrayList*, la capacidad crece automáticamente. Una aplicación puede incrementar la capacidad de la instancia de *ArrayList* antes de agregar una gran cantidad de elementos utilizando *ensureCapacity*. Esto puede reducir la cantidad de reasignaciones de memoria incrementales, lo que resulta muy importante para esta implementación debido a que el tamaño de los ArrayLists utilizados siempre es conocido de antemano desde que se obtiene el modelo de input, por lo que las reasignaciones de memoria podrían llevarse al mínimo, en todos los casos.

Al normalizar se hace una copia de la representación del modelo PCA y se la normaliza. Lo mismo para construir el modelo UCA se hace una copia del modelo Normal. Copiar los modelos no es estrictamente necesario para calcular o construir los certificados, sin embargo se tomó esta decisión para permitir una mejor salida gráfica (poder tener los tres modelos al mismo tiempo) y para poder realizar las autenticaciones de los certificados, en un procesamiento posterior.

También se construyó una aplicación para generar los modelos de entrada. Se generan modelos UCA, PCA pero no UCA y PCA aleatorios, que se guardan en archivos de texto conteniendo un modelo con los extremos ordenados tal como se obtendrían del algoritmo de Deng et al.[3].

Cuando se utilizan modelos de más de 100 vértices se anula la parte gráfica, principalmente porque los gráficos se vuelven muy complejos de analizar visualmente. También porque la salida gráfica usa componentes que podrían no estar implementados con complejidad lineal, lo que aumentaría considerablemente el tiempo de ejecución en modelos grandes con más de 100 arcos. Se utilizan librerías *java.awt.** para la salida de java y *java.io.PrintWriter* para exportar a svg. Para medir los resultados especialmente el tiempo en milisegundos se ejecutó un versión sin salida gráfica que corre en una consola tipo MS-DOS en una PC sin antivirus, firewall ni internet. Sin embargo el sistema operativo Windows ejecuta otros procesos que pueden competir por el procesador y la memoria y aumentar el tiempo medido en milisegundos.

Complejidad y alcance de los algoritmos

Hacemos ahora un breve resumen de la complejidad total del algoritmo, uniendo los resultados que obtuvimos para cada algoritmo por separado. Para determinar si *G* es un grafo PCA y obtener el correspondiente modelo (*C*, ***A***), se puede implementar el algoritmo [3] tiene una complejidad $O(n+m)$ [3]. Este modelo arco circular propio sirve de input para los algoritmos de [11, 12] implementados en esta tesis.

Teorema 3.1 *[11]: El algoritmo normal termina dentro de un tiempo O(n). Hay menos de 6n iteraciones del bloque repetir, durante todo el proceso.*

Durante la normalización los arcos de (*C,* ***A***) pueden ser encogidos como mucho 2 veces durante todo el algoritmo. En consecuencia, las operaciones de encogimiento contribuyen con 2*n* en la cuenta final, y el algoritmo realiza menos de 6*n* iteraciones del bloque *Repetir,* donde cada una de estas iteraciones se realiza en tiempo constante. Por lo tanto la complejidad total para el algoritmo normal es *O(n)*.

Para construir el digrafo de segmento, se construye un full system y luego el sistema reducido. El primero es una familia de *n*-1 ecuaciones, cuyas variables son las longitudes de los segmentos. A cada lado de la ecuación los segmentos son consecutivos. Por lo tanto se requieren 4*n* - 4 índices para representar estas ecuaciones.

Además, en el sistema reducido, cada longitud de segmento aparece a los sumo dos veces. Esto quiere decir que necesita a los sumo 4*n* índices. El digrafo de segmento tiene *n* vértices y a los sumo 2*n* aristas, y se construye en tiempo *O*(*n*) a partir del sistema de ecuaciones reducido.

Para *m* la cantidad de aristas y *n* la cantidad de vértices, las componentes fuertemente conexas de un grafo se pueden construir en tiempo *O(n + m)* [16, 20]. Construir los in/out-arborescences, también consumen *O*(*m*) tiempo. Estos algoritmos son utilizados con digrafos de segmentos, que tienen 2*n* ejes, por lo tanto, la complejidad temporal de estos algoritmos se reduce a *O*(*n*).

El siguiente teorema asegura que los algoritmos que construyen el certificado positivo [11] utilizan enteros de tamaño lineal, evitando así el problema que tiene algoritmo de Tucker [23, 24].

Teorema 3.2 *[11]: Sea (C, A) un modelo UCA construido por el algoritmo entonces los extremos de (C, A) corresponden a números enteros de tamaño < 4n.*

Para la construcción del certificado negativo, cuando el digrafo de segmento *D* no es fuertemente conexo, se encuentra en *D* una componente fuertemente conexa secundaria *CFCs* que sea fuente o sumidero en tiempo *O*(*n*) [12]. Luego se determina la descripción mínima de *CFCs* y se obtienen los selected arcs que representan al subgrafo prohibido *CI*(*n, k*) también en tiempo *O*(*n*) [12].

Para construir las secuencias del (*k, t*)-circuito y (*k, t*)-conjunto-independiente solamente hace falta despejar el valor *t* que utiliza la función *Next*. Este valor se puede determinar a partir de t_{i1} (s_{i1}) si *CFCs* es fuente (sumidero), conociendo su extremo inmediatamente anterior en el sentido horario s $s_{iNext(1)}$ ($t_{iNext(1)}$) que se puede encontrar con *O*(*1*) operaciones constantes. Con lo cual se halla el

valor de *Next*(1) y podemos calcular t = (*Next*(1)-1) *mod k* en tiempo constante [12].

El costo total del algoritmo que construye el certificado negativo es de $O(n)$ tiempo [12].

Por lo tanto la complejidad total para determinar si un grafo es UCA construyendo los certificados es de $O(n+m)$. Cuando el input es directamente un modelo PCA, la complejidad del algoritmo [11, 12] se reduce a $O(n)$), como es el caso del algoritmo implementado en esta tesis.

Resultados computacionales

El principal punto a evaluar en esta sección es que el conjunto de algoritmos necesarios para determinar si el modelo es UCA o no y la construcción de los modelos respectivos se ejecuten en tiempo lineal. Además comprobar empíricamente el Teorema 3...[11] que indica que no se efectuarás más de $6n$ iteraciones del bloque principal del algoritmo que efectúa la transformación normal del modelo PCA del input. Por último corroborar el Teorema 3...[11] que indica que los extremos del certificado construido corresponden a número enteros menores a $4n$.

Métodos

Elección del conjunto de prueba:

Para poder comprobar la eficiencia de los algoritmos se utilizaron 3 conjuntos de modelos de representación, el primero formados por modelos conocidos por ser UCA, el segundo por modelos conocidos por ser PCA-y-no-UCA, y el tercero por modelos PCA construidos de manera aleatoria por lo tanto sin conocer de antemano a que resultado llegarán.

Construcción del conjunto de prueba:

Los modelos UCA se generan utilizando la familia de grafos K_n, ordenando los arcos de tal modo que requieran una alta tasa de normalización, es decir solapando arcos de tal forma que de a dos arcos cubran la circunferencia. Los modelos PCA-y-no-UCA se generan extendiendo la familia de grafos prohibidos descripta por Tucker [21], como grafos $CI(k, t)$, y comentada en el Capítulo 2 de esta tesis. El último grupo se construye con un generador aleatorio de grafos PCA, en todos los casos las longitudes de los arcos son aleatorios, preservando la adyacencia de los arcos. Los 2 primeros grupos de modelos nos sirven para constatar tanto la correctitud del algoritmo como para poder calcular el cumplimiento de la complejidad temporal lineal en la implementación, el tercer grupo se utiliza principalmente para esto último.

Mediciones:

Para calcular los tiempos de ejecución de los algoritmos, fueron contadas las operaciones de suma, resta, comparación, multiplicación y división, y las operaciones sobre listas agregar, asignar y acceder a un elemento de una lista.

También se utilizó un doble control al medir el tiempo en nanosegundos, para poder tener un control sobre el conteo anterior. Para estas mediciones se utilizó una versión de consola DOS del soft, para minimizar interferencias del sistema operativo y no se incluyó en este calculo la parte gráfica, ya que el orden de esta podría no ser lineal y podría tergiversar los resultados cuando el modelo es grande.

Resultados

Para el procedimiento normal se obtuvieron menos de 6*n* iteraciones del bloque "repeat"[3], por lo tanto se cumple con el Teorema 3.1 [11], en la practica no se obtuvieron mas de 3*n* iteraciones, se midió la performance de este algoritmo en particular, ya que es central para el reconocimiento del grafo, además para corroborar el teorema y despejar las dudas que presenta a primera vista los ciclos anidados que contiene éste. El conjunto de prueba esta formado por modelos PCA de la familia de grafos K_n (que ya se sabe son UCA, ver en el Gráfico 3.1 serie UCA), y otro grupo aleatorios (en la ver en el Gráfico 3.1 serie RAN), con un ordenamientos circular de extremos tal que requieran normalización, se testeó con *n* de 4 a 100000, se utilizó este conjunto K_n como input ya que tienen entre *n*/2 y *n*-1 pares de arcos que cubren *C*, lo cual es una tasa alta que requiere encoger arcos en todos los casos, y nos permite probar intensamente el procedimiento de normalización.

[3] El bloque "repeat" es el ciclo más interno del procedimiento normal y debe chequear si dos arcos cubren C y cuando esto ocurre mantener el orden del arreglo de extremos al reinsertar los extremos en orden constante. Asi como pasar al próximo arco a examinar y al próximo extremo a examinar.

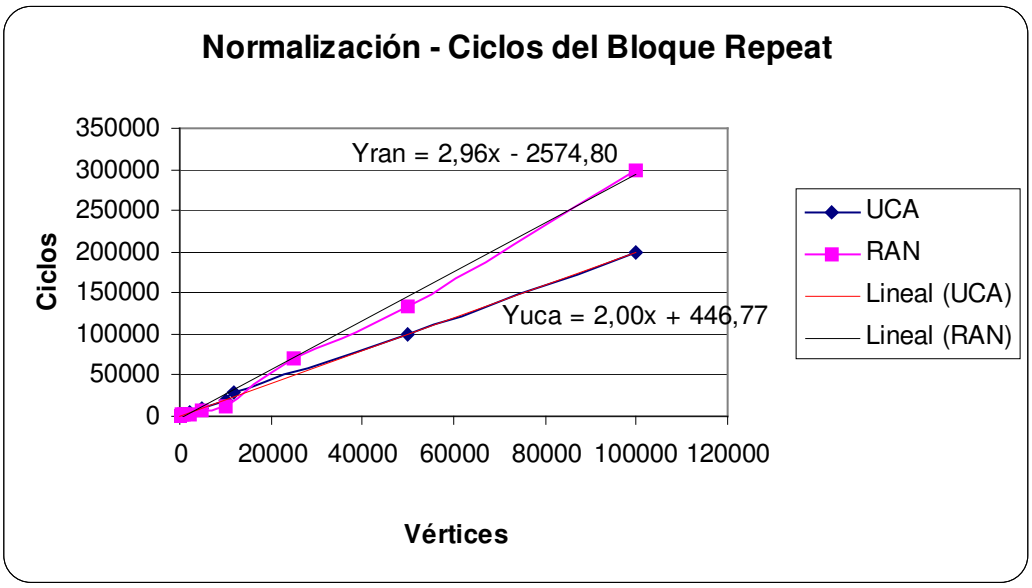


Gráfico 3.1: Gráfico de la cantidad de ciclos del bloque interno del algoritmo normal realizadas para normalizar el modelo PCA según la cantidad de arcos del modelo.

En cuanto al conteo de operaciones[4] y el tiempo de ejecución medido en milisegundos, para el mismo conjunto de datos, ver Gráfico 3.1 y 3.2 se observó un comportamiento lineal y los resultados obtenidos se alinean con la función *f(x)=127x – 88000* (88.000 parece un número alto pero si pensamos en las 13.000.000 de operaciones necesarias para computar 100.000 arcos totales son casi 3 ordenes de magnitud menos por lo tanto no es muy significativo, el R^2=0,997), por su parte el tiempo transcurrido contado en milisegundos para estos ejemplos se ajusta a la función *f(x) = x - 500*.

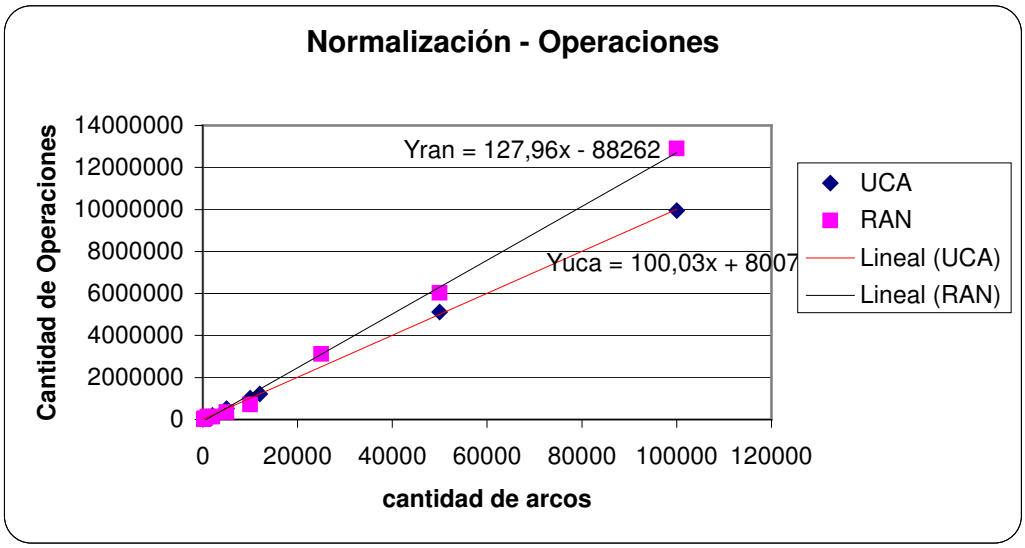


Gráfico 3.2: Gráfico de operaciones realizadas para normalizar el modelo PCA según la cantidad de arcos del modelo.

[4] Se contabilizan operaciones aritméticas de suma, resta, multiplicación, división, módulo, comparaciones lógicas, y otras operaciones sobre colecciones y objetos get(), set(), add ().

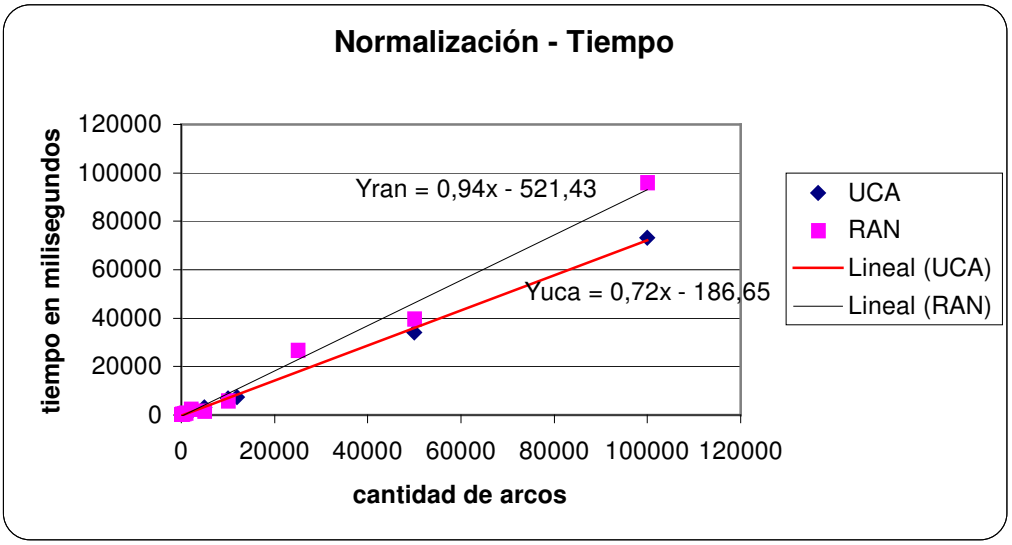


Gráfico 3.3: Gráfico de la cantidad de tiempo medido en milisegundos para normalizar el modelo PCA según la cantidad de arcos del modelo.

Para determinar si el modelo ingresado es UCA o no, se obtuvieron resultados que se ajustan a una función lineal, se contabilizaron las operaciones hasta el momento de la decisión. Se utilizó líneas de tendencia, para calcular las funciones que indican una cota superior e inferior para los resultados obtenidos, como input se usaron modelos PCA de las familias de grafos K_n (desde 8 a 50000 arcos), de la familia descripta por Tucker [21] (desde 4 a 50000 arcos) y modelos generados en forma aleatoria (desde 5 a 50000 arcos), en todos los casos los resultados obtenidos fueron inferiores a $f(x) = 250x+c$ (ver Gráfico 3.2). También se efectuó un control del tiempo en milisegundos con los mismos conjuntos de datos. Los resultados también se ajustan a una función lineal, se obtuvieron como cota superior e inferior las siguientes dos funciones $f_{UCA}(x)=1,47x -190$ y $f_{no\text{-}UCA}(x)=0.58x -129$. (ver Gráfico 3.3).

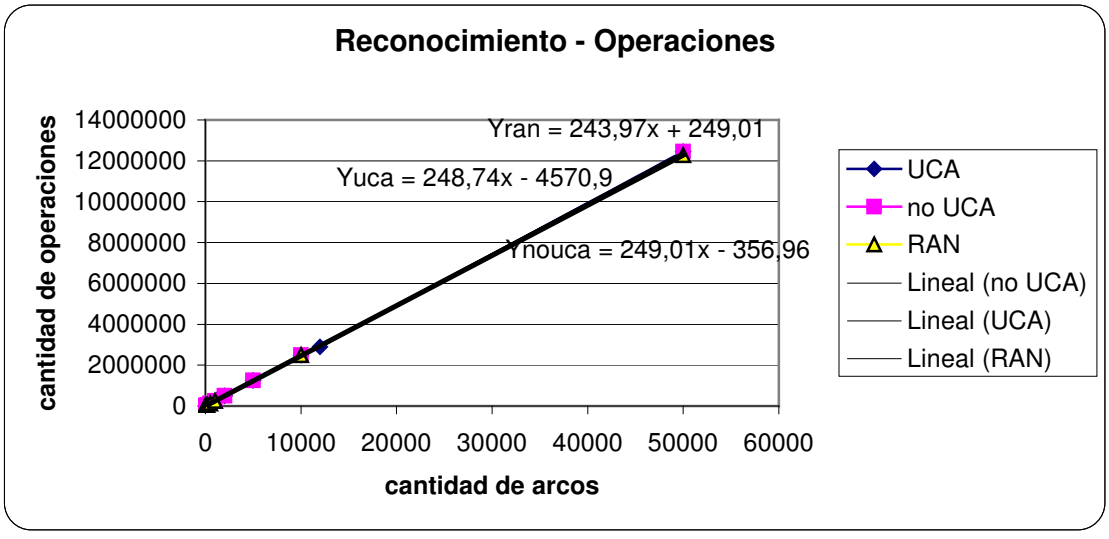


Gráfico 3.5: Gráfico de operaciones realizadas para determinar si el modelo es UCA según la cantidad de arcos del modelo, para tres conjuntos de datos uno conocido UCA, otro conocido no-UCA y otro PCA aleatorios.

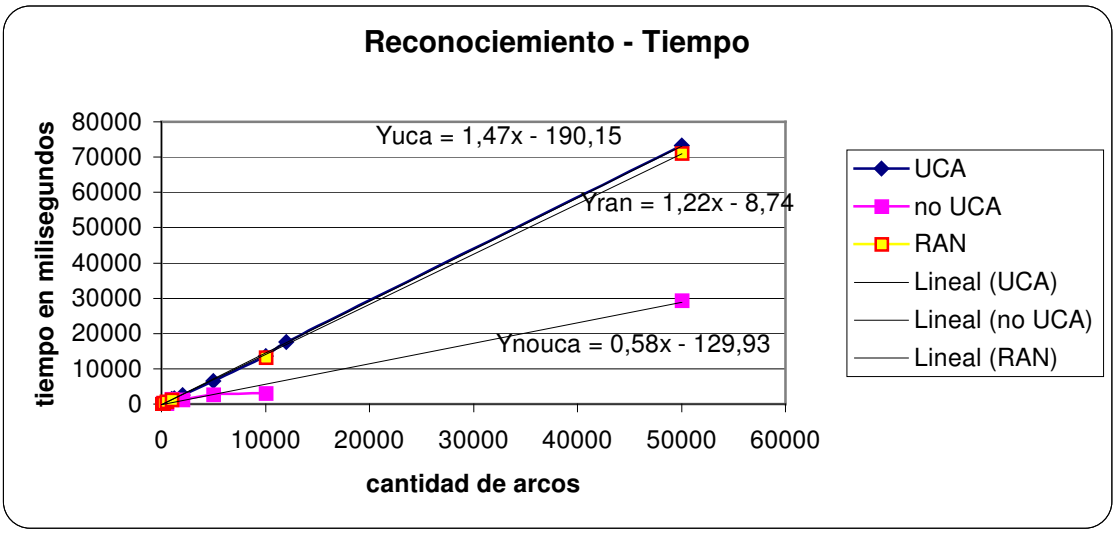


Gráfico 3.6: Gráfico del tiempo medido en milisegundos para determinar si el modelo es UCA según la cantidad de arcos del modelo, para tres conjuntos de datos uno conocido UCA, otro conocido no-UCA y otro PCA aleatorios.

Para la construcción de los certificados (positivos y negativos) se contabilizaron todas las operaciones reconocimiento + construcción, el conjunto de input es el mismo que para la etapa de reconocimiento. Las rectas obtenidas a partir de los resultados van entre $y_1 = 390x + c_1$ e $y_2 = 284x + c_2$, por lo tanto es lineal y en milisegundos las rectas obtenidas a partir de los resultados van entre $y_1 = 3{,}85x + c_1$ a $y_2 = 0{,}66x + c_2$ por lo que también es lineal, donde la grafos generados aleatoriamente muestran una pendiente más marcada y que la construcción del certificado negativo muestra una pendiente bastante inferior a la del certificado positivo.

Por último se comprobó en todos los casos que los extremos del modelo UCA construido corresponden a números enteros de tamaño $< 4n$. Los resultados obtenidos se aproximan a una recta cercana $2n + c$, con c una constante aproximándose a cero para n grande.

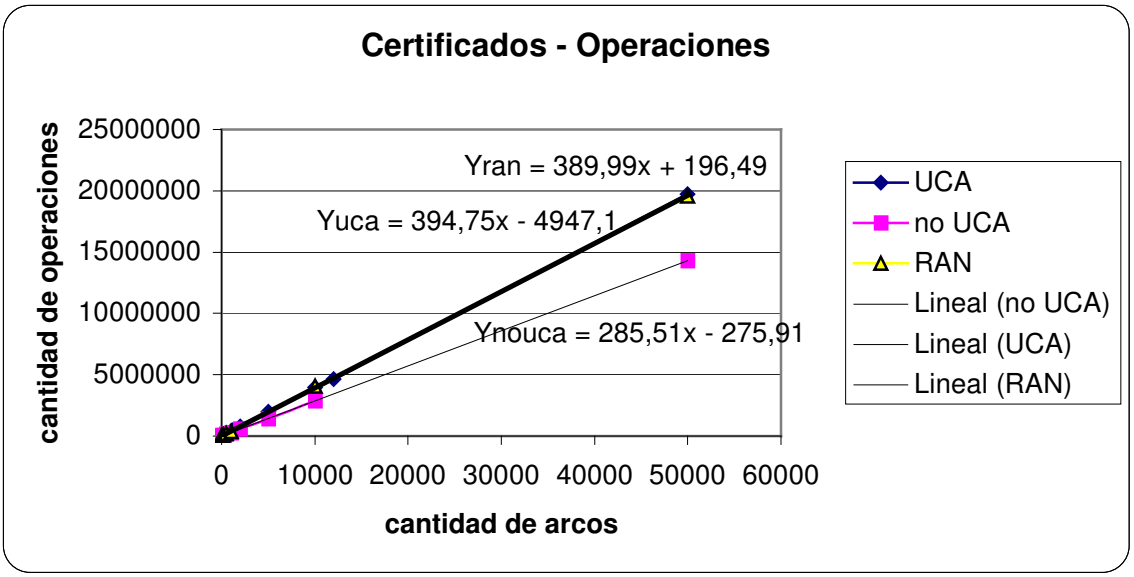


Gráfico 3.7: Gráfico de la cantidad de operaciones realizadas para construir un certificado según la cantidad de arcos del modelo, para tres conjuntos de datos uno conocido UCA, otro conocido no-UCA y otro PCA aleatorios.

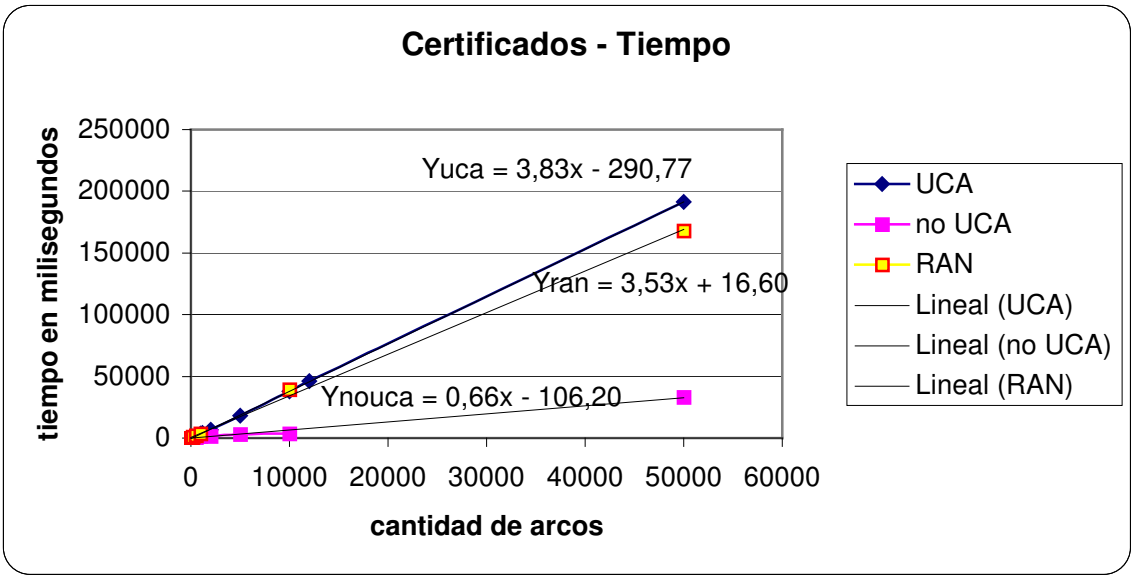


Gráfico 3.8: Gráfico del tiempo medido en milisegundos para construir un certificado según la cantidad de arcos del modelo, para tres conjuntos de datos uno conocido UCA, otro conocido no UCA y otro aleatorios.

Capítulo 4

Autenticación de los Certificados UCA y no UCA

En esta sección se presentan algoritmos para la autenticación de ambos certificados de los grafos arco-circulares unitarios. Se utilizan dos algoritmos principales, uno para el certificado positivo, cuando es UCA y otro para el certificado negativo es decir cuando el modelo PCA no es UCA. Primero se describen los algoritmos y luego se presenta una implementación eficiente de estos para autenticar los certificados emitidos por la implementación de los algoritmos de [11 ,12] explicada en el Capítulo 3. Por último también se presenta un algoritmo para autenticar un modelo PCA, que es utilizado como input, y su implementación.

Descripción de los algoritmos de la autenticación de Certificados Positivos

El siguiente grupo de algoritmos permite verificar la autenticidad de un modelo arco circular unitario.

Algoritmo 4.1 : Autenticación del certificado positivo

1. *verificar que cada arco tiene la misma longitud*
2. *verificar que ambos PCA y UCA son modelos equivalentes, es decir que el grafo de intersección de ambos es el mismo.*

Con estos dos pasos se autentica que el certificado sea realmente un modelo unitario y que por otra parte el grafo subyacente del certificado sea el mismo que el del input.

Al verificar la longitud de los arcos se está diciendo a su vez que el grafo es PCA, ya que si todos los arcos tienen igual longitud no puede haber uno incluido en otro, teniendo en cuenta que no hay dos arcos que comiencen en el mismo punto.

Algoritmo 4.2 : Comparar Grafos de Intersección de modelos UCA y PCA.

1. *recorre todos los arcos y los extremos de los dos modelos en paralelo*
    1.1. *chequea para cada par de arcos que los arcos que se solapan son los mismos*

Chequea que los dos modelos tengan el mismo grafo de Intersección verificando que los arcos que

se solapan son los mismos en ambos modelos, es decir las intersecciones son las mismas, entonces el grafo de intersección subyacente es el mismo.

Detalle de la implementación de la autenticación de Certificados Positivos

A continuación de describe una implementación eficiente de los algoritmos de autenticación de certificados positivos para modelos UCA.

Procedimiento 4.1 : Comparar Grafos de Intersección de modelos UCA y PCA

```
1.      uj←0, pj←0      //posición extremos de pca y uca del arco j
2.      j←1             //contador de arcos
3.      eu←0 ,ep←0      //contadores de ejes entre vértices
// chequear que ambos modelos tengan los mismos vértices
4.       si [tamaño arcos de UCA] <> [tamaño arcos de PCA]
5.              salir
// chequear que ambos modelos tengan los mismos ejes
6.      Mientras(j < [tamaño arcos de PCA]){
7.       pj ← [posición(del extremo inicial del arco PCA (de la posición j-1))]
8.       uj ← [posición(del extremo inicial del arco UCA (de la posición j-1))]
9.       pj ← pj+1
10.      uj ← uj+1
11.      Mientras no haya llegado al final del arco en PCA
12.       extremoPcai ← [Extremo PCA de la posición (pj modulo(2*np))]
13.      Si tipo(extremoPcai) == 's' y label(extremoPcai) <> [j + ep + 1]
14.             devolver false
15.      Si tipo(extremoPcai) == 's' y label(extremoPcai) == [j + ep + 1]
16.             ep ← ep + 1
17.      pj ← pj+1
18.      Mientras no haya llegado al final del arco en UCA
19.      extremoUCAi ← [Extremo UCA de la posición (uj módulo(2*np))]
20.      Si tipo(extremoUCAi)== 's' y label(extremoUCAi) <> [j + eu + 1]
21.       devolver false
22.      Si tipo(extremoUCAi)== 's' y label(extremoUCAi) == [j + eu + 1]
23.             eu ← eu + 1
24.      uj ← uj + 1
25.      si ep <> eu
26.             devolver false
27.      j←j+eu
28.      devolver true
```

Recorre en paralelo los dos modelos chequeando que los arcos que se solapen sean los mismos es decir que tengan el mismo grafo de intersección y aprovecha el ordenamiento circular relativo de extremos para recorrer solo un vez el arreglo de arcos de tamaño *n* pero la cantidad de comparaciones esta dada por las cantidad de aristas *m*. Por lo tanto el orden temporal es *O(n+m)*. Para determinar que todos los arcos tengan la misma longitud simplemente se recorre una vez el arreglo de arcos del modelo.

Descripción de los algoritmos de la autenticación de Certificados Negativos

Los siguientes algoritmos permiten verificar la autenticidad del certificado negativo para grafos UCA, construidos por la implementación de los algoritmos de [12] descripta en el Capítulo 3.

Algoritmo 4.3 : Autenticación certificado negativo
El algoritmo verifica que:

1. *(k, t)-circuito sea efectivamente (k, t)-circuito.*
2. *Lo mismo con (m, l)-conjunto independiente.*
3. *ver que n/k=m/l.*

Se verifica que las partes del grafo *CI(k, t)* cumplan con sus definiciones así se autentica que el certificado negativo contenga al grafo prohibido, en este caso el modelo del input no se ve alterado por lo que no se requiere corroborar que el grafo subyacente sea el mismo. El siguiente corolario del Teorema 2.1 justifica esta autenticación.

Corolario 4.1 [21]: *Sea G un grafo PCA. Entonces G es un grafo no-UCA si y sólo si G tiene un (n, k)-circuito minimal y un (m, l)-conjunto independiente maximal que satisfagan que n/k = m/l.*

Para verificar (*k ,t*)-circuito se recorren los arcos del modelo chequeando que los k arcos se solapen en el mismo orden tal como indica (*k ,t*)-circuito. Y para verificar el (*m, l*)-conjunto-independiente se recorren los arcos del modelo comparándolos para ver que no se solapen los m arcos que pertenecen al conjunto independiente. La idea para verificar *n/k = m/l* es recalcular *n, m, k* y *l* a partir del (*n, k*)-circuito y de (*m, l*)-conjunto independiente y no usar los calculados en el certificado.

Detalle de la implementación de la autenticación de Certificados Negativos

Detalle de una implementación eficiente de los algoritmos para autenticar el certificado negativo para grafos UCA.

Procedimiento 4.2 : verificar (*m, l*)-conjunto independiente

1. *Mientras haya arcos sin visitar y no se encuentre el primer arco*
   1.1. *Avanzar hasta hallar el primero en Independent set*
2. *Mientras haya arcos sin visitar //comparación i-ésimo con ((i+t)mod(k))-ésimo*
   2.1. *Visitar arco a*
   2.2. *Si a es parte de Independent set*
      2.2.1. *comparar a con el arco siguiente en independent set*
         2.2.1.1. *Si no se solapan entonces seguir*
         2.2.1.2. *si se solapan devolver false*
      2.2.2. *comparación n-ésimo arco en independent set con el primero hallado*
         2.2.2.1. *Si no se solapan entonces devolver true*

2.2.3. *sino devolver false*

Este algoritmo recorre el conjunto independiente de tal forma que cada elemento del conjunto independiente se compara con el siguiente, cuando el *CI(k, t)* tiene un *t*>1 el ordenamiento de los arcos no es igual al del conjunto independiente por lo que hay que tener en cuenta el tamaño de *t* para ver como se intercalan, para poder armar un índice y así recorrerlo en tiempo lineal. La Implementación para autenticar el (*k ,t*)-circuito es muy similar a la de (*m, l*)-conjunto independiente.

Para verificar *n/k = m/l* se recorre el modelo arco por arco y se cuenta el tamaño del circuito *n* y del conjunto independiente *m* y la cantidad de vueltas, *k* y *l* respectivamente, que dan a la circunferencia cada uno de ellos, luego compara que *n/k* sea igual a *m/l*.

Algoritmos de autenticación para modelo PCA

A continuación se describe un algoritmo eficiente que valida un certificado PCA. El algoritmo utiliza el ordenamiento de los arcos y extremos para recorrer el modelo sobre los extremos de sus arcos.

Algoritmo 4.4 : autenticar modelo PCA input: modelo PCA, output: valor de verdad.

1. setear rangos de índices de arcos de extremos: abiertos[-1,-1], cerrados[-1,-1]
2. Recorrer los extremos del modelo
3. Mientras haya extremos e sin visitar hacer
3.1. si e es un extremo inicial s
3.1.1. si e es el primer extremo
3.1.1.1. setear abiertos[índice$_e$, índice$_e$]
3.2. sino
3.2.1. si e es el [extremo derecho de abiertos +1] módulo n
3.2.1.1. setear abiertos[x, índice$_e$+1]
3.3. si e es un extremo final t
3.3.1. si e esta en abiertos[]
3.3.1.1. case (e es igual al lado izquierdo de abiertos[arco$_e$,x])
3.3.1.1.1. setear abiertos[índice$_e$ +1,x]
3.3.1.2. case (abiertos[arco$_e$, arco$_e$])
3.3.1.2.1.1. setear abiertos[-1,-1]
3.3.1.3. case (e es interno a abierto[x, y]) salir: NO ES PROPIO
3.3.2. sino
3.3.2.1. si ya hubo algún arco cerrado salir: NO ES PROPIO
3.3.2.2. si el rango cerrados[] esta vacío //inicial
3.3.2.2.1. setear cerrados[índice$_e$, índice$_e$]
3.3.2.3. sino
3.3.2.3.1. si e está en el final del rango cerrados[x, índice$_e$]
3.3.2.3.1.1. setear cerrados[x, índice$_e$+1]
3.3.2.3.2. sino esta en otra posición salir: NO ES PROPIO
4. devolver PROPIO

El algoritmo considera que los extremos de los arcos están ordenados, de esta manera, utilizando rangos de índices de arcos para saber cuales están abiertos y cuales están cerrados, se puede chequear para cada extremo si su ubicación es válida en un modelo propio. Se recorre el arreglo de los extremos de los arcos una sola vez. Se efectúan 4*2*n* comparaciones de enteros, se puede

implementar la autenticación del modelo PCA en tiempo *O(n)*. La implementación de este algoritmo se utiliza para chequear que el modelo arco circular del input sea propio en la aplicación que reconoce y certifica los grafos UCA descripta en el Capítulo 3.

```
verificación: EL MODELO ES PCA
Circuito [ 1 3 5 7 9 ]
verificación: (n,k)circuit es un circuito
Conjunto Independiente [ 2 6 10 4 8 ]
verificación: (m,l)independent set es un conjunto independiente
verificación:  n/k IGUAL a m/l
```

Fig. 4.1 : Ejemplo de salida de la autenticación para el grafo *Cl*(5,2).

Detalles de la implementación

El módulo de autenticación también fue implementado en el lenguaje de programación Java[5]. Para las estructuras de datos se utilizaron las librerías estándar *java.util.**. Los métodos que se implementan en este módulo están dentro de la clase *UCAVerifier* dentro del paquete *uca* . Las autenticaciones se ejecutan al finalizar la construcción de los certificados, el módulo recibe como entrada los modelos PCA Original y UCA y devuelve un archivo de texto conteniendo el resultado de cada punto de las verificaciones. Este módulo también chequea que el input recibido por la aplicación sea un modelo PCA con los extremos de los arcos ordenados circularmente.

Complejidad y alcance de los algoritmos

En la autenticación del certificado positivo, para saber si el modelo es UCA basta saber si es PCA y si los arcos tienen todos la misma longitud, esto último implica a lo primero. Pero además hay que verificar que el modelo UCA sea equivalente al modelo de input. Para verificar la longitud de los arcos simplemente hay que recorrer los n arcos. Para chequear el grafo de intersección de ambos hay que chequear las adyacencias de los vértices es decir que tengan las mismas *m* aristas. Chequear las longitudes de los arcos lleva *O(n)* y chequear que ambos grafos de intersección sean iguales se puede hacer en *O(n+m)*. Por lo tanto el orden total es *O(n+m)*.

[5] Entorno de desarrollo Java 2 SDK, Standard Edition, Version 1.6 Entorno de ejecución Java(TM) Runtime Environment, Standard Edition Version 1.6. La plataforma de desarrollo utilizada fue Eclipse Version: 3.1.2.

Para la *verificación certificado negativo,* se verifica que (*n, k*)-circuito sea efectivamente un circuito, este algoritmo se completa en tiempo lineal, ya que se recorren los *k* arcos y se compara el *i*-ésimo con *i*+1-ésimo, en un circuito se chequean las adyacencias (v_i y v_{i+1} son adyacentes con $1 \leq i \leq n-1$, y v_n es adyacente a v_1). Con (*m, l*)-conjunto independiente se recorren los *m* arcos indicados como pertenecientes al conjunto independiente para verificar que cada arco del conjunto independiente no sea adyacente con su siguiente. Como $m \leq n/2$ el orden total es de O(n). Por lo tanto se realiza en tiempo O(n) y la comparación final n/k=m/l lleva un tiempo O(n) ya que no se utilizan los valores calculados para la construcción del certificado sino que se calculan a partir de recorrer una vez el modelo generado.

El modelo es PCA si y solo si no hay arcos contenidos dentro de otros, Kaplan y Nussbaum [10] propone chequear las adyacencias para cada par de vértices controlando el orden de sus extremos y obtiene así un certificado de tamaño *O(n)* en un tiempo de autenticación *O(n+m)*. En esta tesis se implementó la misma idea utilizando los extremos de los arcos pero utilizando rangos de arcos abiertos y rangos de arcos cerrados para corroborar las adyacencias, de esta forma se recorre una sola vez el arreglo de extremos de tamaño 2n, por lo tanto el tamaño de *O(n)* y con un total de 2*n* iteraciones el tiempo de autenticación es O(*n*).

Resultados computacionales

El conjunto de modelos a evaluar se conformó a partir de un conjunto de archivos de entrada con diferentes modelos de representación de grafos arco circulares, construidos con el generador de modelos mencionado en el capítulo anterior, y compuesto por modelos de grafos PCA, de los que se conoce de antemano si son UCA o si no lo son. De esta forma se puede corroborar si el resultado obtenido es el esperado. También se utilizaron modelos generados aleatoriamente de tamaño pequeño donde la corroboración puede realizarse de manera visual.

Se pudo verificar en todos los casos (con modelos de hasta 50000 arcos) la autenticidad de los certificados construidos por el aplicativo. En los casos en que ya se conocía el resultado de antemano se corroboró lo ya sabido. Y para los modelos generados de forma aleatoria o los casos de test puntuales se pudo confirmar las validez de los certificados emitidos.

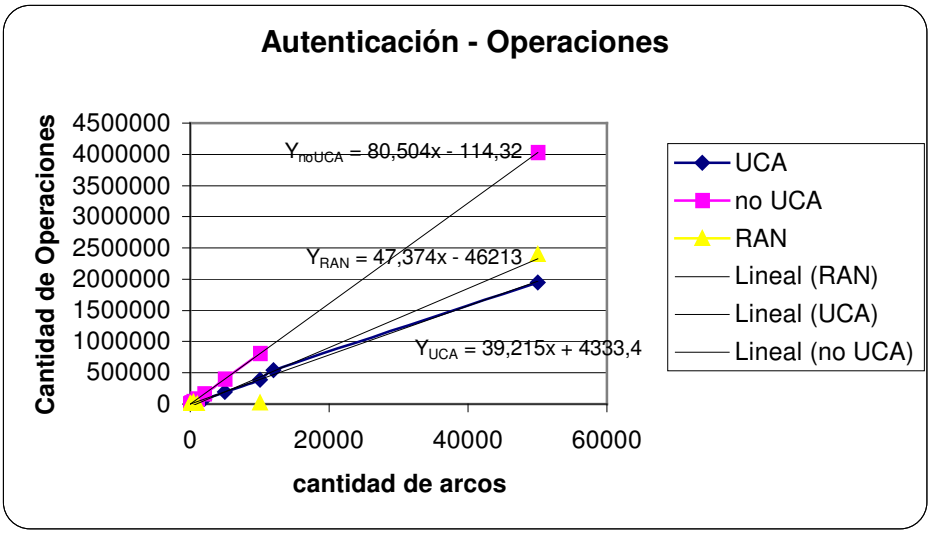


Gráfico. 4.1: Gráfico de la cantidad de operaciones realizadas según la cantidad de arcos del modelo

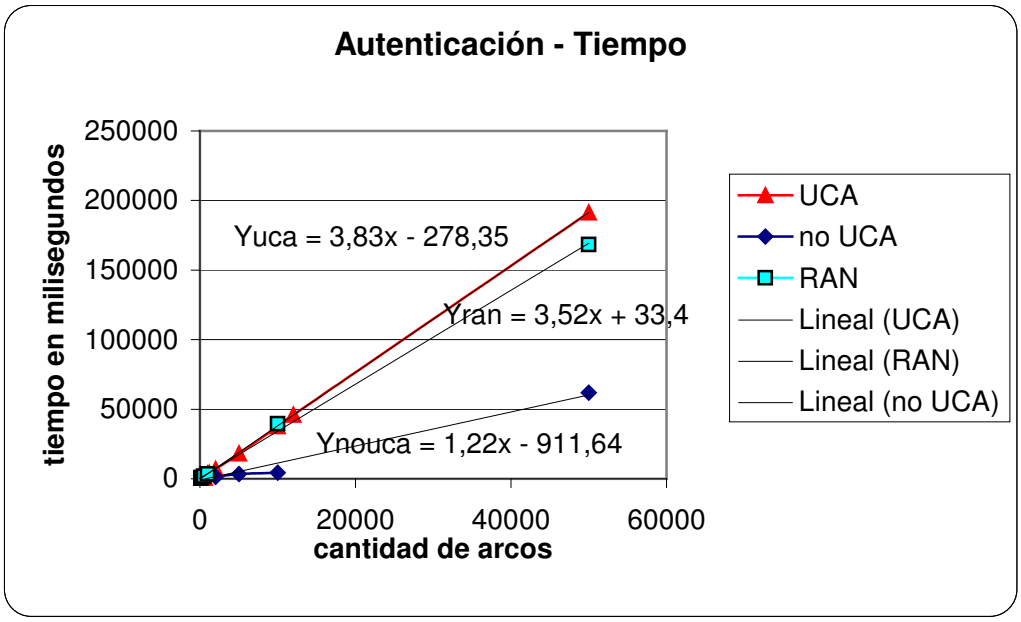


Gráfico. 4.2: Gráfico del tiempo medido en milisegundos según la cantidad de arcos del modelo

En cuanto a la complejidad temporal de los algoritmos utilizados se contaron las operaciones realizadas y el tiempo en milisegundos y nanosegundos como control del primero. Tal como en el capítulo anterior, el resultado obtenido para modelos de hasta 50000 arcos, en ambas mediciones, se ajusta a una función lineal más similar a *O(n)* que a *O(n+m)* que es el orden teórico calculado, siendo *n* la cantidad de arcos y *m* la cantidad de ejes del grafo subyacente del modelo PCA (hay que tener en cuenta que en la serie UCA *m = n*(n-1)*).

Las operaciones se ajustan a una recta con una pendiente inferior 81*n* (Gráfico 4.2), en el caso de las mediciones del tiempo se midió en nanosegundos porque el tiempo transcurrido era muy corto,

sin embargo no dio resultados significativos como para sacar conclusiones en el Gráfico 4.2 se muestra el tiempo completo en milisegundos que incluye la construcción de los certificados + la autenticación, donde se ve una pendiente inferior $4n$.

Conclusiones

En este trabajo llevamos a cabo una implementación de los algoritmos de Lin y Szwarcfiter [11,12], que permiten determinar fehacientemente si un grafo es arco circular unitario (UCA), o bien si no lo es, devolviendo en ambos casos el certificado correspondiente, todo esto en tiempo lineal. Estos algoritmos toman como punto de partida un modelo arco circular propio (PCA), cuyos extremos se encuentren ordenados. Estos modelos de entrada se pueden generar para un grafo PCA utilizando el algoritmo de Deng et al. [3].

Los algoritmos en cuestión son descriptos detalladamente, se calcula su orden teórico y se muestra su implementación. Cada algoritmo fue implementado como un método de Java para poder hacer un mapeo directo de estos con el código fuente. También se implementaron otros algoritmos que están citados en [11,12], como el de determinar las componentes fuertemente conexas de un grafo orientado [16]. Los tiempos de ejecución obtenidos en la práctica corroboraron los resultados teóricos. Para medirlos se construyó un conjunto de prueba y se realizó un doble control, se contaron las operaciones efectuadas y los nanosegundos transcurridos en cada ejecución. Para cada método en particular y para el conjunto total se, confirmó en la práctica que el tiempo de ejecución es de O(n), tanto para la parte de reconocimiento del modelo UCA, como así también para la construcción de ambos certificados.

En el Capítulo 4 desarrollamos algoritmos de autenticación para los certificados construidos según [11,12]. La autenticación se vuelve fundamental para poder verificar los certificados generados ya que, a simple vista, se pueden cometer errores de interpretación, sobretodo cuando la cantidad de vértices es grande. Asimismo su implementación nos permitió encontrar bugs y errores de implementación antes no detectados en la etapa de construcción de los certificados. Por otra parte se mejoró el tiempo del algoritmo de autenticación de un modelo PCA de $O(n+m)$ [10] a $O(n)$.

La verificación de los certificados no representa un costo muy alto ya que lleva un tiempo $O(n+m)$, el cual para grafos pequeños no es relevante. Para grafos grandes se podría asumir el riesgo de excluir de la autenticación, la verificación en la cual se chequea que el modelo final represente al mismo grafo que el original, permitiendo en este caso, reducir la complejidad total para la autenticación de certificados a un tiempo de $O(n)$.

Como trabajos futuros queda el problema de encontrar un modelo UCA, para un grafo UCA, cuyos puntos extremos correspondan a números enteros, de tal forma que se minimice el largo del máximo segmento. Es decir, que se minimice la longitud de la circunferencia. Sobre este punto podemos conjeturar que la cota mínima es el grado del vértice de mayor grado + 1, es decir cota

mínima = $d_G(v)_{máx}$ +1. En este trabajo comparamos los valores de los grados de los vértices con la longitud de los arcos obtenidos, viendo que aunque generalmente dan igual, no siempre son iguales. Por lo tanto esta construcción no necesariamente arroja el mínimo. Otro dato relevante es que no se obtuvieron valores experimentales para la longitud de los arcos en ningún caso superiores a 2 veces la cota mínima, por lo tanto ésta podría ser una cota máxima.

Bibliografía

[1] G. Confessore, P. Dell'Olmo and S. Giordani, An approximation result fora periodic allocation problem, Discrete Applied Mathematics 112 (2001),pp. 53–72.

[2] V. Conitzer, J. Derryberry, and T. Sandholm. Combinatorial auctions with structured item graphs. In Proceedings of the Nineteenth National Conference on Artificial Intelligence (2004), pp. 212–218

[3] X. Deng, P. Hell and J. Huang, Linear time representation algorithms for proper circular–arc graphs and proper interval graphs, SIAM Journal of Computing, 25 (1996), pp. 390 – 403.

[4] G. A. Durán, Sobre grafos intersección de arcos y cuerdas en un círculo, Tesis Doctoral (2000)

[5] G. A. Durán, A. Gravano, R. M. McConnell, J. Spinrad, A. Tucker, Polynomial time recognition of unit circular–arc graphs, Journal of Algorithms, Volume 58 (2006), pp. 67 – 78.

[6] E. Eschen and J. Spinrad, An O(n2) Algorithm for Circular–Arc Graph Recognition, Proceedings of the 4th Annual ACM–SIAM Symposium on Discrete Algorithms (1993), pp. 128–137.

[7] M. C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980.

[8] L. Hubert, Some applications of graph theory and related non–metric techniques to problems of approximate seriation: the case of symmetry proximity measures, British J. Math. Statist Psychology 27 (1974), pp. 133–153.

[9] H. Kaplan, Y. Nussbaum, A Simpler Linear–Time Recognition of Circular–Arc Graphs, The Tenth Scandinavian Workshop on Algorithm Theory (2006).

[10] H. Kaplan and Y. Nussbaum, Certifying algorithms for recognizing proper circular–arc graphs and Unit Circular–Arc Graphs. WG 2006: 289–300

[11] M. C. Lin, J. L. Szwarcfiter, Efficient Construction of Unit Circular–Arc Models, Proceedings of the 17th Annual ACM–SIAM Symposium on Discrete Algorithms (2006), pp. 309 – 315.

[11'] M. C. Lin, J. L. Szwarcfiter, Unit Circular–Arc Graph Representations and Feasible Circulations.

[12] M. Lin and J. Szwarcfiter, Simpler certifying algorithm for recognizing unit circular–arc graphs using segment digraphs, Manuscrito.

[13] M. Lin and J. Szwarcfiter, Characterization and Recognition of

Circular-Arc Graphs and Subclasses: a Survey(2007).

[14] R. M. McConnell, A certifying algorithm for the consecutive ones property, Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete. Algorithms (SODA'04) (2004), pp. 716-770.

[15] F. S. Roberts, Graph Theory and Its Applications to Problems of Society, Society for Industrial and Applied Mathematics, Philadelphia, 1978.

[16] M. Sharir. A Strong Connectivity algorithm anf its applications in data flow analisys. Computers and Mathematics with applications,7:67-72,1981

[17] F. Stahl, Circular genetic maps, J. Cell Physiol 70(Suppl. 1)(1967),1-12.

[18] S. Stefanakos, T. Erlebach, Routing in All-Optical Ring Networks Revisited, Proceedings of the 9th IEEE Symposium on Computers and Communications (ISCC) (2004), pp. 288 – 290.

[19] K. Stouffers, Scheduling of traffic lights – A new approach, Transportation Res. 2 (1968), pp. 199 – 234.

[20] R. E. Tarjan. Depth first search and linear graph algorithms. SIAM Journal on Computing, 1(2):146-160, 1972.

[21] A. Tucker, Structure theorems for some circular-arc graphs, Discrete Mathematics 7 (1974), pp. 167 – 195.

[22] A. Tucker, Matrix characterizations of circular-arc graphs, Pacific J. Math. 38 (1971), pp. 535 – 545.

[23] A. Tucker, Characterizing circular-arc graphs, Bull. American Mathematical Society 76 (1970), pp. 1257-1260.

[24] A. Tucker, An effcient test for circular-arc graphs, SIAM J. Computing 9 (1980), pp. 1-24.

[25] A.Tucker, Coloring a family of circular-arc graphs, SIAM J. Appl. Math. 29 (1975), pp. 493 – 502.

[26] S. Benzer, On the topology of the genetic fine structure, Proc. Nat. Acad. Sci. U.S.A., 45:1607-1620, 1959.

Anexo I

Formato del archivo de entrada

Formato
-las líneas con # son ignoradas
-cada línea representa a un extremo de un arco
compuesto por:
Nro_arco, posicion_del_extremo, tipo_de_extremo

Definiciones

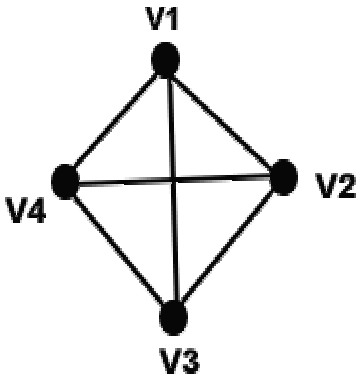
Nro_arco:	desde 1 a cantidad de arcos
posicion_del_extremo:	un valor entero mayor al del extremo anterior
tipo_de_extremo:	determina si es un extremo inicial s o final t

Se eligió un archivo de entrada donde se indican los extremos de los arcos y no los arcos directamente. El sistema requiere que ambos, extremos y arcos, se encuentren ordenados. De haber utilizado el ingreso por arcos ordenados se debería haber construido una estructura intermedia para rescribirlo por extremos (esta estructura puede construirse en tiempo lineal, ya que tanto los extremos iniciales como finales tienen un ordenamiento circular relativo entre sí), ya que el sistema necesita recorrer el modelo tanto por arcos como por extremos, por lo tanto este formato de input facilita la implementación.

En el Anexo II hay varios ejemplos de archivos de entrada con su respectivo grafo y modelo PCA .

Anexo II

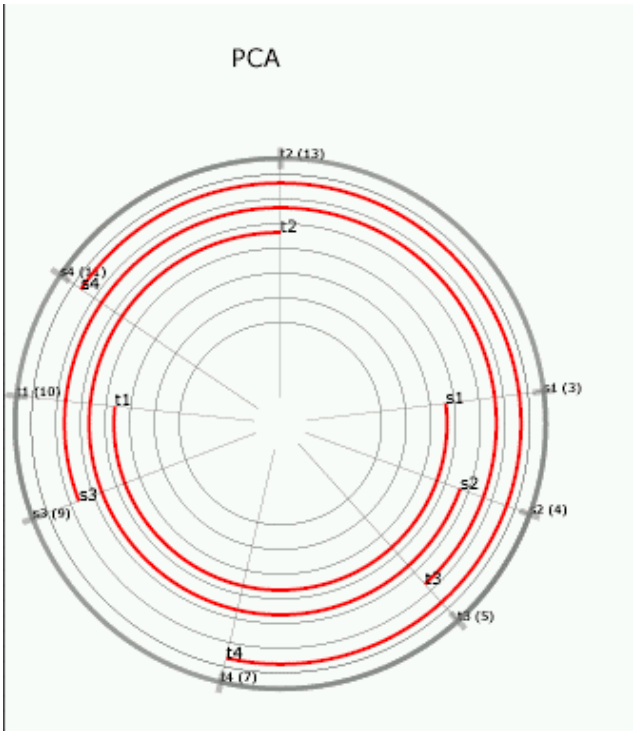
Corrida de un ejemplo donde el grafo donde se requiere normalizar.



Grafo k4

#archivo de input modelo PCA del Grafo k4
1,3.0,s
2,4.0,s
3,5.0,t
4,7.0,t
3,9.0,s
1,10.0,t
4,11.0,s
2,13.0,t

Fig. aII.1.1 Grafo k4 y secuencia de extremos del input



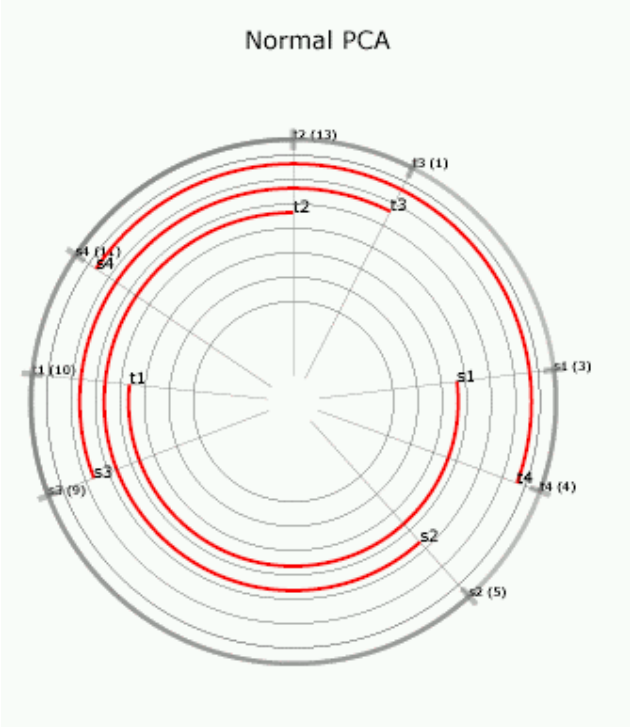
Modelo PCA

Mapea el vértice v_i de G con A_i, para i de 1 a n

Arcos:
A_1, A_2, A_3, A_4

Ordenamiento de los extremos del modelo PCA:
<extremo(valor)>
$s_1(3)$, $s_2(4)$, $t_3(5)$, $t_4(7)$, $s_3(9)$, $t_1(10)$, $s_4(11)$, $t_2(13)$

Fig. aII.1.2 modelo PCA para el grafo de la Fig. aII.1.2



Normalizacion:

Input: modelo PCA

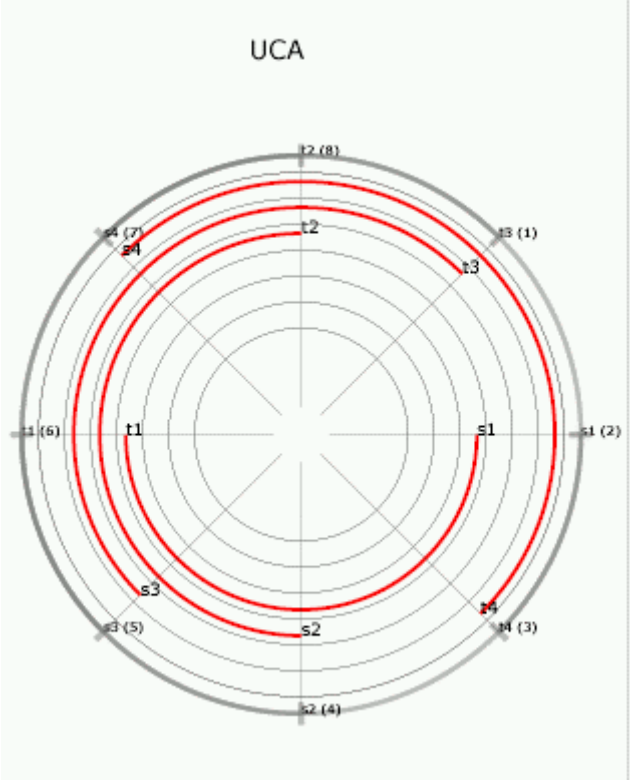
CUBRE s_1:3 t_1:10 s_3:9 t_3:5

CUBRE s_2:4 t_2:13 s_4:11 t_4:7

Repeticiones del bloque interno: 9

Ordenamiento de los extremos del modelo normal:
<extremo(valor)>
$t_3(1)$, $s_1(3)$, $t_4(4)$, $s_2(5)$, $s_3(9)$, $t_1(10)$, $s_4(11)$, $t_2(13)$

Fig. aII.1.3 Corrida del procedimiento de normalización para el grafo de la Fig. aII.1.1



Input: modelo normal

Arcos y su rango de segmentos:
$A_1[2,6]$, $A_2[4,8]$, $A_3[5,1]$, $A_4[7,3]$

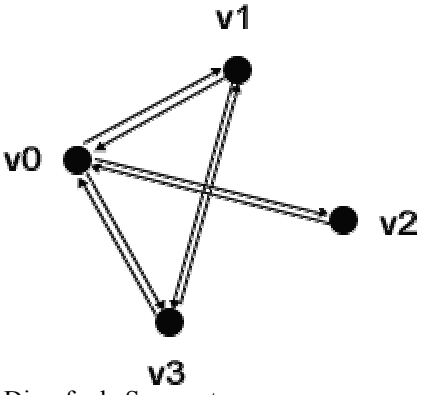
Ecuaciones del sistema reducido<label,[lb,le],[rb,re]>:
$q_1([2|3],[6|7])$, $q_2([4|4],[8|8])$, $q_3([5|6],[1|2])$

Vértices Digrafo de Segmento/pesos w^- y w^-(size:4)
v_0 (w^-: 3, w^+: 3),
v_1 (w^-: 2, w^+: 2),
v_2 (w^-: 1, w^+: 1),
v_3 (w^-: 2, w^+: 2)

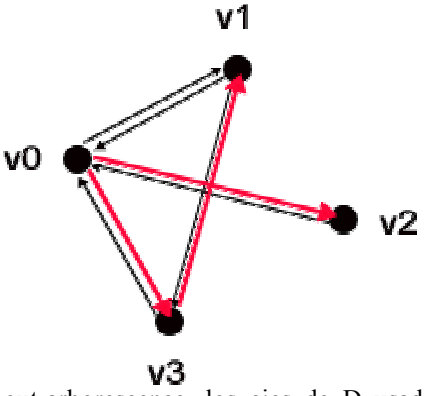
Ejes Digrafo de Segmento/peso de e/segmento (size:8)
(0, 3)w:1, Seg:1,
(1, 3)w:1, Seg:2,
(1, 0)w:1, Seg:3,
(2, 0)w:1, Seg:4,
(3, 0)w:1, Seg:5,
(3, 1)w:1, Seg:6,
(0, 1)w:1, Seg:7,
(0, 2)w:1, Seg:8

Correr depht first forest(D), depht first forest(Dt) y SCC.

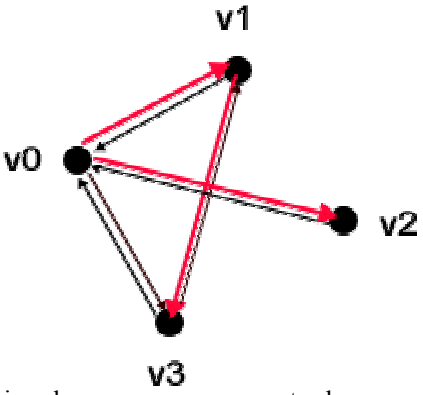
Fig. aII.1.4 Corrida de los procedimientos para reconocer si el grafo de la Fig. aII.1.1 es UCA



Digrafo de Segmento



out-arborescence, los ejes de D usados se dirigen a las hojas



in-arborescence, es una out-arborescence de D^t, los ejes de D usados se dirigen a la raíz

Out-arborescence T
OUT- PROCEDURE:
Ejes de T (size:3) : {(0, 3)w:1 Seg:0, (3, 1)w:1 Seg:1, (0, 2)w:1 Seg:2}

Orden de los vértices hoja-raiz: $v_2|v_0|v_1|v_3$

in-arborescence: T
IN- PROCEDURE:
Ejes de T (size:3)
(0, 1)w:1 Seg:0, (1, 3)w:1 Seg:1, (0, 2)w:1 Seg:2

Orden de los vértices hoja-raiz: $v_2|v_0|v_3|v_1$

Modelo UCA en el digrafo:
Vértices del digrafo de segmento (size:4)
$v_0(w^-: 3\ w^+: 3)$, $v_1(w^-: 2\ w^+: 2)$, $v_2(w^-: 1\ w^+: 1)$, $v_3(w^-: 2\ w^+: 2)$

mapeo ejes del digrafo de segmento/ segmentos del modelo UCA (size:8):
(0, 3)w:1 Seg:1, (1, 3)w:1 Seg:2, (1, 0)w:1 Seg:3, (2, 0)w:1 Seg:4, (3, 0)w:1 Seg:5, (3, 1)w:1 Seg:6, (0, 1)w:1 Seg:7, (0, 2)w:1 Seg:8

segmentos del modelo UCA<label,start point,lenght>:
{<1|1|1>, <2|2|1>, <3|3|1>, <4|4|1>, <5|5|1>, <6|6|1> <7|7|1>, <8|8|1>}

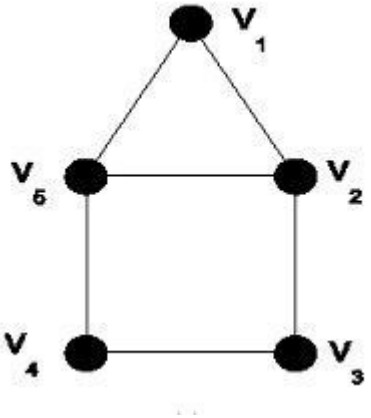
Longitud del Circulo C: 8

Ordenamiento de los extremos del modelo UCA <extremo(valor)> :
$t_3(1)$, $s_1(2)$, $t_4(3)$, $s_2(4)$, $s_3(5)$, $t_1(6)$, $s_4(7)$, $t_2(8)$

verificación:
- TODOS LOS ARCOS TIENEN LA MISMA LONGITUD
- AMBOS MODELOS SON EQUIVALENTES

Fig. aII.1.5 Corrida de los procedimientos para construir un certificado positivo para el grafo de la Fig. aII.1.1

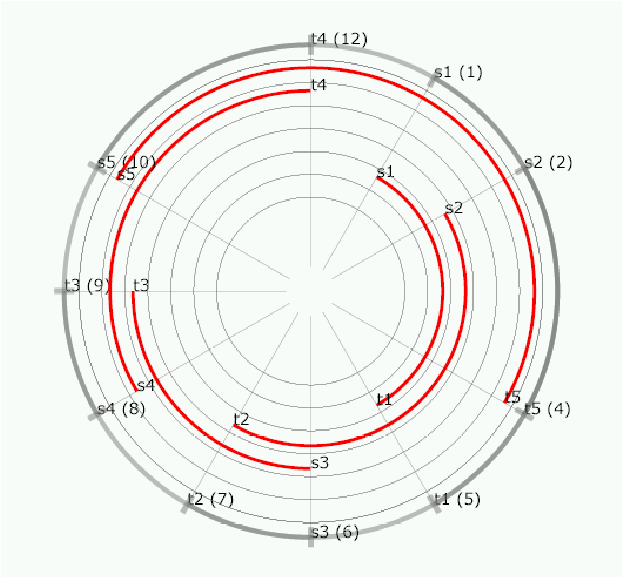
Corrida de un ejemplo donde el grafo donde el grafo es arco circular unitario y requiere equilibrar los vértices de las.



Grafo k4

#archivo de input modelo PCA del Grafo k4
1,1,s
2,2,s
5,4,t
1,5,t
3,6,s
2,7,t
4,8,s
3,9,t
5,10,s
4,12,t

Fig. aII.2.1 Grafo de 5 vértices y secuencia de extremos del input



Modelo PCA

Mapea el vértice v_i de G con A_i, para i de 1 a n

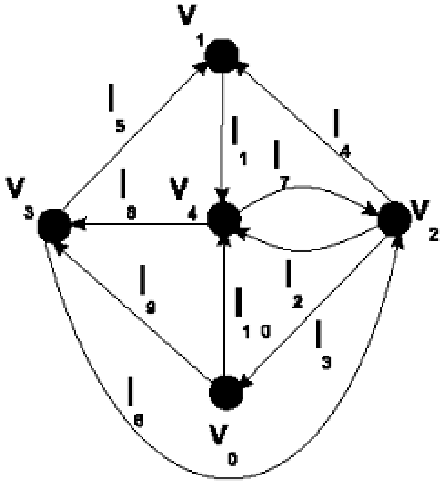
Arcos:
A_1, A_2, A_3, A_4, A_5

Ordenamiento de los extremos del modelo PCA:
<extremo(valor)>
s_1 (1), s_2 (2), t_5 (4), t_1 (5), s_3 (6), t_2 (7), s_4 (8), t_3 (9), s_5 (10), t_4 (12)

Normalizacion:
Repeticiones del bloque interno: 8
NO REQUIERE NORMALIZACION

Fig. aII.2.2 modelo PCA para el grafo de la Fig. aII.1.2

Digrafo de segmento



Input: modelo normal

Arcos y su rango de segmentos:
A_1 [1,4]a2[2,6] , A_3 [5,8] , A_4 [7,10] , A_5 [9,3]

Ecuaciones del sistema reducido y sus rangos de índices de segmentos:
q_1([1|1],[4|5]), q_2([2|4],[6|7]), q_3([5|6],[8|9]), q_4([7|8],[10|2])

Vértices Digrafo de Segmento/pesos w^- y w^-(size:5)
v_0 (w^-: 1, w^+: 2),
v_1 (w^-: 2, w^+: 1),
v_2 (w^-: 2, w^+: 3),
v_3 (w^-: 2, w^+: 2),
v_3 (w^-: 3, w^+: 2)

Ejes Digrafo de Segmento/peso de e/segmento size:10
(1, 4)W:1 Seg:1,
(2, 4)W:1 Seg:2,
(2, 0)W:1 Seg:3,
(2, 1)W:1 Seg:4,
(3, 1)W:1 Seg:5,
(3, 2)W:1 Seg:6,
(4, 2)W:1 Seg:7,
(4, 3)W:1 Seg:8,
(0, 3)W:1 Seg:9,
(0, 4)W:1 Seg:10

Correr depht first forest(D), depht first forest(Dt) y SCC.
---------- ----------

Fig. aII.2.3 construcción del digrafo de segmento

Out-arborescence

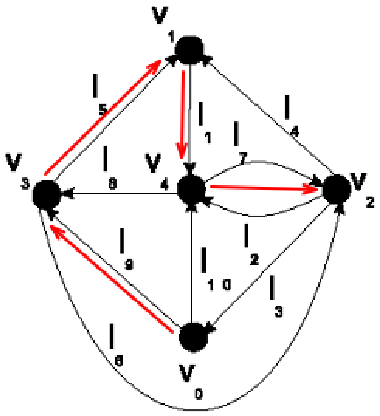


Fig. : out-arborescence de *D*, los ejes de D usados se dirigen a las hojas.

OUT- PROCEDURE:
Ejes de T size:4 (0, 3)w:1 Seg:0 (3, 1)w:1 Seg:1 (1, 4)w1 Seg:2 (4, 2)w:1 Seg:3

Vertices de D: (0 w-: 1 w+: 2) (1 w-: 2 w+: 1) (2 w-: 2 w+: 3) (3 w-: 2 w+: 2) (4 w-: 3 w+: 2)

Orden de los vértices hoja-raiz: $v_2|v_4|v_1|v_3|v_0$

[i: 4 (v_2→u_4)| $e_{7D} == e_{4T}$ | $v(w^-)$:2 < $v(w^+)$:3]

Set $u(w^+)$:4 = $u(w^+)$:2 + $v(w^+)$:3 - $v(w^-)$:2 = 3
Set $w(e_7)$:1 = 2
Set $v(w^-)$:2 = $v(w^+)$:3 = 3

[i: 3 (v_4→u_1) | $v(w^-)$:3 == $v(w^+)$:3]
[i: 2 (v_1→u_3) | $v(w^-)$:2 == $v(w^+)$:1]
[i: 1 (v_3→u_0) | $v(w^-)$:2 == $v(w^+)$:2]

Vertices de D (v0 w-: 1 w+: 2) (v1 w-: 2 w+: 1) (**v2 w-: 3 w+: 3**) (v3 w-: 2 w+: 2) (**v4 w-: 3 w+: 3**)

---------- ----------

In-arborescence:

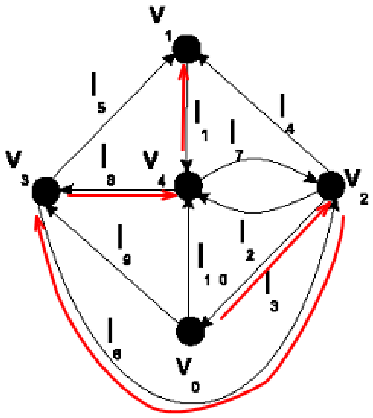


Fig. :in-arborescence, es una out-arborescence de D^t, los ejes de D usados se dirigen a la raíz.

IN- PROCEDURE:
Ejes de T size:4 (0, 2)w:1 Seg:0 (2, 3)w:1 Seg:1 (3, 4)w:1 Seg:2 (4, 1)w:1 Seg:3

Vertices de D (v0 w-: 1 w+: 2) (v1 w-: 2 w+: 1) (v2 w-: 3 w+: 3) (v3 w-: 2 w+: 2) (v4 w-: 3 w+: 3)

Orden de los Vértices hoja-raiz: $v_1|v_4|v_3|v_2|v_0$

[i: 4 (v1→u4) | $e_{1D} == e_{4T}$ |$v(w^-)$:2 > $v(w^+)$:1]

Set $u(w^-)$:4 = $u(w^+)$:3 + $v(w^-)$:2 - $v(w^+)$:1 = 4
Set $w(e_1)$:1= 2
Set vW+1 = $v(w^-)$:2 = 2

[i: 3 (v_4→u_3) | $e_{8D} == e_{3T}$ | $v(w^-)$:4 >$v(w^+)$:3]

Set $u(w^-)$:3 = $u(w^+)$:2 + $v(w^-)$:4 - $v(w^+)$:3 = 3
Set $w(e_8)$:1= 2
Set $v(w^+)$:4 = $v(w^-)$:4 = 4

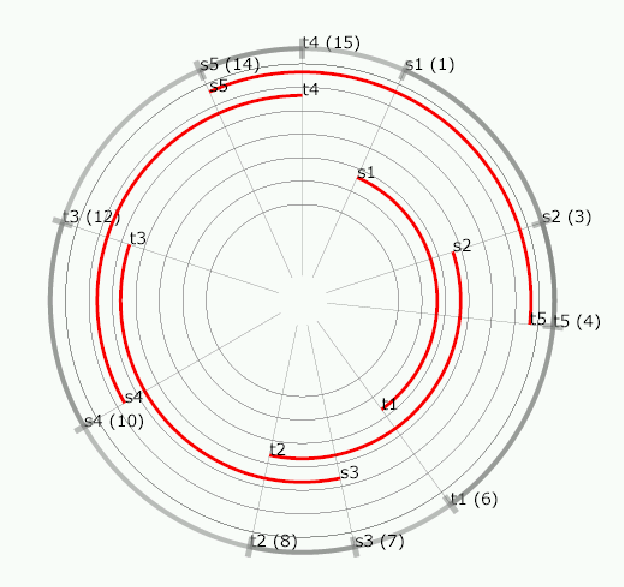
[i: 2 (v3→u2) | $e_{6D} == e_{2T}$ | $v(w^-)$:3) > $v(w^+)$:2]

Set $u(w^-)$:2 = $u(w^+)$:3 + $v(w^-)$:3 - $v(w^+)$:2 = 4
Set $w(e_6)$:1= 2
Set $v(w^+)$:3 = $v(w^-)$:3) = 3

[i: 1 (v2→u0) | $e_{3D} == e_{1T}$ | $v(w^-)$:4 > $v(w^+)$:3]

Set $u(w^-)$:0 = $u(w^+)$:2 + $v(w^-)$:4 - $v(w^+)$:3 = 3
Set $w(e_3)$:1= 2
Set $v(w^+)$:2 = $v(w^-)$:4 = 4

Vertices de D: **(v_0 w-: 2 w+: 2) (v_1 w-: 2 w+: 2) (v_2 w-: 4 w+: 4) (v_3 w-: 3 w+: 3) (v_4 w-: 4 w+: 4)**



Modelo UCA en el digrafo:
Vértices del digrafo de segmento (v0 w-: 2 w+: 2) (v1 w-: 2 w+: 2) (v2 w-: 4 w+: 4) (v3 w-: 3 w+: 3) (v4 w-: 4 w+: 4)

mapeo ejes del digrafo de segmento/ segmentos del modelo UCA (1, 4)W:2 Seg:1 (2, 4)W:1 Seg:2 (2, 0)W:2 Seg:3 (2, 1)W:1 Seg:4 (3, 1)W:1 Seg:5 (3, 2)W:2 Seg:6 (4, 2)W:2 Seg:7 (4, 3)W:2 Seg:8 (0, 3)W:1 Seg:9 (0, 4)W:1 Seg:10

Segmentos del modelo UCA :<label,start,lenght>:
<1|1|2> <2|3|1> <3|4|2> <4|6|1> <5|7|1> <6|8|2> <7|10|2> <8|12|2> <9|14|1> <10|15|1>

Longitud del Circulo C: 15

Ordenamiento de los extremos del modelo UCA:
s1 (1) s2 (3) t5 (4) t1 (6) s3 (7) t2 (8) s4 (10) t3 (12) s5 (14) t4 (15)

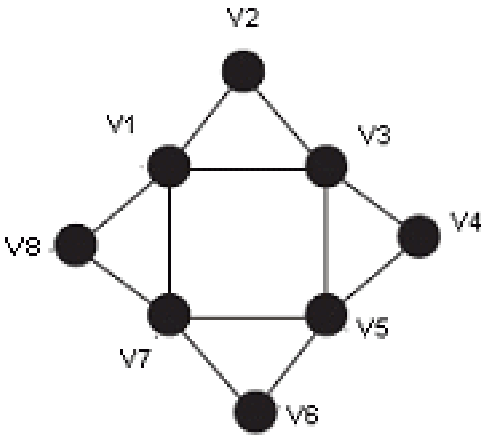
verificación: TODOS LOS ARCOS TIENEN LA MISMA LONGITUD

verificación: AMBOS MODELOS SON EQUIVALENTES

grado máximo de un vértice: 3 longitud de los arcos: 4

Fig. aII.2.4 Out e in procedures completos y construccion del modelo UCA final

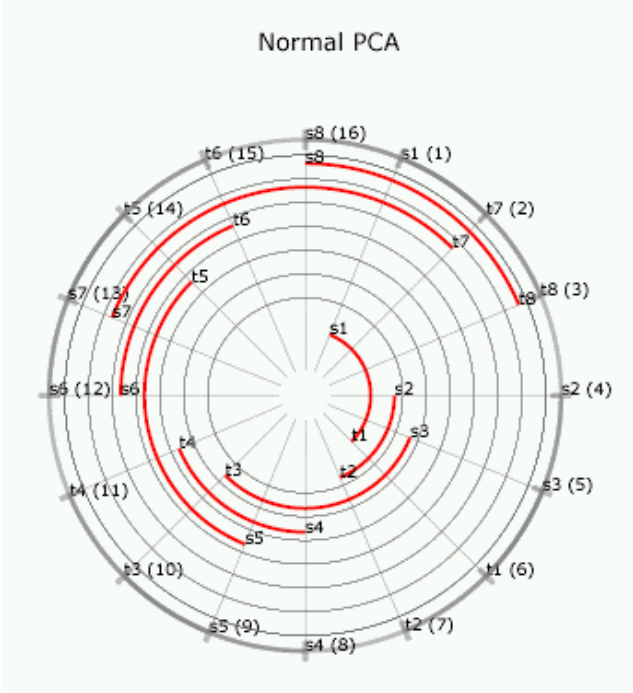
Corrida de un ejemplo donde el grafo no es arco circular unitario



Grafo CI(4,1)

E8 grafo 8
1,1.0,s
7,2.0,t
8,3.0,t
2,4.0,s
3,5.0,s
1,6.0,t
2,7.0,t
4,8.0,s
5,9.0,s
3,10.0,t
4,11.0,t
6,12.0,s
7,13.0,s
5,14.0,t
6,15.0,t
8,16.0,s

Fig. aII.3.1 grafo prohibido CI(4 ,1) y secuencia de extremos del input



Modelo PCA (es normal)

Mapea el vértice v_i de G con A_i, para i de 1 a n

Arcos:
A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8

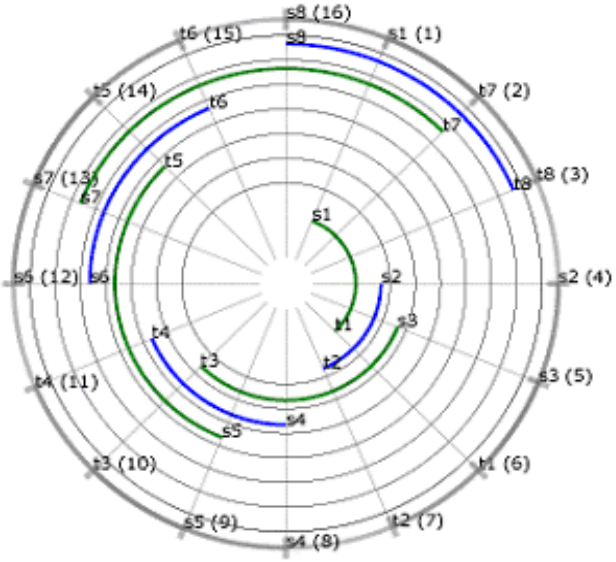
Ordenamiento de los extremos del modelo PCA
<extremo(valor)> :
$s_1(1)$, $t_7(2)$, $t_8(3)$, $s_2(4)$, $s_3(5)$, $t_1(6)$, $t_2(7)$, $s_4(8)$, $s_5(9)$, $t_3(10)$, $t_4(11)$, $s_6(12)$, $s_7(13)$, $t_5(14)$, $t_6(15)$, $s_8(16)$

Normalización : Input: modelo PCA

no hay 2 arcos que cubran el círculo C

Repeticiones del bloque interno: 10

Fig. aII.3.2: Modelo PCA del grafo de la Fig. aII 3.1, el modelo es normal, sin embargo para determinarlo, se debe ejecutar el mismo procedimiento de normalización.



Input: modelo normal

Arcos del modelo y su rango de segmentos:
A_1[1,6], A_2 [4,7], A_3 [5,10], A_4 [8,11], A_5 [9,14], A_6 [12,15], A_7[13,2], A_8[16,3]

Ecuaciones del sistema reducido <label,[lb,le],[rb,re]>:
q_1([1|3],[6|6]), q_2([4|4],[7|9]), q_3([5|7],[10|10]), q_4([8|8],[11|13]), q_5([9|11],[14|14]), q_6([12|12],[15|1]), q_7([13|15],[2|2])

Vértices Digrafo de Segmento/pesos (size:8):
v_0(w^-: 3 w^+: 1), v_1(w^-: 1 w^+: 3), v_2(w^-: 3 w^+: 1), v_3(w^-: 1 w^-: 3), v_4(w^-: 3 w^+: 1), v_5(w^-: 1 w^+: 3), v_6(w^-: 3 w^+: 1), v_7(w^-: 1 w^+: 3)

Ejes Digrafo de Segmento/peso/segmento (size:16):
(1, 6) w:1 Seg:1, (1, 7)w:1 Seg:2, (1, 0)w:1 Seg:3, (2, 0)w:1 Seg:4, (3, 0)w:1 Seg:5, (3, 1)w:1 Seg:6, (3, 2)w:1 Seg:7, (4, 2)w:1 Seg:8, (5, 2)w:1 Seg:9, (5, 3)w:1 Seg:10, (5, 4)w:1 Seg:11, (6, 4)w:1 Seg:12, (7, 4)w:1 Seg:13, (7, 5)w:1 Seg:14, (7, 6)w:1 Seg:15, (0, 6)w:1 Seg:16

Correr depht first forest(D), depht first forest(Dt) y SCC.

Determinar componentes fuente (1) o sumidero (2)
<0 |w^+: 0 w^-: 8| fos:2 startingNode: 0>
<1 |w^+: 8 w^-: 0| fos:1 startingNode: 1>
componente:1 (no contiene a 0)
fos:1 (FUENTE)

Descripción Mínima:
[1,1] [3,3] [5,5] [7,7]

Selected Arcs :
[1 3 5 7 2 4 6 8]

next_1: 2 → t = 1

Modelo NO UCA:
(k,t)Circuit: t=1 k=4 [1 3 5 7]
(k,t)Independent Set: t=1 k=4 [2 4 6 8]

verificación:
-EL MODELO ES PCA
- (n,k)circuit es un circuito
- (m,l)independent set es un conjunto independiente
- n/k IGUAL a m/l

Fig. aII.3.3 Corrida de los procedimientos para construir un certificado positivo para el grafo de la Fig. aII.3.1