Algoritmos eficientes para calcular el clique transversal mínimo en grafos

Tesis de Licenciatura

Javier Ignacio Arregui

jarregui@dc.uba.ar

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Director: Dr. Min Chih Lin

Diciembre de 2011

Dedicatoria
A mis padres, Raúl y Titi.

Agradecimientos
A Oscar, quien hizo posible esta tesis. A Ibis que siempre confió en mí. Y a todos los que me acompañaron y acompañan a seguir creciendo. Especialmente a Gabriela.

Resumen

Un clique transversal de un grafo G es un subconjunto de vértices que se intersecan con todos los cliques del grafo G. Es un problema NP-Hard determinar la cardinalidad ($\tau_c(G)$) del clique transversal de tamaño mínimo para un grafo G. En este trabajo proponemos algoritmos para generar un clique transversal de tamaño mínimo y por consiguiente determinar el valor de $\tau_c(G)$, mejorando algunos resultados de [15].

Los principales algoritmos con los que contribuimos en esta tesis son:

1) Un algoritmo general de tiempo $O(n^{\tau_c(G)-1}m^{\frac{\tau_c(G)}{2}})$ que genera un clique transversal de tamaño mínimo para cualquier grafo G. Este algoritmo es una modificación del algoritmo propuesto en [15] de tiempo $O(n^{2\tau_c(G)})$.

2) Un algoritmo robusto de tiempo $O(n+m)$ que genera un clique transversal de tamaño mínimo para cualquier grafo arco-circular sin $\overline{3K_2}$ como subgrafo inducido. Este algoritmo primero genera un modelo arco-circular a partir del grafo (cuesta tiempo $O(n+m)$) y luego lo utiliza para generar un clique transversal mínimo o bien detectar la existencia de un $\overline{3K_2}$ como subgrafo inducido del grafo (todo esto cuesta tiempo $O(n)$). Por lo tanto, si la entrada del algoritmo ya es un modelo arco-circular del grafo, la complejidad total se reduce a $O(n)$. El mejor algoritmo para esta clase de grafos hasta este momento era uno de tiempo $O(n^4)$ propuesto en [15]. Al igual que este algoritmo, nuestro algoritmo también es robusto en el sentido que si el grafo de entrada contiene algún $\overline{3K_2}$ como subgrafo inducido, o bien el algoritmo genera un clique transversal de tamaño mínimo del grafo o encuentra un subgrado inducido $\overline{3K_2}$ en el grafo.

3) Un algoritmo de reconocimiento de tiempo $O(nm^{\frac{3}{2}})$ para los grafos arco-circulares sin $\overline{3K_2}$.

4) Un algoritmo de tiempo $O(\tau_c(G)nm^{\frac{\tau_c(G)}{2}-1}(\tau_c(G)+\sqrt{m}))$ que genera un clique transversal de tamaño mínimo para cualquier grafo dualmente cordal G. Este algoritmo emplea la técnica llamada ≪simplicial augmentation≫ que fue introducida en [15] para grafos arco-circulares Helly.

Índice general

Capítulo 1

Introducción

Los clique transversales han sido estudiados desde el paper de Tuza (1990)[28]. También fueron incluídos en el paper de Payan (1979)[27]. Fue el trabajo de Erdös (1992)[17] que probó que el problema de clique transversal mínimo es NP-hard. Este problema sigue siendo NP-hard aún restringido a cualquiera de las siguientes subclases de grafos:

- grafos split
- grafos totales
- grafos de co-comparabilidad
- grafos de línea
- grafos planares
- grafos de caminos no orientados
- k-árboles con k no acotada

Por otro lado, para las siguientes subclases este problema admite una solución polinomial:

- grafos fuertemente cordales (Chang et al., 1993, 1996; Guruswami and Pandu Rangan 2000)[10, 8, 18]
- grafos doblemente cordales (Brandstädt et al. 1997)[5]
- grafos cordales con tamaño acotado de cliques (Guruswami and Pandu Rangan 2000)[18]
- k-árboles con k acotada (Chang et al. 1996)[10]
- grafos de comparabilidad (Balachandranet al. 1996)[2]
- grafos balanceados (Bonomo et al. 2006; Dahlhaus et al. 1998)[12, 3]

- grafos de distancia hereditaria (Lee et al. 2002)[23]
- grafos con cuerdas cortas sin 3-fans ni 4-wheels (Durán et al. 2002)[13]
- grafos arco-circulares Helly (Guruswami and Pandu Rangan 2000, Durán et al. 2008)[18, 15].
- grafos arco-circulares sin $\overline{3K_2}$ (Durán et al. 2008)[15].

Explicaremos brevemente la estructura de este trabajo de tesis: el resto del capítulo 1 consiste de una sección de definiciones y notaciones generales y otra sección que explica cómo transformar el problema de clique transversal al problema de conjunto dominante; en el capítulo 2, en primer lugar describiremos el algoritmo existente para determinar un clique transversal mínimo dado un grafo cualquiera para después mostrar de qué manera se puede mejorar; en el capítulo 3, utilizaremos nuestro conocimiento de modelos arco-circulares para diseñar un algoritmo robusto de tiempo lineal que resuelve el problema de clique transversal mínimo cuando el grafo en cuestión es un grafo arco-circular sin $\overline{3K_2}$, y a su vez como valor agregado, presentaremos un algoritmo de reconocimiento eficiente para este tipo de grafos; en el capítulo 4, ponemos nuestra atención en los grafos dualmente cordales, donde exhibimos cómo atacar el problema recurriendo a la técnica contenida en la última sección del capítulo 1, luego realizamos una variante que permite no examinar todos los cliques del grafo; finalmente, en el último capítulo resumimos los resultados de este trabajo y mostramos los problemas abiertos.

1.1. Definiciones, notación y algunas subclases de grafos de interés

Sea $G = (V(G), E(G))$ un grafo no dirigido donde $V(G)$ y $E(G)$ son los conjuntos de vértices y aristas de G respectivamente, $|V(G)| = n$ y $|E(G)| = m$. Un grafo $H = (V(H), E(H))$ es **subgrafo** de G si $V(H) \subseteq V(G)$ y $E(H) \subseteq E(G) \cap (V(H) * V(H))$. H es **subgrafo inducido** de G si además se cumple $E(H) = E(G) \cap (V(H) * V(H))$, en este caso también se dice que H es el subgrafo inducido por el subconjunto de vértices $V(H)$ y se denota como $G[V(H)]$. Es fácil ver que H es exactamente el grafo resultante después de quitar a G los vértices que están en $V(G) \setminus V(H)$. Por eso también se denota $G[V(H)]$ como $G \setminus (V(G) \setminus V(H))$.

Para $v \in V(G)$, denotamos por $N(v)$ al conjunto de vecinos de v. Este concepto también es conocido como la **vecindad abierta** de v, y la **vecindad cerrada** de v es $N[v] = N(v) \cup \{v\}$. Definimos la **no vecindad** de v como $\overline{N[v]} = V(G) \setminus N[v]$. La **vecindad común** de un subconjunto de vértices $W \subseteq V(G)$ es $N[W] = \bigcap_{v \in W} N[v]$.

Decimos que v es un **vértice universal** cuando $N[v] = V(G)$. Un **completo** de G es un subconjunto de vértices que son todos adyacentes entre sí. Un **clique** es un completo maximal. Un **conjunto dominante** de G es un subconjunto $W \subseteq V(G)$ donde cada vértice que no está incluído en W es adyacente a algún vértice de W, es decir, $\bigcup_{v \in W} N[v] = V(G)$
.

Un vértice v se llama **simplicial** si $N[v]$ es un clique.

Un vértice u es **vecino máximo** de un vértice v si $N[w] \subseteq N[u]$ para todo $w \in N[v]$ (u y v pueden ser un mismo vértice).

Sea $\mathcal{V}$ una familia de subconjuntos de $V(G)$, y $W \subseteq V(G)$. Decimos que W es **transversal** a $\mathcal{V}$, cuando W tiene interseccíon no vacía con cada subconjunto de $\mathcal{V}$. En particular, si $\mathcal{V}$ es la familia de los cliques de G entonces W es un **clique transversal** de G.

Emplearemos las siguientes notaciones:

- $\tau_c(G)$, la cardinalidad de un clique transversal de menor tamaño de G que lo llamamos como el número de clique transversal de G.
- $\gamma(G)$, la cardinalidad de un conjunto dominante de menor tamaño de G que lo llamamos como el número de conjunto dominante de G.

Cabe destacar que $\tau_c(G) \geq \gamma(G)$ porque todo clique transversal de G es conjunto dominante de G.

Una familia $\mathcal{F}$ de subconjuntos verifica la **propiedad de Helly** si toda subfamilia intersecante $\mathcal{F}' \subseteq \mathcal{F}$, es decir que sus miembros tienen intersección no vacía de par en par, tiene intersección no vacía.

Dado un **modelo arco-circular** (CA) $\mathcal{M} = (C, \mathcal{A})$ donde C es un círculo y $\mathcal{A}$ es una colección de arcos $A_1, \ldots, A_n$ alrededor de C. El grafo asociado $G(\mathcal{M})$ a este modelo (también se dice que $G(\mathcal{M})$ admite el modelo $\mathcal{M}$) es el grafo de intersección de los arcos de $\mathcal{A}$, es decir, los arcos de $\mathcal{A}$ corresponden uno a uno con los vértices de $G(\mathcal{M})$ y dos vértices de $G(\mathcal{M})$ son adyacentes si y sólo si sus arcos correspondientes en $\mathcal{A}$ tienen intersección no vacía. Un grafo G se llama **grafo arco-circular** (CA) si admite al menos un modelo arco-circular.

Dado un modelo arco-circular $\mathcal{M} = (C, \mathcal{A})$, si $\mathcal{A}$ satisface la propiedad de Helly, entonces $\mathcal{M}$ es un modelo **arco-circular Helly** (HCA). Ahora, un grafo que admite un modelo de este tipo es llamado **arco-circular Helly** (HCA).

Un grafo arco-circular $\overline{3K_2}$-free es un grafo arco-circular que no contiene al grafo de la Figura 1.1. como subgrafo inducido. No es díficil ver que ningún grafo arco-circular Helly puede contener $\overline{3K_2}$.

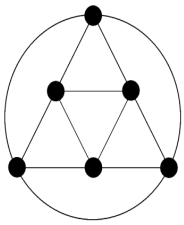

Figura 1.1: Grafo $\overline{3K_2}$.

Un ordenamiento $v_1, \ldots, v_n$ de los vértices de $V(G)$ es un orden de **eliminación perfecta** si para todos los $i \in \{1, \ldots, n\}$ el vértice v_i es simplicial del subgrafo inducido $G_i = G \setminus \{v_1, \ldots, v_{i-1}\}$. Los grafos que tienen un orden de eliminación perfecta son exactamente los grafos cordales (también conocidos como los grafos triangulados).

Existen otros tipos de ordenamientos, nos interesa en particular el siguiente.

Un ordenamiento $v_1, \ldots, v_n$ de los vértices de $V(G)$ es un orden de **vecindades máximas** si el vértice v_i posee algun vecino máximo u_i en el subgrafo inducido $G_i = G \setminus \{v_1, \ldots, v_{i-1}\}$. Los grafos que tienen un orden de vecindades máximas son los grafos dualmente cordales.

Sea $G = (V, E)$ un grafo y $r : V \to Z_{\geq 0}$ una función. Un subconjunto $D \subseteq V$ es un **conjunto r-Dominante** si para cada vértice $v \in V$, existe un vértice $u \in D$ donde $d(u, v) \leq r(v)$. El problema de la r-Dominancia es encontrar un conjunto r-Dominante de cardinal mínimo.

Sea $G = (V, E)$ un grafo, $\mathcal{F} = \{C_1, \ldots, C_k\}$ una familia de completos y $r : \mathcal{F} \to Z_{\geq 0}$ una función. Un subconjunto $D \subseteq V$ es un **conjunto clique r-Dominante** si para cada completo $C \in \mathcal{F}$, existe un vértice $u \in D$ tal que para todo vértice $w \in C$ tal que $d(u, w) \leq r(C)$. El problema de la Clique r-Dominancia es encontrar un conjunto clique r-Dominante de cardinal mínimo.

Vamos a utilizar frecuente el término **certificado** a lo largo de esta tesis. En la teoría de NP, para probar que un problema de decisión Π sea NP, debemos encontrar para las instancias de SI, un certificado que se puede chequear en tiempo polinomial. Este tipo de certificados (ciertas estructuras

que contienen información del dominio del problema en cuestión) se llama certificados positivos ya que permite autenticar las respuestas afirmativas del problema. Similarmente, para probar que Π sea co-NP, hay que hallar un certificado para las instancias de NO que se puede verificar en tiempo polinomial y este tipo de certificados son certificados negativos. Por ejemplo, si consideramos el problema de determinar si un grafo es o no bipartito. Un certificado positivo para este problema sería la bipartición de los vertices del grafo y un certificado negativo podría ser un ciclo de longitud impar del grafo.

Hay términos particulares que, dada su amplia difusión en el ambiente, se dejaron en inglés, con el objetivo de lograr un mayor entendimiento del texto.

1.2. Clique Transversal, Conjunto Dominante y Simplicial Augmentation

En [15] se resolvió eficientemente el problema de clique transversal mínimo para los grafos HCA realizando una transformación llamada ≪simplicial augmentation≫ al problema de conjunto dominante mínimo. Describiremos a continuación esta transformación: dado un grafo G, el ≪simplicial augmentation≫ $\mathcal{S}(G)$ es el grafo resultante de agregar un vértice nuevo w_j por cada clique C_j del grafo G y este vértice w_j es adyacente a cada vértice $v_i \in C_j$. En [15] se muestra que si el grafo G es HCA entonces $\mathcal{S}(G)$ también lo es y es más, se puede obtener en tiempo $O(n)$ un modelo HCA de $\mathcal{S}(G)$ a partir un modelo HCA de G. A su vez, se probó el siguiente teorema en el mismo trabajo:

Teorema 1.1 ([15])**.** *Sea $G = (V(G), E(G))$ un grafo HCA y $U \subseteq V(G)$. Entonces U es un clique transversal de G si y solo si U es un conjunto dominante de $\mathcal{S}(G)$.*

En realidad, este resultado sigue siendo válido para cualquier grafo G aunque en este caso $\mathcal{S}(G)$ puede tener tamaño exponencial con respecto al grafo G en lugar de tamaño $O(n)$. Abajo es la extensión del teorema 1.1 para cualquier grafo.

Teorema 1.2. *Sea $G = (V(G), E(G))$ un grafo y $U \subseteq V(G)$. Entonces U es un clique transversal de G si y solo si U es un conjunto dominante de $\mathcal{S}(G)$.*

Demostración. En primer lugar, vamos a mostrar que dado un subconjunto $U \subseteq V(G)$ es clique transversal de G entonces U es conjunto dominante de $\mathcal{S}(G)$. Supongamos que U no es conjunto dominante de $\mathcal{S}(G)$ pero sabemos que U es conjunto dominante de G, por lo tanto, existe algún vértice w_j de $\mathcal{S}(G)$ que corresponde a un clique C_j de G que no es adyacente a ningún vértice de U. Entonces U no cubre C_j en G, lo cual es absurdo. Nos falta

probar solamente la vuelta. Ahora $U \subseteq V(G)$ es conjunto dominante de $\mathcal{S}(G)$ y suponemos que no es clique transversal de G, por lo cual existe un clique C_j de G que no está cubierto por U, entonces el vértice w_j no es adyacente a ningún vértice de U, lo cual también es absurdo. □

Se puede utilizar este resultado para resolver en tiempo polinomial el problema de clique transversal mínimo para un grafo G tal que la cantidad de cliques de G es polinomial y $\mathcal{S}(G)$ pertenece a una subclase de grafos donde existe algoritmo de tiempo polinomial para el problema de conjunto dominante mínimo para esa subclase. Por ejemplo se podría aplicar esta idea para las siguientes subclases:

- Los grafos (P_5, $K_4 \setminus \{e\}$, C_4, C_5)-Free. La cantidad de cliques es $O(m)$ porque no contiene $K_4 \setminus \{e\}$. Dado un grafo G de esta subclase, $\mathcal{S}(G)$ es (P_5, $K_4 \setminus \{e\}$)-Free (no contiene a P_5 porque G no contiene a P_5, C_4 ni C_5; no contiene a $K_4 \setminus \{e\}$ porque G tampoco lo contiene) y para esta subclase existe algoritmo de tiempo lineal [4] para resolver el problema de conjunto dominante mínimo.

- Los grafos (P_5, K_q, C_4, C_5)-Free con q fija. La cantidad de cliques es $O(n^{q-1})$ porque no contiene K_q. Dado un grafo G de esta subclase, $\mathcal{S}(G)$ es (P_5, K_{q+1})-Free (no contiene a P_5 porque G no contiene a P_5, C_4 ni C_5; no contiene a K_{q+1} porque G no contiene a K_q) y para esta subclase existe algoritmo de tiempo polinomial [29] para resolver el problema de conjunto dominante mínimo.

Capítulo 2

Grafos generales

En 2008, Durán, Lin, Mera y Szwarcfiter [15] presentaron un algoritmo que resuelve el problema de clique transversal mínimo para cualquier grafo G. La complejidad temporal de este algoritmo es $O(n^{2\tau_c(G)})$. Cabe destacar que si podemos considerar a $\tau_c(G)$ como un valor constante, entonces este algoritmo es de tiempo polinomial.

A continuación, enunciamos el teorema que caracteriza un clique transversal y es la piedra fundamental para construir el algoritmo mencionado y para las mejoras que elabora esta tesis.

Teorema 2.1 ([15]). *Sea $G = (V, E)$ un grafo y $W = \{v_1, \ldots, v_k\} \subseteq V$. W no es clique transversal de G si y sólo si existe un completo $C \subseteq V$ con a lo sumo k vértices que satisface $C \cap \overline{N[v_i]} \neq \emptyset$, para $1 \leq i \leq k$.*

Es importante señalar que el completo C de Teorema 2.1 sirve como un certificado negativo de que el subconjunto W no es clique transversal de G.

Claramente, como consecuencia directa del teorema anterior, tenemos el siguiente algoritmo.

Algoritmo 2.1 ([15]). *Dado un grafo $G = (V, E)$ y un subconjunto $W = \{v_1, \ldots, v_k\} \subseteq V$, determina si W es o no clique transversal de G. En el caso negativo, genera un completo C como certificado.*

La implementación de Algoritmo 2.1 dada en [15] es el siguiente: generar un conjunto C eligiendo un vértice de cada $\overline{N[v_i]}$, $1 \leq i \leq k$, y luego verificar si C es un completo; en el caso que falla, volver a intentar con otro conjunto C eligiendo otros vértices; se repite este procedimiento hasta agotar todas las posibles combinaciones para componer C. La complejidad de esta implementación es $O(n^k k^2)$ ya que la cantidad de posibles C es $O(n^k)$ y cada verificación de C es completo o no cuesta $O(k^2)$.

Usando el Algoritmo 2.1 se puede determinar si $\tau_c(G) \leq k$ $(k \geq 2)$ dado un grafo G y un número entero k con simplemente generar todos los subconjuntos W de k vértices y verificar si alguno de ellos es clique transversal de G. Cabe destacar que para verificar si $\tau_c(G) = 1$ $(\tau_c(G) \leq 1)$ es lo mismo que

preguntar si el grafo G tiene vértice universal y eso cuesta $O(m+n)$ ($O(m)$ si G es conexo).

Algoritmo 2.2 ([15]). *Dado un grafo $G = (V, E)$ y un número entero k, determina si existe un clique transversal de G con a lo sumo k vértices.*

La complejidad de Algoritmo 2.2 es $\binom{n}{k}$ veces la complejidad de Algoritmo 2.1, es decir, $O(\binom{n}{k} n^k k^2)$

Finalmente, podemos determinar un clique transversal mínimo de un grafo G, invocando iterativamente al Algoritmo 2.2 e incrementando el valor de k hasta encontrar un clique transversal.

Algoritmo 2.3 ([15]). *Dado un grafo $G = (V, E)$, determina un clique transversal mínimo de G.*

En [15], se probó que la complejidad de Algoritmo 2.3 es $O(n^{2\tau_c(G)})$ que es la sumatoria de los costos asociados a las $O(\tau_c(G))$ invocaciones de Algoritmo 2.2.

En las siguientes secciones de este capítulo, introducimos distintas modificaciones que mejoran las complejidades de los algoritmos presentados en [15].

Para fines algorítmicos, podemos suponer que los grafos a tratar tienen al menos 4 aristas, pués en el caso contrario, se puede encontrar muy fácilmente un clique transversal mínimo en tiempo $O(n)$.

2.1. Búsqueda de certificado negativo

Nosotros proponemos una nueva implementación de Algoritmo 2.1 que consiste en generar todos los completos C de hasta k vértices y luego verificar si alguno de ellos es transversal a $\{\overline{N[v_i]}\}_{v_i \in W}$. En el caso positivo, encontramos el certificado negativo para que W no sea clique transversal de G. Caso contrario, W es clique transversal de G.

Para generar los completos de hasta k vértices, invocamos el algoritmo de Chiba y Nishizeki [11] que permite listar todos los completos de un cierto tamaño l, para $2 \leq l \leq k$. El costo de cada invocación es $O(lm^{\frac{l}{2}})$, por lo tanto el costo total sería $O(n + \sum_{l=2}^{k} lm^{\frac{l}{2}})$. Se puede acotar fácilmente esta complejidad por $O(k^2 m^{\frac{k}{2}} + n)$. Por otro lado, la cantidad de completos generados de cada iteración está acotada por $O(m^{\frac{l}{2}})$. A continuación vamos a probar el siguiente lema que afirma que la cantidad de completos de hasta k vértices es $O(m^{\frac{k}{2}} + n)$.

Lema 2.1. *La cantidad de completos de tamaño hasta k de un grafo con n vértices es $O(m^{\frac{k}{2}} + n)$.*

Demostración. Vamos a separar en tres casos:

$m \leq 3$) en este caso la cantidad total de completos de G es a lo sumo n.

$k = 1$) la cantidad de completos de tamaño 1 es n.

$m \geq 4$ y $k \geq 2$) primero, probamos por inducción que $\sum_{i=2}^{k} m^{\frac{i}{2}} \leq 2m^{\frac{k}{2}}$. El caso base es $k = 2$ que vale trivialmente. Ahora suponemos que es cierto para $k = k_0$ ($k_0 \geq 2$) y queremos probarlo para $k = k_0 + 1$. Tenemos $\sum_{i=2}^{k_0+1} m^{\frac{i}{2}} \leq m^{\frac{k_0+1}{2}} + \sum_{i=2}^{k_0} m^{\frac{i}{2}} \leq$ (por hipótesis inductiva) $m^{\frac{k_0+1}{2}} + 2m^{\frac{k_0}{2}} \leq m^{\frac{k_0+1}{2}} + m^{\frac{k_0+1}{2}} = 2m^{\frac{k_0+1}{2}}$, lo que queremos probar. Entonces la cantidad de cliques de tamaño entre 2 y k es $O(m^{\frac{k}{2}})$ y agregando los n completos de tamaño 1 son en total $O(m^{\frac{k}{2}} + n)$.

□

Para completar el análisis de complejidad, falta sumar el costo de las verificaciones de la transversabilidad de los completos que sería $O((m^{\frac{k}{2}} + n)k^2))$. En consecuencia la nueva complejidad del Algoritmo 2.1 es $O((m^{\frac{k}{2}} + n)k^2))$. Cabe aclarar que para grafos conexos y $k \geq 2$, se podría simplificar como $O(m^{\frac{k}{2}}k^2)$.

Con esta nueva implementación, se reducen automáticamente las complejidades de los Algoritmos 2.2 y 2.3 a $O(\binom{n}{k} m^{\frac{k}{2}}k^2)$ y $O(n^{\tau_c(G)}m^{\frac{\tau_c(G)}{2}})$ (la prueba es similar a la del teorema 2.4), respectivamente.

2.2. Rescatando a W

Hasta ahora hemos visto que el algoritmo más importante es el Algoritmo 2.1 (en el cual se basan los demás). Dicho algoritmo permite dado un subconjunto W de k vértices, determinar si W es o no un clique transversal del grafo, tratando de localizar un certificado negativo. De alguna manera, este certificado evidencia la existencia de cliques que no están cubiertos por W. Ahora bien, podríamos aprovechar estos certificados para ver si es posible ≪arreglar≫ W adicionándole otro vértice. Esta es la idea principal de la mejora que vamos a proponer en esta sección. El siguiente teorema permite inferir la existencia o no de estos vértices.

Teorema 2.2. *Sea $G = (V, E)$ un grafo y $W = \{v_1, \ldots, v_k\} \subseteq V$. W es clique transversal de G si y sólo si $W \setminus \{v_k\}$ ya es clique transversal de G o v_k es un vértice universal del subgrafo inducido G' por $\bigcup_{i=1}^{p} N[C_i]$, donde $C_1, \ldots, C_p$ son los certificados negativos de que $W \setminus \{v_k\}$ no es clique transversal de G.*

Demostración. Probaremos la ida por el absurdo. Sea $W = \{v_1, \ldots, v_k\} \subseteq V$ un clique transversal de G, supongamos que $W \setminus \{v_k\}$ no es clique transversal

de G y v_k no es vértice universal del subgrafo G'. Entonces en G', v_k no es vecino de algún vértice $u \in N[C_i]$ donde C_i es un certificado negativo de que $W \setminus \{v_k\}$ no es clique transversal de G. Sea $C'_i = C_i \cup \{u\}$, claramente C'_i es un completo y $|C'_i| \leq k$. Veamos que C'_i es un certificado negativo para W. Necesitamos verificar que $C'_i \cap \overline{N[v_j]} \neq \emptyset$, para $1 \leq j \leq k$. Para $1 \leq j \leq k-1$, vale $C_i \cap \overline{N[v_j]} \neq \emptyset$ porque C_i es certificado negativo para $W \setminus \{v_k\}$, como $C_i \cap \overline{N[v_j]} \subseteq C'_i \cap \overline{N[v_j]}$ entonces $C'_i \cap \overline{N[v_j]} \neq \emptyset$. Falta solamente verificar que $C'_i \cap \overline{N[v_k]} \neq \emptyset$, pero sabemos que $u \in C'_i \cap \overline{N[v_k]}$, listo.
Ahora probamos la vuelta. Si $W \setminus \{v_k\}$ es clique transversal de G, claramente W también lo es. Ahora suponemos que $W \setminus \{v_k\}$ no es clique transversal de G pero v_k es vértice universal del subgrafo G', supongamos que W no es clique transversal de G. Por lo tanto existe un clique M que no está cubierto por W, es decir que $M \cap \overline{N[v_i]} \neq \emptyset$ para todo $v_i \in W$. En particular, podemos escoger un vértice por cada $M \cap \overline{N[v_i]}$, $1 \leq i \leq k-1$. Estos vértices seleccionados conforman un certificado negativo C_j de que $W \setminus \{v_k\}$ no es clique transversal de G. Como $M \subseteq N[C_j]$, por definición de G' y el hecho de que v_k es vértice universal en G', $M \subseteq N[v_k]$ y M debe contener a v_k ya que es un clique, por lo tanto v_k cubre a M, lo cual es una contradicción. □

Del teorema anterior, se deduce el siguiente algoritmo.

Algoritmo 2.4. *Dado un grafo $G = (V, E)$ y $W = \{v_1, \ldots, v_{k-1}\} \subseteq V$, devuelve alguno de los siguientes posibles resultados.*

a) W es clique transversal de G.

b) no es cierto a) y existe un vértice $v_k \in V \setminus W$ tal que $W \cup \{v_k\}$ es clique transversal de G. (En este caso devuelve este vértice v_k)

c) no son ciertos ni a) ni b).

La implementación del Algoritmo 2.4 es la siguiente: genera todos los completos C de hasta $|W| = k-1$ vértices (usando Chiba y Nishizeki [11]) y verifica cuáles de ellos son transversales a $\{\overline{N[v_i]}\}_{v_i \in W}$. Si ninguno cumple esta condición entonces W es un clique transversal de G. Caso contrario, busca un vértice universal en el subgrafo G' inducido por $\bigcup_{i=1}^{p} N[C_i]$, donde $C_1, \ldots, C_p$ son los certificados negativos de que W no es clique transversal de G. Si pudiera encontrar tal vértice universal v_k, $W \bigcup \{v_k\}$ sería un clique transversal de G.

La complejidad de este algoritmo es la suma de los siguientes costos asociados:

1. chequear si W es un clique transversal tratando de generar certificados negativos, esto cuesta $O((m^{\frac{k-1}{2}} + n)(k-1)^2)$ ($O(m^{\frac{k-1}{2}}(k-1)^2)$ para grafos conexos y $k \geq 3$). Por cada certificado negativo C_i generado, marcar los vértices que están en $N[C_i]$. Esto último incrementa el costo a $O((m^{\frac{k-1}{2}}+n)(k-1)n)$ ($O(m^{\frac{k-1}{2}}(k-1)n)$ para grafos conexos y $k \geq 3$).

Es importante notar que si $k = 2$, los certificados negativos son los conjuntos unitarios formados por cada no vecino de v_1. Cuesta $O(n)$ encontrarlos y $O(m)$ marcar los vértices que son vecinos de alguno de estos vértices. Por lo tanto, se reduce la complejidad a $O(n+m)$ ($O(m)$ para grafos conexos).

2. generar G' a partir de los vértices marcados y buscar el vértice universal v_k en G' cuesta $O(n+m)$ ($O(m)$ para grafos conexos).

Consecuentemente, el costo total del Algoritmo 2.4 es $O((m^{\frac{k-1}{2}}+n)(k-1)n)$ ($O(m^{\frac{k-1}{2}}(k-1)n)$ para grafos conexos y $k \geq 3$; $O(m)$ para grafos conexos y $k = 2$).

Ahora, se puede tener una nueva implementación para el Algoritmo 2.2 que es ejecutar el Algoritmo 2.4 para cada subcobjunto $W \subseteq V$ de $k-1$ vértices. De esta manera, la complejidad del Algoritmo 2.2 se reduce a $O(\binom{n}{k-1}(m^{\frac{k-1}{2}}+n)(k-1)n)$ ($O(\binom{n}{k-1}m^{\frac{k-1}{2}}(k-1)n)$ para grafos conexos y $k \geq 3$; $O(nm)$ para grafos conexos y $k = 2$).

Con esta implementación, la complejidad del Algoritmo 2.3 se reduce a $O(n^{\tau_c(G)}m^{\frac{\tau_c(G)-1}{2}})$* para grafos conexos y si además $\tau_c(G) = 2$, la complejidad sería $O(nm)$.

2.3. Reducir el número de candidatos W's

Hasta ahora en la implementación del Algoritmo 2.2 siempre se consideran todos los subconjuntos de un tamaño fijo como W. Cada uno de ellos es examinado para saber si es un clique transversal, o es extensible a un clique transversal. Entonces, si podemos disminuir la cantidad de posibles W's para esta verificación (obviamente sin afectar la correctitud), el algoritmo sería más eficiente. Esto es exactamente lo que vamos a proponer en esta sección.

Una técnica bastante utilizada ([1, 16, 22]) para mejorar eficiencias de algoritmos es considerar separadamente los vértices de alto (≪high≫) grado y bajo (≪low≫) grado y analizar si se puede obtener algun tipo de beneficio con esta separación. En nuestro caso, proponemos este tipo de separación para restringir la composición de un candidato W.

Dado un grafo G y un número entero $k \geq 1$, decimos que un vértice v es ≪high≫ si $d(v) > g(k) = \sqrt{\frac{2m}{k}} - 1$, caso contrario, v es ≪low≫. Denotamos

*La prueba es similar a la del teorema 2.4

los conjunto de vértices ≪high≫ y ≪low≫ como $H(k)$ y $L(k)$, respectivamente. A continuación, mostramos en el siguiente teorema una propiedad que debe verificar cualquier clique transversal.

Teorema 2.3. *Si W es un clique transversal, entonces $W \cap H(|W|) \neq \emptyset$.*

Demostración. Vamos a probar por el absurdo. Sea $|W| = k$ y supongamos que $W \cap H(k) = \emptyset$, es decir que $W \subseteq L(k)$. Ahora queremos contar la cantidad de aristas que tienen un extremo en W o tales que sus dos extremos tienen un vecino común en W y denotamos este número como $\beta(W)$. No es difícil de ver que $\beta(W) = \bigcup_{v \in W}\{(u,w) \in E(G) \ / \ u \neq w \wedge u, w \in N[v]\}$. Claramente, $\beta(W) \leq \sum_{v \in W} \frac{(d(v)+1)d(v)}{2} \leq \frac{k(g(k)+1)g(k)}{2} = \frac{k\sqrt{\frac{2m}{k}}(\sqrt{\frac{2m}{k}}-1)}{2} = m - \sqrt{\frac{mk}{2}} < m$ (suponiendo que $m > 0$). Es decir que hay al menos una arista e que no tiene ningún extremo que sea vértice de W o vecino de algún vértice de W. Esta arista e está contenida en algun clique C (puede ser más de uno) y $C \cap W = \emptyset$, lo cual es absurdo. □

Como consecuencia del teorema anterior, podemos solamente considerar los subconjuntos W's que tengan por lo menos un vértice ≪high≫. Entonces una nueva implementación del Algoritmo 2.2 es elegir el vértice v_1 siempre del conjunto $H(k)$. Ahora para cuantificar cúal es el ahorro obtenido necesitamos el siguiente lema.

Lema 2.2. $|H(k)| = O(\sqrt{mk})$.

Demostración. Sabemos que $d(v) > \sqrt{\frac{2m}{k}} - 1$ para cada vértice v de $H(k)$ y como $d(v)$ es un número entero entonces $d(v) \geq \left\lfloor \sqrt{\frac{2m}{k}} \right\rfloor$. Por lo tanto tenemos:
$|H(k)| \left\lfloor \sqrt{\frac{2m}{k}} \right\rfloor \leq \sum_{v_i \in H(k)} d(v_i) \leq \sum_{v_i \in V(G)} d(v_i) = 2m$
como $1 \leq k \leq n$, $|H(k)| \leq \frac{2m}{\left\lfloor \sqrt{\frac{2m}{k}} \right\rfloor} = O(\sqrt{mk})$ □

La nueva complejidad del Algoritmo 2.2 es $O(\binom{H(k)}{1}\binom{n}{k-2}(m^{\frac{k-1}{2}}+n)(k-1)n) = O(\sqrt{mk}\binom{n}{k-2}(m^{\frac{k-1}{2}}+n)(k-1)n)$ ($O(\binom{n}{k-2}m^{\frac{k}{2}}(k-1)\sqrt{k}n)$ para grafos conexos y $k \geq 3$; $O(m^{\frac{3}{2}})$ para grafos conexos y $k = 2$).

De acuerdo al párrafo anterior, la complejidad del Algoritmo 2.3 para grafos conexos se reduce de la siguiente manera: si $\tau_c(G) = 2$ es $O(m^{\frac{3}{2}})$ y si $\tau_c(G) \geq 3$ es $O(n^{\tau_c(G)-1}m^{\frac{\tau_c(G)}{2}})$, este último caso se fundamenta en el Teorema 2.4 que describiremos más adelante.

Para probar el Teorema 2.4, introducimos el siguiente lema:

Lema 2.3. $g'(i) = \frac{\sqrt{i}(i-1)}{(i-2)!} \leq 2\sqrt{3}$, *para todo número entero* $i \geq 3$.

Demostración. Es fácil de verificar los siguientes valores de $g'(i) \leq 2\sqrt{3} \approx 3{,}4641$:

$g'(3) = \frac{\sqrt{3}(3-1)}{(3-2)!} = 2\sqrt{3}$

$g'(4) = \frac{\sqrt{4}(4-1)}{(4-2)!} = \frac{2*3}{2} = 3$

$g'(5) = \frac{\sqrt{5}(5-1)}{(5-2)!} = \frac{4\sqrt{5}}{6} = \frac{2\sqrt{5}}{3} \approx 1{,}49$

$g'(6) = \frac{\sqrt{6}(6-1)}{(6-2)!} = \frac{5\sqrt{6}}{24} = \frac{5\sqrt{3}\sqrt{2}}{24} \approx 0{,}51$

$g'(7) = \frac{\sqrt{7}(7-1)}{(7-2)!} = \frac{6\sqrt{7}}{120} = \frac{\sqrt{7}}{20} \approx 0{,}1323$

A continuación vamos a probar que $g'(i) < g'(i-1)$ para cualquier número entero $i \geq 8$. Con lo cual la demostración del lema estaría completa.

$$g'(i) < g'(i-1) \Leftrightarrow \frac{\sqrt{i}(i-1)}{(i-2)!} < \frac{\sqrt{i-1}(i-2)}{(i-3)!} \Leftrightarrow \sqrt{i}(i-1) < \sqrt{i-1}(i-2)^2 \Leftrightarrow$$
$$i(i-1)^2 < (i-1)(i-2)^4 \Leftrightarrow i(i-1) < (i-2)^4 (\text{porque}$$
$$i-1 \neq 0) \Leftrightarrow i^2 - i < i^4 - 8i^3 + 24i^2 - 32i + 16 \Leftrightarrow 0 < i^4 - 8i^3 + 23i^2 - 31i + 16$$

Se verifican fácilmente que las siguientes desigualdades son válidas porque $i \geq 8$:

$$0 < 153i + 16 = 8i^3 - 8i^3 + 8 * 23i - 31i + 16 \leq i^4 - 8i^3 + 23i^2 - 31i + 16$$

□

Teorema 2.4. *Si $G = (V(G), E(G))$ es un grafo conexo ($n = |V(G)|$ y $m = |E(G)|$) con $\tau_c(G) \geq 3$, entonces $\sum_{k=3}^{\tau_c(G)} \binom{n}{k-2} m^{\frac{k}{2}}(k-1)\sqrt{k}n = O(n^{\tau_c(G)-1} m^{\frac{\tau_c(G)}{2}})$*

Demostración. $\sum_{k=3}^{\tau_c(G)} \binom{n}{k-2} m^{\frac{k}{2}}(k-1)\sqrt{k}n \leq \sum_{k=3}^{\tau_c(G)} \frac{n^{k-2}}{(k-2)!} m^{\frac{k}{2}}(k-1)\sqrt{k}n = \sum_{k=3}^{\tau_c(G)} \frac{\sqrt{k}(k-1)}{(k-2)!} m^{\frac{k}{2}} n^{k-1} = \sum_{k=3}^{\tau_c(G)} g'(k) m^{\frac{k}{2}} n^{k-1} \leq \sum_{k=3}^{\tau_c(G)} 2\sqrt{3} m^{\frac{k}{2}} n^{k-1}$ (por Lema 2.3)$= 2\sqrt{3} \sum_{k=3}^{\tau_c(G)} m^{\frac{k}{2}} n^{k-1} = 2\sqrt{3}\sqrt{m} \sum_{k=3}^{\tau_c(G)} (n\sqrt{m})^{k-1} \leq 2\sqrt{3}\sqrt{m}(2(n\sqrt{m})^{\tau_c(G)-1})$ (porque podemos asumir que $m \geq 4$, por lo tanto $n\sqrt{m} \geq 2$ y podemos usar la propiedad geométrica) $= O(n^{\tau_c(G)-1} m^{\frac{\tau_c(G)}{2}})$. □

Capítulo 3

Grafos arco-circulares $\overline{3K_2}$-free

Además del Algoritmo 2.3, en el trabajo [15] también se exhibe un algoritmo robusto (Algoritmo 3.1) que resuelve el problema de clique transversal mínimo para grafos arco-circulares $\overline{3K_2}$-free en tiempo $O(n^4)$.

A continuación, enunciaremos el teorema en el cual se basa dicho algoritmo y una breve descripción del mismo.

Teorema 3.1 ([15]). *Sea G es un grafo arco-circular que no tiene a $\overline{3K_2}$ como subgrafo inducido, y no es HCA. Entonces, $\tau_c(G) \leq 3$.*

Algoritmo 3.1 ([15]). *Dado un grafo arco-circular $\overline{3K_2}$-free G, determina un clique transversal mínimo de G. En el caso que el grafo G viole la condición de ser $\overline{3K_2}$-free, el algoritmo puede resolver el problema de todas maneras o mostrar la existencia de un subgrafo inducido $\overline{3K_2}$. Se detallan a continuación los pasos del algoritmo.*

1. *En el caso en que el grafo no viene con un modelo arco-circular dado, es necesario, generar uno por medio de alguno de los algoritmos conocidos para tal fin [21, 26]. Podemos asumir que el modelo dado o generado es $\mathcal{M} = (C, \mathcal{A})$.*

2. *Verificar si G posee algún vértice universal. En el caso positivo, cualquiera de estos vértices constituye un clique transversal de G y por lo tanto $\tau_c(G) = 1$ (termina aquí la ejecución del algoritmo). Cabe destacar que no fue utilizada la condición de $\overline{3K_2}$-free, así que es válido para cualquier grafo arco-circular.*

3. *Utilizar el Algoritmo 2.2 con $k = 2$, para ver si $\tau_c(G) \leq 2$. En el caso afirmativo, $\tau_c(G) = 2$ (termina aquí la ejecución del algoritmo). Aquí tampoco fue utilizada la condición de $\overline{3K_2}$-free, por lo tanto es válido también para cualquier grafo arco-circular.*

4. *Tratar de encontrar si hay tres arcos A_1, A_2 y $A_3 \in \mathcal{A}$ que cubran C. Si no existen tales arcos, $\mathcal{M}$ es un modelo HCA y se puede determinar*

$\tau_c(G)$ aplicando el algoritmo específico, también dado en [15], para encontrar un clique transversal mínimo para grafos arco-circulares Helly (termina aquí la ejecución del algoritmo).

5. *Los arcos A_1, A_2 y A_3 encontrados en el punto anterior forman un clique transversal de G siempre que no sean parte de un subgrafo inducido $\overline{3K_2}$ de G. A diferencia de los puntos 2 y 3, aquí sí prohibe la existencia de $\overline{3K_2}$'s aunque solamente aquellos que involucren a los arcos A_1, A_2 y A_3. Por lo tanto si el grafo G contiene algunos $\overline{3K_2}$'s pero ninguno de ellos contiene a A_1, A_2 y A_3, se puede obtener de todas maneras un clique transversal mínimo de G. Para completar este punto, hay que buscar si existen otros tres arcos A_4, A_5, $A_6 \in \mathcal{A}$ que conjuntamente con A_1, A_2 y A_3 formen un $\overline{3K_2}$ en G. En el caso positivo, informar que G contiene un $\overline{3K_2}$ compuesto por estos seis arcos. Caso contrario, $\tau_c(G) = 3$.*

Es claro que el algoritmo anterior es robusto en el sentido que el grafo G puede no cumplir la condición de $\overline{3K_2}$-free, sin embargo es posible determinar un clique transversal mínimo de G en los puntos 2, 3 y 5 o bien informar la existencia de uno de estos subgrafos prohibidos en el punto 5.

A continuación, se describe la complejidad de cada paso: el punto 1 que es opcional dependiendo de si el input viene con un modelo dado o no, cuesta $O(m+n)$; el punto 2 se puede hacer en $O(n)$ [19]; la complejidad del punto 3 es $O(n^4)$ [15] (mediante las mejoras propuestas en el capítulo anterior se reduce a $O(m^{\frac{3}{2}})$); existe un algoritmo de tiempo $O(n)$ [19] para hallar tres arcos que cubran C para el punto 4 y por lo tanto todo este punto se puede hacer en $O(n)$ (el problema de clique transversal en grafos arco circulares Helly se puede resolver en $O(n)$ [15] tomando un modelo HCA como input); el punto 5 se puede implementar fácilmente en $O(n^3)$.

Se puede observar que el paso 3 ($O(n^4)$) era el que determinaba la complejidad del Algoritmo 3.1 en [15]. Con las mejoras propuestas del capítulo anterior, el paso que determina la complejidad del algoritmo pasa a ser el punto 5 ($O(n^3)$). Claramente, si pudieramos mejorar las implementaciones de los puntos 3 y 5 podemos reducir aún más la complejidad del algoritmo 3.1.

En la próxima sección daremos unas definiciones y elementos auxiliares que usaremos a lo largo del capítulo. Luego en la subsiguiente sección, mostraremos una implementación $O(n)$ del punto 5 que a su vez permite construir un algoritmo de reconocimiento de tiempo $O(n^4)$ para los grafos arco-circulares sin $\overline{3K_2}$. Y en la sección final detallamos una implementación del punto 3 que explota la estructura de los modelos arco-circulares, logrando así una implementación del Algoritmo 3.1 en tiempo lineal.

3.1. Elementos auxiliares

Cuando recorremos el círculo C, si no aclaramos explícitamente, siempre lo hacemos en la dirección de las agujas del reloj, es decir, en el sentido horario. Sean s, t dos puntos en C, escribimos (s, t) para denotar al arco definido sobre C que va desde el punto s hasta el punto t. Llamamos s y t a los extremos de (s, t), donde s es el punto de partida (**start point**) y t es el punto de finalización del arco (**end point**). Para cada $A_i \in \mathcal{A}$, escribimos $A_i = (s_i, t_i)$. Sin pérdida de generalidad, en este capítulo vamos a suponer que todos los arcos de los modelos arco-circulares son abiertos, los extremos son todos distintos dentro de un mismo modelo y no puede haber un sólo arco que cubre todo el círculo.

Sea $\mathcal{M} = (C, \mathcal{A})$ un modelo arco-circular. Una s-sequence (t-sequence) de $\mathcal{M}$ es una secuencia maximal de start points (end points) consecutivos de $\mathcal{A}$ en el orden circular de C. Entonces una secuencia de extremos significa una s-sequence o una t-sequence. Los $2n$ puntos extremos estarán particionados en s-sequences y t-sequences, que se alternan en C. Para una secuencia de extremos E, la notación $NEXT(E)$ and $NEXT^{-1}(E)$ representa la secuencia de extremos que siguen o preceden a E en C, respectivamente. Para un extreme point $p \in A$, denotaremos por $SEQUENCE(p)$ la secuencia de extremos que contiene a p, y $NEXT(p)$ significa la secuencia $NEXT(SEQUENCE(p))$.

Sea s_i un start point de A y $S = SEQUENCE(s_i)$. Podemos decir que s_i es estable cuando $i = j$ o $A_i \bigcap A_j = \varnothing$, para todo $t_j \in NEXT^{-1}(S)$. Un modelo (C,A) es estable cuando todos sus start points son estables. Sea t_j un end point de A y $T = SEQUENCE(t_j)$. Podemos decir que t_j es estable cuando $i = j$ o $A_i \bigcap A_j = \varnothing$, para todo $s_i \in NEXT(T)$.

Podemos observar en el ejemplo de la Figura 3.1. Hay 6 arcos en este modelo donde el orden circular de sus 12 extremos es: s_1, t_5, s_2, s_3, t_2, t_3, s_4, s_5, t_4, s_6, t_1, t_6. Enumeramos primero las secuencias de extremos: $S_1 = s_1, T_1 = t_5, S_2 = s_2 s_3, T_2 = t_2 t_3, S_3 = s_4 s_5, T_3 = t_4, S_4 = s_6, T_4 = t_1 t_6$. Ahora mostramos los valores de las funciones $NEXT$, $NEXT^{-1}$: $NEXT(S_i) = T_i, NEXT^{-1}(T_i) = S_i, i = 1, \ldots, 4$; $NEXT(T_i) = S_{i+1}, NEXT^{-1}(S_{i+1}) = T_i, i = 1, \ldots, 3$; $NEXT(T_4) = S_1$ y $NEXT^{-1}(S_1) = T_4$. Finalmente, la función $SEQUENCE$: $SEQUENCE(s_1) = S_1$, $SEQUENCE(t_5) = T_1$, $SEQUENCE(s_2) = SEQUENCE(s_3) = S_2$, $SEQUENCE(t_2) = SEQUENCE(t_3) = T_2$, $SEQUENCE(s_4) = SEQUENCE(s_5) = S_3$, $SEQUENCE(t_4) = T_3$, $SEQUENCE(s_6) = S_4$, $SEQUENCE(t_1) = SEQUENCE(t_6) = T_4$.

Lema 3.1. *[19] Un modelo es estable precisamente cuando todos sus end points son estables.*

Podemos ver los siguientes ejemplos: el primero no estabilizado (Figura 3.1) y el segundo si (Figura 3.2).

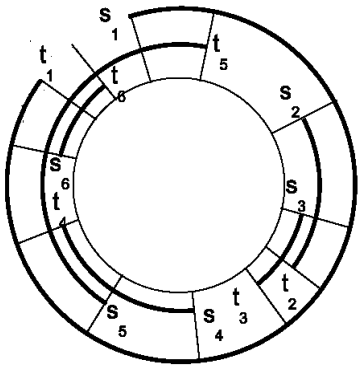

Figura 3.1: Ejemplo de modelo no estable.

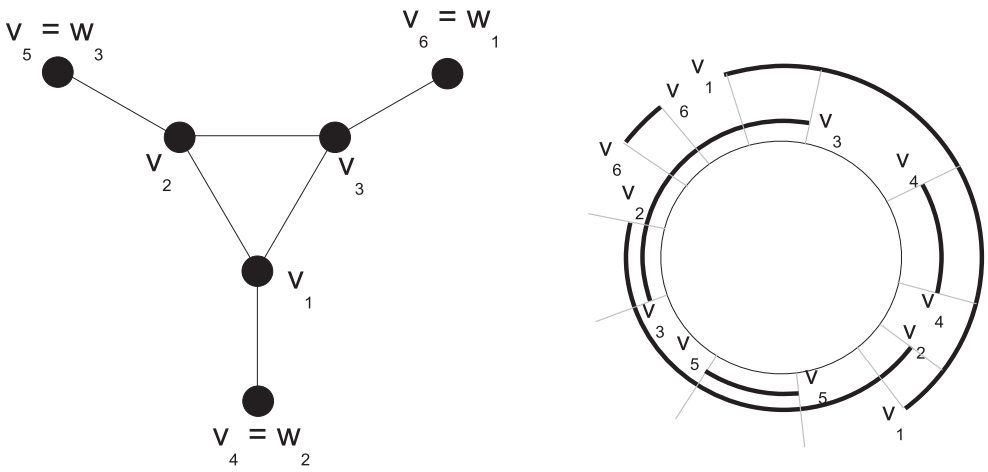

Figura 3.2: Ejemplo de modelo estable.

Si $\mathcal{M} = (C, \mathcal{A})$ es un modelo estable y G es el grafo intersección de A, entonces podemos decir que $\mathcal{M}$ es un modelo estable de G.

En [19], los autores presentan un algoritmo de tiempo $O(n)$ que dado un modelo arco-circular $\mathcal{M}$ de un grafo G, lo transforma a otro modelo arco-circular $\mathcal{M}'$ estable equivalente. Dicho algoritmo se basa en la idea de estirar lo más posible todos los extremos de los arcos, preservando las adyacencias.

Algoritmo 3.2 ([19]). *Dado un grafo arco-circular y un modelo arco-circular $\mathcal{M}$ de G, devuelvo un modelo arco-circular estable $\mathcal{M}'$ de G.*

Dentro de la implementación del Algoritmo 3.2, se utiliza una operación llamada **REORDER** para ordenar los extremos de cada s-sequence de acuerdo al siguiente criterio: si s_i y s_j son start points de la s-sequence en cuestión y t_j precede a t_i (tomando como punto de partida el primer extremo después de la secuencia en el sentido horario), entonces s_i debe preceder a s_j. Se puede observar, después de esta operación, cada conjunto de arcos que tienen su start point en una misma s-sequence quedan linealmente ordenados por inclusión.

A continuación, describimos una manera fácil de determinar si existe un par de arcos que cubre todo el círculo C, dado un modelo arco-circular $\mathcal{M} = (C, \mathcal{A})$. En primer lugar, se puede obtener el modelo complemento $\overline{\mathcal{M}} = (C, \overline{\mathcal{A}})$ (si $(s_i, t_i) \in \mathcal{A}$ entonces $(t_i, s_i) \in \overline{\mathcal{A}}$). Es fácil de ver que no hay dos arcos que cubren C en $\mathcal{M}$ si y sólo si en $\overline{\mathcal{M}}$ todo par de arcos tiene intersección no vacía. Lo cual implica que el grafo de intersección de $\overline{\mathcal{M}}$ es un grafo completo y el conjunto independiente máximo tiene tamaño 1. Con lo cual podemos aplicar el algoritmo descripto en [20] para determinar un conjunto independiente máximo en tiempo $O(n)$ tomando como entrada $\overline{\mathcal{M}}$. En el caso en que el conjunto resultante tiene dos arcos disjuntos (t_i, s_i) y (t_j, s_j), entonces (s_i, t_i) y (s_j, t_j) cubren todo C.

3.2. Hallar un $\overline{3K_2}$ que contenga a A_1, A_2 y A_3

En esta sección vamos a proponer un subalgoritmo de tiempo $O(n)$ para el punto 5 del Algoritmo 3.1 que recibe como input un modelo arco-circular $\mathcal{M} = (C, \mathcal{A})$ de un grafo G y tres arcos distintos $A_1, A_2, A_3 \in \mathcal{A}$ que cubren C y tal que $\tau_C(G) > 2$ (por lo tanto ningún par de arcos de $\mathcal{A}$ pueden cubrir C) y determina si existe algún subgrafo inducido $\overline{3K_2}$ de G que contenga a A_1, A_2 y A_3.

Sin pérdida de generalidad podemos suponer que el orden circular relativo de los extremos de los arcos A_1, A_2 y A_3 es el siguiente: $s_1, t_3, s_2, t_1, s_3, t_2$ (sino podemos intercambiar los nombres de los arcos A_2 y A_3). Ahora necesitamos agregar los extremos de otros tres arcos A_4, A_5 y A_6 para que juntos con los anteriores formen un submodelo arco-circular que representa un $\overline{3K_2}$ y la única forma de hacerlo es la siguiente (con los renombramientos adecuados

de los últimos tres arcos): s_1, t_3, s_4, t_6, s_2, t_1, s_5, t_4, s_3, t_2, s_6, t_5 (ver la Figura 3.3).

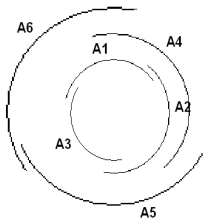

Figura 3.3: Modelo arco-circular de un grafo $\overline{3K_2}$.

Dado el orden relativo anterior, notemos que los arcos A_1, A_2 y A_3 nos vienen dados, por lo tanto lo que tenemos que hacer es encontrar, si existen, los arcos A_4, A_5 y A_6 que junto con los anteriores forman el subgrafo inducido $\overline{3K_2}$ de G. Por lo tanto los posibles candidatos a A_6 son los arcos que comienzan entre t_2 y s_1 y terminan entre t_3 y s_2. Llamamos al primer segmento y al segundo segmento como área de gestación y área de terminación respectivamente de los candidatos a A_6. No es díficil de ver que si un arco A candidato a A_6 contiene a otro arco A' que también es candidato a A_6, podemos descartar a A' ya que en cualquier $\overline{3K_2}$ de G que involucra a A_1, A_2, A_3 y A' podemos sustituir A' por A y sigue siendo $\overline{3K_2}$. Por lo tanto, no consideramos los arcos candidatos a A_6 que que están contenidos en otro arco candidato a A_6 (ver la Figura 3.4). Podemos proceder de la misma manera con los arcos candidatos a A_5 definiendo como el área de gestación entre t_1 y s_3 y el área de terminación entre t_2 y s_1.

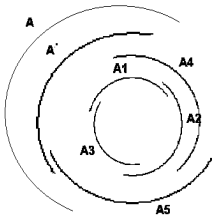

Figura 3.4: Ejemplo de inclusión entre candidatos a A_6.

Llamamos $A_6^1, \ldots, A_6^p$ los arcos candidatos (sin inclusiones) a A_6 ordenados en el sentido anti-horario de acuerdo a sus end points, es decir que el orden relativo de los extremos de estos arcos es: $s_6^p, \ldots, s_6^1, t_6^p, \ldots, t_6^1$. De modo similar llamamos $A_5^1, \ldots, A_5^q$ los arcos candidatos (sin inclusiones) a A_5 ordenados en el sentido horario de acuerdo a sus start points, es decir que el orden relativo de los extremos de estos arcos es: $s_5^1, \ldots, s_5^q, t_5^1, \ldots, t_5^q$.

Para cada arco candidato A a A_4, previamente definiendo el área de gestación y área de terminación de A_4, determinamos el arco A_6^i que lo interseca con el mayor superíndice i y lo mismo con el arco A_5^j que lo interseca

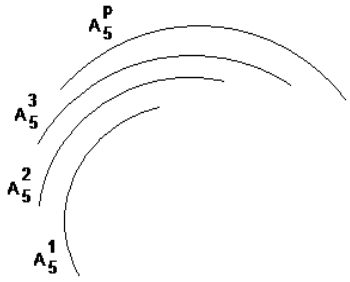

Figura 3.5: Escalonamiento de los candidatos a A_5.

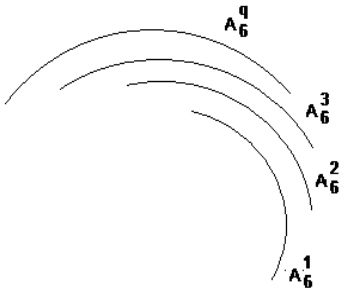

Figura 3.6: Escalonamiento de los candidatos a A_6.

con el mayor superíndice j. Claramente, si existe un $\overline{3K_2}$ de G integrado en parte por A_1, A_2, A_3 y A, entonces en particular $\{A_1, A_2, A_3, A, A_5^j, A_6^i\}$ induce un $\overline{3K_2}$. Por lo tanto, basta con verificar si A_5^j y A_6^i tiene o no intersección en común para saber si A realmente puede tomar el rol de A_4 o no. Esta verificación claramente cuesta $O(1)$. El problema radica en cómo hallar A_6^i y A_5^j en forma eficiente. En el pseudocódigo de abajo, se muestra cómo ir actualizando el mejor arco candidato A_6^i (A_5^j) a medida que se van recorriendo los extremos en el área de gestación (terminación) de los arcos candidatos a A_4 en el sentido anti-horario (horario), dicha área coincide con el área de terminación (gestación) de los $A_6^1, \ldots, A_6^p$ ($A_5^1, \ldots, A_5^q$).

1. El propósito de este paso es armar una lista de candidatos L6 para jugar el papel de A_6. Inicialmente la lista L6 está vacía y p=s_1. Vamos a recorrer cada extremo t_j a partir de s_2 hasta t_3 (excluyéndolo) en el sentido anti-horario. Si s_j está dentro de (t_2,p), entonces L6:=L6 $\bigcup$ $\{t_j\}$ y p:=s_j. De esta manera, los arcos que corresponden a extremos que están en L6 no tienen relación de contención y cada arco empieza antes que su arco anterior en la lista.

2. De la misma manera se puede armar una lista de candidatos L5 para jugar el papel de A_5. Inicialmente la lista L5 está vacía y p=t_2. Ahora que hay que recorrer cada extremo s_i a partir de t_1 hasta s_3 (excluyéndolo) en el sentido horario. Si t_i está dentro de (p,s_1), entonces L5:=L5 $\bigcup$ $\{s_i\}$ y p:=t_i. De esta manera, los arcos que están en L5 no tienen relación de contención y cada arco empieza antes que su arco posterior en la lista.

3. Ahora para cada arco A_q candidato a A_4, es decir, que s_q está en (t_3, s_2) y t_q está en (t_1, s_3), vamos a buscar el mejor candidato A_6 para él siempre que sea posible y también el mejor candidato A_5 mientras que sea posible. Para ello se debe hacer lo siguiente:

   a) p:=vacío y recorrer cada extremo e a partir de s_2 (excluyéndolo) hasta t_3 en el sentido anti-horario y analizar e de acuerdo a su tipo de extremo:

      e es de tipo start point $e = s_q$) Si t_q está en (t_1, s_3), entonces A_q es candidato a A_4, y ponemos MejorA6(A_q):=p.

      e es de tipo end point $e = t_j$) Si t_j es el primer elemento de L6 entonces p:=A_j y borrarlo de L6.

   b) p:=vacío y recorrer cada extremo e a partir de t_1 (excluyéndolo) hasta s_3 en el sentido horario y analizar e de acuerdo a su tipo de extremo:

      e es de tipo end point $e = t_q$) Si s_q está en (t_3, s_2), entonces A_q es candidato a A_4, y ponemos MejorA5(A_q):=p, si MejorA5(A_q)

y MejorA6(A_q) tiene intersección no vacía, hemos encontrado los arcos A_4=A_q, A_5=MejorA5(A_q) y A_6=MejorA6(A_q) para formar con A_1, A_2 y A_3, un $\overline{3K_2}$

e es de tipo start point $e = s_i$) Si s_i es el primer elemento de L5 entonces p:=A_i y borrarlo de L5.

La complejidad de esta implementación es $O(n)$ porque hay que recorrer a lo sumo 4 veces todos los extremos. Por lo tanto, la complejidad de hallar en G un $\overline{3K_2}$ que contenga a A_1, A_2 y A_3 es ahora lineal. Con esta mejora la complejidad del Algoritmo 3.1 pasa a ser $O(m^{\frac{3}{2}})$.

Para reconocer si un grafo G es o no un grafo arco-circular sin $\overline{3K_2}$, podemos en primer lugar verificar si G es arco-circular y en el caso afirmativo conseguir un modelo arco-circular $\mathcal{M} = (C, \mathcal{A})$, todo esto en tiempo $O(n+m)$ [26]. Luego aplicamos el subalgoritmo descripto en esta sección para cada trio de arcos A_1, A_2 y A_3 que cubren todo el círculo C para encontrar un submodelo de $\overline{3K_2}$. Claramente, la cantidad de aplicaciones es a lo sumo $O(n^3)$ y por lo tanto la complejidad final es $O(n^4)$.

Teorema 3.2. *Existe un algoritmo de tiempo $O(n^4)$ para reconocer grafos arco-circulares sin $\overline{3K_2}$.*

3.3. Completando la implementación lineal

La complejidad de la implementación anterior del Algoritmo 3.1 radica en el uso del Algoritmo 2.2 que es para grafos generales en el punto 3). En esta sección vamos a mejorar el orden del Algoritmo 3.1 reemplazando la llamada al Algoritmo 2.2 por una llamada al Algoritmo 3.3 que será presentado en esta sección. Este último algoritmo utiliza las características específicas de los grafos arco circulares, llegando a un algoritmo de tiempo $O(n)$ si el input es un modelo arco-circular (en caso contrario hay que encontrar un modelo arco-circular del grafo dado, y eso puede hacerse en tiempo $O(m)$).

De acuerdo al siguiente teorema, sabemos que dado un modelo arco-circular $\mathcal{M} = (C, \mathcal{A})$ de un grafo G, si este modelo es anormal, es decir que posee dos arcos $A_1, A_2 \in \mathcal{A}$ tal que $A_1 \cup A_2 = C$, entonces los vértices correspondientes a $\{A_1, A_2\}$ forman un clique transversal de G ($\tau_c(G) \leq 2$).

Teorema 3.3 ([15])**.** *Dado un grafo $G = (V, E)$ y un modelo arco circular $\mathcal{M} = (C, \mathcal{A})$ del mismo, si existen dos arcos $A_1, A_2 \in A$, que cubren enteramente a C, $\tau_c(G) \leq 2$.*

Entonces para saber si un grafo arco-circular G tiene o no un clique transversal de tamaño 2, una manera sería tratar de hallar algun modelo anormal de G. El algoritmo de estabilización de modelos que mencionamos en la sección 3.1, justamente intenta modificar el modelo para que tenga la mayor cantidad de pares de arcos que cubren a l círculo C, es decir que trata

de anormalizar el modelo. Pero, existen grafos que tienen clique transversales de tamaño 2 y no poseen ningún modelo anormal, por ejemplo C_4. A pesar de ello, el algoritmo de estabilización juega un rol principal en nuestro Algoritmo 3.2 que describimos a continuación, ya que restringue notoriamente las posibles configuraciones de los clique transversales de tamaño 2.

Algoritmo 3.3. *Dado un grafo $G = (V, E)$ sin vértice universal y un modelo arco circular $\mathcal{M} = (C, \mathcal{A})$ del mismo, determina si $\tau_c(G) = 2$.*

a) Estabilizar el modelo. Esto se hace en $O(n)$. (ver punto 3.1. Elementos auxiliares de este capítulo, Construir un modelo estable [19])

b) Reordenar los extremos de cada secuencia de extremos tal que para cada par de arcos que tienen extremos en una misma secuencia de extremos, uno debe contener al otro. Esto se puede hacer también en $O(n)$. (ver punto 3.1. Elementos auxiliares de este capítulo, operación $REORDER$ [19])

c) Buscar si hay 2 arcos que cubren todo el círculo (ver punto 3.1. Elementos auxiliares de este capítulo [19]), en el caso que encontrar tales arcos, entonces τ_c=2. Esto también se puede hacer en $O(n)$.

d) Examinar circularmente si existe un end point t_j seguido inmediatamente por un start point s_i, y tal que al end point t_i también le sigue inmediamente s_j. Este chequeo se puede hacer en $O(n)$. Si existen tales arcos A_i y A_j, entonces $\tau_c = 2$, sino $\tau_c > 2$.

En el punto (b) del algoritmo propuesto aplicamos la operación $REORDER$ al modelo arco circular estable $\mathcal{M} = (C, \mathcal{A})$. La operación $REORDER$, realiza únicamente reordenamientos de extremos que están dentro de una secuencia, por lo que esta operación claramente no altera las adyacencias del grafo asociado ni la estabilidad del modelo. Luego de aplicar $REORDER$ obtenemos en consecuencia otro modelo estable y equivalente al anterior (es decir que el grafo asociado es el mismo). Llamaremos a este otro modelo obtenido $\mathcal{M}'$.

Teorema 3.4. *El Algoritmo 3.3 es correcto.*

Demostración. En primer lugar, probaremos que no genera falsos positivos, es decir que cuando el algoritmo afirma que $\tau_c = 2$ entonces realmente lo es. Los pasos en los cuales puede haber hecho esta afirmación son (c) y (d). En el caso de (c), lo avala el Teorema 3.3 porque se encontró un par de arcos que cubren todo el círculo C. Solamente tenemos que probar que las afirmaciones provenientes de (d) son ciertas. En este caso, se debe haber encontrado un par de arcos A_i y A_j disjuntos tal que todo extremo de cualquier otro arco $A_k \notin \{A_i, A_j\}$ debe estar contenido en A_i o en A_j. Supongamos que $\{A_i, A_j\}$ no es clique transversal, entonces existe un certificado negativo

que consiste de un sólo vértice o un par de vértices adyacentes. En el caso de un sólo vértice, entonces no debe ser adyacente con ninguno de los vértices correspondientes a A_i y A_j. Lo cual es imposible de ocurrir. Entonces, el certificado negativo debe ser un par de vértices adyacentes, uno de ellos cuyo arco se interseca con A_i pero no con A_j entonces debe estar contenido en A_i, ocurre exactamente lo contrario con el otro vértice cuyo arco debe estar contenido en A_j, por lo tanto no pueden ser adyacentes, lo cual contradice que son un par de vértices adyacentes.

Ahora vamos a probar que el algoritmo no genera falsos negativos. Supongamos que sí, es decir que existe un par de arcos A_1, A_2 cuyos vértices conforman un clique transversal R (en este caso debe ser mínimo porque no hay vértice universal en el grafo) y el algoritmo no fue capaz de determinar que $\tau_c = 2$.

Se pueden ver en las siguientes figuras las configuraciones posibles de un par de arcos, debemos analizar cuales son válidas para A_1 y A_2 en M'.

Figura 3.7: son disjuntos

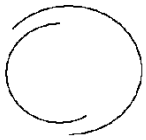

Figura 3.8: cubren el círculo

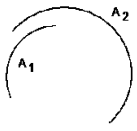

Figura 3.9: se superponen (overlap)

Claramente, el caso de la Figura 3.10 se puede descartar porque el clique transversal R no sería mínimo.

El paso c) del Algoritmo 3.3 detecta precisamente la configuración de la figura 3.8. Así que debe ser reportado afirmativamente en ese paso. Por lo tanto, esta configurción no puede ocurrir.

Figura 3.10: uno contiene al otro

Quedan solamente posibles las configuraciones de las figuras 3.7 y 3.9 para los arcos A_1 y A_2. En cualquiera de ellas, se puede observar que el segmento $(t_2, s_1) \cap A_2 = \emptyset$ y $(t_2, s_1) \cap A_1 = \emptyset$.

Sea T la t-sequence que contiene a t_2 y S la s-sequence que contiene a s_1. Consideramos las siguientes y únicas dos posibilidades:

i) $NEXT(T) = S$

ii) $NEXT(T) = S' \neq S$ y $NEXT^{-1}(S) = T' \neq T$. Además se puede ver que el orden circular relativo de ellas es: T,S',T',S

Veamos que no es posible que pase ii). Supongamos que sí. Sea A_3 un arco cuyo end point t_3 está en T'. Claramente, $A_3 \cap A_1 = \emptyset$ pués caso contrario, el modelo no sería estable y es una contradicción. Entonces $A_3 \cap A_2 \neq \emptyset$ porque sino el vértice correspondiente a A_3 sería un certificado negativo para el clique transversal R, lo cual es una contradicción. Entonces $s_3 \in (t_1, t_2)$. Ahora sea A_4, un arco tal que su start point $s_4 \in S$'. Nuevamente, podemos afirmar que $A_4 \cap A_2 = \emptyset$ usando el argumento de que el modelo es estable. Pero debe ocurrir $A_4 \cap A_3 \neq \emptyset$ porque $s_4 \in A_3$. Por lo tanto los vértices correspondientes a A_4 y A_3 constituyen un certificado negativo para el clique transversal R, lo que es una contradicción. Por lo tanto, ii) no puede ocurrir y siempre ocurre i).

Supongamos que la configuración es la de la Figura 3.9, es decir que $A_1 \cap A_2 \neq \emptyset$ y como verifica i) de acuerdo al párrafo anterior, contradice que el modelo es estable. Lo cual implica que el caso de la Figura 3.9 nunca puede ocurrir.

Queda entonces posible solamente, la configuración de la Figura 3.7, cumpliendo las siguientes igualdades:

$$NEXT(SEQUENCE(t_2) = T) = SEQUENCE(s_1) = S$$

$$NEXT(SEQUENCE(t_1) = T') = SEQUENCE(s_2) = S'$$

Podemos tomar s_i como el primer extremo de S, como s_i y s_1 están en una misma s-sequence S, entonces t_i debe estar en T' o una t-sequencia posterior, este último caso es imposible de ocurrir porque el modelo no sería estable. A su vez, t_i debe ser el último extremo de T' porque sino el modelo no sería estable nuevamente. Ahora procedemos de la misma manera, tomando s_j como el primer extremo de S', llegamos a que t_j es el último extremo de T.

Esto implica que s_i sigue inmediatamente a t_j y s_j sigue inmediamente t_i. Por lo tanto el algoritmo debe haber afirmado que $\tau_c = 2$. Lo cual es una contradicción. □

Finalmente, con esta última mejora la complejidad del Algoritmo 3.1 es $O(n)$ (si el input es un modelo arco-circular, en caso contrario hay que generar el modelo y esto cuesta tiempo $O(m)$).

Capítulo 4

Grafos dualmente cordales

4.1. Introducción

En la introducción, habíamos definido los grafos dualmente cordales como exactamente los grafos que admiten un orden de vecindades máximas. Pero también se pueden caracterizar como los grafos clique de los grafos cordales. De todas maneras, vamos a utilizar como base los órdenes de vecindades máximas para desarrollar nuestros algoritmos en este capítulo. Los grafos dualmente cordales pueden ser reconocidos en tiempo $O(m+n)$ [7] y dado un grafo dualmente cordal es posible obtener un orden de vecindades máximas también en este tiempo [5, 7, 6].

A su vez, estos mismos autores en [5], propusieron un algoritmo para resolver el problema de Clique r-Dominancia para esta clase de grafos. La complejidad del algoritmo es lineal de acuerdo al tamaño de su entrada que consiste en un grafo dualmente cordal, una familia de completos $C_1, \ldots, C_k$ (pueden no ser cliques) de este grafo y una función $r : \{C_1, \ldots, C_k\} \longrightarrow \mathbf{Z}_{\geq 0}$. Se podría utilizar este algoritmo para resolver el problema de clique transversal mínimo para grafos dualmente cordales, simplemente con poner como entrada el grafo en cuestión, la familia de todos los cliques del grafo y $r(C_i) = 1$ para todo clique C_i de dicha familia. El único inconveniente es que la cantidad de cliques puede ser exponencial con respecto al tamaño del grafo. Con lo cual no sería un algoritmo polinomial.

Por otro lado, en [6] se desarrolla un algoritmo lineal para la r-Dominancia en grafos dualmente cordales. Usa un orden de vecindades máximas generado por el algoritmo MNO de [5]. Vamos a dar una simplificación de este algoritmo teniendo en cuenta que $r = 1$. El algoritmo va construyendo golosamente un subconjunto D de vértices que domina a los vértices recorridos hasta el momento de acuerdo al orden de vecindades máximas.

Algoritmo 4.1 ([6]). *Dado un grafo dualmente cordal G, genera un conjunto dominante mínimo*

1. *Obtener un orden de vecindades máximas $v_1, \ldots, v_n$ con sus vecinos máximos $u_i, \ldots, u_n$.*

2. $D := \emptyset$

3. Para cada vértice v_i, $1 \leq i \leq n$ *hacer*

Si $N[v_i] \cap D = \emptyset$ *entonces* $D := D \cup \{u_i\}$

4. Retornar D.

En el siguiente lema probamos que el ≪simplicial augmentation≫ de un grafo dualmente cordal también lo es.

Lema 4.1. *Dado un grafo dualmente cordal* G*, su ≪simplicial augmentation≫* $\mathcal{S}(G)$ *es también dualmente cordal.*

Demostración. Si G es un grafo dualmente cordal, quiere decir que existe un ordenamiento $v_1, \ldots, v_n$ de los vértices de $V(G)$, que es un orden de vecindades máximas. Notemos que para que la ≪simplicial augmentation≫ $\mathcal{S}(G)$ sea también un grafo dualmente cordal, $\mathcal{S}(G)$ tiene que tener también un orden de vecindades máximas para sus vértices. La diferencia entre el conjunto de vértices $V(G)$ y el conjunto de vértices de $V(\mathcal{S}(G))$ está dada por los vértices w_j que se sumaron durante la ≪simplicial augmentation≫. Entonces, la idea es generar un orden de vecindades máximas de $\mathcal{S}(G)$ a partir de un orden de vecindades máximas de G, insertando adecuadamente los vértices w_j. Concretamente, w_j debe ser agregado justo antes al primer vértice del clique C_j de G correspondiente a w_j, podemos suponer que es el vértice v_i. De esta manera, el orden de vecindades máximas de G con el agregado de todos los vértices de la ≪simplicial augmentation≫ en la ubicación especificada es un orden de vecindades máximas de $\mathcal{S}(G)$. No es difícil de verificar que el vecino máximo de cada vértice original de G sigue siendolo en este nuevo orden y el vecino máximo de v_i es vecino máximo de w_j. En consecuencia $\mathcal{S}(G)$ también es un grafo dualmente cordal. □

El siguiente algoritmo aprovecha la reducción propuesta en la prueba del Lema 4.1 para resolver el problema clique transversal mínimo a través del problema de conjunto dominante mínimo, de manera similar en que se usa para los grafos HCA [19].

Algoritmo 4.2. *Algoritmo para resolver el problema de encontrar el clique transversal en grafos dualmente cordales*

1) generar todos los cliques C_j *de* G *para obtener la ≪simplicial augmentation≫* G' *de* G*, esto cuesta* $O(\#Cmn)$ *donde* $\#C$ *es la cantidad de cliques de* G.

2) Obtener la ≪simplicial augmentation≫ G' que cuesta $O(|C_1|+...+|C_{\#C}|)$

3) Aplicar el Algoritmo 4.1 para G'.

Se puede observar que este algoritmo tiene la misma complejidad (exponencial) que el algoritmo de Clique r-Dominancia aplicado para todos los cliques del grafo [5].

En la siguiente sección mostraremos una manera de implementar el Algoritmo 4.2 sin tener que generar los cliques del grafo G, por lo tanto la complejidad no depende de la cantidad de cliques sino del tamaño de clique transversal mínimo como los algoritmos generales vistos en el Capítulo 2. También vamos a desarrollar distintas versiones de este algoritmo que permiten ir mejorando la complejidad.

4.2. Nuestro primer algoritmo

La idea es similar al Algoritmo 4.1 donde recorre un orden de vecindades máximas de G que es una subsecuencia de un orden de vecindades máximas de G', los únicos vértices que faltan corresponden a cliques de G. Pero se debe analizar adicionalmente cuando el vértice v_i es considerado, la influencia que pudieron haber tenido los vértices faltantes delante de v_i que corresponden a cliques que contienen a v_i como primer vértice dentro del orden de vecindades máximas de G. En el caso que $N[v_i] \cap D = \emptyset$ obliga a incluir u_i al subconjunto D, como u_i tambíen es máximo vecino de estos vértices faltantes, están cubiertos por el mismo, por lo tanto no hay que hacer nada. El caso que podría tener problemas es cuando $N[v_i] \cap D \neq \emptyset$, aquí no tenemos certeza si los vértices faltantes están cubiertos o no por D (cabe destacar que estos vértices corresponden a cliques de G que contienen a v_i como primer vértice y es posible que hayan otros cliques de G que contienen a v_i pero deben corresponder a vértices que están delante de algun vértice v_j con $j < i$, es decir que ya deben estar cubiertos por D). Para verificar esto es equivalente chequear si $D \cap N(v_i)$ es clique tranversal del subgrafo inducido $G[N[v_i]]$ (ver Lema 4.2). Aquí podemos utilizar el Algoritmo 2.1 para realizar esta tarea.

Previamente, vamos a definir los siguientes conceptos: dado un grafo $G = (V, E)$, denotamos $\mathcal{C}_G(v_i) = \{C \text{ clique de } G / v_i \in C\}$ y decimos que un subconjunto de vértices $W \subseteq V$ es tranversal de $\mathcal{C}_G(v_i)$ si $\forall C \in \mathcal{C}_G(v_i), C \cap W \neq \emptyset$. No es difícil de ver que un subconjunto de vértices $W \subseteq V$ es clique tranversal de G si y sólo si W es transversal de $\mathcal{C}_G(v), \forall v \in V$.

Lema 4.2. *Dado un grafo $G = (V, E)$, un vértice $v \in V$ y un subconjunto de vértices $W \subseteq V$. W es clique transversal de $\mathcal{C}_G(v) \Leftrightarrow W$ es clique transversal de $G[N[v]]$.*

Demostración. Basta con probar que los cliques de $\mathcal{C}_G(v)$ son exactamente los cliques de $G[N[v]]$. Sea C un clique de $\mathcal{C}_G(v)$, claramente $C \subseteq N[v]$ por lo tanto, C sigue siendo un clique en $G[N[v]]$. Falta ver que todo clique C de $G[N[v]]$,$C \in \mathcal{C}_G(v)$. Supongamos que existe un clique C de $G[N[v]]$,$C \notin$

$\mathcal{C}_G(v)$. Entonces $v \notin C$, entonces C no es maximal pués $C \cup \{v\}$ sigue siendo completo de G, absurdo. □

Algoritmo 4.3. *Dado un grafo dualmente cordal G, genera un clique transversal de G*

1. *Obtener un orden de vecindades máximas $v_1, \ldots, v_n$ con sus vecinos máximos $u_i, \ldots, u_n$.*
2. *$D := \emptyset$*
3. *Para cada vértice v_i, $1 \leq i \leq n$ hacer*

   Si $N[v_i] \cap D = \emptyset$ entonces $D := D \cup \{u_i\}$

   De lo contrario, si $D \cap N(v_i)$ no es clique transversal del subgrafo inducido $G[N(v_i)]$ entonces $D := D \cup \{u_i\}$
4. *Retornar D.*

La complejidad de este algoritmo es determinada fundamentalmente por verificar si un subconjunto es clique transversal de un grafo o no, pero esto es exactamente el Algoritmo 2.1 de esta tesis ($O(k^2(m^{\frac{k}{2}} + n))$. Podemos suponer en el peor de los casos, que se necesita aplicar $n-1$ veces dicho algoritmo (una vez por cada vértice $v_i, 2 \leq i \leq n$) y la cardinalidad de D ($1 \leq |D| \leq \tau_c(G)$) puede incrementar pero nunca decrementar entre dos aplicaciones consecutivas. Por lo tanto, la peor distribución del valor de $|D|$ para estas $n-1$ aplicaciones es, una vez 1, una vez 2, ..., una vez $\tau_c(G)-1$ (A) y $n-1-(\tau_c(G)-1) = n-\tau_c(G)$ veces $\tau_c(G)$ (B).

Siendo la complejidad de (A) es $O(A) = O(\sum_{k=1}^{\tau_c(G)-1} k^2(m^{\frac{k}{2}} + n))$ y la complejidad de (B) es $O(B) = O((n-\tau_c(G))\tau_c(G)^2(m^{\frac{\tau_c(G)}{2}} + n))$. Podemos suponer que G no tiene vértice universal y es conexo. Entonces $O(A) \cong O(2n + \sum_{k=2}^{\tau_c(G)-1} k^2 m^{\frac{k}{2}}) = O(2n + n\tau_c(G)^2 m^{\frac{\tau_c(G)}{2}}) = O(n\tau_c(G)^2 m^{\frac{\tau_c(G)}{2}})$ y $O(B) \cong O(n\tau_c(G)^2 m^{\frac{\tau_c(G)}{2}})$. Con lo cual, el orden de este Algoritmo 4.3 es $O(O(A) + O(B)) \cong O(\tau_c(G)^2 n m^{\frac{\tau_c(G)}{2}})$.

4.3. Algoritmo Propuesto Mejorado

Presentamos a continuación un mejor algoritmo modificando el Algoritmo 4.3. La diferencia consiste en agregar una verificación para ver si D es un clique transversal de G, cada vez que aumenta el tamaño de D.

Algoritmo 4.4. *Dado un grafo dualmente cordal G, genera un clique transversal de G*

1. *Obtener un orden de vecindades máximas $v_1, \ldots, v_n$ con sus vecinos máximos $u_i, \ldots, u_n$.*

2. $D := \emptyset$

3. Para cada vértice v_i, $1 \leq i \leq n$ *hacer*

Si $N[v_i] \cap D = \emptyset$ *entonces* $D := D \cup \{u_i\}$

De lo contrario, si $D \cap N(v_i)$ *no es clique transversal del subgrafo inducido* $G[N(v_i)]$ *entonces* $D := D \cup \{u_i\}$

En cualquier caso, si hubo cambio en D*, verificar si* D *es clique transversal de* G *(aplicando el Algoritmo 2.1)*

En caso positivo, retornar D *y termina la ejecución del algoritmo.*

Nuevamente, en el peor de los casos, se necesita aplicar (a) $n-1$ veces el algoritmo 2.1 para verificar si $D \cap N(v_i)$ es o no clique transversal en el subgrafo inducido $G[N(v_i)]$ y (b) $\tau_c(G)$ veces el mismo algoritmo para ver si D es clique transversal de G (con $\tau_c(G)-1$ fracasos y éxito en el último intento). Pero ahora la peor distribución del valor de $|D|$ en para las $n-1$ aplicaciones de (a) es, una vez 1, una vez 2, ..., una vez $\tau_c(G)$-2, una vez $\tau_c(G)$ y $n-\tau_c(G)$ veces $\tau_c(G)-1$. En consecuencia, el orden del Algoritmo 4.4 es: $O(\sum_{k=1}^{\tau_c(G)-2} k^2(m^{\frac{k}{2}}+n)+(n-\tau_c(G))(\tau_c(G)-1)^2(m^{\frac{(\tau_c(G)-1)}{2}}+n)+\tau_c(G)^2(m^{\frac{\tau_c(G)}{2}}+n)+\sum_{k=1}^{\tau_c} k^2(m^{\frac{k}{2}}+n)) \cong O(\tau_c(G)^2 n m^{\frac{(\tau_c(G)-1)}{2}})$ (suponiendo que G no tiene vértice universal y es conexo).

4.4. Algoritmo Propuesta Final

En el Capítulo 2, propusimos el Algoritmo 2.4 para verificar si un subconjunto de vértices $W = \{v_1, \ldots, v_{k-1}\}$ es o no un clique transversal de un grafo G y en el caso negativo, si era posible encontrar algun vértice adicional v_k para completar junto a W un clique transversal de G. Entonces, en lugar de utilizar el Algoritmo 2.1 para chequear si D es clique transversal de G cada vez que D aumenta su tamaño en el Algoritmo 4.4, utilizamos ahora el Algoritmo 2.4. Con este cambio, la complejidad final es: $O(\tau_c(G) n m^{\frac{\tau_c(G)}{2}-1}(\tau_c(G)+\sqrt{m}))$.

Capítulo 5

Conclusiones y Trabajo Futuro

Tabla 1: esta tabla sumariza las complejidades del problema de clique transversal en distintas subclases de grafos: algoritmos propuestos en esta tesis y los mejores algoritmos previos.

Problema	Clase de Grafo	Orden del Alg. Propuesto	Alg. Previo
¿$\tau_c(G)$=2?	Grafos Generales	$O(m^{\frac{3}{2}})$	$O(n^4)$
¿$\tau_c(G)$?	Grafos Generales	$O(n^{\tau_c(G)-1}m^{\frac{\tau_c(G)}{2}})$	$O(n^{2\tau_c(G)})$
¿$\tau_c(G)$?	Arco circulares sin $\overline{3K_2}$	$O(n)$	$O(n^4)$
¿$\tau_c(G)$?	Dualmente Cordal	$O(\tau_c(G)nm^{\frac{\tau_c(G)}{2}-1}(\tau_c(G)+\sqrt{m}))$	$O(n^{2\tau_c(G)})$

Se podría analizar la posibilidad de usar la ≪simplicial augmentation≫ en otras clases de grafos donde el problema de dominación tengan soluciones polinomiales sin preocuparnos en que la cantidad de cliques sea polinomial o no.

Bibliografía

[1] N. Alon, R. Yuster and U. Zwick (1997). Finding and counting given length cycles. Algorithmica, 17 (3), 209–223.

[2] V. Balachandran, P. Nagavamsi and C. Pandu Rangan (1996). Clique transversal and clique independence on comparability graphs. Information Processing Letters, 58, 181-184.

[3] F. Bonomo, G. Durán, M. C. Lin and J. L. Szwarcfiter (2006). On Balanced Graphs. Mathematical Programming B, 105, 233–250.

[4] A. Brandstädt (2004). (P_5, diamond)-free graphs revisited: Structure and linear time optimization. Discrete Applied Mathematics, 138(1-2), 13–27.

[5] A. Brandstädt, V. D. Chepoi and F. F. Dragan (1997). Clique r-Domination and Clique r-Packing Problems on Dually Chordal Graphs. SIAM Journal on Discrete Mathematics, 10(1), 109–127.

[6] A. Brandstädt, V. D. Chepoi and F. F. Dragan (1998). The algorithmic use of hypertree structure and maximum neighbourhood orderings. Discrete Applied Mathematics, 82, 43–77.

[7] A. Brandstädt, F. F. Dragan, V. D. Chepoi and V. Voloshin (1998). Dually Chordal graphs. SIAM Journal on Discrete Mathematics, 11, 437–455.

[8] G. J. Chang, M. Farber and Z. Tuza (1993). Algorithmic aspects of neighbourhood numbers. SIAM Journal on Discrete Mathematics, 6, 24–29.

[9] M. S. Chang (1998). Efficient algorithms for the domination problems on interval and circular-arc graphs. SIAM Journal on Computing, 27, 1671–1694.

[10] M. S. Chang, Y. H. Chen, G. J. Chang and J. H. Yan (1996). Algorithmic aspects of the generalized clique transversals problem on chordal graphs. Discrete Applied Mathematics, 66, 189–203.

[11] N. Chiba and T. Nishizeki (1985). Arboricity and subgraph listing algorithms. SIAM Journal on Computing, 14(1), 210–223.

[12] E. Dahlhaus, P. D. Manuel and M. Miller (1998). Maximum h-colourable subgraph problem in balanced graphs. Information Processing Letters, 65, 301-303.

[13] G. Durán, M. L. Lin and J. L. Szwarcfiter (2002). On clique-transversals and clique-independent sets. Annals of Operations Research, 116, 71–77.

[14] G. Durán, M. L. Lin, S. Mera and J. L. Szwarcfiter (2006). Algorithms for clique-independent sets on subclasses of circular-arc graphs. Discrete Applied Mathematics, 154(13), 1783–1790.

[15] G. Durán, M. L. Lin, S. Mera and J. L. Szwarcfiter (2008). Algorithms for finding clique-transversals of graphs. Annals of Operations Research, 157(1), 37–45.

[16] D. Eppstein and E. S. Spiro (2009). The h-index of a graph and its application to dynamic subgraph statistics. In: The 11th Algorithms and Data Structures Symposium (WADS'09). Lecture Notes in Computer Science, 5664, 278–289.

[17] P. Erdös, T. Gallai and Z. Tuza (1992). Covering the cliques of a graph with vertices. Discrete Mathematics, 108, 279–289.

[18] V. Guruswami and C. Pandu Rangan (2000). Algorithmic aspects of clique transversals and clique independent sets. Discrete Applied Mathematics, 100, 183–202.

[19] B. L. Joeris, M. C. Lin, R. M. McConnell, J. P. Spinrad and J. L. Szwarcfiter (2011). Linear-Time Recognition of Helly Circular-Arc Models and Graphs. Algorithmica, 59(2), 215–239.

[20] W. L. Hsu and K. H. Tsai (1991). Linear time algorithms on circular-arc graphs. Information Processing Letters, 40, 123-129.

[21] H. Kaplan and Y. Nussbaum (2006). A simpler linear-time recognition of circular-arc graphs. In: The Tenth Scandinavian Workshop on Algorithm Theory (SWAT'06). Lecture Notes in Computer Science, 4059, 289–300.

[22] T. Kloks, D. Kratsch and H. Müller (2000). Finding and counting small induced subgraphs efficiently. Information Processing Letters, 74(3-4), 115–121.

[23] C. M. Lee, M. S. Chang and S. C. Sheu (2002). The clique transversals and clique independence of distance hereditary graphs. In Proceedings of the 19th Workshop on Combinatorial Mathematics and Computation Theory, Taiwan, 64–69.

[24] M. C. Lin, R. M. McConnell, F. J. Soulignac and J. L. Szwarcfiter (2008). On cliques of Helly Circular-arc Graphs. Electronic Notes in Discrete Mathematics, 30, 117–122.

[25] M. C. Lin and J. L. Szwarcfiter (2006). Characterizations and linear time recognition of Helly circular-arc graphs. In: The 12th Annual International Conference on Computing and Combinatorics (COCOON'06). Lecture Notes in Computer Science, 4112, 73–82.

[26] R. M. McConnell (2003). Linear-time recognition of circular-arc graphs. Algorithmica, 37(2), 93–147.

[27] C. Payan (1979). Remarks on cliques and dominating sets in graphs. Ars Combinatoria, 7, 181–189.

[28] Z. Tuza (1990). Covering all cliques of a graph. Discrete Mathematics, 86, 117–126.

[29] I. E. Zverovich (2003). The domination number of (K_p,P_5)-free graphs. The Australasian Journal of Combinatorics, 27, 95–100.