Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Maestría en Ciencias en Ciencias de la Computación

Algoritmo de enumeración de cliques maximales en grafos k-outerplanares, usando descomposición de árbol

Tesis

para cubrir parcialmente los requisitos necesarios para obtener el grado de Maestro en Ciencias

Presenta:

Francisco González Domínguez

Ensenada, Baja California, México

2021

Tesis defendida por

Francisco González Domínguez

y aprobada por el siguiente Comité

Dr. José Alberto Fernández Zepeda

Codirector de tesis

Dr. Alejandro Flores Lamas

Codirector de tesis

Dr. Edgar Leonel Chávez González

Dr. J. Apolinar Reynoso Hernández

Dr. Israel Marck Martínez Pérez

Coordinador del Posgrado en Ciencias de la Computación

Dr. Pedro Negrete Regagnon

Director de Estudios de Posgrado

Resumen de la tesis que presenta Francisco González Domínguez como requisito parcial para la obtención del grado de Maestro en Ciencias en Ciencias de la Computación.

Algoritmo de enumeración de cliques maximales en grafos k-outerplanares, usando descomposición de árbol

Resumen aprobado por:

Dr. José Alberto Fernández Zepeda	Dr. Alejandro Flores Lamas
Codirector de tesis	Codirector de tesis

Un grafo $G = (V, E)$ es una estructura discreta compuesta por el conjunto de vértices V y el de aristas E. Para un grafo G, un clique es un subgrafo para el cual cada par de vértices está conectado por una arista. Un clique maximal en G es aquel que no está contenido en otro clique más grande en G. El problema de enumeración de cliques consiste en enumerar todos los cliques maximales en G. Este problema pertenece a la clase $\mathcal{NP}$-Difícil cuando G es arbitrario, y no se conocen algoritmos eficientes que lo resuelva. En el presente trabajo de investigación se propone el algoritmo DP-CLIQUE, el cual resuelve el problema anterior cuando el grafo de entrada es k-outerplanar. Un grafo k-outerplanar es un grafo plano, que al quitarle los vértices que tocan la región infinita del plano que rodea al grafo, así como sus aristas incidentes, se genera un grafo $(k-1)$-outerplanar. El algoritmo DP-CLIQUE se auxilia de dos algoritmos del estado del arte. El primer algoritmo calcula la descomposición de árbol de un grafo k-outerplanar. Una descomposición de árbol es una transformación del grafo de entrada a un árbol, en el cual, cada uno de sus nodos tiene atributos especiales. Posteriormente, se utiliza otro algoritmo que genera una descomposición de árbol agradable a partir de la descomposición anterior. La descomposición de árbol agradable tiene restricciones adicionales que facilitan su procesamiento. El tiempo de ejecución del algoritmo DP-CLIQUE es de $O\left((\tau+1)\cdot n\cdot 2^{\tau+1}\right)$ unidades de tiempo, donde n es el número de vértices en el grafo de entrada, y τ la anchura de árbol de G. La anchura de árbol de un grafo G es una medida que indica qué tan cercano es G a un árbol. Este tiempo de ejecución es polinomial en n, si τ es una constante. En particular, τ es una constante cuando k es una constante. Este tiempo de ejecución es equivalente al del mejor algoritmo del estado del arte, cuando k es constante, pero usa una técnica alternativa. Hasta donde el autor de este trabajo tiene conocimiento, el algoritmo DP-CLIQUE es el primero en resolver el problema mencionado a través de la descomposición de árbol. Este algoritmo es el primer paso para resolver este problema en otros grafos en tiempo polinomial en n.

Palabras clave: Algoritmo para grafos, enumeración de cliques maximales, descomposición de árbol, anchura de árbol, grafo outerplanar, grafo k-outerplanar, problema del clique

Abstract of the thesis presented by Francisco González Domínguez as a partial requirement to obtain the Master of Science degree in Computer Sciences.

Maximal clique enumeration algorithm in k-outerplanar graphs, using tree decomposition

Abstract approved by:

Dr. José Alberto Fernández Zepeda
Thesis Co-Director

Dr. Alejandro Flores Lamas
Thesis Co-Director

A graph $G = (V, E)$ is a discrete structure consisting of the vertex set V and the edge set E. For a graph G, a clique is a subgraph for which an edge connects each vertices pair. A maximal clique in G is a clique that is not a subset of a larger clique in G. The maximal clique enumeration problem consists of enumerating all maximal cliques in G. This problem is $\mathcal{NP}$-Hard when G is arbitrary, and there are no known efficient algorithms that solve it. This work proposes the algorithm DP-CLIQUE, which solves the problem above when the input graph is k-outerplanar. A k-outerplanar graph is a plane graph, such that by removing each vertex that touches the infinite region of the plane that surrounds the graph, and its incident edges, generates a $(k-1)$-outerplanar graph. The DP-CLIQUE algorithm employs two algorithms from state of the art. The first algorithm calculates the tree decomposition of a k-outerplanar graph. A tree decomposition is a transformation of the input graph to a tree, in which each of its nodes has unique attributes. Then, the second algorithm generates a nice tree decomposition from the previous decomposition. A nice tree decomposition has additional restrictions that make it easier to process. The algorithm DP-CLIQUE requires $O\left((\tau+1)\cdot n\cdot 2^{\tau+1}\right)$ units of time, where n is the number of vertices in the input graph, and τ the treewidth in G. The treewidth of a graph G is a measure that indicates how close G is to a tree. This execution time is polynomial in n if τ is a constant. In particular, τ is a constant when k is a constant. This execution time is equivalent to that of the best-known state-of-the-art algorithm when k is a constant, but it uses an alternative technique. As far as the author of this work knows, the DP-CLIQUE algorithm is the first one that solves the problem mentioned above through tree decomposition. This algorithm is the first step for solving this problem in other graphs on polynomial time in n.

Keywords: Graph algorithms, maximal clique enumeration, tree decomposition, treewidth, outerplanar graph, k-outerplanar graph, clique problem

Dedicatoria

A mis familiares

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por brindarme el apoyo económico para realizar mis estudios de maestría/doctorado. No. de becario: 733012.

Al Centro de Investigación Científica y de Educación Superior de Ensenada, por darme la oportunidad de iniciar mi posgrado en su institución.

A los profesores y compañeros que tuve el gusto de conocer en el departamento de Ciencias de la Computación.

A mis codirectores de tesis, el Dr. José Alberto Fernández Zepeda y el Dr. Alejandro Flores Lamas, cuya disposición y profesionalismo fueron guías clave en mi formación académica.

A los miembros de mi comité de tesis, el Dr. Édgar Leonel Chávez González y el Dr. J. Apolinar Reynoso Hernández, cuyas sinceras observaciones coadyuvaron en este trabajo de investigación.

Por último pero no menos importante, a mis familiares por su incondicional apoyo.

Tabla de contenido

Página

Tabla de contenido (continuación)

Lista de figuras

Figura Página

Lista de tablas

Tabla Página

Capítulo 1. Introducción

El presente trabajo de investigación se ubica en el área de análisis y diseño de algoritmos para grafos. En particular, se propone un algoritmo que resuelve el problema de enumeración de cliques maximales en un grafo k-outerplanar. A continuación, se presentan un conjunto de definiciones básicas para entender dicho problema.

1.1. Conceptos básicos y preliminares

Un grafo G es un par de conjuntos $G = (V, E)$, tal que V es el conjunto de *vértices*, y $E \subseteq V \times V$, el de *aristas* (Diestel, 2017). Cada *arista*, denotada $(u, v) \in E$ se representa por medio de su par de vértices. La Figura 1a ilustra un grafo de cuatro vértices y seis aristas. Un grafo *dirigido* es un grafo en el que cada una de sus aristas tiene una dirección asociada (Gross *et al.*, 2014); en caso contrario, el grafo es *no dirigido*.

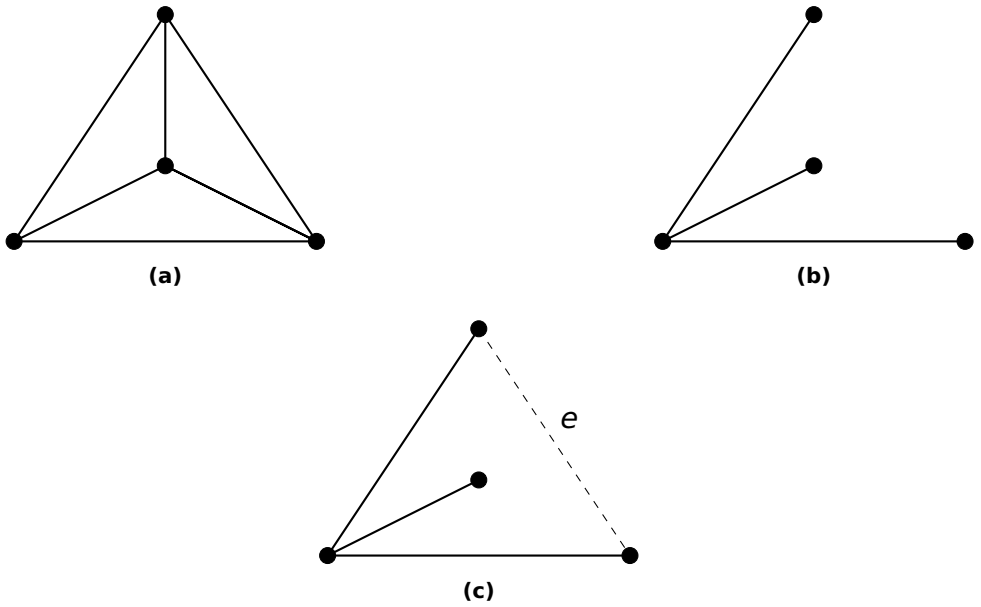


Figura 1. Ejemplo de un grafo, un posible árbol de expansión y su ciclo fundamental. a) Grafo conectado G; b) Árbol de expansión de G; c) La arista e forma un ciclo fundamental.

El grafo G es *simple* si no tiene autociclos (i.e., aristas de la forma (v, v)) ni aristas múltiples o redundantes (Gross *et al.*, 2014). En este trabajo, se utilizan únicamente grafos simples y no dirigidos. Dos vértices u, v de G son *adyacentes*, o vecinos, si (u, v) es una arista en E. Dos aristas distintas son adyacentes si tienen un vértice

en común (Gross *et al.*, 2014) El *orden* de G, representado por $|V|$, es el número de vértices en el grafo (Gross *et al.*, 2014). El conjunto de *vecinos* o *vecindario abierto* de un vértice v en G, denotado por $N(v)$, es el conjunto $U \subseteq V$ de vértices, tal que $u \in U$ y $(v, u) \in E$ (Diestel, 2017). Similarmente, el vecindario de distancia κ para el vértice v, es el subconjunto de vértices en V, tal que cada elemento del subconjunto esté a una distancia a lo más κ de v (Behar y Cohen, 2018). El grado de un vértice v, denotado por $d(v)$, es igual al número de aristas incidentes a v. Esto también es igual al número de vecinos de v, i.e., $d(v) = |N(v)|$. Un vértice de grado cero está aislado (Diestel, 2017). Una caminata es un grafo no vacío $W = (V, E)$ de G, cuya forma es $V = \{v_0, v_1, \ldots, v_m\}$, $E = \{(v_0, v_1), \ldots, (v_{m-1}, v_m)\}$ (Diestel, 2017). La caminata anterior tiene una longitud m. Si en la caminata W todos los vértices son distintos, entonces se llama camino. Un *ciclo* es una caminata de longitud $m \geq 3$, en donde $v_0 = v_m$ (Diestel, 2017). La *distancia* entre dos vértices $u, v \in V$ de G, denotado como $d_G(u, v)$, es la longitud del camino más corto, medido en aristas, que conecta a u con v en G; si ese camino no existe, entonces $d_G(u, v) = \infty$ (Diestel, 2017). A partir de esta definición, el *diámetro* de $G = (V, E)$, denotado por $diam(G[V])$, es la distancia más grande entre cualquiera de los vértices de G (Butenko y Prokopyev, 2007).

Un grafo $G = (V, E)$ es *plano* si tiene un empotramiento en $\mathbb{R}^2$, donde sus aristas únicamente se tocan en sus vértices (Diestel, 2017). En un grafo plano, una *cara* es la región delimitada por aristas. Toda cara tiene frontera con G, y para un G plano, exactamente una de sus caras, la *cara exterior*, es ilimitada (Diestel, 2017). Cualquier otra cara en el grafo, si existe, es una *cara interior*. Un grafo con $|V|$ vértices enumerados se puede representar por medio de una *matriz de adyacencias* $M = (a_{u,v})$ de tamaño $|V| \times |V|$, donde cada elemento $a_{u,v}$ de M es 1 si $(u, v) \in E$ y 0 de otro modo (Cormen *et al.*, 2009).

Dentro de la clase de los grafos planos, existe la subclase 1-*outerplanar* o simplemente *outerplanar* (de ahora en adelante, se utiliza este último término), cuyos vértices tocan la cara exterior. Un grafo outerplanar es *maximal* si al agregar cualquier arista al grafo, éste deja de ser outerplanar. La Figura 2a muestra un ejemplo de un grafo outerplanar; la región delimitada por las aristas $\{(1, 2), (2, 8), (1, 8)\}$ es una cara interior del grafo. De manera generalizada, un grafo *k-outerplanar* es aquel que al remover sus vértices de la cara exterior, así como sus aristas incidentes, resulta en un

grafo $(k-1)$-outerplanar (Katsikarelis, 2013).

Los grafos *Halin* son una subclase de los grafos 2-outerplanar. Un grafo Halin se puede particionar en un árbol de al menos cuatro vértices, sin vértices de grado dos, y un conjunto de aristas que forma un ciclo al conectar las hojas del árbol (Gross *et al.*, 2014). A la unión del conjunto de hojas y al conjunto de aristas que las conecta, se le conoce como *ciclo de hojas*. La Figura 2b muestra un grafo Halin, donde los vértices $\{1, 2, 3, 4, 5\}$ forman el ciclo de hojas.

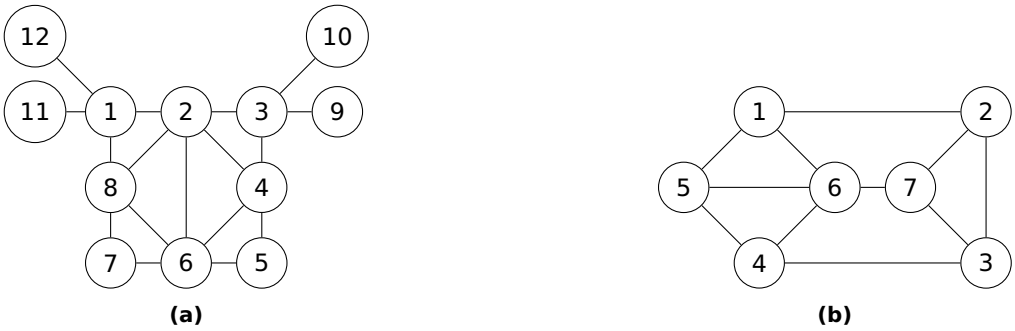


Figura 2. Ejemplos de grafo outerplanar y Halin. a) Grafo outerplanar. La región delimitada por $(1, 2), (2, 8), (1, 8) \in E$ es una cara interna. b) Grafo Halin H; las aristas uniendo a los vértices $\{1, 2, \ldots, 5\}$ forman el ciclo de hojas de H.

En un grafo G, un *subgrafo inducido* en un conjunto de vértices $V' \subseteq V$, denotado por $G[V']$, tiene a V' como el conjunto de vértices y contiene toda arista de G cuyos vértices están en V' (Gross *et al.*, 2014). El grafo G es *conectado* si no está vacío y cualquier par de vértices está vinculado por un camino en G (Diestel, 2017). Un *subgrafo de expansión* $G' = (V', E')$ de G es aquél tal que V' cubre a todo G; i.e., $V' = V$ (Diestel, 2017). Si G es un grafo conectado, su *árbol de expansión* $T = (V', E')$ es un subgrafo de expansión en G que es un árbol (Gross *et al.*, 2014). La Figura 1b ilustra un posible árbol de expansión del grafo de la Figura 1a.

Considere un grafo conectado $G = (V, E)$ con un árbol de expansión $T \subseteq G$. Para cada arista $e \in E \setminus E'$, existe un ciclo fundamental único C_e en $T + e$ con respecto a T (Diestel, 2017). La Figura 1c muestra que la arista e forma un ciclo fundamental al agregarlo al árbol de expansión de la Figura 1b. Un grafo G es *completo* si cada par de vértices de V está conectado por una arista. La Figura 1a muestra un grafo completo de cuatro vértices.

Un *problema computacional de decisión* es un conjunto de entradas, típicamente infinita, junto con una pregunta cuya respuesta es ‘sí’ o ‘no’ para cada entrada (Lewis y Papadimitriou, 1998). Un problema computacional es de *optimización* si su salida es el valor máximo o mínimo entre el universo de soluciones posibles. Esta solución también se conoce como ‘la mejor’ dentro del espacio de soluciones. Además, es posible expresar estos problemas como problemas de decisión (Cormen *et al.*, 2009).

Un *algoritmo* es un procedimiento para un problema computacional que toma un conjunto de valores como entrada y produce un conjunto de valores como salida (Cormen *et al.*, 2009). La *complejidad computacional* o simplemente *complejidad*, es la cantidad de recursos necesarios para la ejecución del algoritmo. Una manera de medir la complejidad de un algoritmo es a través del *tiempo de ejecución*, que es el número de operaciones básicas o “pasos” que el algoritmo tarda en ejecutarse (Cormen *et al.*, 2009). Usualmente, el tiempo de ejecución está representado por una función $f(n)$, donde n es el tamaño de la entrada del problema.

El *modelo computacional* de implementación en el que estos algoritmos corren se divide en determinísticos y no determinísticos. En un modelo *determinístico*, los estados y operaciones sucesivas dependen del estado presente para su ejecución. Ejemplos de modelos deterministicos incluyen el modelo RAM y la máquina de Turing determinística. En un modelo *no determinístico*, estando éste en un cierto estado y para un cierto conjunto de valores fijos en sus variables, el modelo puede tener cero, una, o más transiciones aplicables, y puede elegir cualquiera de ellas. Un ejemplo de este modelo es la máquina de Turing no determinística (Lewis y Papadimitriou, 1998).

Se dice que un problema computacional es *manejable* o eficiente si existe un algoritmo que lo resuelve se ejecuta en *tiempo polinomial*, i.e., su tiempo de ejecución se representa por una función de la forma $O(n^c)$, donde c es una constante positiva y el polinomio está representado por n. De lo contrario, se dice que el problema es *inmanejable*. En la práctica, comúnmente existen problemas de optimización de carácter inmanejable (Cormen *et al.*, 2009).

Un problema computacional P está en la clase $\mathcal{P}$ si el algoritmo que lo resuelve se ejecuta en tiempo polinomial en modelos computacionales deterministicos; i.e, es manejable y $f(n)$ es un polinomio en n. Un problema computacional P está en la clase

$\mathcal{NP}$ si cualquier certificado o solución propuesta para P se puede verificar por un algoritmo en tiempo polinomial. Esta clase de problemas corre en tiempo polinomial en la máquina de Turing no determinística. Note también que los problemas de la clase $\mathcal{P}$ están en $\mathcal{NP}$, ya que éstos pueden verificarse también en tiempo polinomial (Cormen *et al.*, 2009).

Un problema P está en la clase $\mathcal{NP}$*-difícil* si cualquier problema P′ $\in \mathcal{NP}$ se pueda expresar en términos de P bajo un procedimiento u algoritmo polinomial. Dicho procedimiento se conoce como algoritmo de *reducción*. Por otra parte, P está en la clase $\mathcal{NP}$*-completo* si P $\in \mathcal{NP}$ y P $\in \mathcal{NP}$-difícil (Cormen *et al.*, 2009).

Un ligero cambio en las especificaciones de un problema puede cambiar su clase (i.e., pasar de $\mathcal{P}$ a $\mathcal{NP}$-completo o $\mathcal{NP}$-difícil). Por ejemplo, el problema de determinar el camino más corto entre dos vértices arbitrarios de un grafo está en la clase $\mathcal{P}$; mientras que determinar el camino más largo en el mismo grafo es un problema $\mathcal{NP}$-difícil (Cormen *et al.*, 2009).

Muchos problemas prácticos están en la clase $\mathcal{NP}$-difícil y no se conocen algoritmos eficientes para resolverlos, i.e., algoritmos que corran en tiempo polinomial. Por ello, a continuación se describen algunas técnicas para tratar de resolver algunos de estos problemas de manera eficiente.

Una forma sencilla de diseñar algoritmos para resolver problemas de la clase $\mathcal{NP}$-difícil es a través de la técnica de *fuerza bruta*. Esta técnica inspecciona exhaustivamente todas las posibles soluciones al problema y elige la mejor de ellas. En este tipo de problemas, normalmente el espacio de búsqueda es exponencial en función de n (Cormen *et al.*, 2009). Por lo que en la práctica, el uso de esta técnica no es factible, a menos de que n sea pequeña. Un ejemplo de esta técnica se encuentra en el Pseudocódigo 1, en la Sección 2.1; este pseudocódigo determina si existe en el grafo de entrada un clique de tamaño Q.

Los algoritmos de *aproximación* buscan en el espacio de soluciones candidatas, valores ‘cercanos’ al óptimo, para problemas que están en la clase $\mathcal{NP}$-difícil. Este tipo de algoritmos garantiza que la solución propuesta C se acerque a la solución óptima C^* por un factor de a lo más $\rho(n)$. Para problemas de maximización, $\rho(n) \geq \frac{C^*}{C}$, donde $0 < C \leq C^*$; para problemas de minimización, $\rho(n) \geq \frac{C}{C^*}$, donde $0 < C^* \leq C$

(Cormen *et al.*, 2009). Un ejemplo de un algoritmo de aproximación para el problema del *cubrimiento de vértices mínimo* es el propuesto en Hochbaum (1997), cuyo factor de aproximación $\rho(n)$ es 2; esto quiere decir que el valor de salida del algoritmo de aproximación no pasa del doble del valor óptimo. Dado un grafo G, un *cubrimiento de vértices* $V' \subseteq V$, es aquel tal que cada arista $(u, v) \in E$, ya sea u, v, o ambos, están en V'. La versión de optimización de este problema consiste en encontrar el conjunto V' de menor cardinalidad. La versión de decisión del problema del cubrimiento de vértices es $\mathcal{NP}$-completo (Cormen *et al.*, 2009).

Otra técnica utilizada para tratar de resolver problemas de optimización que están en la clase $\mathcal{NP}$-difícil son las *heurísticas*. Estas técnicas no necesariamente manejan un factor de aproximación como los algoritmos de aproximación. Por lo tanto, no se puede saber con certeza qué tan cerca está la solución aproximada de la solución óptima. No obstante, el tiempo de ejecución de estos algoritmos suele ser menor que el de los algoritmos de fuerza bruta. Las heurísticas se utilizan mucho en la práctica, son fáciles de programar y generan soluciones cercanas a la óptima. La base teórica de las heurísticas no siempre proporcionan un análisis del tiempo de ejecución en sus algoritmos. En su lugar, normalmente se realiza una serie de experimentos para obtener un tiempo de ejecución promedio. Los *benchmarks* o comparaciones entre estos algoritmos determinan su desempeño y utilidad (Eiben y Smith, 2015).

Un ejemplo de heurística mencionado en Eiben y Smith (2015) son los *algoritmos genéticos*, los cuales están inspirados en la teoría de la selección natural para obtener respuestas cercanas al óptimo. La *población* es un grupo de respuestas candidatas, cuyos individuos se conocen como *genes*. Cada gen está constituido por una serie de atributos. Cada gen se somete a una mutación con probabilidad p. Una *mutación* es una alteración intencional de algún o algunos atributos en un gen. La calidad de cada gen se mide por medio de una función de aptitud (‘fitness’). Después, los genes de mejor calidad realizan el cruzamiento para generar una nueva población. Típicamente, el *cruzamiento* es el proceso que genera nuevos genes a partir de una combinación de los atributos de los genes padres. A todo el proceso anterior se le llama *generación*, y éste se repite hasta que la calidad de los genes ya no cambie significativamente. El gen de mejor calidad se considera como la respuesta aproximada al problema y se espera que éste sea cercano al óptimo.

Un ejemplo de heurística aplicada al problema de cubrimiento de vértices mínimo se observa en Khuri y Bäck (1994). Este trabajo describe un algoritmo genético que produce respuestas aproximadas para este problema con una calidad mayor a la que produce el algoritmo de aproximación con $\rho(n) = 2$, descrito en (Cormen *et al.*, 2009).

La teoría de la *complejidad parametrizada* investiga cómo diseñar algoritmos de tal forma que su tiempo de ejecución incluya un parámetro adicional e independiente al tamaño de la entrada del problema. Bajo esta teoría, ciertos problemas se pueden reformular de tal manera que su tiempo de ejecución se exprese como el producto de dos funciones. La primera función depende únicamente del parámetro k, que es independiente de la entrada del problema; mientras que la segunda depende del parámetro n, el cual es el tamaño del problema. Existen distintas clases de problemas parametrizados. Una de ellas, descrito en Cygan *et al.* (2015), es la clase de problemas con *parámetro fijo manejable* (fixed parameter tractable, FPT por sus siglas en inglés). El tiempo de ejecución de estos algoritmos consiste de una función computable f y un parámetro k, que en conjunto son de la forma $O\left(f(k) \cdot n^{O(1)}\right)$. Note que $f(k)$ puede ser una función exponencial de k, que para muchos problemas si k se considera constante, el costo computacional es manejable.

Uno de los parámetros considerados en complejidad parametrizada es el tamaño de la solución. Un ejemplo de ello es el problema de cubrimiento de vértices de tamaño a lo más k. Para este problema, existe un algoritmo FPT cuya complejidad es de $O\left(2^k \cdot |E|\right)$ unidades de tiempo en un grafo general (Nisse, 2018).

1.2. Problema de enumeración de cliques maximales

En un grafo simple, no dirigido y finito G, un *clique* es un subconjunto de vértices $V' \subseteq V$ tal que entre cualquier par de vértices u, v de V', existe una arista $(u, v) \in E$ que los conecta (Cormen *et al.*, 2009). El grafo G de n vértices es *completo*, denotado por K_n, si los n vértices forman un clique. Note que el subgrafo inducido por el clique V' es $K_{|V'|}$. En la Figura 3a, se muestran los grafos completos K_1 a K_4.

Un clique V' de G es *maximal* si V' no es un subgrafo de otro clique más grande en G. Un clique maximal es *máximo* si es el de mayor orden en G. El *tamaño del clique máximo* (clique number) de G se denota por $\omega(G)$. En un grafo, el clique máximo

no necesariamente es único (Butenko y Prokopyev, 2007). En el grafo de la Figura 3b se tienen dos conjuntos cliques maximales, éstos son $\{1, 2, 6\}$ y $\{3, 4, 5, 7\}$. En particular, el conjunto $\{3, 4, 5, 7\}$ es el único clique máximo del grafo.

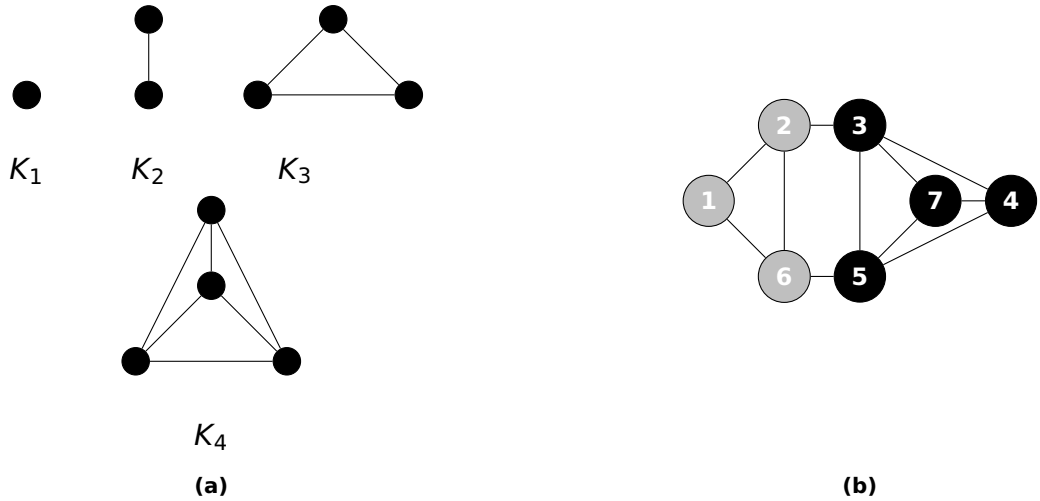


Figura 3. Ejemplos de cliques y conjuntos cliques maximales. a) Grafos planos K_1, K_2, K_3, K_4; b) Grafo con cliques maximales $\{1, 2, 6\}$ y $\{3, 4, 5, 7\}$, siendo este último el clique máximo en el grafo.

El *problema de optimización del clique* toma como entrada un grafo G y su salida es el clique máximo en G. La versión de decisión de este problema pregunta si existe un clique de al menos Q vértices en G. Este problema es $\mathcal{NP}$-completo en su versión de decisión y es $\mathcal{NP}$-difícil en la de optimización (Karp, 1972). Por otra parte, el *problema de enumeración de cliques maximales consiste en identificar todos los cliques maximales en G. Este problema es una generalización del problema de optimización del clique, y también es $\mathcal{NP}$-difícil.*

El tamaño del clique máximo en un grafo outerplanar es tres. Dicho grafo no puede tener un K_4 como subgrafo, ya que el empotramiento plano del grafo K_4 tiene un vértice que no toca la cara externa. Además, no existen empotramientos planos para el grafo K_n, con $n > 4$.

Un grafo no dirigido G es *s-degenerado*, si cada subgrafo no vacío $G' \subseteq G$ tiene al menos un vértice v con grado a lo más s. La degeneración de un grafo G, denotada por d, es el valor s más pequeño tal que G es s-degenerado. Para un grafo con degeneración d existe un *ordenamiento degenerado*. Una forma de obtener este ordenamiento

es redibujando al grafo G, de tal manera que los vértices se dibujen sobre el eje horizontal, y para cada vértice existan a lo más d vecinos a su derecha. El ordenamiento degenerado consiste de la secuencia de vértices, de izquierda a derecha, en el grafo redibujado (Eppstein *et al.*, 2010). Un algoritmo que calcula el ordenamiento degenerado para un grafo outerplanar es el mostrado por el Pseudocódigo 6, en el Capítulo 3. Para un grafo con degeneración d, su clique máximo no tiene una cardinalidad mayor a $d+1$. En un grafo outerplanar, cuya degeneración es dos, su clique máximo es tres. Más información sobre el problema del clique se discute en la Sección 2.1 del Capítulo 2.

1.2.1. Clique de distancia κ

A continuación se presenta una variante del problema de optimización del clique, propuesta por Luce (1950). Dado un grafo no dirigido $G = (V, E)$, un clique máximo V' de *distancia* κ (adaptado del término en inglés de *k-clique* Butenko y Prokopyev (2007)), para una constante entera κ, es un subconjunto de máxima cardinalidad de V, tal que para cualquier par de vértices $u, v \in V'$, la distancia entre ellos es a lo más κ. Note que el problema del clique máximo de distancia uno es igual al problema de optimización del clique. En la Figura 4a se muestra un grafo outerplanar de seis vértices. Las Figuras 4b y 4c muestran ejemplos de cliques máximos de distancia dos, mostrados en negro.

1.3. Otras variantes del problema del clique máximo de distancia κ

Un problema similar al clique máximo de distancia κ es el problema del κ-clan máximo. Dado un grafo no dirigido $G = (V, E)$, un κ-*clan* máximo V' es un subconjunto de vértices de máxima cardinalidad de V, tal que $diam(G[V']) = \kappa$, y además V' forma un clique maximal de distancia κ en G (Butenko y Prokopyev, 2007). Una forma de encontrar el κ-clan máximo en un grafo G es el siguiente. Primero, encontrar todos los cliques maximales de distancia κ en G. Segundo, determinar cuál de ellos forma un κ-clan. Tercero, de éstos últimos, seleccionar el de máxima cardinalidad.

Una variante del κ-clan es el κ-club. Dado un grafo no dirigido $G = (V, E)$, un κ-*club* V' es un subconjunto de vértices de V, tal que $diam(G[V']) = \kappa$, pero V' no

necesariamente forma un clique maximal de distancia κ en G (Butenko y Prokopyev, 2007). Note que cualquier κ-clan es un κ-club, pero no necesariamente lo contrario. Similarmente, un κ-clan es un clique de distancia κ maximal, pero no necesariamente lo contrario.

En la Figura 4c, el conjunto S_2 forma un 2-clan máximo. En la Figura 4d, el conjunto S_1 forma un clique de distancia 2 maximal en G, pero no es un 2-clan. Las Figuras 4e a 4g muestran algunos conjuntos 2-club de G.

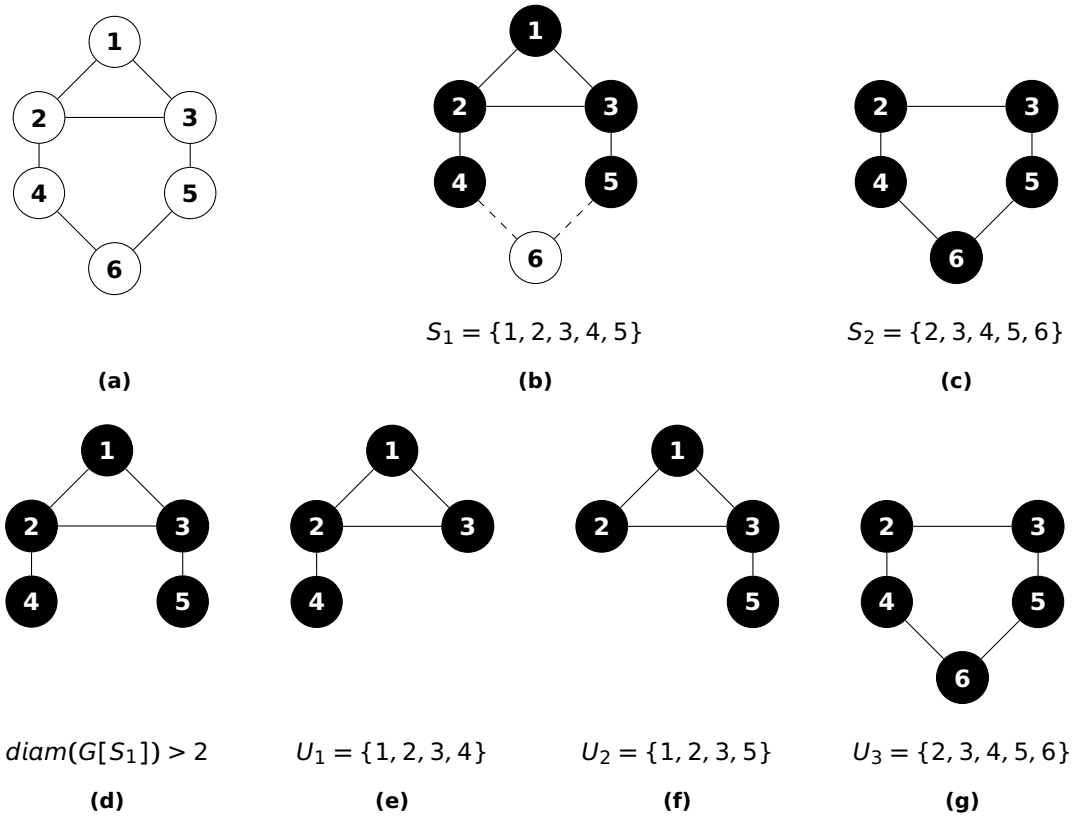


Figura 4. Ejemplos de cliques y variantes del clique. Los vértices de color negro forman estos conjuntos. **a)** Grafo outerplanar de entrada G. **b)** El conjunto S_1 forma un clique máximo de distancia 2; note que $d(4, 5) \leq 2$ porque utiliza el camino $4, (4, 6), 6, (6, 5), 5$. **c)** El conjunto S_2 forma un clique máximo de distancia 2. Este conjunto también forma un 2-clan máximo, porque el grafo inducido por S_2 tiene un diámetro igual a dos. **d)** En G, S_1 no es un 2-clan, porque el diametro de $G[S_1]$ es mayor que dos. **e)-g)** Los conjuntos U_1, U_2 y U_3 son 2-club porque el diámetro de los grafos $G[U_1]$, $G[U_2]$ y $G[U_3]$ es igual a dos, pero no necesariamente son cliques maximales en G.

1.4. Descomposición de árbol y sus variantes

Además del tamaño de la solución, en la complejidad parametrizada existen parámetros que representan la estructura de la entrada del problema, como en el caso de los grafos. A continuación se describen las descomposiciones de camino y de árbol en un grafo, así como sus variantes agradables, según lo descrito por Cygan *et al.* (2015).

Sea G un grafo simple, no dirigido y conectado. Una *descomposición de árbol* del grafo G es un par $\mathcal{T} = (T, X)$. Donde $T = (I, F)$ es un árbol con un conjunto de nodos I y un conjunto de aristas F. El conjunto X es una colección de *bolsas*, donde cada bolsa es un subconjunto de vértices del grafo de entrada. Además, cada bolsa $X_i \in X$ está asociada al nodo $i \in I$. La bolsa X_i se puede ver como información satelital del nodo i.

Nota. En este documento, para cualquier grafo de entrada $G = (V, E)$, los elementos de V se llaman *vértices*. En la transformación de árbol y sus variantes, los elementos del conjunto I se llaman *nodos*.

Cualquier descomposición de árbol tiene las siguientes propiedades:

1. Cada $v \in V$ debe estar contenido en al menos una bolsa X_i.
2. Para cada arista (u, v) de E, existe al menos una bolsa X_i que incluye a los vértices u y v.
3. Para cada $v \in V$, el conjunto de nodos $\{i \in I : v \in X_i\}$ induce un subárbol conectado en T, llamado T_v.

La Figura 5a muestra un grafo outerplanar maximal G, donde cada círculo es un vértice, y la letra circunscrita su identificador. La Figura 5b muestra una posible descomposición de árbol de G. Cada círculo es un nodo, y cada uno de ellos tiene asociada una bolsa con un conjunto de vértices de G. Note que los vértices de cada arista del grafo G están contenidos en la misma bolsa de al menos algún nodo en el árbol. En el Capítulo 3 se describe el algoritmo de Katsikarelis (2013) con el cual se puede generar dicha descomposición.

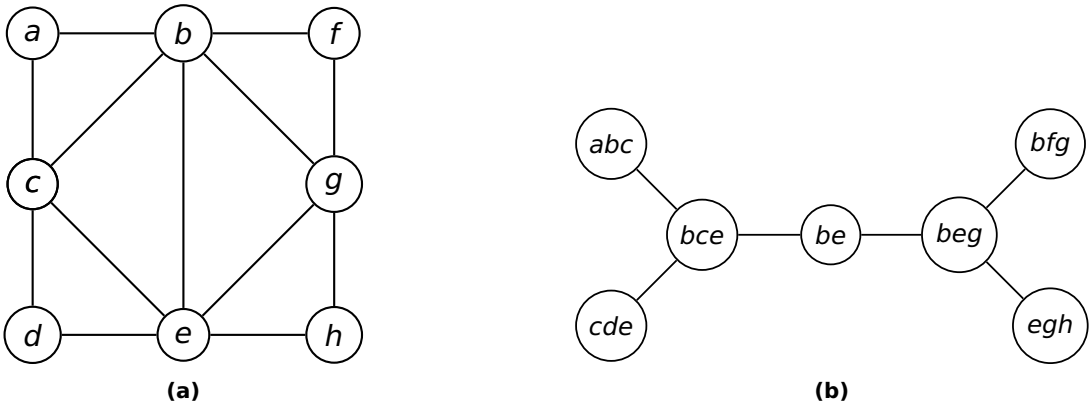


Figura 5. Ejemplo de descomposición de árbol a partir de un grafo outerplanar. a) Grafo outerplanar de ocho vértices; b) Una posible descomposición de árbol de siete nodos del grafo outerplanar. Las letras dentro de cada nodo representan el contenido de su bolsa.

Una *descomposición de árbol agradable* $\mathcal{T}'$ satisface las condiciones de una descomposición de árbol y, además, las que se enumeran a continuación:

1. El árbol T de $\mathcal{T}'$ está enraizado en un nodo r de grado uno, llamado raíz.
2. Las bolsas asociadas a cualquier nodo hoja y al nodo raíz están vacías.
3. Cualquier nodo que no sea hoja ni raíz es del tipo introductor, eliminador o unión.
4. Los nodos introductores y eliminadores tienen un solo descendiente. Un nodo unión t tiene dos nodos descendientes, denominados t_1 y t_2. Los vértices en las bolsas X_t, X_{t_1} y X_{t_2} son los mismos.
5. Para el nodo introductor p y su descendiente c, $X_p = X_c \cup v$, donde v es el vértice que se introduce en la bolsa del nodo p. Similarmente, para el nodo eliminador p y su descendiente c, $X_p = X_c \setminus v$, donde v es el vértice que se elimina de la bolsa del nodo c.
6. El grado de cada nodo en el árbol T es a lo más tres.

La Figura 6 muestra una posible descomposición de árbol agradable del árbol de la Figura 5b. Los tres nodos blancos de grado uno son las hojas. El único nodo negro de grado uno es la raíz. Cada nodo negro de grado dos es eliminador. Cada nodo gris de grado dos es introductor. Finalmente, cada nodo blanco de grado tres es unión.

Note que la diferencia en la cardinalidad de las bolsas de dos nodos vecinos o adyacentes es a lo más uno. En el Capítulo 3 se describe un algoritmo, mostrado en el Pseudocódigo 7, con el cual se transforma una descomposición de árbol arbitraria a una agradable.

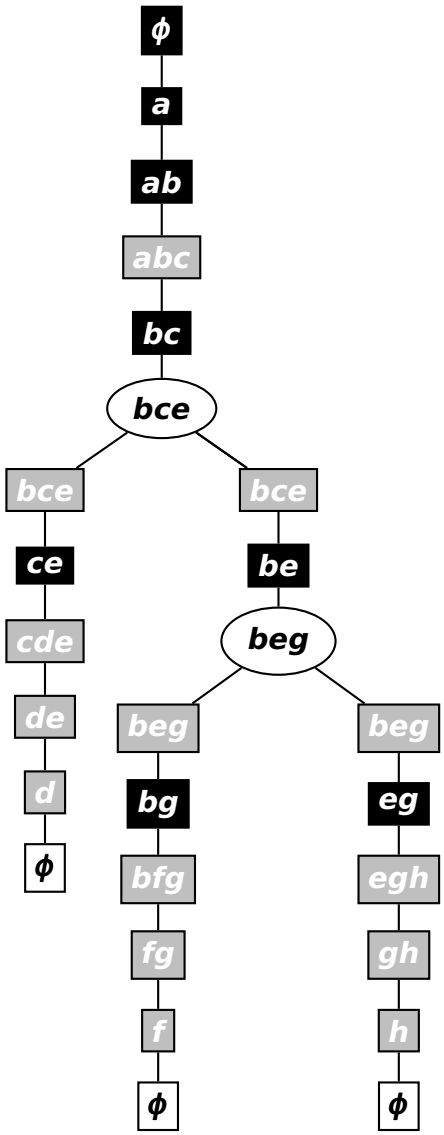


Figura 6. Una posible descomposición de árbol agradable del árbol de la Figura 5b. Los nodos blancos de grado uno son las hojas. El nodo negro de grado uno es la raíz. Cualquier otro nodo en negro, gris o blanco es eliminador, introductor o unión, respectivamente.

Dada una descomposición de árbol $\mathcal{T}$, la *anchura de la descomposición de árbol* de $\mathcal{T}$ es la cardinalidad de su bolsa más grande menos uno. La *anchura de árbol* (*tree-width*) de un grafo G, denotada por τ o τ_G, es la anchura mínima de todas las posibles anchuras de las descomposiciones de árbol en G. Pese a que encontrar la anchura de árbol en un grafo arbitrario es un problema $\mathcal{NP}$-difícil (Cygan *et al.*, 2015), existen

algunos algoritmos polinomiales que obtienen la descomposición de árbol para ciertas clases de grafos. Un ejemplo de este tipo de grafos es el outerplanar, cuya descomposición de árbol se obtiene a través del algoritmo de Katsikarelis (2013), el cual se describe en el Capítulo 3. La anchura de árbol en el árbol de la Figura 5b es dos. Su descomposición de árbol agradable sigue manteniendo la misma anchura de árbol, como se puede ver en la Figura 6.

Una descomposición de camino $\mathcal{P}$ es un caso especial de la descomposición de árbol. Para este tipo de descomposición, el grafo T es una cadena lineal de nodos. En la descomposición de camino también se cumplen las condiciones de una descomposición de árbol. Similar a la anchura de árbol, una *anchura de camino* de $\mathcal{P}$ es la cardinalidad de la bolsa más grande menos uno. La *anchura de camino* (*pathwidth*), denotado por $pw(G)$, es la anchura mínima de todas las descomposiciones de camino posibles de G. Además, una descomposición de camino $\mathcal{P}'$ es *agradable* si cumple las mismas propiedades de una descomposición de árbol agradable. En $\mathcal{P}'$ no existen nodos unión, y el grado máximo de cualquier nodo es dos (Cygan *et al.*, 2015).

De manera intuitiva, la anchura de árbol τ de G indica qué tan similar es el grafo G a un árbol. Entre más pequeño sea este valor, más cercano es G al árbol. Si G es un árbol, su anchura de árbol es uno. Similarmente, la anchura de camino $pw(G)$ indica qué tan similar es el grafo G a una cadena lineal de vértices. Si G es una cadena lineal de vértices, entonces $pw(G) = 1$.

1.5. Contribución

La contribución de esta tesis consta del algoritmo DP-CLIQUE, el cual utiliza la técnica de programación dinámica. Este algoritmo resuelve el problema de optimización del clique para un grafo k-outerplanar en $O\left((\tau+1)\cdot n\cdot 2^{\tau+1}\right)$ unidades de tiempo. En este trabajo se compara el algoritmo propuesto contra los algoritmos del estado del arte que resuelven el mismo problema, y discute los parámetros que cada uno utiliza. El algoritmo propuesto se auxilia de la técnica de descomposición de árbol, ya que la entrada para el algoritmo DP-CLIQUE es la transformación de árbol agradable del grafo de entrada. Se puede observar que el tiempo de ejecución de este algoritmo es polinomial, si τ se considera constante.

1.6. Organización de la tesis

El presente documento de tesis se organiza de la siguiente manera. El Capítulo 2 presenta el estado del arte del problema de enumeración de cliques maximales, y enumeración de cliques maximales de distancia κ. Para estos problemas, se discute el algoritmo de Bron y Kerbosch (1973), que es básico para resolver estos problemas. El Capítulo 3 explica la descomposición de árbol y su variante agradable. En particular, describe el algoritmo de Katsikarelis (2013), que obtiene la descomposición de árbol de un grafo k-outerplanar. También presenta un algoritmo, diseñado por el autor de este documento, que obtiene una descomposición de árbol agradable a partir de una descomposición de árbol arbitraria. El Capítulo 4 describe el algoritmo propuesto DP-Clique, la principal contribución de este trabajo de investigación. También se demuestra que este algoritmo es correcto y se hace el análisis de su tiempo de ejecución. Finalmente, el Capítulo 5 presenta las conclusiones de este trabajo de investigación, diferenciándolo de los trabajos del estado del arte. También se proponen algunas líneas de investigación relacionadas con el problema estudiado.

Capítulo 2. Enumeración de cliques maximales

En este capítulo se describe el problema de enumeración de cliques maximales y una variante denominada enumeración cliques maximales de distancia $\boldsymbol{\kappa}$. Para cada problema, se describen algunos algoritmos en la literatura que los resuelven. Primero, se presenta un algoritmo de fuerza bruta, propuesto por el autor de este documento, que encuentra si existe o no, un clique de cardinalidad Q en un grafo arbitrario.

2.1. Algoritmo de fuerza bruta para el problema de clique

Una forma de determinar si existe un clique de tamaño Q en un grafo arbitrario, consiste en enlistar todos los subconjuntos de vértices de tamaño Q y revisar si cada uno forma un clique. El tiempo de ejecución de este algoritmo es $O\left(Q^2\binom{n}{Q}\right)$ u.t., donde el factor $\binom{n}{Q}$ representa el número de distintas formas en que se pueden elegir Q vértices de un conjunto de tamaño n; el factor Q^2 es el tiempo requerido para verificar que los Q vértices forman un clique. El tiempo de ejecución de este algoritmo es polinomial si Q es una constante, i.e., $O\left(n^Q\right)$ u.t. (Cormen *et al.*, 2009).

El algoritmo recursivo SIZE-Q-CLIQUE, mostrado en el Pseudocódigo 1 y propuesto por el autor de este trabajo de tesis, determina por fuerza bruta si existe en el grafo de entrada G al menos un clique de tamaño Q. La lista E almacena las aristas en G. La lista *chain* almacena una secuencia de vértices a evaluar para determinar si existe un clique de tamaño Q; la longitud máxima de la lista es Q. La lista R almacena los vértices disponibles para insertarlos en *chain*. La variable booleana $found$ indica si se encontró un clique de tamaño Q cuando es verdadera. La lista *clique* almacena el último clique de tamaño Q encontrado, si existe. La función CHECK determina si la cadena *chain* es un clique; si es así, esta función es verdadera.

Inicialmente, $R = V$, $clique = nil$, $found = false$ y $chain = nil$. La instrucción *if* de las Líneas $1-7$ del Pseudocódigo 1 verifica si se han insertado Q elementos a *chain*, para que ésta la pueda inspeccionar la función CHECK. En el caso que *chain* forme un clique, el Pseudocódigo regresa la lista *clique* y el booleano $found$ como verdadero. De otro modo, las Líneas $8-14$ del Pseudocódigo modifican los valores en *chain* y R, llamando recursivamente a SIZE-Q-CLIQUE, para continuar con la verificación de otra cadena distinta. Su tiempo de ejecución es de $O\left(Q^2 \cdot \binom{n}{Q}\right)$ u.t.

Pseudocódigo 1: SIZE-Q-CLIQUE(Q, $G = (V, E)$, R, $chain$, $found$, $clique$)

```
   Entrada: Un grafo simple, no dirigido y conectado G (representado por su
            matriz de adyacencias M) y un entero Q.
   Salida: Si existe un clique de tamaño Q, éste se almacena en la variable
           clique; en caso contrario, clique es un conjunto vacío.
 1 if Q > R.size ∨ k = 0 then
 2 |   if k = 0 then
 3 |   |   if CHECK(chain, G = (V, E))= true then
 4 |   |   |   found ← true;
 5 |   |   |   clique ← chain;
 6 |   |   end
 7 |   end
 8 else
 9 |   x ← POP(R);
10 |   PUSH(chain, x);
11 |   SIZE-K-CLIQUE(Q − 1, G = (V, E), R, chain, found, clique);
12 |   x ← POP(chain);
13 |   SIZE-K-CLIQUE(Q, G = (V, E), R, chain, found, clique);
14 end
15 return found, clique;
```

2.2. Algoritmo de Bron y Kerbosch

Otro algoritmo de carácter recursivo que resuelve el problema de enumeración de cliques maximales es el propuesto por Bron y Kerbosch (1973). Una implementación de este algoritmo se muestra en el Pseudocódigo 2. Dicho pseudocódigo se tomó de Eppstein *et al.* (2010).

Pseudocódigo 2: BRON-KERBOSCH($G = (V, E), R, P, X$)

```
   Entrada: Un grafo G = (V, E) no dirigido.
   Salida: Todos los cliques maximales en G.
 1 if P = X = ∅ then
 2 |   reportar R como un conjunto clique maximal;
 3 end
 4 foreach v ∈ P do
 5 |   BRON-KERBOSCH(G = (V, E), R ∪ {v}, P ∩ N(v), X ∩ N(v));
 6 |   P = P \ {v};
 7 |   X = X ∪ {v};
 8 end
```

El algoritmo de Bron y Kerbosch (1973), mostrado en el Pseudocódigo 2, es un algoritmo recursivo, comúnmente llamado de *backtracking*. Su entrada es el conjunto P, que inicialmente contiene el conjunto de vértices V del grafo de entrada. Los con-

juntos R y X inicialmente son vacíos. Un ejemplo de ejecución de este algoritmo se muestra en el árbol derecho de la Figura 8. Las Líneas $1-3$ del pseudocódigo determinan cuándo el algoritmo encontró un clique maximal. Estos cliques maximales se pueden observar en las hojas 1 y 4, de izquierda a derecha, resaltados en negro. El algoritmo realiza un backtracking para poder revisar en forma exhaustiva todos los posibles cliques maximales del grafo. En general, el conjunto P consiste de los vértices potenciales conectados a R, y el conjunto X consiste de los vértices ya procesados al realizar el backtracking. En cada llamada al algoritmo, P y X son conjuntos disjuntos cuya unión consiste de los vértices que forman un clique cuando se añaden a R. Además, el objetivo del conjunto X es evitar que se repita el reporte de algún clique maximal durante la ejecución del algoritmo. El algoritmo posee la invariante de que los vértices en R forman un clique. Desafortunadamente, los autores de este algoritmo no realizan el análisis del tiempo de ejecución, pero se supone que éste es exponencial, por el resultado de Moon y Moser (1965), que establece que el número de cliques en un grafo arbitrario puede llegar a ser exponencial. El tiempo de ejecución de este algoritmo es $O(3^{n/3})$ u.t. Adicionalmente, dicho algoritmo mejora el tiempo de ejecución del algoritmo de Bierstone, que en algún momento se consideró el algoritmo del estado del arte para la enumeración de cliques maximales en un grafo.

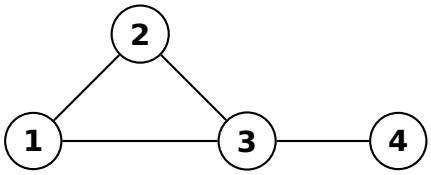


Figura 7. Grafo utilizado para ejemplificar de enumeración de cliques maximales.

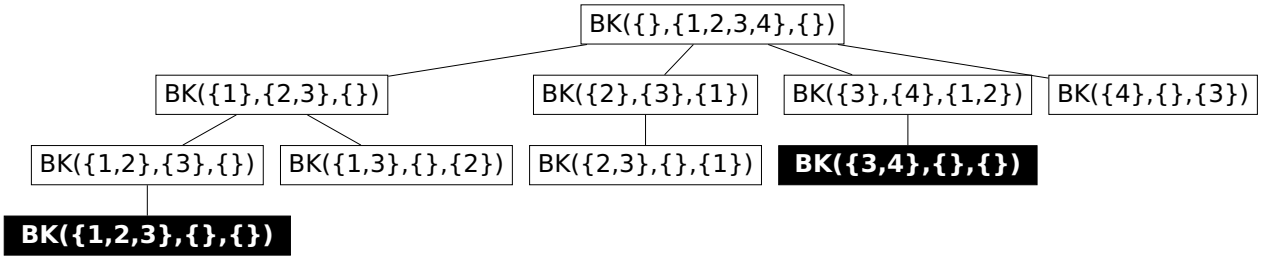


Figura 8. Un posible árbol de recursividad generado al utilizar el algoritmo de Bron y Kerbosch (1973) para el grafo de la Figura 7. Los hojas en negro reportan los cliques maximales.

Tomita *et al.* (2006) diseñaron una variante del algoritmo anterior, llamado BRON-KERBOSCH-PIVOT, mostrado en el Pseudocódigo 3. Dicho algoritmo selecciona al vértice que posee al vecindario de mayor tamaño en el conjunto en cuestión (ver Línea 4 del

Pseudocódigo 3).

Si la entrada al algoritmo es un grafo arbitrario no dirigido, G, su tiempo de ejecución es $O(3^{n/3})$ u.t., donde n es el número de vértices en G. El trabajo de Moon y Moser (1965) determina que el máximo número de cliques en G es $3^{n/3}$, por lo que el Algoritmo 3 resulta óptimo.

Pseudocódigo 3: BRON-KERBOSCH-PIVOT($G = (V, E), R, P, X$)

Entrada: Un grafo $G = (V, E)$ no dirigido.
Salida: Enumeración de todos los conjuntos cliques maximales en G.

```
1 if P = X = ∅ then
2 |   reportar R como un conjunto clique maximal;
3 end
4 Escoger un pivote u ∈ P ∪ X, tal que u tenga el mayor vecindario en P;
5 foreach v ∈ P \ N(u) do
6 |   BRON-KERBOSCH-PIVOT(G = (V, E), R ∪ {v}, P ∩ N(v), X ∩ N(v));
7 |   P = P \ {v};
8 |   X = X ∪ {v};
9 end
```

Eppstein *et al.* (2010) toman como base el algoritmo de Tomita *et al.* (2006), y le agregan el ordenamiento degenerado de los vértices. Este algoritmo, mostrado en el Pseudocódigo 4, se utiliza para enumerar los cliques maximales en G. Con esta adición, ellos obtienen un algoritmo FPT, donde el parámetro adicional es la degeneración del grafo, denotada por d. Este algoritmo requiere $O\left(d \cdot n \cdot 3^{d/3}\right)$ u.t.

Pseudocódigo 4: BRON-KERBOSCH-DEGENERACY($G = (V, E)$)

Entrada: Un grafo $G = (V, E)$ no dirigido.
Salida: Enumeración de todos los conjuntos cliques maximales en G.

```
1 foreach v_i en un orden degenerado v_0, v_1, ... v_n de (V, E) do
2 |   P ← N(v_i) ∩ {v_{i+1}, ..., v_{n-1}};
3 |   X ← N(v_i) ∩ {v_0, ..., v_{i-1}};
4 |   BRON-KERBOSCH-PIVOT(G = (V, E), {v_i}, P, X);
5 end
```

Wood (2007) y Eppstein y Strash (2011) mencionan que el número total de cliques, no necesariamente maximales, en un grafo con degeneración dos es lineal respecto al tamaño de sus vértices o aristas. Para calcular la degeneración de un grafo G, los algoritmos de (Matula y Beck, 1983) y (Batagelj y Zaversnik, 2011) lo encuentran en $O(|V| + |E|)$ u.t., con base en un ordenamiento degenerado. Otro ejemplo de un algoritmo que obtiene la degeneración de un grafo outerplanar, es el mostrado en el

Pseudocódigo 6, y propuesto por el autor de este documento. Dicho algoritmo corre en tiempo lineal en el número de vértices en el grafo.

La Tabla 1 resume los trabajos más relevantes relacionados con los problemas de enumerar todos los cliques maximales y máximos en un grafo. En particular, los trabajos de Tomita y Kameda (2007), Tomita *et al.* (2017) y Konc y Janezic (2007) encuentran el clique máximo sin necesidad de enumerar todos los cliques maximales.

Nota. A partir de la Sección 2.3, el problema de optimización del clique se renombra como el problema del optimización del clique de distancia 1. Dado que este problema es un caso especial del problema de optimización del clique de distancia $\boldsymbol{\kappa}$.

Tabla 1. Estado del arte de los algoritmos que enumeran los cliques maximales y máximos.

Autor	Problema	Tiempo de ejecución	Técnica	Comentarios
Bron y Kerbosch (1973)	Enumeración de cliques maximales	Exponencial	Branch & Bound (BnB)	
Tomita *et al.* (2006)	Enumeración de cliques maximales	Exponencial		Alg. de Bron-Kerbosch con estrategia de pivote.
Tomita y Kameda (2007)	Enumeración de cliques maximales	Exponencial	Coloreado aproximado + *pruning*	Efectivo en grafos dispersos bajo experimentación
Konc y Janezic (2007)	Clique máximo	Exponencial	Coloreado aproximado que toma tiempo polinomial	$O(n^2)$ bajo aproximación
Eppstein y Strash (2011)	Enumeración de cliques maximales	$O\left(d \cdot n \cdot 3^{d/3}\right)$	Degeneración del grafo	Para un grafo arbitrario, $d \leq n$; para una subclase de grafos como el outerplanar, d es un valor fijo
San Segundo *et al.* (2016)	Encontrar clique máximo	Exponencial	*BnB, bit parallelism*	Formulación matemática del problema

2.3. Enumeración de cliques maximales de distancia κ

Por sus posibles aplicaciones en la práctica, algunos investigadores (Pattillo *et al.*, 2013; Behar y Cohen, 2018) han optado por 'relajar' la propiedad de distancia entre los elementos conectados que constituyen al *clique*, por un valor entero $\boldsymbol{\kappa} > 1$.

Behar y Cohen (2018) propusieron una variante del algoritmo de Bron y Kerbosch (1973) para enumerar todos los cliques maximales de distancia $\boldsymbol{\kappa}$, a través del vecindario de distancia $\boldsymbol{\kappa}$ del vértice a procesar. Este algoritmo, llamado CkCliques, se

muestra en el Pseudocódigo 5.

Pseudocódigo 5: CkCliques($G = (V, E), R, P, X, \kappa$)

Entrada: Un grafo $G = (V, E)$ no dirigido.
Salida: Enumeración de todos los conjuntos cliques maximales de distancia κ en G.

```
1 if P = X = ∅ then
2 |  reportar R como un conjunto clique maximal;
3 end
4 foreach v ∈ P do
5 |  CkCliques(G = (V, E), R ∪ {v}, P ∩ N^k(v), X ∩ N^k(v), k);
6 |  P = P \ {v};
7 |  X = X ∪ {v};
8 end
```

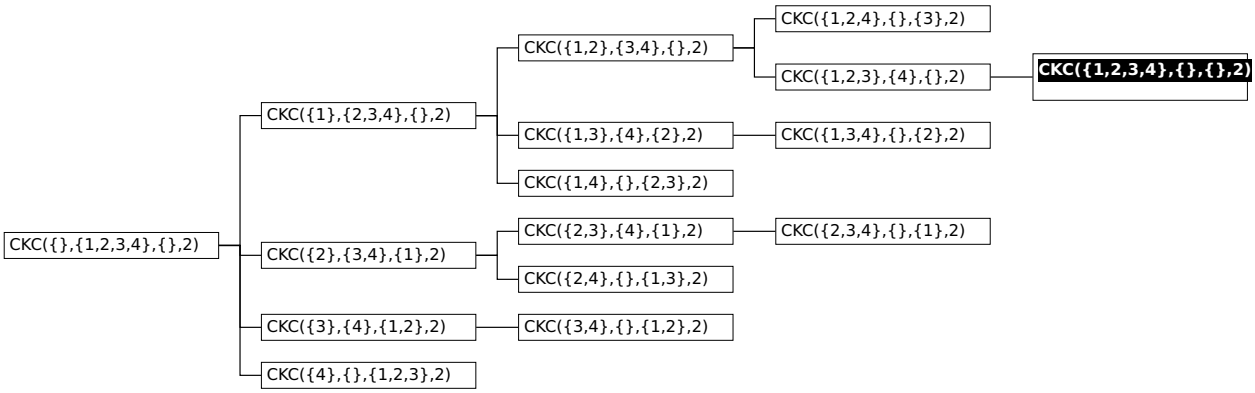


Figura 9. Ejemplo de enumeración de cliques maximales de distancia κ con el algoritmo CkCliques, con $\kappa = 2$, utilizando el grafo de la Figura 7 como entrada. La hoja en negro muestra el único clique maximal de distancia dos.

El Algoritmo CkCliques es recursivo, al igual que el de Bron-Kerbosch (Pseudocódigo 2). Su entrada es el conjunto P, que inicialmente contiene el conjunto de vértices V del grafo de entrada. Los conjuntos R y X inicialmente están vacíos. Un ejemplo de ejecución de este algoritmo se muestra en la Figura 9. En el árbol inferior de esta figura, cada vértice representa una ejecución del algoritmo con ciertos parámetros. Las Líneas $1-3$ del pseudocódigo determinan cuándo el algoritmo encontró un clique maximal de distancia κ, con $\kappa = 2$. El clique maximal de distancia κ se puede observar en la hoja resaltada en negro. Este algoritmo realiza un backtracking para poder revisar en forma exhaustiva todos los posibles cliques maximales de distancia κ del grafo. En general, los conjuntos P, R y X tienen la misma función que en el algoritmo de Bron-Kerbosch.

Adicionalmente, en tiempo polinomial, los autores de dicho algoritmo encuentran otro clique maximal de distancia κ, y así sucesivamente. Como el número de cliques maximales de distancia κ en G puede ser exponencial, el algoritmo para encontrar todos ellos también corre en tiempo exponencial.

Capítulo 3. Descomposición de árbol

En este capítulo se describe el algoritmo de Katsikarelis (2013), que encuentra una descomposición de árbol en un grafo k-outerplanar. También se propone un algoritmo que, dada una descomposición de árbol arbitraria, obtiene una descomposición de árbol agradable.

3.1. Algoritmo de Katsikarelis

El algoritmo de Katsikarelis (2013) toma como entrada un grafo k-outerplanar no dirigido y obtiene su descomposición de árbol. La anchura de árbol de dicha descomposición es a lo más $3k-1$. El tiempo de ejecución de este algoritmo en $O(kn)$ u.t. A continuación se explican los procedimientos para los grafos k-outerplanares, cuando $k = 1$ y $k > 1$, respectivamente.

3.1.1. El algoritmo de Katsikarelis para $k = 1$

Para un grafo outerplanar $G = (V, E)$, Katsikarelis (2013) propuso un procedimiento para encontrar su descomposición de árbol. Para un grafo de este tipo, el orden degenerado siempre es igual o menor que dos. Katsikarelis no propone en forma explícita un algoritmo para encontrar la descomposición de árbol de G; sin embargo, él demuestra, por medio de inducción, que esta descomposición se puede construir de forma recursiva. Grosso modo, su algoritmo primero tendría que calcular un ordenamiento degenerado de G. Después, siguiendo este ordenamiento, su algoritmo eliminaría vértices del grafo, y al mismo tiempo, crearía los nodos de la transformación de árbol con sus bolsas correspondientes. En particular, para un vértice v de grado dos, se eliminarían v y sus aristas incidentes (v, u) y (v, w). En este punto, su algoritmo añadiría una arista entre los vértices u y w, si ésta no estuviera presente. Este procedimiento crearía un nodo de la descomposición de árbol con bolsa $\{u, v, w\}$, y además estaría conectado al nodo que tuviera que en su bolsa al par $\{u, w\}$. En el caso en que el vértice v tuviera grado uno, su eliminarían a v y a su arista incidente, (v, u), crearía un nodo con bolsa $\{v, u\}$, y estaría conectado a un nodo cuya bolsa tuviera a $\{u\}$. Este procedimiento continuaría vértice por vértice hasta llegar a un clique de tamaño

dos. En este punto, se constituiría el último nodo de la descomposición, con una bolsa que tendría los vértices de dicho clique.

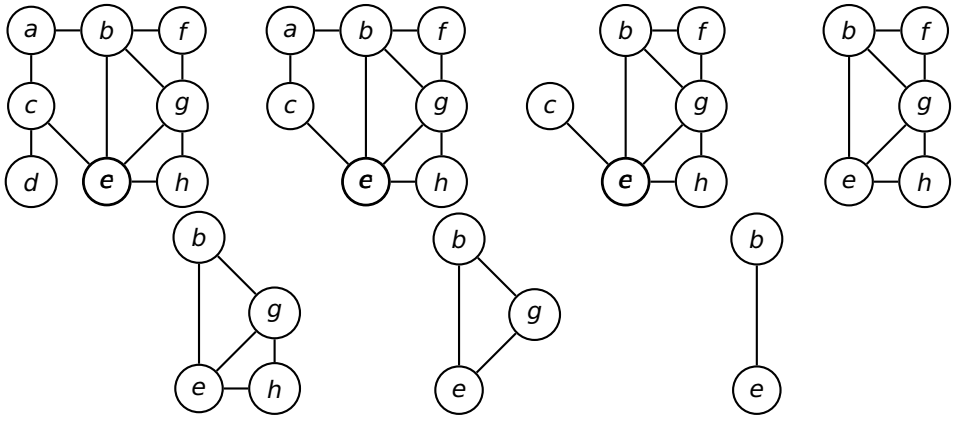


Figura 10. Ejemplo de reducción degenerada de un grafo de un grafo outerplanar, usando el algoritmo DEGENERATE-SORT. Para llegar a la condición de paro, los vértices se removieron en el siguiente orden: $\{d, a, c, f, h, g, b, e\}$.

Proponer un pseudocódigo explícito del algoritmo de Katsikarelis excede los objetivos de este trabajo de tesis; sin embargo, el autor de este documento de tesis propone el algoritmo DEGENERATE-SORT (Pseudocódigo 6), el cual obtiene un ordenamiento 2-degenerado de un grafo outerplanar. Más aún, dicho algoritmo se podría utilizar como el bloque principal para generar el algoritmo de Katsikarelis, para $k = 1$, debido a que gran parte de las acciones que realizaría su algoritmo, también las realiza el algoritmo DEGENERATE-SORT. Este algoritmo se describe a continuación.

Dado un grafo outerplanar $G = (V, E)$, el algoritmo DEGENERATE-SORT reduce gradualmente a G hasta llegar a la condición de paro; i.e., cuando el grafo resultante es un clique de tamaño dos. Para cualquier grafo outerplanar, existe al menos un vértice v cuyo grado es menor o igual a dos. Si el grafo de entrada es outerplanar maximal (i.e., si al agregar una arista adicional al grafo, éste deja de ser outerplanar), no existen vértices de grado uno. Si el grafo outerplanar no es maximal, entonces pueden existir vértices de grado uno. Este algoritmo siempre procesa los vértices de grado uno antes que los de grado dos. Este algoritmo remueve cada vértice de grado uno y su arista incidente. Para un vértice de grado dos, el algoritmo lo remueve, así como sus aristas incidentes. El orden en el que se van removiendo los vértices del grafo es el ordenamiento degenerado. Del clique de tamaño dos generado por la condición de paro, se toman sus dos vértices en cualquier orden para finalizar el ordenamiento.

Pseudocódigo 6: DEGENERATE-SORT($L(1), \ldots L(n)$)

Entrada: La lista de adyacencias de cada vértice del grafo G.
Salida: Ordenamiento degenerado en G.

```
1  C_1 ← NIL;                                  ▷ Inicializa C_1 y C_2 con NIL
2  C_2 ← NIL;
3  for i ← 1 to n do
4  |  D(i) ← FIND-DEG(i, L(i));                ▷ Calcula el grado de cada vértice i y lo almacena en D(i)
5  |  if D(i) = 1 then PUSH(C_1, i);           ▷ Inserta los vértices candidatos de grado 1 y 2 en C_1 y C_2
6  |  if D(i) = 2 then PUSH(C_2, i);
7  end
8  for k ← 1 to n − 2 do
9  |  if C_1 ≠ ∅ then                          ▷ Continúa si existen vértices de grado 1
10 |  |  x ← POP(C_1);
11 |  |  S(k) ← x;                            ▷ Inserta el vértice candidato en el arreglo de salida
12 |  |  D(x) ← 0 ;                           ▷ Elimina a x de G
13 |  |  v ← FIND-NEIGH_1(x, L(x));           ▷ Encuentra al único vecino de v, tal que D(v) ≠ 0
14 |  |  D(v) ← D(v) − 1;                     ▷ Decrementa el grado de v
15 |  |  if D(v) = 1 then PUSH(C_1, v);       ▷ Si v es candidato, se van a su cola
16 |  |  if D(v) = 2 then PUSH(C_2, v);
17 |  else
18 |  |  flag ← falso;
19 |  |  while flag = falso do                ▷ Determina si x todavía existe; pudo haber desaparecido
20 |  |  |  x ← POP(C_2);
21 |  |  |  if D(x) = 0 then flag = verdadero;
22 |  |  end
23 |  |  S(k) ← x;                            ▷ Inserta el vértice candidato en S
24 |  |  D(x) ← 0;                            ▷ Elimina a x de G
25 |  |  u, v ←FIND-NEIGH_2(x, L(x));         ▷ Encuentra a los vecinos de x, ambos con grado 0
26 |  |  D(u) ← D(u) − 1;
27 |  |  D(v) ← D(v) − 1;
28 |  |  if D(u) = 1 then PUSH(C_1, u);       ▷ Vértices candidatos se insertan en la cola
29 |  |  if D(v) = 1 then PUSH(C_1, v);
30 |  |  if D(u) = 2 then PUSH(C_2, u);
31 |  |  if D(v) = 2 then PUSH(C_2, v);
32 |  end
33 end
34 x ← POP(C_1);                               ▷ En este punto, G es un clique de tamaño 2
35 S(n − 1) ← x;
36 x ← POP(C_1);
37 S(n) ← x;
38 return S;
```

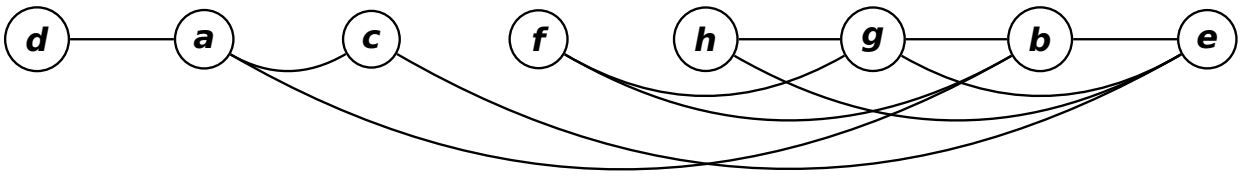


Figura 11. Orden degenerado del grafo outerplanar de la Figura 10.

A continuación se explica el algoritmo DEGENERATE-SORT (Pseudocódigo 6). En el ciclo de las Líneas $3-7$ se inspecciona si existe en el arreglo D algún vértice v con grado menor o igual a dos en G. Si es así, entonces el vértice v se inserta en la cola del tipo 'primero entra, primero sale' ('FIFO' por sus siglas en inglés) C_1 o C_2, si v tiene grado uno o dos, respectivamente.

El ciclo de las Líneas $8-33$ se describe como sigue. En el subciclo de las Líneas $9-16$, si existe algún vértice x con grado uno en G, x se va al arreglo de salida S y se cambia su grado a cero. Posteriormente, la función FIND-NEIGH$_1$ busca al vértice vecino v de x en su lista de adyacencias, y decrementa el grado de v en el arreglo D. Se inspecciona el grado de v y se va a la cola correspondiente, si su grado es menor o igual a dos, como lo describen las Líneas 15 y 16. El subciclo de las Líneas $17-31$ es el caso análogo para los vértices de grado dos. Las Líneas $34-37$ describen la condición de paro, cuando el clique es de tamaño dos. Cada uno de los vértices se va al arreglo S, la salida de este algoritmo. El ordenamiento degenerado del grafo consiste del arreglo S. El tiempo de ejecución de este algoritmo es de $O(n)$ unidades de tiempo.

En la Figura 10 se presenta un ejemplo de cómo un grafo outerplanar se va reduciendo hasta llegar a un clique de tamaño dos. Note que en esta figura, siempre que existe un vértice de grado uno, éste se remueve antes que cualquier vértice de grado dos. En la Figura 11 se ilustra el ordenamiento degenerado de los vértices. Este ordenamiento, de izquierda a derecha, muestra el momento en que cada vértice se removió del grafo G. Las aristas mostradas conectan a cada vértice con sus vecinos al momento de la eliminación. Note que el número de aristas que conectan a cualquier vértice con sus vecinos a su derecha nunca es mayor que dos; por lo tanto, la degeneración es dos.

En la Figura 12 se muestra una descomposición de árbol del grafo de la Figura 11 al

usar el algoritmo de Katsikarelis. Note que dicho árbol consiste de siete nodos, y cada uno de ellos tiene indicado con letras el conjunto de vértices en su bolsa correspondiente. Note también que la cardinalidad de cada bolsa nunca es mayor que tres, de aquí que la anchura de árbol de esta descomposición sea dos.

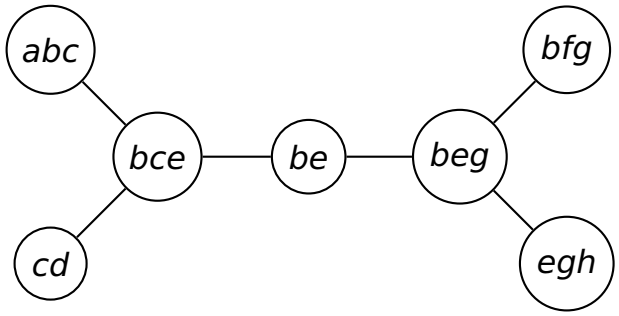


Figura 12. Ejemplo de la descomposición de árbol resultante al aplicar el algoritmo de Katsikarelis al grafo outerplanar de la Figura 10.

3.1.2. El algoritmo de Katsikarelis para $k > 1$

Esta sección describe el algoritmo de Katsikarelis (2013) para un grafo k-outerplanar G, con $k > 1$. En este tipos de grafos siempre existe un vértice con grado mayor o igual a tres. A continuación se describe el funcionamiento de este algoritmo.

Paso 1. Se realiza una 'expansión' para cada vértice v cuyo grado sea mayor que tres. Esta expansión consiste en reemplazar al vértice v por una cadena de vértices de longitud $deg(v) - 2$. A cada vértice extremo en la cadena se le agregan dos aristas, y a cada vértice restante de la cadena, una arista. Estas aristas se conectan al conjunto de vértices vecinos de v, manteniendo el empotramiento plano y respetando las caras adyacentes a las aristas incidentes al vértice v. Al concluir este paso, cada vértice del grafo 'expandido', llamado G', tiene a lo más grado tres. En la Figura 13 se muestra la expansión del vértice v de grado ocho, a una cadena de seis vértices, cada uno de grado tres. En esta figura, f_i denota la cara i. Se puede ver que el número de caras no se afecta con este proceso. La Figura 14a muestra un grafo 2-outerplanar. Note que en este grafo, los vértices c y f tienen grado cuatro. Al aplicar este paso a dichos vértices, se obtiene el grafo de la Figura 14b, el cual sigue siendo 2-outerplanar.

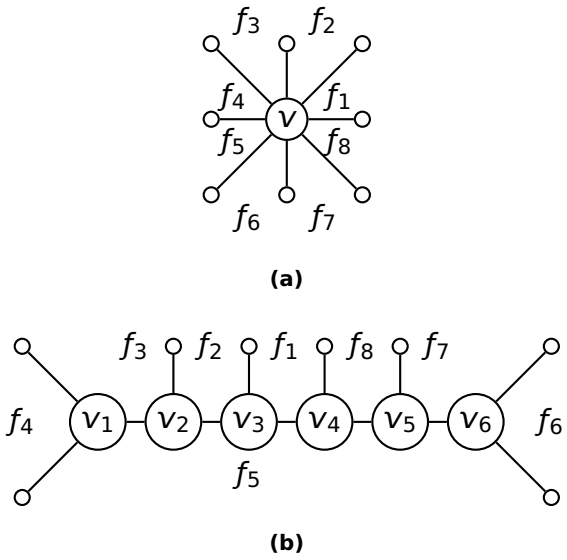


Figura 13. Paso 1 de la descomposición de árbol en un grafo k-outerplanar, $k > 1$; **a)** el vértice v tiene un grado $d(v) = 8$; **b)** el número de vértices expandidos es de $d(v) - 2 = 6$, y cada uno de los vértices expandidos tiene un grado igual a tres. Note que el número de caras en ambas figuras es el mismo. Esta figura se tomó de (Katsikarelis, 2013).

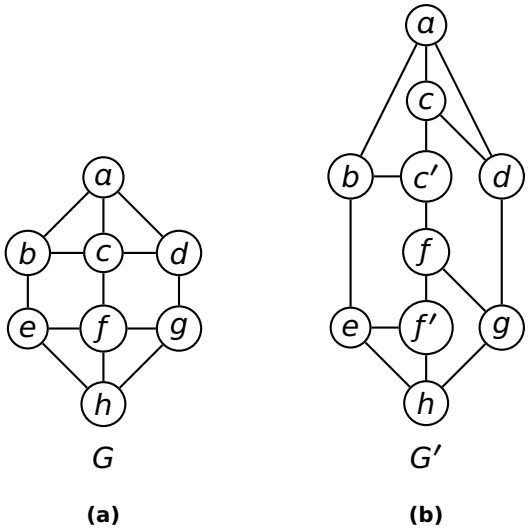


Figura 14. Ejemplo del paso 1 de la descomposición de árbol en un grafo k-outerplanar, para $k = 2$; **a)** el grafo de entrada G tiene dos vértices, c y f, con grado igual a cuatro; **b)** la expansión de vértices con grado mayor a tres en el inciso (a) produce el grafo G'.

Paso 2. A partir de G', al que también se llama G'_k, se construye un bosque de expansión maximal, denominado T. Este proceso consiste de a lo más k pasos. En cada paso i, se remueven todas las aristas de la cara externa del grafo G'_{k-i+1}, conservando las aristas removidas, R_{k-i+1}, en alguna estructura de datos. Este paso genera un

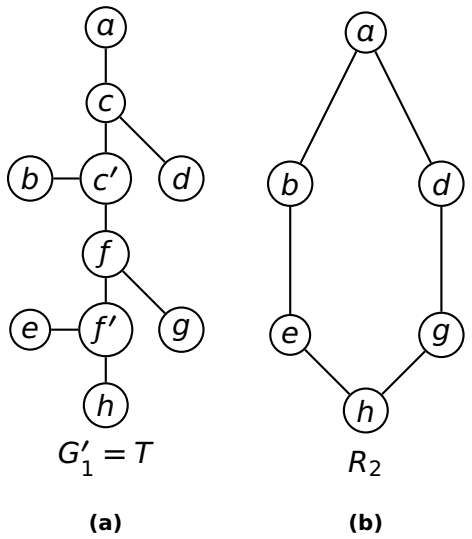


Figura 15. Paso 2 de la descomposición de árbol en un grafo k-outerplanar, $k > 1$. Continuación de la Figura 14b. Sea $G' = G'_k = G'_2$ la entrada en este paso; **a)** Grafo G'_1 obtenido de G'_2 al remover todas las aristas de la cara externa; **b)** Grafo inducido por las aristas removidas del grafo G'_2. Particularmente en esta entrada, G'_1 es un árbol de expansión maximal de G'_2, por lo que no es necesario realizar otra iteración. Por lo tanto, $G'_1 = T$.

grafo $(k-i)$-outerplanar, llamado G'_{k-i}. Al finalizar este proceso, el grafo resultante, que puede ser G'_1 o incluso G'_0, es un bosque, también llamados T_1 o T_0, según sea el caso. Posteriormente, siguiendo un procedimiento en orden inverso de a lo más k pasos, se van agregando aristas a T_1 o a T_0, hasta formar un bosque de expansión maximal de G'_k, llamado $T_k = T$. En cada paso i de este proceso constructivo, el árbol T_i se descompone en el árbol T_{i+1} al agregar un subconjunto de aristas de R_{i+1}. Se debe tener cuidado en el paso i, que a la hora de introducir aristas, no se genere un ciclo. Este procedimiento se utiliza para garantizar que el 'número de vértice recordado', asociado a T_k, sea menor o igual que $3k-1$. El *número de vértice recordado* es el máximo número de aristas del conjunto $G'_k \setminus T_k$ que son incidentes a un mismo vértice de G'_k. Esto garantiza que la anchura de árbol de la descomposición sea a lo más $3k-1$. En la Figura 15, G'_1 ya es un árbol, por lo que ya no es necesario generar G'_0.

Paso 3. A partir de G' y T se empieza a construir una descomposición de árbol $\mathcal{T}^*$ con anchura de árbol de a lo más $3k-1$. Sea $\mathcal{T}^* = (T, X^*)$, donde X^* es un conjunto de bolsas tal que la $X_u = \{u\}$. Después, por cada arista $(v, w) \in T$, se añade un nodo en $\mathcal{T}^*$, cuya bolsa es $\{v, w\}$, y se conecta entre los nodos con bolsas v y w. En la Figura 16a, se puede ver cómo se agregaron nueve nodos, cada una con bolsa de dos vértices, entre pares de nodos que incluyen una bolsa con un solo vértice.

Por cada arista (a, b) de G' que no esté en T, se realiza el siguiente procedimiento. Como existe en T una trayectoria simple que va del vértice a al vértice b, esta trayectoria también existe en $\mathcal{T}^*$, del nodo con bolsa a al nodo con bolsa b. Se elige un vértice arbitrario entre a y b (suponga que el elegido es a), entonces el vértice a se agrega a cada bolsa en cada nodo de la trayectoria que va desde el nodo con bolsa a hasta el nodo con bolsa b en $\mathcal{T}^*$, con excepción de las bolsas en los nodos extremos. En esta trayectoria se incluyen también los nodos que se agregaron como se describe en el párrafo anterior. En la Figura 16b se puede ver cómo se agregan entre cada trayectoria especificada por una arista que no está en T, pero sí en G', un vértice en cada una de las bolsas de cada nodo de $\mathcal{T}^*$, con excepción de las bolsas en los nodos extremos. Por ejemplo, la trayectoria que va del nodo con bolsa a al nodo con bolsa d, en la parte superior de la Figura 16b, se agregó el vértice d, excepto en los extremos. Este proceso se repite en las trayectorias definidas por las aristas (a, b), (b, e), (e, h), (g, h) y (g, d).

Paso 4. A partir de $\mathcal{T}^*$, los vértices expandidos se re-etiquetan con el nombre del vértice original antes de la expansión. Este proceso puede generar nodos contiguos con bolsas repetidas; si ésto ocurre, estos nodos se colapsan en uno solo, conservando la misma bolsa. La salida de este algoritmo es el grafo $\mathcal{T}$, y tiene una anchura de árbol de a lo más $3k - 1$. En la Figura 17 se observa cómo los vértices expandidos f' y c' de la Figura 16b se re-etiquetan por f y c. Note que el nodo $eff'g$ de la Figura 16b se re-etiqueta a efg; y cómo éste es contiguo a otro nodo con los mismos vértices en la bolsa, ambos se colapsan en un solo nodo.

3.2. Descomposición de árbol agradable

En Fomin *et al.* (2019) se define una descomposición de árbol agradable como una descomposición de árbol que además posee las siguientes propiedades:

Propiedad 1. $\mathcal{T}'$ está enraizado en el nodo r.

Propiedad 2. $X_r = \emptyset$, y para cada nodo hoja ℓ, $X_\ell = \emptyset$.

Propiedad 3. $\mathcal{T}'$ es una descomposición de árbol binaria.

Propiedad 4. Si el nodo t tiene dos hijos, t_1 y t_2, entonces $X_t = X_{t_1} = X_{t_2}$.

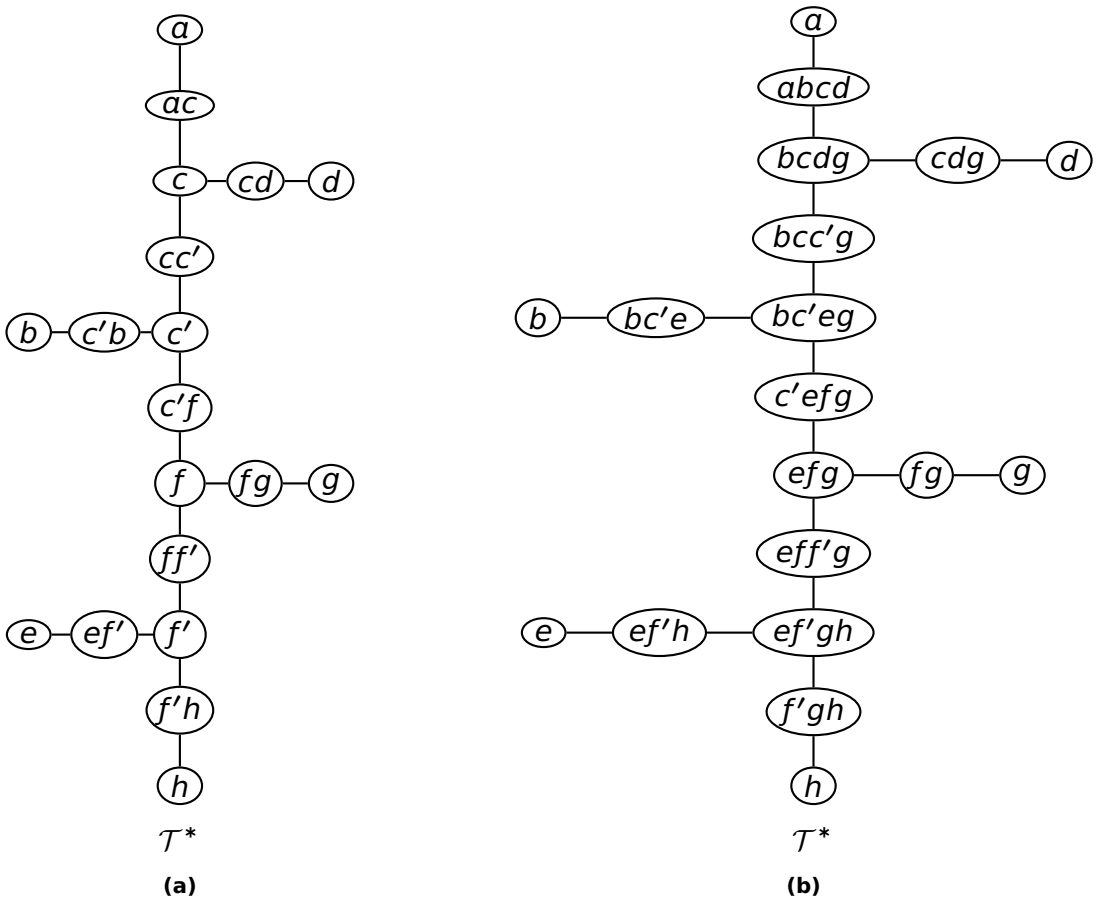


Figura 16. Paso 3 de la descomposición de árbol en un grafo k-outerplanar, $k > 1$. Continuación de la Figura 15; **a)** A partir del árbol de la Figura 15a, se genera un árbol $\mathcal{T}^*$ al cual, por cada arista de T, se le agrega un nodo a $\mathcal{T}^*$ con una bolsa que incluye a los dos vértices extremos de la arista de T; **b)** A partir de la Figura 16a, se genera un árbol en el que a cada nodo de una trayectoria simple definida por un ciclo fundamental o arista que no esté en el bosque, con excepción de los nodos extremos, se le agrega un vértice extremo a sus bolsas.

Propiedad 5. Si el nodo t tiene un hijo, t', entonces una de las siguientes condiciones es verdadera:

- $X_{t'} \subset X_t$ y $|X_t| = |X_{t'}| + 1$.
- $X_t \subset X_{t'}$ y $|X_{t'}| = |X_t| + 1$.

A continuación se describe un procedimiento para generar una descomposición de árbol agradable $\mathcal{T}'$ a partir de una descomposición de árbol arbitraria $\mathcal{T}$. Si $\mathcal{T}$ tiene una anchura de árbol τ, entonces $\mathcal{T}'$ también tiene una anchura de árbol τ. Adicionalmente, si $\mathcal{T}$ tiene $|I|$ nodos, entonces $\mathcal{T}'$ tiene $c \cdot |I|$ nodos, para una constante entera c. Este procedimiento requiere $O\left(\tau^{O(1)} \cdot n\right)$ u.t.

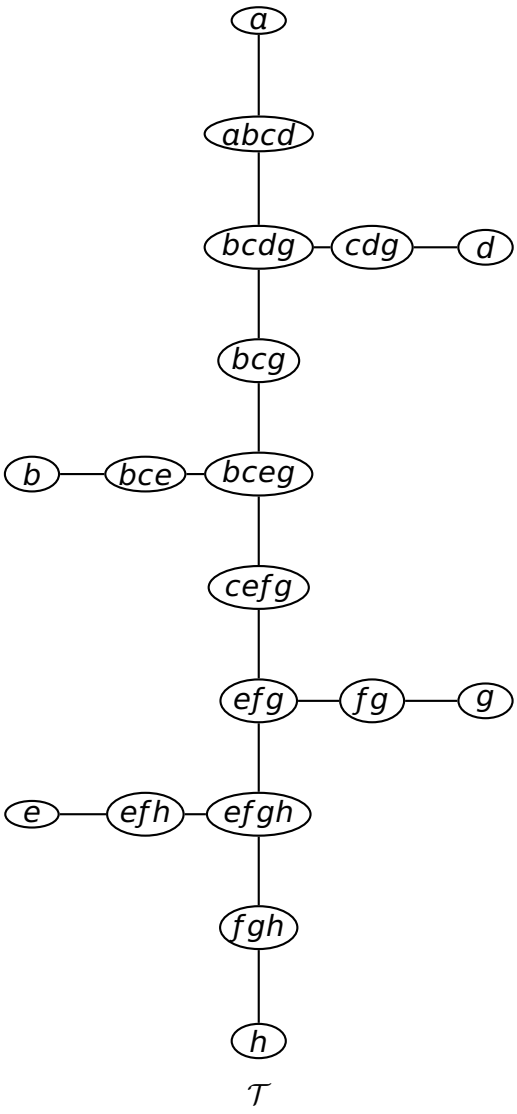


Figura 17. Paso 4 de la descomposición de árbol en un grafo k-outerplanar, $k > 1$. A partir de la Figura 16b, se reemplazan los vértices expandidos f' y c' por f y c, respectivamente, y se colapsan los nodos contiguos con bolsas repetidas. Su salida es una descomposición de árbol con anchura a lo más $3k - 1$.

Paso 1. Haga una copia de $\mathcal{T}$ en $\mathcal{T}'$, y conecte un nuevo nodo t' a cada nodo t de grado uno de $\mathcal{T}'$, donde $X_{t'} \leftarrow \emptyset$. Este paso se muestra en la Línea 2 del algoritmo NICE-TREE-DECOMPOSITION (Pseudocódigo 7). La Figura 18 muestra un árbol en el que se agregaron seis vértices con bolsa vacía (pintados de negro).

Paso 2. Enraice el árbol $\mathcal{T}'$ en un nodo arbitario r de grado uno. Existen varios algoritmos en la literatura para realizar este paso en tiempo lineal con respecto al número de nodos. En el árbol de la Figura 18 se supone que la raíz es el vértice superior del mismo.

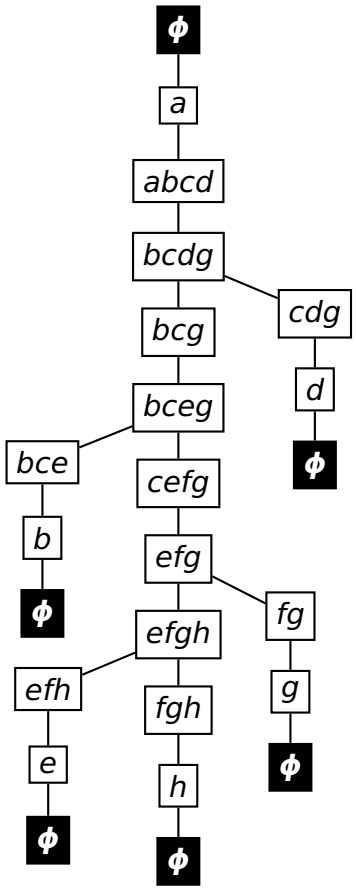


Figura 18. Pasos 1 y 2 para generar una descomposición de árbol agradable. Sea $\mathcal{T}$ el grafo de la Figura 17, y sea $\mathcal{T}'$ su copia. A cada nodo $t \in \mathcal{T}'$ con grado uno, se conecta un nuevo nodo t' con $X_{t'} = \emptyset$. Posteriormente, se enraiza $\mathcal{T}'$ a partir de la elección de uno de los nodos con grado uno.

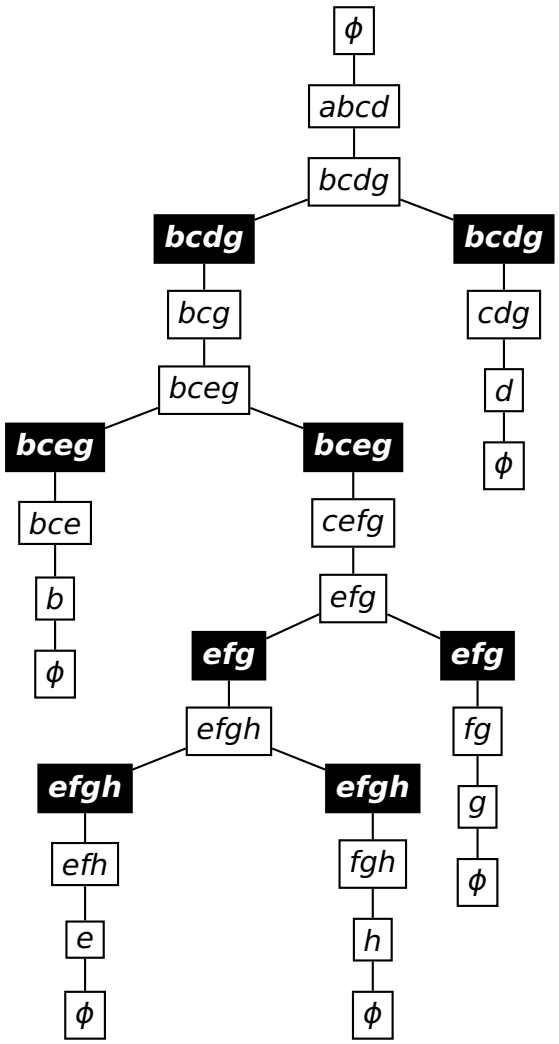


Figura 19. Paso 4 para generar una descomposición de árbol agradable. Note que el árbol de la Figura 18 ya es un árbol binario, por lo que no es necesario realizar el paso 3. En este paso, a cada nodo t con más de un hijo, se agregan dos nuevos nodos hijos t'_1 y t'_2 a t, donde $X_{t'_1} \leftarrow X_{t'_2} \leftarrow X_t$. Posteriormente se asigna a t'_1 y a t'_2 como el padre de t_1 y t_2, respectivamente.

Paso 3. Si $\mathcal{T}'$ no es un árbol binario, transfórmelo a binario. Una forma de hacer este procedimiento es a través del algoritmo *N*-ARY-TO-BINARY-TREE (Pseudocódigo 8), el cual se aplica a cada nodo de $\mathcal{T}'$ con grado mayor a tres. Como el árbol de la Figura 18 es binario, no se ejemplifica este paso.

Paso 4. Para cada nodo t con dos hijos en $\mathcal{T}'$, agregue dos nodos hijos t'_1 y t'_2 a t, con $X_{t'_1} \leftarrow X_{t'_2} \leftarrow X_t$. Asigne a t'_1 y a t'_2 como el padre de t_1 y t_2, respectivamente. Este paso corresponde al ciclo 'for' de las Líneas $7-10$ en el algoritmo NICE-TREE-DECOMPOSITION (Pseudocódigo 7). En el árbol de la Figura 19 se muestran en negro los ocho vértices agregados en el Paso 4.

Paso 5. Mientras exista un par de nodos contiguos t y t' en $\mathcal{T}'$, tal que $|X_t \setminus X_{t'}| + |X_{t'} \setminus X_t| > 1$, agregue un nodo t'' con bolsa $X_{t''}$ entre ambos, de tal forma que $|X_{t'} \setminus X_{t''}| + |X_{t''} \setminus X_{t'}| = 1$. Una implementación de este paso se presenta en el algoritmo CLOSEUP-ROUTINE (Pseudocódigo 9). En el árbol de la Figura 20 se muestran en negro los vértices agregados en el Paso 5.

Pseudocódigo 7: NICE-TREE-DECOMPOSITION($\mathcal{T}$)

Entrada: Una descomposición de árbol $\mathcal{T} = (X, T)$ con anchura de árbol τ, no enraizada.
Salida: Descomposición de árbol agradable $\mathcal{T}' = (X', T')$ con anchura de árbol τ.

Inicialice $\mathcal{T}' = (X', T') \leftarrow (X, T)$;
A cada nodo hoja t de $\mathcal{T}'$ conecte un nuevo nodo hoja t', donde $X_{t'} \leftarrow \emptyset$;
Seleccione uno de estos nuevos nodos hojas en forma arbitraria como raíz r, y enraice el árbol $\mathcal{T}'$ en r;
foreach $t \in \mathcal{T}'$, *con* $\boldsymbol{\kappa} > 2$ **do**
    N-ARY-TREE-TO-BINARY-TREE(t,κ);
end
foreach $t \in \mathcal{T}'$ *que tenga dos hijos,* t_1 *y* t_2, *tal que* $X'_t \neq X'_{t_1} \neq X'_{t_2}$ **do**
    Agregue dos nodos hijos t'_1 y t'_2 a t en $\mathcal{T}'$, con $X'_{t'_1} \leftarrow X'_{t'_2} \leftarrow X'_t$;
    Asigne a t'_1 y a t'_2 como el padre de t_1 y t_2, respectivamente;
end
foreach $t \in \mathcal{T}'$ *que tenga un hijo único* t_1 **do**
    CLOSEUP-ROUTINE(t);
end

Pseudocódigo 8: *N*-ARY-TO-BINARY-TREE(t,κ)

Entrada: Un nodo t de una descomposición de árbol $\mathcal{T}'$ cuyo número de descendientes κ es mayor a 2.
Salida: Subárbol binario en $\mathcal{T}'$ con el nodo t como la raíz del subárbol.

if $\kappa = 2$ **then**
    exit;
else
    Agregue p como hijo de t en $\mathcal{T}'$, con $X'_p \leftarrow X'_t$;
    De los κ hijos originales de t, asigne $\kappa - 1$ de ellos como hijos a p;
    N-ARY-TO-BINARY-TREE(p, $\kappa - 1$);
end

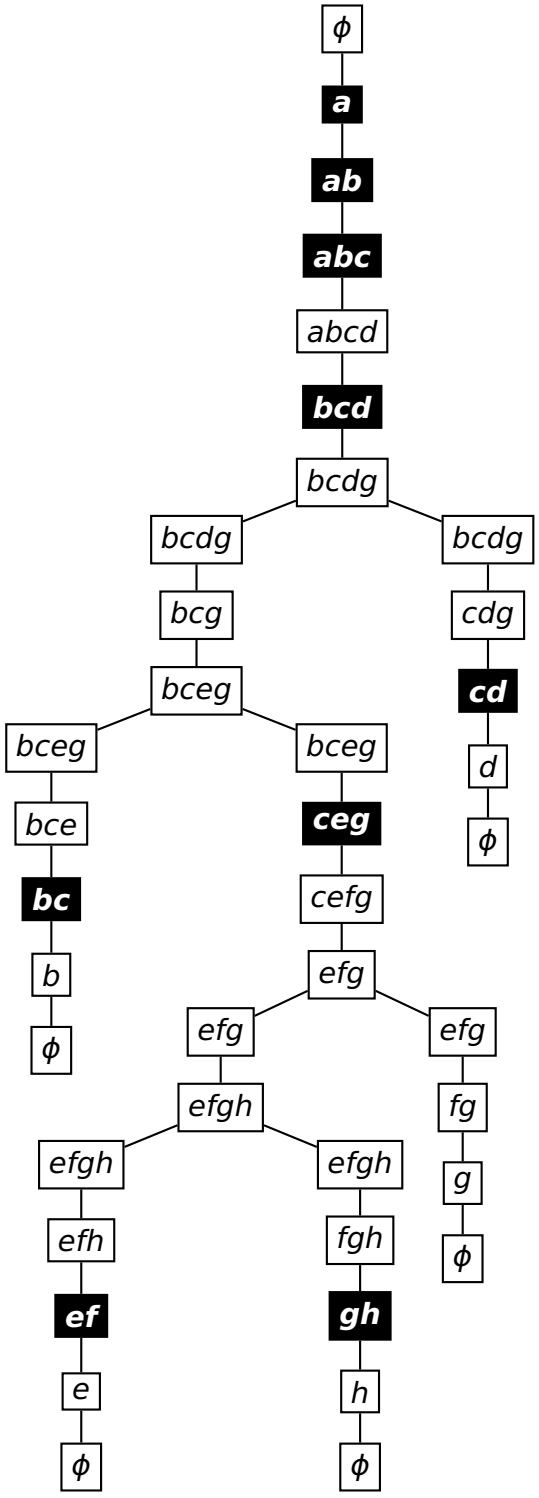


Figura 20. Paso 5 para generar una descomposición de árbol agradable. A partir de la descomposición de árbol $\mathcal{T}'$ en la Figura 19 se realiza lo siguiente. Para cada nodo t con un descendiente t', se cuenta el número de vértices en que difieren las bolsas de los nodos t y t'. Si este número es mayor que uno, se ejecuta el algoritmo mostrado en el Pseudocódigo 9.

Pseudocódigo 9: CLOSEUP-ROUTINE(t)

Entrada: Un nodo t de una descomposición de árbol $\mathcal{T}'$ cuya diferencia entre X_t y la bolsa $X_{t'}$ de su hijo sea mayor que uno.
Salida: La diferencia entre la bolsa X_t y $X_{t'}$ es igual a uno.

```
1  if |X_t \ X_t'| + |X_t' \ X_t| = 1 then
2  |  exit;
3  else
4  |  if |X_t \ X_t'| ≥ 1 then
5  |  |  Sea u uno de los vértices de {X_t \ X_t'};
6  |  |  Inserte el nodo p entre t y t';
7  |  |  X_p ← X_t \ u;
8  |  |  CLOSEUP-ROUTINE(p);
9  |  else
10 |  |  Sea u uno de los vértices de {X_t' \ X_t};
11 |  |  Inserte el nodo p entre t y t';
12 |  |  X_p ← X_t' \ u;
13 |  |  CLOSEUP-ROUTINE(p);
14 |  end
15 end
```

Capítulo 4. Algoritmo DP-Clique

Este capítulo describe el Algoritmo DP-Clique que resuelve el problema de enumeración de cliques maximales de distancia $\boldsymbol{\kappa}$, para $\boldsymbol{\kappa} = 1$, en el grafo k-outerplanar. Grosso modo, la resolución de dicho problema se lleva a cabo como se describe a continuación. Primero, dado un grafo k-outerplanar $G = (V, E)$ no dirigido, se utiliza el algoritmo descrito por Katsikarelis (2013) para obtener una descomposición de árbol de G. Después, mediante el algoritmo Nice-Tree-Decomposition (Pseudocódigo 7), se transforma la descomposición anterior en una nueva, llamada descomposición de árbol agradable. Finalmente, utilizando la técnica de programación dinámica, se propone un algoritmo que enumera todos los cliques maximales de distancia 1 en G.

4.1. Conceptos preliminares

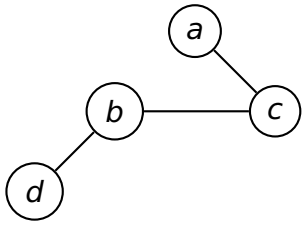
Sean $G = (V, E)$ un grafo k-outerplanar, $\mathcal{T} = (X, T)$ una descomposición de árbol de G, y $\mathcal{T}' = (X', T')$ una descomposición de árbol agradable, obtenida a partir de $\mathcal{T}$. Sea C un clique maximal de distancia 1 en G. A continuación se describen algunos conceptos básicos necesarios para la explicación del algoritmo DP-Clique.

4.1.1. Tablas de programación dinámica

Dada una descomposición de árbol agradable $\mathcal{T}'$ y su nodo raíz, obtenida por medio del algoritmo Nice-Tree-Decomposition (Pseudocódigo 7), se puede asignar una etiqueta a cada nodo t de $\mathcal{T}'$ por medio de un recorrido postorder. Cada nodo t de $\mathcal{T}'$ tiene asociada una tabla, denotada por c^t. Dicha tabla se puede ver como información satelital del nodo t, en forma similar a su bolsa X_t. La tabla consiste de un renglón de longitud $2^{|X_t|}$, y cada elemento de dicho renglón se denota como $c^t(j)$, para $0 \leq j \leq 2^{|X_t|} - 1$.

Como $\mathcal{T}'$ es un arbol enraizado, cada nodo t, con excepción de la raiz, tiene asociado a un padre. Similarmente, cada nodo que no sea hoja tiene asociado uno o dos nodos hijos.

Un *mintérmino* para el nodo t es una expresión algebraica booleana que contiene los vértices de la bolsa X_t. Esta función es verdadera únicamente para una combina-



(a)

(b)

$\bar{a}\,\bar{b}\,\bar{c}$	$\bar{a}\,\bar{b}\,c$	$\bar{a}\,b\,\bar{c}$	$\bar{a}\,b\,c$	$a\,\bar{b}\,\bar{c}$	$a\,\bar{b}\,c$	$a\,b\,\bar{c}$	$a\,b\,c$
000	001	010	011	100	101	110	111
∅	{c}	{b}	{b,c}	{a}	{a,c}	∅	∅

(c)

Figura 21. Ejemplo de mintérmino. **a)** Grafo G no dirigido y conectado. **b)** Un nodo t con bolsa $X_t = \{a, b, c\}$ en su descomposición de árbol. **c)** Tabla correspondiente con los mintérminos en el renglón superior y sus representaciones binarias en el renglón central.

ción de dichas variables. Para un conjunto de $|X_t|$ variables, existen $2^{|X_t|}$ mintérminos distintos. Cada mintérmino se puede mapear a una cadena binaria de tamaño $|X_t|$. La localidad j de la tabla c^t está asociada al mintérmino j (expresado en binario). En la Figura 21b se puede ver un nodo de una descomposición de árbol cuya bolsa $X_t = \{a, b, c\}$. En la Figura 21c, se puede ver la tabla correspondiente con ocho mintérminos distintos, mostrados en el renglón superior. Las cadenas binarias en el renglón central representan el número binario j asociado a cada mintérmino. El renglón inferior muestra al conjunto de vértices, derivado de la representación binaria, que forman un clique de distancia 1.

Para un nodo hoja t, dado que la cardinalidad de $|X_t| = 0$, c^t consiste de una tabla de una sola localidad de memoria que corresponde al conjunto vacío.

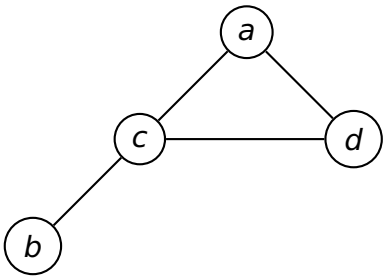
4.1.2. Operador estrellita '$*$'

Sean A y B dos subconjuntos de vértices del grafo G. El operador *estrellita*, denotado como '$*$', es binario y conmutativo. Este operador está relacionado con la conectividad del grafo, y tiene como operandos a los conjuntos A y B. El conjunto A es un clique, mientras que el conjunto B es un solo vértice. La operación $A * B$ determina

si el conjunto $A \cup B$ forma un clique de G, como se muestra en la Ecuación 1.

$$A * B = \begin{cases} A \cup B & \text{Si } A \cup B \text{ forman un clique} \\ \emptyset & \text{de cualquier otro modo} \end{cases} \quad (1)$$

La Figura 22a muestra un grafo de cuatro vértices. La Figura 22b muestra dos ejemplos del operador estrellita. El primer ejemplo, $\{a, c\} * \{b\} = \emptyset$ porque el conjunto $\{a, b, c\}$ no forma un clique en el grafo de la Figura 22a. Por otro lado, $\{a, c\} * \{d\} = \{a, c, d\}$ porque dicho conjunto sí forma un clique en la Figura 22a.



(a)

$$\{a, c\} * \{b\} = \emptyset$$

$$\{a, c\} * \{d\} = \{a, c, d\}$$

(b)

Figura 22. Ejemplo del operador estrellita. **a)** Grafo outerplanar. **b)** Ejemplos de dos operaciones estrellita de acuerdo a la estructura del grafo. La operación $\{a, c\} * \{b\}$ regresa el conjunto vacío porque el conjunto $\{a, b, c\}$ no forma un clique en el grafo. Por otro lado, $\{a, c\} * \{d\} = \{a, c, d\}$ porque dicho conjunto forma un clique en el grafo.

4.2. Descripción general de DP-Clique

El algoritmo de programación dinámica DP-Clique tiene como entrada una descomposición de arbol agradable $\mathcal{T}'$ del grafo k-outerplanar G, y su salida es la numeración de cliques maximales de distancia uno. Primero, se enumeran todos los nodos de $\mathcal{T}'$ por medio de un recorrido postorder. Para cada nodo t en $\mathcal{T}'$, el algoritmo llena la tabla c^t, tal y como se describe en la Sección 4.3. Durante el llenado de algunas tablas, se ejecuta el Algoritmo Find-Maximal-Clique (Pseudocódigo 10). Este último algoritmo encuentra los cliques maximales de distancia uno en dichas tablas. Dicho algoritmo se describe la Sección 4.4. Al terminar el recorrido sobre $\mathcal{T}'$, la salida de DP-Clique son todos los cliques maximales de distancia uno que se encontraron, los cuales se fueron almacenando paulatinamente dentro de alguna estructura de datos.

4.3. Llenado de tablas

La única localidad de las tablas para los nodos hojas se llena con el símbolo conjunto vacío. Lo anterior obedece a que las bolsas de estos nodos están vacías. Para el llenado del resto de las tablas, se tienen tres casos, uno para cada tipo de nodo: introductor, eliminador y unión.

4.3.1. Tablas de nodos introductores

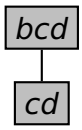
Sea c^t la tabla del nodo introductor t, y sea $c^{t'}$ la de su único hijo t'. Como t es un nodo introductor, la bolsa X_t tiene un vértice más que la bolsa $X_{t'}$. Sea el vértice $v = X_t \setminus X_{t'}$ el vértice introducido.

Sea $\gamma_{|X_t|-1}, \gamma_{|X_t|-2}, \ldots, \gamma_1, \gamma_0$ el conjunto de vértices o variables asociados a la bolsa X_t. Sea γ_i el vértice introducido en el nodo t, donde i puede tomar cualquier valor entre 0 y $|X_t|-1$. Por lo tanto, las variables asociadas a la bolsa $X_{t'}$ son $\gamma_{|X_t|-1},\ldots,\gamma_{i+1},\gamma_{i-1},\ldots,\gamma_0$; estas variables corresponden a $\gamma'_{|X_{t'}|-1}, \ldots, \gamma'_0$ en la bolsa $X_{t'}$. En ambas tablas, cada mintérmino contiene alguna combinación de las variables asociadas (negadas o no negadas) a sus bolsas. El llenado de la tabla del nodo introductor c^t se realiza de la siguiente manera:

- Para una localidad en la tabla X_t, la cual esté asociada al mintérmino $\gamma_{|X_t|-1}, \ldots, \gamma_0$, y que además tenga γ_i negado, se llena con el contenido extraído de la tabla $X_{t'}$ en el mintérmino $\gamma'_{|X_{t'}|-1}, \ldots, \gamma'_0$.
- Para una localidad en la tabla X_t, la cual esté asociada al mintérmino $\gamma_{|X_t|-1}, \ldots, \gamma_0$, y que además tenga γ_i no negado, se llena con el resultado de la operación estrellita entre la variable γ_i y el contenido extraído de la tabla $X_{t'}$ en el mintérmino $\gamma'_{|X_{t'}|-1}, \ldots, \gamma'_0$.

En el ejemplo de la Figura 23, sean $X_t = \{b, c, d\}$ y $X_{t'} = \{c, d\}$. En consecuencia, $v = b$. Aquellas localidades en c^t cuyos mintérminos tengan a b negado, se llenan con la localidad extraída de $c^{t'}$, donde las combinaciones de las variables cd coincidan. El resto de localidades en c^t, esto es, aquellas que tengan a b verificado, se llenan

con el resultado de la operación estrellita entre b y el contenido extraído de $c^{t'}$ en el mintérmino cd, donde las combinaciones de las variables cd coincidan.



(a)

$\overline{b}\,\overline{c}\,\overline{d}$	$\overline{b}\,\overline{c}\,d$	$\overline{b}\,c\,\overline{d}$	$\overline{b}\,c\,d$	$b\,\overline{c}\,\overline{d}$	$b\,\overline{c}\,d$	$b\,c\,\overline{d}$	$b\,c\,d$
000	001	010	011	100	101	110	111
$\emptyset$	$\{d\}$	$\{c\}$	$\{c,d\}$	$\{b\}$	$\emptyset$	$\{b,c\}$	$\emptyset$

$\overline{c}\,\overline{d}$	$\overline{c}\,d$	$c\,\overline{d}$	$c\,d$
00	01	10	11
$\emptyset$	$\{d\}$	$\{c\}$	$\{c,d\}$

(b)

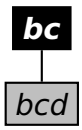
Figura 23. Ejemplo de llenado de tabla de un nodo introductor. **a)** Fragmento de la descomposición de árbol del grafo de la Figura 22a; el nodo superior 'introduce' al vértice b. **b)** Llenado de tablas de los nodos en el árbol mostrado. La tabla superior se llena conforme a las instrucciones descritas en la Sección 4.3.1.

4.3.2. Tabla de nodos eliminadores

Sea c^t la tabla del nodo eliminador t, y sea $c^{t'}$ la de su único hijo t'. Como t es un nodo eliminador, la bolsa X_t tiene un vértice menos que la bolsa $X_{t'}$. Sea el vértice $w = X_{t'} \setminus X_t$ el vértice eliminado.

Sea $\gamma'_{|X_{t'}|-1}, \ldots, \gamma'_{i+1}, \gamma'_i, \gamma'_{i-1}, \ldots, \gamma'_0$ el conjunto de vértices o variables asociados a la bolsa $X_{t'}$. Sea γ'_i el vértice eliminado del nodo t', donde i puede tomar cualquier valor entre 0 y $|X_{t'}| - 1$. Por lo tanto, las variables asociadas a la bolsa X_t son $\gamma'_{|X_{t'}|-1}, \ldots, \gamma'_{i+1}, \gamma'_{i-1}, \ldots, \gamma'_0$; estas variables corresponden a $\gamma_{|X_t|-1}, \ldots, \gamma_0$ en la bolsa X_t. El llenado de la tabla del nodo eliminador c^t se realiza de la siguiente manera:

- Para una localidad en la tabla X_t, la cual está asociada al mintérmino $\gamma_{|X_t|-1}, \ldots, \gamma_0$, se llena con el contenido extraído de la tabla $X_{t'}$ en el mintérmino $\gamma'_{|X_{t'}|-1}, \ldots, \gamma'_{i+1}, \gamma'_i, \gamma'_{i-1}, \ldots, \gamma'_0$, donde γ'_i aparece negado.



(a)

$\bar{b}\,\bar{c}$	$\bar{b}\,c$	$b\,\bar{c}$	$b\,c$
00	01	10	11
$\emptyset$	$\{c\}$	$\{b\}$	$\{b, c\}$

$\bar{b}\,\bar{c}\,\bar{d}$	$\bar{b}\,\bar{c}\,d$	$\bar{b}\,c\,\bar{d}$	$\bar{b}\,c\,d$	$b\,\bar{c}\,\bar{d}$	$b\,\bar{c}\,d$	$b\,c\,\bar{d}$	$b\,c\,d$
000	001	010	011	100	101	110	111
$\emptyset$	$\{d\}$	$\{c\}$	$\{c, d\}$	$\{b\}$	$\emptyset$	$\{b, c\}$	$\emptyset$

(b)

Figura 24. Ejemplo de llenado de tabla de un nodo eliminador. **a)** Fragmento de la descomposición de árbol del grafo de la Figura 22a; el nodo superior 'elimina' al vértice *d*. **b)** Llenado de tablas de los nodos en el árbol mostrado. La tabla superior se llena conforme a las instrucciones descritas en la Sección 4.3.2.

En el ejemplo de la Figura 24, sean $X_t = \{b, c\}$ y $X_{t'} = \{b, c, d\}$. En consecuencia, $w = d$. Las localidades en c^t se llenan con las localidades extraídas de $c^{t'}$, cuyos mintérminos tengan a *d* negado, y las combinaciones de las variables *bc* coincidan.

4.3.3. Tabla de nodos unión

Sea c^t la tabla del nodo unión t, y sean $c^{t'}$ y $c^{t''}$ las tablas de sus dos hijos t' y t''. Las bolsas X_t, $X_{t'}$ y $X_{t''}$ son iguales. El llenado de la tabla del nodo unión c^t se realiza de la siguiente manera:

- Cada localidad de la tabla c^t, para un mintérmino dado, se llena con el contenido de cualquiera de las dos tablas hijas en el mismo mintérmino; i.e., la tabla c^t, $c^{t'}$ y $c^{t''}$ son iguales.

4.4. Determinación de cliques maximales en tablas

Sea c^t la tabla de un nodo eliminador t, y sea $c^{t'}$ la tabla de su único hijo, t'. Como t es un nodo eliminador, la bolsa X_t tiene un vértice menos que la bolsa $X_{t'}$. Sea el vértice $w = X_{t'} \setminus X_t$ el vértice que se pierde en t.

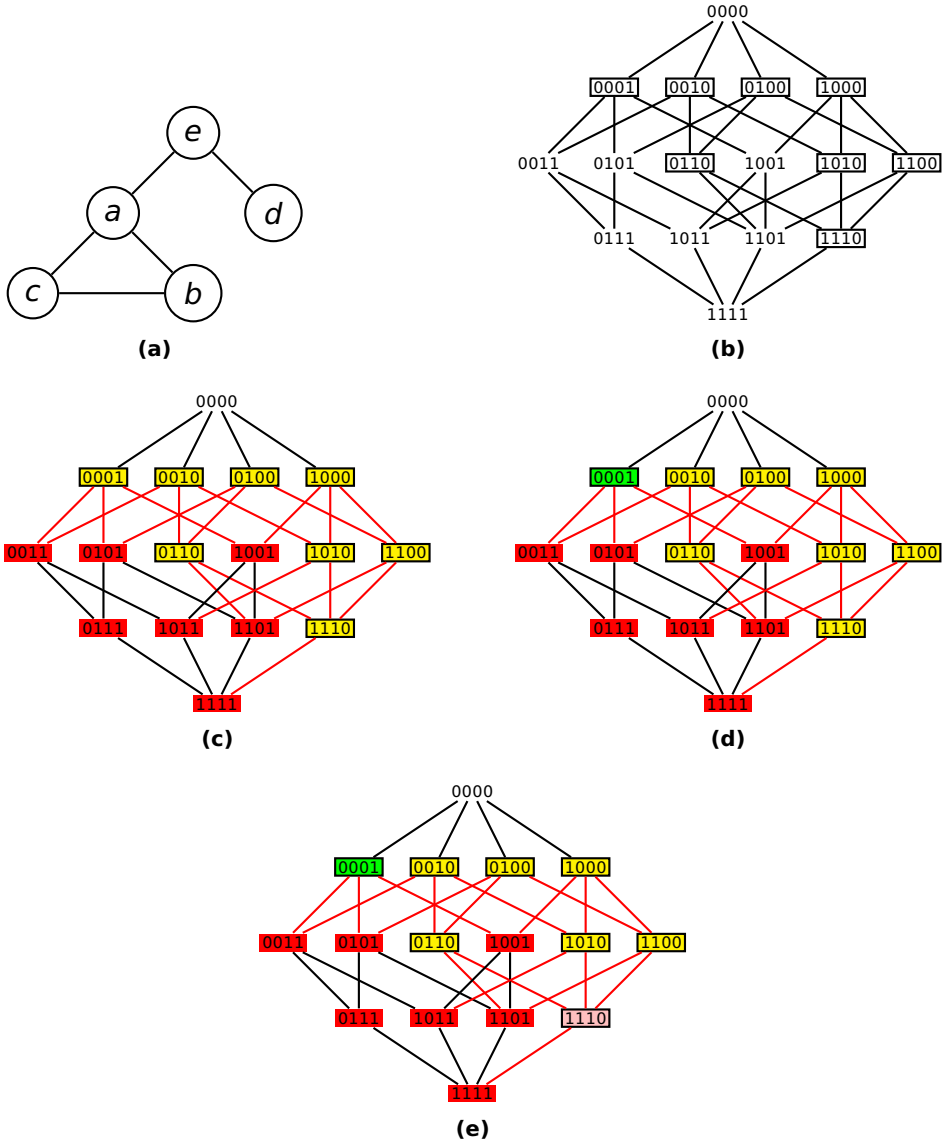


Figura 25. Ejemplo de determinación de cliques maximales en una tabla $c^{t'}$ con cuatro vértices. **a)** El grafo G de cinco vértices. **b)** Los mintérminos de las variables a, b, c y d conectados como un hipercubo. Los mintérminos en rectángulos representan cliques, de acuerdo al grafo del inciso a. **c)** Ejecución de las Líneas $2-13$ del algoritmo FIND-MAXIMAL-CLIQUE. Los rectángulos en amarillo representan los cliques encontrados en $c^{t'}$. Los rectángulos en rojo representan conjuntos vacíos en la tabla $c^{t'}$. Las aristas rojas son los enlaces creados. Suponga que $w = a$. **d)** Después de la ejecución de la primera iteración del ciclo **for** de las Líneas 14-28, se detecta que el clique del mintérmino 0001 es candidato maximal (mostrado en verde). **e)** Después de la ejecución de la tercera iteración del ciclo **for** de las Líneas 14-28, se detecta que el clique del mintérmino 1110 es maximal (mostrado en rosa).

El procedimiento para determinar los candidatos a ser cliques maximales de distancia uno en una tabla se encuentra en el Algoritmo FIND-MAXIMAL-CLIQUE, mostrado en el Pseudocódigo 10. Note que por las propiedades de $\mathcal{T}'$, siempre existe una tabla eliminadora como raíz. Este algoritmo utiliza la información almacenada en la tabla $c^{t'}$ para encontrar dichos conjuntos.

El Algoritmo FIND-MAXIMAL-CLIQUE se describe a continuación. La entrada de este algoritmo es la tabla $c^{t'}$. En el ciclo **for** de las Líneas $2-12$, en forma iterativa, para cada mintérmino i de la tabla, se crea una lista vacía $L(i)$ por medio de la función CREATE-LIST. Si el mintérmino i forma un clique de distancia uno, entonces se crean enlaces a otros mintérminos por medio del ciclo **for** de las Líneas $6-8$. Estos enlaces son un subconjunto de los enlaces existentes en un hipercubo de dimensión $|X_{t'}|$, suponiendo que los mintérminos son los vértices del hipercubo. La Figura 25b muestra un hipercubo, donde cada vértice está asociado a un mintérmino, y las líneas son las aristas. Los enlaces generados en el ciclo **for** de las líneas $6-8$ corresponden a las aristas rojas mostradas en la Figura 25c. La Línea 13 ordena los mintérminos de acuerdo al número de unos que contiene su etiqueta binaria. El ciclo **for** de las Líneas $14-28$ procesa cada mintérmino de acuerdo al ordenamiento anterior para determinar si un mintérmino es candidato a ser clique maximal o un clique maximal confirmado. En la Figura 25, un clique en el mintérmino i es por lo menos candidato a ser clique maximal si no existen cliques más grandes en los mintérminos vecinos indicados por aristas rojas. Los mintérminos marcados en verde son cliques maximales; aquellos marcados en rosado son cliques máximos. En la Línea 16 del Algoritmo FIND-MAXIMAL-CLIQUE, un mintérmino que forma un clique es candidato a maximal si el encabezado de la lista es vacío, o si se cumple la condición anterior y además cada arista roja del mintérmino no forma un clique. Las Líneas $18-25$ determinan si el mintérmino candidato a clique maximal es clique maximal confirmado o no.

Ejemplo de ejecución del Algoritmo FIND-MAXIMAL-CLIQUE. Suponga que se tiene un grafo de cinco vértices, como se muestra en la Figura 25a. Suponga que la bolsa $X_{t'}$ del árbol $\mathcal{T}'$ contiene a los vértices $\{a, b, c, d\}$, pero no el $\{e\}$. También suponga que la bolsa X_t tiene los vértices $\{b, c, d\}$; por lo tanto, $w = a$. Al ejecutarse el primer ciclo **for** y ordenar los mintérminos de acuerdo al número de unos, se pueden visualizar los mintérminos como el grafo de la Figura 25a. Los mintérminos en rojo no forman

cliques. Después de ejecutar la primera iteración del ciclo **for** de las Líneas $14-28$, se detecta que el clique en el mintérmino 0001, mostrado en verde en la Figura 25d, es candidato a maximal. En la segunda iteración del ciclo **for**, no se detecta ningún clique maximal. En la tercera iteración del ciclo, se detecta que el clique del mintérmino 1110 es maximal, mostrado en rosa en la Figura 25e. Finalmente, la cuarta iteración del ciclo **for**, no detecta algún clique maximal.

El tiempo de ejecución de este algoritmo es $O(\chi \cdot 2^{\chi})$ u.t., donde $\chi = |X_{t'}|$. Este tiempo de ejecución obedece a que en el peor caso se tienen que crear todas las aristas del hipercubo de dimensión χ, las cuales son $O(\chi \cdot 2^{\chi})$ u.t.

En $\mathcal{T}'$, existe un nodo eliminador por cada vértice del grafo original. Por tal razón, el algoritmo FIND-MAXIMAL-CLIQUE se ejecuta en a lo más $O(n)$ tablas. Determinar los cliques maximales de distancia uno en $\mathcal{T}'$ tiene un tiempo de ejecución de $O(\chi \cdot 2^{\chi}) \cdot O(n) = O(\chi \cdot n \cdot 2^{\chi})$ u.t.

4.5. Demostración de que DP-CLIQUE es correcto

Para demostrar que el algoritmo DP-CLIQUE es correcto, primero se demuestra el Lema 4.1.

Lema 4.1 *Sea C un clique del grafo de entrada G, entonces debe existir un nodo t del árbol agradable $\mathcal{T}'$ de G, tal que $C \subseteq X_t$.*

Demostración. La demostración es por inducción en la cardinalidad del clique C.

Sin pérdida de generalidad, suponga que C es un clique maximal en G, cuya cardinalidad es k.

Caso base: Suponga que $k = 2$. Por propiedad del árbol agradable, cada arista de G debe estar en la bolsa de algún nodo t de $\mathcal{T}'$. Por lo tanto, el caso base se cumple.

Hipótesis inductiva: Ahora suponga que todo clique C_0 de cardinalidad menor o igual a $k-1$ se encuentra en la bolsa de al menos un nodo t de $\mathcal{T}'$.

Paso inductivo: Ahora se demuestra que si existe el clique C de cardinalidad k en G, éste también se encuentra en la bolsa de al menos un nodo t de $\mathcal{T}'$.

Por la hipótesis inductiva, el clique C_0 se encuentra en la bolsa de al menos un nodo t de $\mathcal{T}'$. Sea $\mathcal{T}'_{C_0}$ el conjunto de nodos t de $\mathcal{T}'$, cuyas bolsas contienen al clique C_0. Por propiedad de $\mathcal{T}'$, dicho conjunto es un árbol conectado.

Como C tiene cardinalidad k, se puede descomponer a C en un clique C_0 de tamaño $k-1$ y un vértice v, de tal manera que cada vértice $u \in C_0$ se conecta a v; i.e., $C = C_0 \cup v$.

De aquí en adelante, la demostración es por contradicción. Se supone que el clique C de cardinalidad k se encuentra en G, pero éste no se encuentra en $\mathcal{T}'$. Sea $\mathcal{T}'_v$ el conjunto de nodos t cuyas bolsas contienen al vértice v. Note que $\mathcal{T}'_v$ es conectado, y por la suposición anterior, este árbol y $\mathcal{T}'_{C_0}$ son disjuntos; i.e., estos dos árboles no comparten ningún nodo.

Sea F' el bosque generado al remover $\mathcal{T}'_{C_0}$ de $\mathcal{T}'$. Si dicho bosque tiene más de un árbol, entonces $\mathcal{T}'_v$ debe pertenecer solamente a uno de ellos (digamos, $\mathcal{T}'_{v^+}$). De no ser así, como $\mathcal{T}'_v$ es conectado, este árbol tendría un traslape con $\mathcal{T}'_{C_0}$. Lo cual no es posible por la suposición inicial.

Como $\mathcal{T}'$ es conectado, $\mathcal{T}'_{C_0}$ y $\mathcal{T}'_{v^+}$ deben de estar conectados por alguna arista, y ambos árboles son disjuntos. Por propiedades de la transformación de árbol, cualquier arista de G debe estar en la bolsa de algún nodo de $\mathcal{T}'$. Esto implica que cada una de las aristas que conecta a v con C_0 debe de estar en $\mathcal{T}'$. Esto también implica que cada uno de los vértices de C_0 también debe de estar contenido en al menos una bolsa de algún nodo de $\mathcal{T}'_{v^+}$. Lo anterior es una contradicción, porque esto quiere decir que $\mathcal{T}'_{C_0}$ y $\mathcal{T}'_{v^+}$ no son disjuntos, violando la suposición inicial. ■

Lema 4.2 *Si C es un clique maximal de G, el Algoritmo DP-Clique lo identifica en forma correcta.*

Demostración. La demostración es por contradicción. Suponga que C es un clique maximal en G y el Algoritmo DP-Clique no lo identifica.

Como C es un clique maximal en G, por el Lema 4.1, C debe estar contenido en la bolsa de al menos un nodo t en $\mathcal{T}'$. Sea $\mathcal{T}'_C$ el conjunto de nodos t de $\mathcal{T}'$ cuyas bolsas contienen a C. Note que $\mathcal{T}'_C$ es conectado. Sea t' el nodo de $\mathcal{T}'_C$ más cercano a la raíz de $\mathcal{T}'$ y sea t el padre de t'. Note que t debe ser un nodo eliminador porque éste ya no contiene a C. Como C es un clique maximal, no existe en la tabla $c^{t'}$ un clique C' tal que $C \subset C'$. Como el vértice eliminado w en t debe estar en C, el Algoritmo FIND-MAXIMAL-CLIQUE lo detecta. Lo cual es una contradicción a la suposición inicial ■

Durante el recorrido postorder de $\mathcal{T}'_C$, previo al llenado de t'_C, se detectan los cliques maximales en la tabla c^{t_C} con el Algoritmo FIND-CLIQUE-MAXIMAL que se utiliza como subrutina en DP-CLIQUE. En consecuencia, se encuentra a C.

4.6. Análisis del algoritmo DP-CLIQUE

Lema 4.3 *El algoritmo DP-CLIQUE identifica en forma correcta todos los cliques maximales de distancia* 1 *de G, en* $O\left((\tau+1)\cdot n\cdot 2^{\tau+1}\right)$ *u.t.*

Demostración. Del resultado de Fomin *et al.* (2019), se sabe que el número de nodos en $\mathcal{T}'$ es $O(n)$. El llenado de las tablas se realiza siguiendo un recorrido postorder de los nodos en $\mathcal{T}'$. Durante el recorrido, se obtienen los conjuntos clique maximales de distancia 1 en la tabla de cada nodo eliminador t, cuyo nodo hijo t' sea introductor o unión.

El tiempo de ejecución del llenado de cada tabla c^t, en el peor de los casos, es el siguiente:

- Caso 1. Si t es un nodo introductor, la mitad de la tabla c^t se llena copiando el contenido de la tabla hija. La segunda parte de la tabla c^t, se llena después de realizar la operación estrellita entre cada clique de la tabla hija y el vértice introducido. Cada operación estrellita requiere $O(\tau)$ pasos. Por lo tanto, llenar la tabla introductora más grande requiere $O(\tau\cdot 2^{\tau+1})$ u.t.
- Caso 2. Si t es un nodo eliminador, la tabla c^t se llena copiando el contenido de la mitad de la tabla hija. No se requiere ninguna operación adicional. Por lo tanto, llenar la tabla eliminadora más grande requiere $O(2^{\tau})$ u.t.

- Caso 3. Si t es un nodo unión, la tabla c^t se llena copiando el contenido de la tabla de cualquiera de sus hijos. Por lo tanto, llenar la tabla unión más grande requiere $O(2^{\tau+1})$ u.t.
- Caso 4. Si t es un nodo unión o introductor cuyo padre es eliminador, se ejecuta FIND-MAXIMAL-CLIQUE, cuyo tiempo de ejecución es $O\left((\tau+1)\cdot 2^{\tau+1}\right)$ u.t.

Como existen $O(n)$ tablas en $\mathcal{T}'$, llenar todas las tablas requiere $O\left((\tau+1)\cdot n\cdot 2^{\tau+1}\right)$ u.t. Éste también es el tiempo para enumerar todos los cliques maximales de distancia 1 en G. ■

Se puede encontrar el clique máximo de distancia 1 por medio de una búsqueda secuencial sobre todos los cliques maximales de distancia 1 en tiempo lineal.

Pseudocódigo 10: FIND-MAXIMAL-CLIQUE($c^{t'}, w, parameter_1, parameter_2$)

Entrada: Tabla $c^{t'}$, donde t' es el nodo hijo de t. El vértice eliminado w en la tabla c^t. Si t' es eliminador, $parameter_1 \leftarrow c^{t''}$, donde t'' es el hijo de t' y $parameter_2 \leftarrow w'$, donde w' es el vértice eliminado en t'. En caso contrario, $parameter_1 \leftarrow NIL$ y $parameter_2 \leftarrow NIL$.

Salida: Enumeración de cliques maximales en $c^{t'}$.

```
1  χ = |X_{t'}|;
2  for i = 0 to 2^χ − 1 do
3  |  CREATE-LIST(L(i));
4  |  if c^{t'}(i) forma un clique then
5  |  |  flag(i) ← 1;
6  |  |  for k = 1 to χ− NUM-ONES(i) do
7  |  |  |  LIST-INSERT(L(i), i^(k));
8  |  |  end
9  |  else
10 |  |  flag(i) ← 0;
11 |  end
12 end
13 Ordenar una copia de los índices i de acuerdo al número de unos en su
     representación binaria (i' ← i);
14 for i' = 0 to 2^χ − 1 do
15 |  if flag(i) = 1 then
16 |  |  if (L(i).head ≠ NIL ∧ ∀j ∈ L(i) | flag(j) = 0) ∨ (L(i).head = NIL) then
17 |  |  |  flag(i) ← 2;
18 |  |  |  if w está en el clique de c^{t'}(i) then
19 |  |  |  |  flag(i) ← 3;
20 |  |  |  |  if t' es un nodo eliminador y w' su vértice eliminado then
21 |  |  |  |  |  if clique en c^{t'}(i) ∗ w' es un clique en c^{t''} then
22 |  |  |  |  |  |  flag(i) ← 2;
23 |  |  |  |  |  end
24 |  |  |  |  end
25 |  |  |  end
26 |  |  end
27 |  end
28 end
```

Capítulo 5. Conclusiones

El trabajo de esta tesis presenta una solución al problema de enumeración de cliques maximales de distancia 1 en un grafo k-outerplanar. Este problema para un grafo general es $\mathcal{NP}$-Difícil, pero resulta polinomial para la clase de grafos estudiados, si k es una constante. También este capítulo resume las actividades realizadas durante el trabajo de investigación y discute el trabajo a futuro que esta línea de investigación puede tomar.

En esta tesis se propone el algoritmo DP-CLIQUE que utiliza la técnica de programación dinámica, y se auxilia de los algoritmos Katsikarelis (2013), Fomin *et al.* (2019) y FIND-MAXIMAL-CLIQUE.

Primero, con base en el algoritmo de Katsikarelis (2013), se realiza una descomposición de árbol del grafo de entrada G. La anchura de la descomposición de árbol $\mathcal{T}$ es menor o igual a $3k-1$, y se calcula en tiempo lineal. El algoritmo de descomposición de árbol para el grafo G depende de la k-outerplanaridad del grafo y del vértice de mayor grado.

Posteriormente, con base en las observaciones de Fomin *et al.* (2019), el autor de este trabajo propuso una serie de algoritmos que transforman una descomposición de árbol arbitraria $\mathcal{T}$ a una descomposición de árbol agradable $\mathcal{T}'$, en $O(n \cdot \tau)$ u.t. Los números de nodos en $\mathcal{T}$ y en $\mathcal{T}'$ tienen el mismo orden.

Finalmente, el algoritmo DP-CLIQUE llena la tabla c^t para cada nodo t en $\mathcal{T}'$, siguiendo un recorrido postorder. Cada tabla se llena de acuerdo a una serie de reglas, y éstas dependen del tipo de nodo; i.e., hoja, introductor, eliminador, y unión. Se utiliza el algoritmo FIND-MAXIMAL-CLIQUE para identificar todos los cliques maximales de distancia $\boldsymbol{\kappa} = 1$. Esta identificación se lleva a cabo durante el recorrido postorder en el árbol $\mathcal{T}'$. El tiempo de ejecución del algoritmo DP-CLIQUE es $O\left((\tau+1) \cdot n \cdot 2^{\tau+1}\right)$ u.t.

Para demostrar que DP-CLIQUE es correcto, primero se demuestra que cualquier clique en G también se encuentra contenido en al menos una bolsa de $\mathcal{T}'$. Esta demostración se hace por inducción y contradicción. Adicionalmente, se demuestra por contradicción que el algoritmo FIND-MAXIMAL-CLIQUE detecta todos los cliques maximales que aparecen en las tablas de cada nodo de t'.

Hasta donde el autor de este trabajo de investigación tiene conocimiento, el algoritmo propuesto es el primero que se basa en descomposición de árbol para resolver el problema de enumeración de cliques maximales de distancia 1 para grafos k-outerplanares. Los algoritmos del estado del arte existentes utilizan técnicas de branch and bound.

El algoritmo diseñado por el autor de este trabajo de investigación tiene un tiempo de ejecución que es equivalente al algoritmo diseñado por Eppstein *et al.* (2010). El algoritmo propuesto utiliza la anchura de árbol τ como el parámetro que no depende del tamaño. Los algoritmos del estado del arte utilizan la degeneración del grafo. Para el grafo outerplanar, la anchura de árbol es dos y la degeneración también es dos. Por lo que ambos algoritmos son equivalentes para este tipo de grafos. Una desventaja de nuestro algoritmo propuesto es que para un grafo k-outerplanar, la anchura de árbol se incrementa linealmente conforme aumenta la k; mientras que la degeneración del grafo se mantiene constante para cualquier grafo plano.

Una desventaja de la técnica empleada en este trabajo de investigación es que se supone la existencia de un algoritmo óptimo para obtener la descomposición de árbol del grafo de entrada. A pesar de que existen algoritmos para ciertos grafos, en general, este es un problema $\mathcal{NP}$-Difícil.

Una posible mejora del algoritmo FIND-MAXIMAL-CLIQUE podría ser la siguiente. Como la cardinalidad del clique máximo de un grafo plano es cuatro, en una tabla c^t de 2^τ columnas, sólo sería necesario revisar $\sum_{n=1}^{4} \binom{2^\tau}{n}$ columnas; i.e., no sería necesario revisar mintérminos de más de cuatro variables verificadas, porque ningún grafo plano tiene cliques de tamaño cinco o mayores.

Una línea de investigación futura es el de generalizar el algoritmo propuesto para que pueda resolver el problema de enumerar todos los cliques maximales de distancia $\boldsymbol{\kappa} > 1$. Algunos de los retos que hemos identificado son los siguientes. Se tendría que modificar el operador estrellita de manera que verifique que el conjunto resultante sea un clique de distancia $\boldsymbol{\kappa}$. Se conjetura que el algoritmo de Floyd-Warshall se podría utilizar para realizar dicho cálculo. En esta modificación, la entrada al operador estrellita modificado es un clique A de distancia $\boldsymbol{\kappa}$, y un vértice B. Con el algoritmo de Floyd-Warshall, se podría verificar si el nuevo vértice B tiene una distancia a lo más

κ de todos los vértices de A. Otro reto que habría que resolver es que los cliques de distancia κ pueden ser grandes, incluso $O(n)$, por lo que se tendría que diseñar un mecanismo eficiente para evitar ir arrastrando las soluciones parciales en cada una de las tablas.

Por otra parte, existen otras subclases de grafos planos en los cuales se puede trabajar este problema; el reto consiste en encontrar su descomposición de árbol con una anchura de árbol óptima o una cota ajustada a ésta.

Literatura citada

Batagelj, V. y Zaversnik, M. (2011). An $O(m)$ algorithm for cores decomposition of networks. *Advances in Data Analysis and Classification*, **5**(2): 129–145.

Behar, R. y Cohen, S. (2018). Finding all maximal connected s-cliques in social networks. En: *Advances in Database Technology – Extending Database Technology*. OpenProceedings.org, pp. 61–72.

Bron, C. y Kerbosch, J. (1973). Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, **16**(9): 575–577.

Butenko, S. y Prokopyev, O. (2007). On k-club and k-clique numbers in graphs. Reporte técnico, Texas A and M University.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., y Stein, C. (2009). *Introduction to algorithms*. 3ra. ed., The MIT Press.

Cygan, M., Fomin, F. V., Kowalik, Ł., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., y Saurabh, S. (2015). *Parameterized algorithms*. Springer.

Diestel, R. (2017). *Graph Theory*. 5ta. ed., Springer.

Eiben, A. E. y Smith, J. E. (2015). *Introduction to evolutionary computing*. 2da. ed., Springer. pp. 25–48.

Eppstein, D. y Strash, D. (2011). Listing all maximal cliques in large sparse real-world graphs. En: *International Symposium on Experimental Algorithms*. Springer, pp. 364–375.

Eppstein, D., Löffler, M., y Strash, D. (2010). Listing all maximal cliques in sparse graphs in near-optimal time. En: *International Symposium on Algorithms and Computation*. Springer, pp. 403–414.

Fomin, F. V., Lokshtanov, D., Saurabh, S., y Zehavi, M. (2019). *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press.

Gross, J. L., Yellen, J., y Zhang, P. (2014). *Handbook of graph theory*. 2da. ed., CRC press.

Hochbaum, D. S. (1997). *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company. pp. 102–104.

Karp, R. M. (1972). Reducibility among combinatorial problems. En: *Complexity of computer computations*. Springer, pp. 85–103.

Katsikarelis, I. (2013). Computing bounded-width tree and branch decompositions of k-outerplanar graphs. *Computing Research Repository (CoRR)*, **abs/1301.5896**.

Khuri, S. y Bäck, T. (1994). An evolutionary heuristic for the minimum vertex cover problem. En: *Genetic Algorithms within the Framework of Evolutionary Computation – Proc. of the KI-94 Workshop*. Saarbrücken, Germany, pp. 86–90.

Konc, J. y Janezic, D. (2007). An improved branch and bound algorithm for the maximum clique problem. *MATCH. Communications in Mathematical and in Computer Chemistry*, **4**(5): 569–590.

Lewis, H. R. y Papadimitriou, C. H. (1998). *Elements of the Theory of Computation*. 2da. ed., Prentice-Hall.

Luce, R. D. (1950). Connectivity and generalized cliques in sociometric group structure. *Psychometrika*, **15**(2): 169–190.

Matula, D. W. y Beck, L. L. (1983). Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, **30**(3): 417–427.

Moon, J. W. y Moser, L. (1965). On cliques in graphs. *Israel journal of Mathematics*, **3**(1): 23–28.

Nisse, N. (2018). Graph theory and optimization parameterized algorithms. `http://www-sop.inria.fr/members/Nicolas.Nisse/lectures/9parameterized.pdf`.

Pattillo, J., Youssef, N., y Butenko, S. (2013). On clique relaxation models in network analysis. *European Journal of Operational Research*, **226**(1): 9–18.

San Segundo, P., Lopez, A., y Pardalos, P. M. (2016). A new exact maximum clique algorithm for large and massive sparse graphs. *Computers & Operations Research*, **66**: 81–94.

Tomita, E. y Kameda, T. (2007). An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global optimization*, **37**(1): 95–111.

Tomita, E., Tanaka, A., y Takahashi, H. (2006). The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, **363**(1): 28–42.

Tomita, E., Matsuzaki, S., Nagao, A., Ito, H., y Wakatsuki, M. (2017). A much faster algorithm for finding a maximum clique with computational experiments. *Journal of Information Processing*, **25**: 667–677.

Wood, D. R. (2007). On the maximum number of cliques in a graph. *Graphs and Combinatorics*, **23**(3): 337–352.