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

Doctorado en Ciencias en Ciencias de la Computación

Estudio de factibilidad para resolver el problema 2-packing máximo en tiempo polinomial en grafos outerplanares

Tesis

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

Presenta:

Alejandro Flores Lamas

Ensenada, Baja California, México

2018

Tesis defendida por

Alejandro Flores Lamas

y aprobada por el siguiente Comité

Dr. José Alberto Fernández Zepeda

Director de tesis

Dr. Carlos Alberto Brizuela Rodríguez

Dr. Edgar Leonel Chávez González

Dr. Joel Antonio Trejo Sánchez

Dr. Jesús Favela Vara

Coordinador del Posgrado en Ciencias de la Computación

Dra. Rufina Hernández Martínez

Directora de Estudios de Posgrado

Resumen de la tesis que presenta Alejandro Flores Lamas como requisito parcial para la obtención del grado de Doctor en Ciencias en Ciencias de la Computación.

Estudio de factibilidad para resolver el problema 2-packing máximo en tiempo polinomial en grafos outerplanares

Resumen aprobado por:

Dr. José Alberto Fernández Zepeda
Director de tesis

Este trabajo de investigación se centra en el área de análisis y diseño de algoritmos para grafos, tanto centralizados como distribuidos. Sea $G = (V_G, E_G)$ un grafo no dirigido, donde $|V_G| = n$. Un subconjunto $\hat{S} \subseteq V_G$ es un conjunto 2-*packing* si para cada par de vértices en $\hat{S}$, la longitud del camino más corto entre ellos es de al menos tres aristas. Un conjunto 2-packing $\hat{S}$ es *maximal* si no existe otro conjunto 2-packing $\hat{S}'$ tal que $\hat{S} \subset \hat{S}'$. Un conjunto 2-packing es *máximo* si es el conjunto de mayor cardinalidad entre todos los conjuntos 2-packing maximales. En este documento se discute el problema de encontrar un conjunto 2-packing en algunos grafos planos en tiempo polinomial. Un *grafo plano* es el que se puede dibujar en un plano de tal forma que las aristas sólo se intersectan en sus extremos. El *cactus* es un grafo plano donde cualquier arista pertenece a lo más a un ciclo. Un grafo 1-*outerplanar* es un grafo plano donde todos sus vértices yacen en el borde externo del mismo. Un *grafo Halin* es un grafo plano que se construye al conectar las hojas de un dibujo plano de un árbol (con al menos cuatro vértices, ninguno de ellos de grado dos) para que formen un ciclo. La primera contribución del presente trabajo de investigación consiste del diseño de un algoritmo de programación dinámica que encuentra un conjunto 2-packing máximo en un grafo cactus en $O(n^2)$ unidades de tiempo. La segunda contribución consiste del diseño de un algoritmo que encuentra un conjunto 2-packing máximo en un grafo 1-outerplanar en $O(n)$ unidades de tiempo. Finalmente, la tercera contribución consiste del diseño de un algoritmo distribuido que encuentra un conjunto 2-packing maximal en un grafo Halin en $O(n)$ unidades de tiempo. Se conjetura que las técnicas empleadas para el desarrollo de estos algoritmos podrían servir como base para desarrollar algoritmos similares para el problema del k-packing maximal y máximo en este tipo de grafos en tiempo polinomial.

Palabras clave: Algoritmos para grafos, conjunto 2-packing, grafo 1-outerplanar, desmembración de árbol, cactus.

Abstract of the thesis presented by Alejandro Flores Lamas as a partial requirement to obtain the Doctor of Science degree in Computer Sciences.

Feasibility study to solve the maximum 2-packing set problem in polynomial time in outerplanar graphs

Abstract approved by:

Dr. José Alberto Fernández Zepeda
Thesis Director

This research project focuses on the analysis and design of graph algorithms, both centralized and distributed. Let $G = (V_G, E_G)$ be an undirected graph, where $|V_G| = n$. A subset $\hat{S} \subseteq V_G$ is a 2-packing set if for every pair of vertices in $\hat{S}$, the shortest path between them is at least three edges long. A 2-packing set $\hat{S}$ is maximal if it does not exist other 2-packing set $\hat{S}'$ such that $\hat{S} \subset \hat{S}'$. A maximum 2-packing set is the one of largest cardinality among all maximal 2-packing sets. This document addresses the problem of finding a 2-packing set in some planar graphs in polynomial time. A *planar graph* has an embedding on a plane in such a way that its edges intersect only at their endpoints. The *cactus* is a planar graph such that any edge belongs to at most one cycle. A *1-outerplanar* graph is a planar graph for which all its vertices lie on the boundary of the graph. A Halin graph is a planar graph constructed by connecting the leaves of a planar embedding of a tree (with at least four vertices, none of them of degree two) into a cycle. The first contribution of this work consists of the design of a dynamic programming algorithm that finds a maximum 2-packing set in a cactus graph in $O(n^2)$ time units. The second contribution consists of the design of an algorithm that finds a maximum 2-packing set on a 1-outerplanar graph in $O(n)$ time units. Finally, the third contribution consists of the design of a distributed algorithm that finds a maximal 2-packing set on a Halin graph in $O(n)$ time units. We conjecture that the techniques used for the design of these algorithms could serve as a basis to develop similar algorithms for the maximal and maximum k-packing set problem in these type of graphs in polynomial time.

Keywords: Graph algorithms, 2-packing set, 1-outerplanar graph, tree decomposition, cactus.

Dedicatoria

A mis padres

Agradecimientos

A Dios, por todos estos años, por todo aquello que me has dejado ver, ser y hacer.

A mis padres Rebeca y Juan, por impulsarme desde pequeño, creer siempre en mí y apoyarme en todo momento.

A mi director de tesis, el Dr. José Alberto Fernández Zepeda, mi más profundo respeto y sincera admiración por compartir desinteresadamente sus conocimientos. Gracias por su invaluable apoyo y guía.

A los miembros de mi comité de tesis: el Dr. Carlos Alberto Brizuela Rodríguez, el Dr. Edgar Leonel Chávez González y el Dr. Joel Antonio Trejo Sánchez. Les agradezco su tiempo, observaciones y sugerencias para mejorar mi formación.

A todos mis amigos y compañeros del Departamento de Ciencias de la Computación, especialmente a los del laboratorio de algoritmos. Nada de esto sería tan especial sin su apoyo y amistad.

Al Centro de Investigación Científica y de Educación Superior de Ensenada: por su apoyo durante mi estancia en esta institución y por la enseñanza académica.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por brindarme el apoyo económico para realizar mis estudios de doctorado. No. de becario: 275910.

Me disculpo por omitir más nombres aquí, son muchos para nombrarlos a todos y no sería apropiado olvidar a uno de ellos.

A todos ustedes, gracias.

Alejandro Flores Lamas.

Tabla de contenido

Página

Tabla de contenido (continuación)

Lista de figuras

Figura Página

Lista de tablas

Capítulo 1. Introducción

1.1. Preliminares

Un *problema computacional* consiste de una *entrada* codificada en algún alfabeto, donde lo que se busca es devolver una solución que satisfaga cierta propiedad; es decir, un problema computacional está descrito por la propiedad que la salida debe satisfacer con respecto a la entrada. Esta tesis se enfoca en dos tipos de problemas computacionales: de 'decisión' y de 'optimización'.

Los problemas de *decisión* se responden con un *sí* o *no*; es decir, lo que se pide es verificar si la entrada satisface cierta propiedad. Por su parte, los problemas de *optimización* buscan encontrar la mejor solución dentro del conjunto de todas las soluciones factibles.

Un *algoritmo* es una secuencia de pasos computacionales que transforman una entrada, mediante un procedimiento específico, en una salida (Cormen *et al.*, 2009). Donde la *complejidad* de un algoritmo es la cantidad de recursos necesarios para su ejecución; i.e, número de pasos, cantidad de espacio, ancho de banda en la red, entre otros. Para este trabajo se considera al *tiempo de ejecución* como la cantidad de pasos que requiere un algoritmo para su ejecución. Este parámetro se expresa en unidades de tiempo o *u.t.*

Un *sistema distribuido* es una colección heterogénea de componentes de hardware, software y datos interconectados por una red que proveen un servicio (Kshemkalyani y Singhal, 2008). Para los sistemas distribuidos, un algoritmo es un conjunto de acciones replicadas en cada nodo participante del sistema que dicta las interacciones entre ellos (Jiménez y Estrada, 2014). El tiempo de ejecución para los algoritmos distribuidos en un 'sistema síncrono' se mide en 'rondas'. Una *ronda* es la ejecución de una operación local en la cual los nodos del sistema envían un mensaje a sus vecinos (Tanenbaum y Steen, 2006; Ghosh, 2014).

Un algoritmo es *autoestabilizante* si en un sistema distribuido se garantiza la convergencia del sistema a un estado legal (y mantenerse en éste si no existen fallos) en un número finito de pasos, sin importar el estado inicial del mismo (Dijkstra, 1974).

Por convención, se dice que un algoritmo es *factible* si este se ejecuta en tiempo polinomial; i.e., si existe algún polinomio p, tal que el algoritmo requiere a lo más $p(n)$ pasos en entradas de longitud n. A lo largo de esta tesis se hará mención a tres clases de problemas:

- La clase **P** consiste de aquellos problemas que se pueden resolver en tiempo polinomial.
- La clase **NP** son los problemas que, dada una solución candidata, se pueden verificar si ésta es una solución correcta al problema en tiempo polinomial.
- La clase **NP-completo** son los problemas más difíciles en NP. Un problema L es NP-completo si 1) éste se encuentra en NP y 2) si cualquier problema en NP se puede expresar en términos de L a través de un algoritmo de transformación que corra en tiempo polinomial. Si sólo se cumple el punto 2) pero no necesariamente el 1), entonces el problema es **NP-difícil**.

Un *grafo* es un par $G = (V_G, E_G)$ de conjuntos tal que $E_G \subseteq V_G \times V_G$; es decir, los elementos de E_G son subconjuntos de dos elementos de V_G. A los elementos de V_G se les llama *vértices* (o *nodos* o *puntos*) del grafo G, mientras que a los elementos de E_G se les denomina *aristas*. El *orden* de G es el número de vértices en V_G, escrito como $|V_G| = n$, mientras que el número de aristas, $|E_G| = m$, es el *tamaño* de G. En esta definición de grafo se considera que las aristas no son dirigidas; es decir $\{u, v\} = \{v, u\}$. Asimismo, no se consideran autociclos, i.e., esto es una arista de un vértice a sí mismo (Diestel, 2018). Tampoco se consideran aristas múltiples, ya que un conjunto no contiene elementos repetidos.

Se dice que un vértice v es *incidente* a una arista $e \in E_G$ si $v \in e$. Al par de vértices $\{u, v\}$ incidentes en una arista e se les denomina *vértices extremos*, donde la arista e *conecta* a u con v o viceversa. Para este trabajo, la arista compuesta por el par de vértices $\{u, v\}$ se escriben como $(u, v) \in E_G$. Un *camino* es un grafo no vacío es una secuencia alternada de vértices y aristas en donde todos los vértices son distintos. El número de aristas en el camino es su *longitud*. Por lo tanto, la *distancia* entre dos vértices en G es la longitud del camino más corto entre ellos. Un *ciclo* es un camino

$\{x_0, x_1, \ldots, x_i\}$ en donde solamente x_0 y x_i son iguales. La *longitud* del ciclo es el número de aristas que lo componen.

Se dice que $G' = (V'_G, E'_G)$ es un subgrafo de $G = (V_G, E_G)$, denotado por $G' \subseteq G$, si $V'_G \subseteq V_G$, $E'_G \subseteq E_G$ y cada arista en E'_G tiene sus dos vértices extremos en V'_G. Un subgrafo *inducido* G' es un subgrafo de G de la forma $G-X$, donde $X \subset V_G$. El subgrafo inducido G' debe tener todas las aristas de G que unen dos vértices de V'_G. Un grafo G es *conectado* si no es vacío y cualquier par de vértices de V_G están conectados por un camino; caso contrario G es *desconectado*.

Un grafo *geométrico* es aquel cuyos vértices están representados como puntos distintos en cualquier posición del plano y cuyas aristas se dibujan como segmentos de línea recta, con posibles cruces (Pach, 2013); a cada región rodeadas por aristas se le llama *cara*. Un *grafo plano* es el que admite un dibujo en el plano de tal manera que las aristas sólo se intersectan en los vértices. Un *grafo geométrico plano* es un grafo plano $G = (V_G, E_G, F)$ encajado en el plano donde F es un conjunto de caras. Entre las suposiciones que se pueden realizar en estos grafos es que cada vértice tiene su par de coordenadas $[x, y]$ (Chávez *et al.*, 2004).

Un grafo sin ningún ciclo es un *bosque*, y un *árbol* es un grafo conectado sin ciclos. Si el grafo contiene exactamente un ciclo, entonces éste es un *grafo unicíclico* o simplemente *uniciclo*.

Un grafo $K = (V_K, E_K)$ es un *cactus* si cada arista $e \in E_K$ pertenece a lo más a un ciclo de K (es decir, cualquier par de ciclos adyacentes comparten a lo más un vértice). El cactus es un subconjunto de la familia de los grafos planos.

Un grafo es 1-*outerplanar* si se puede encajar en el plano de tal forma que todos sus vértices toquen la cara externa (Mitchell, 1979). Un empotramiento de G es r-outerplanar si es plano, y al borrar todos los vértices de la cara exterior, el empotramiento restante es $(r-1)$-outerplanar (Chekuri *et al.*, 2003). Note que bosques, árboles, ciclos y cactus pertenecen al grafo 1-outerplanar.

Dentro de los grafos 2-outerplanares está el grafo 'Halin'. Sea $T = (V_T, E_T)$ un árbol tal que $|V_T| \geq 4$ y que el grado de cada vértice $v \in V_T$ es distinto de dos. Sea E_C el conjunto de aristas que conectan todas las hojas de T en el orden cíclico dado por el

empotramiento (en el plano) de las hojas de T. El grafo Halin $H = (V_H, E_H)$, es aquel tal que $V_H \leftarrow V_T$ y $E_H \leftarrow \{E_T \cup E_C\}$ (Halin, 1971).

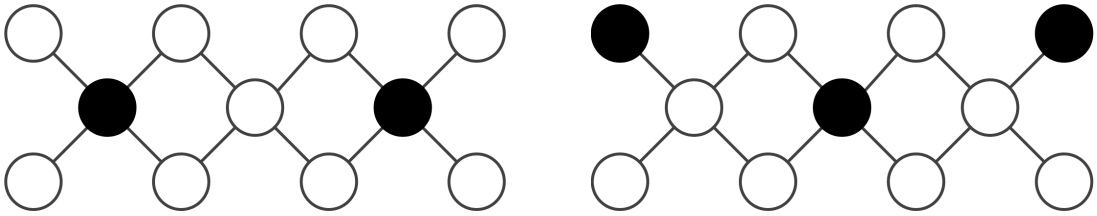
Un conjunto *k-packing* en G es una selección de vértices, tal que el camino más corto entre cada par de vértices seleccionados tiene al menos $k + 1$ aristas. Encontrar un conjunto k-packing máximo en un grafo arbitrario es un problema NP-difícil para cualquier k (Hochbaum y Shmoys, 1985); sin embargo, algunas clases de grafos admiten soluciones de tiempo polinomial para este tipo de problemas (por ejemplo, anillos y árboles) (Mjelde, 2004).

Un par de versiones restringidas bien conocidas de este problema son las siguientes. Cuando $k = 1$, el problema 1-*packing* también se llama el problema del *conjunto independiente* (Meir y Moon, 1975). Butenko (2003) discute algunas aplicaciones relacionadas con esta versión restringida del problema, tales como en recuperación de información, teoría de clasificación, visión por computadora, ingeniería biomédica y mercados financieros.

Cuando $k = 2$, el problema 2-*packing* también se llama el problema del *conjunto independiente fuerte* (Hochbaum y Shmoys, 1985). Existen varias aplicaciones para conjuntos 2-packing, especialmente cuando el problema a resolver requiere la exclusión mutua entre los vértices (Gairing *et al.*, 2004). Un ejemplo es el problema de asignación de frecuencia (Hale, 1980) cuyo propósito es evitar la interferencia de canales entre los transmisores.

Un conjunto 2-packing S es *maximal* si no existe otro conjunto 2-packing S' tal que $S \subset S'$. Un conjunto 2-packing es *máximo* si es el de mayor cardinalidad entre todos los conjuntos 2-packing maximales. Castro *et al.* (2011) usa la notación $\rho(G)$ para indicar la cardinalidad de un conjunto 2-packing máximo. Los vértices oscuros en los cactus de las figuras 1(a) y 1(b) son ejemplos de conjuntos 2-packing maximales. Además, los vértices resaltados de la Figura 1(b) también forman un conjunto 2-packing máximo. Observe que para este cactus pueden existir más de un conjunto 2-packing máximo. Hochbaum y Shmoys (1985) mostraron que encontrar un conjunto 2-packing máximo en un grafo arbitrario es al menos tan difícil como calcular el conjunto independiente máximo en el mismo grafo, es decir, un problema NP-difícil.

Otro concepto relacionado con el conjunto 2-packing es el ‘conjunto dominante’.



(a) Un conjunto 2-packing maximal en un cactus

(b) Un conjunto 2-packing máximo en un cactus

Figura 1. Ejemplos de un conjunto 2-packing maximal y máximo en un grafo cactus.

Dado un grafo $G = (V_G, E_G)$, un conjunto de vértices $D \subseteq V_G$ es un *conjunto dominante* si cada vértice $x \notin D$ es adyacente a al menos un vértice $y \in D$. Algunos autores suelen usar $\gamma(G)$ para denotar al conjunto dominante mínimo (Chellali *et al.*, 2012). Imrich *et al.* (2008) mostraron que la relación $\gamma(G) \geq \rho(G)$ es válida para todo grafo G. Asímismo, Haynes *et al.* (1998) demostraron que el conjunto dominante mínimo es el problema dual, en el sentido de la formulación de la programación lineal, al problema del conjunto 2-packing máximo.

1.2. Objetivo general

El punto central de esta tesis es evaluar la factibilidad de encontrar un conjunto 2-packing máximo en el grafo 1-outerplanar. Por tal razón, se revisó el trabajo existente en la literatura y con base en ello se diseñó una propuesta de solución. Esta propuesta de solución se basa en transformar un grafo G de tal forma que el nuevo grafo transformado tenga una estructura semejante a la de un árbol. A partir de esta estructura, el problema suele simplificarse y, adicionalmente, posibilita la deducción de ciertas propiedades de conectividad de G.

El procedimiento anterior permite aplicar técnicas bien conocidas de la literatura para construir algoritmos eficientes que resuelvan el problema planteado en el árbol generado. En muchas ocasiones, estos problemas son del tipo NP-difícil para el grafo general, como en el caso del conjunto 2-packing. No obstante, mediante esta transformación de G, algunos algoritmos pueden ejecutarse en tiempo polinomial en esta estructura.

1.3. Trabajo relacionado

Esta sección discute algunos algoritmos existentes que resuelven el problema del conjunto 2-packing maximal y máximo así como algunas de sus variantes para distintos grafos.

1.3.1. Conjunto *k*-packing maximal

En el contexto autoestabilizante, Shi (2012) diseñó un algoritmo autoestabilizante que encuentra un conjunto 2-packing maximal en un grafo arbitrario en $O(n \times m)$ u.t., suponiendo el calendarizador adversario. Trejo-Sánchez y Fernández-Zepeda (2012) propusieron un algoritmo autoestabilizante que calcula un conjunto 2-packing maximal en un cactus en $O(D_K)$ rondas. Posteriormente, Trejo-Sánchez y Fernández-Zepeda (2014) diseñaron un algoritmo autoestabilizante que encuentra un conjunto 2-packing maximal en un grafo geométrico outerplanar en $O(n)$ u.t. Turau (2012) propuso un mecanismo que permite el diseño de un algoritmo autoestabilizante que calcula un conjunto 2-packing máximo en un grafo arbitrario en $O(n \times m)$ u.t.

Con respecto al problema general de calcular un conjunto k-packing maximal, Manne y Mjelde (2006) diseñaron un algoritmo que resuelve este problema en tiempo exponencial en un grafo arbitrario. Goddard *et al.* (2008) propuso un mecanismo que permite leer información de vértices a una distancia k en un grafo arbitrario. Cuando $k = 2$, este mecanismo permite diseñar un algoritmo que encuentra un conjunto 2-packing máximo en $O\left(n^2 \times m\right)$ u.t. Tanto Trejo-Sánchez y Fernández-Zepeda (2012) y Shi (2012) proveen una lista detallada de algoritmos para calcular un conjunto 2-packing maximal.

1.3.2. Conjunto *k*-packing máximo

Fleischner *et al.* (2010) mostraron que encontrar un conjunto independiente máximo en grafos Hamiltonianos 3 y 4-regulares es un problema NP-completo. Este resultado se mantiene para un grafo arbitrario para ‘conjuntos independientes grandes,’

i.e., conjuntos cuyo tamaño es al menos $n/(d-1)$, donde el grado máximo de cualquier vértice es a lo más d. Posteriormente, Brause *et al.* (2015) estudiaron la complejidad del conjunto independiente máximo en subclases de grafos subcúbicos. Su resultado principal es un algoritmo polinomial para resolver dicho problema en esos grafos. Xiao y Nagamochi (2016) presentaron un algoritmo cuyo tiempo de ejecución es $1.1736^n n^{O(1)}$ que calcula un conjunto independiente máximo en grafos donde el grado de cualquier vértice es a lo más cinco. Finalmente Lozin (2017) analizó algunas propiedades y restricciones en un grafo que pueden conducir a resolver los problemas del empatamiento máximo y conjunto independiente en algunas familias de grafos, entre ellas la del grafo 1-outerplanar.

Otros enfoques que estudian este tipo de problemas son las heurísticas y métodos probabilísticos. Por ejemplo, Xu *et al.* (2006) propusieron un algoritmo de recocido simulado para encontrar, con alta probabilidad, una solución óptima para el conjunto independiente. Adicionalmente, Murat y Paschos (2002) estudiaron la complejidad de resolver, mediante un enfoque probabilista, el problema del conjunto independiente usando a priori alguna estrategia de optimización.

Un problema relacionado al conjunto independiente máximo es el conjunto independiente con pesos máximo. Sakai *et al.* (2003) presentan tres algoritmos voraces que calculan conjuntos independientes que satisfacen al menos cierto peso. Otros investigadores estudian este problema y sus variantes en ciertas topologías de grafos (Orlovich *et al.*, 2011; Karthick y Maffray, 2017; Karthick, 2016; Keil *et al.*, 2017; Mosca, 2017).

El algoritmo propuesto por Trejo-Sánchez y Fernández-Zepeda (2014) encuentra un conjunto 2-packing máximo cuando el grafo 1-outerplanar de entrada es un ciclo. Mjelde (2004) diseñó un algoritmo de programación dinámica que resuelve el problema del conjunto k-packing máximo en un árbol. Su algoritmo primero enraiza el árbol, luego a través de un recorrido en postorden, para cada vértice v, calcula la cardinalidad del conjunto k-packing máximo para cada subárbol enraizado en v. Luego, a través de un recorrido en preorden, el algoritmo marca los vértices en el árbol que pertenecen al conjunto buscado. Su algoritmo se ejecuta en $O(k \times n)$ u.t.

1.3.3. Pruebas de planaridad

Para algunos problemas de grafos, presentes en este trabajo, es importante conocer si el grafo de entrada G pertenece a una familia Γ de grafos; e.g., si G es 'desmembrable' o es plano. Para saber si un grafo es plano existen *pruebas de planaridad* que determinan si el grafo tiene una representación plana. Algunas de estas pruebas se basan en caracterizaciones mediante grafos prohibidos. Por ejemplo, un grafo es plano (Kuratowski, 1930) si no contiene como subgrafo una subdivisión del grafo completo de 5 vértices K_5 o del grafo bipartito de 6 vértices $K_{3,3}$. Para el grafo plano los subgrafos prohibidos son K_5 y $K_{3,3}$. De forma semejante, el grafo outerplanar se caracteriza por los subgrafos prohibidos K_4 y $K_{2,3}$.

Otros enfoques para verificar la planaridad de un grafo se basan en algoritmos que realizan recorridos o que aplican reglas de reducción. Hopcroft y Tarjan (1974) presentan un algoritmo de $O(n)$ u.t. que emplea una búsqueda en profundidad (DFS) para determinar si un grafo tiene una representación plana. Asimismo, en $O(n)$ u. t. es posible reconocer mediante el algoritmo de Gavril (1975) si un grafo es cordal y posteriormente, modificar el algoritmo para encontrar una representación plana.

Para reconocer al grafo 1-outerplanar, existen al menos dos enfoques. El primero es el algoritmo de Brehaut (1977) que se basa en el trabajo de Hopcroft y Tarjan (1974) que en tiempo lineal encuentra tanto una representación plana y la cara externa del grafo 1-outerplanar. El segundo enfoque es el trabajo de Mitchell (1979) que también identifica si el grafo es 1-outerplanar 'maximal'. Un grafo 1-outerplanar es *maximal* si al agregar una arista adicional entre cualquier par de vértices no adyacentes, deja de ser 1-outerplanar.

El algoritmo de Mitchell (1979) (MOP-TEST) no se basa en DFS, pero sí recorre los vértices del grafo. MOP-TEST maneja 2 estructuras, *LIST* y *PAIRS*, además de la lista de vértices y aristas del grafo. De forma general, MOP-TEST funciona de la siguiente manera:

Primero se contabilizan las aristas del grafo. Si $|E_G| > 2n-3$ entonces G no es outerplanar. Si $|E_G| \leq 2n-3$ en la estructura llamada *LIST* se almacenan los vértices que tienen grado 2. Luego, por cada vértice v en *LIST* se agregan a *PAIRS* los vecinos

de v, se revisa si dicha arista existe en el grafo y se elimina v de E_G. Cada vez que se elimina v de V_G se revisa si existen nuevos vértices de grado 2, si esto sucede, se agregan a *LIST*. El procedimiento continúa hasta que el número de vértices en *LIST* menos el índice del elemento procesado (l) sea menor que 2; i.e., $l \leq n-2$. Al finalizar este procedimiento se revisa que todas las aristas en *PAIRS* se encuentren en E_G. De ser así, el grafo es 1-outerplanar maximal, caso contrario sólo es 1-outerplanar.

Es posible modificar los algoritmos de Brehaut (1977) y Mitchell (1979) para encontrar la cara externa del grafo 1-outerplanar. Posteriormente, se puede numerar las aristas del grafo para utilizar el algoritmo de Trejo-Sánchez y Fernández-Zepeda (2014) para encontrar un conjunto 2-packing maximal en el grafo 1-outerplanar. Note que el algoritmo de Trejo-Sánchez y Fernández-Zepeda (2014) está diseñado para ejecutarse en grafos geométricos; no obstante, conociendo la cara externa del grafo y una numeración de aristas es posible ejecutar su algoritmo en el grafo 1-outerplanar no geométrico. Una idea semejante se usa en el Capítulo 3 para el grafo Halin.

1.4. Motivación

Ahora se presentan las aplicaciones e importancia teórica de los conjuntos k-packing y de los grafos 1 y 2-outerplanares.

Respecto a los conjuntos, Butenko (2003) muestra la aplicación del conjunto independiente en áreas como recuperación de información, teoría de la clasificación e ingeniería biomédica. Michael y Battiston (2009) y Butenko (2003) presentaron al conjunto independiente como herramienta para modelar interacciones entre mercados cuyos precios están en fluctuación. Gairing *et al.* (2004) menciona la aplicación del conjunto 2-packing en problemas que requieren exclusión mutua, como la asignación de recursos donde se requiere que los vecindarios no se empalmen y como subrutina en otros algoritmos. Particularmente Hale (1980) emplea la exclusión de vecindarios para designar servidores en la red y asignar diferentes frecuencias a estaciones transmisoras de radio para minimizar interferencias.

Los grafos 1 y 2-outerplanares modelan distintos escenarios. Por ejemplo, el problema de ubicación de instalaciones (Manne y Mjelde, 2006), asignación de red (Kariv

y Hakimi, 1979), modelado de ciertos tipos de redes telefónicas (Koontz, 1980), diseño de circuitos VLSI (Bodlaender, 1993) y comparador de genomas relacionados (Paten *et al.*, 2010).

Además, los grafos 1 y 2-outerplanares también tienen importancia teórica. Diversos problemas NP-difícil para topologías generales admiten soluciones en tiempo polinomial (Zmazek y Žerovnik, 2004; Ben-Moshe *et al.*, 2005) en estos grafos, incluso lineal por ejemplo: el conjunto k-packing máximo en árboles (Mjelde, 2004), el conjunto dominante mínimo para árboles (Cockayne *et al.*, 1975; Laskar *et al.*, 1984) y en cactus (Hedetniemi *et al.*, 1986), el problema del vendedor viajero (Cornuéjols *et al.*, 1983) y árbol de Steiner (Winter, 1987) en grafos Halin, entre otros.

1.5. Organización y contribución

Esta tesis presenta varios algoritmos para calcular un conjunto 2-packing en sus versiones máximo y maximal en los grafos 1 y 2-outerplanares. El enfoque que se sigue se basa en distintas desmembraciones del grafo. De forma general, el resto de este trabajo de investigación se organiza de la siguiente manera:

- En el Capítulo 2 se propone el algoritmo MAXIMUM-2-PACK-CACTUS que encuentra un conjunto 2-packing máximo en un cactus arbitrario en $O\left(n^2\right)$ u.t. Para diseñar este algoritmo, primero se propuso el algoritmo MAXIMUM-2-PACK-UNICYCLE que encuentra un conjunto 2-packing máximo en un uniciclo. También se propone el algoritmo UNICYCLE-TO-TREE que transforma un uniciclo U con un marcado de vértices, que representa un conjunto 2-packing máximo para U, a un árbol $T(U)$ que tiene el mismo marcado de vértices y que también representa un conjunto 2-packing máximo para $T(U)$. Esta transformación es útil para demostrar que el algoritmo MAXIMUM-2-PACK-CACTUS es correcto.
- El Capítulo 3 propone el algoritmo distribuido MAXIMAL_2-PACKING-SET_HALIN que calcula un conjunto 2-packing maximal en el grafo Halin en $O(n)$ u.t. Para alcanzar este objetivo, primero se encuentra una de las caras externas del grafo mediante reglas de contracción de vértices y aristas. Posteriormente, se realiza una descomposición de orejas del grafo siguiendo la cara externa previamente

encontrada. Finalmente, se aplica un algoritmo de la literatura, sobre las orejas del grafo, que calcula el conjunto 2-packing maximal.

- En el Capítulo 4 se discuten las propiedades de los ‘grafos desmembrables’, ‘k-árbol parcial’, ‘lógica monádica de segundo orden’ y la ‘desmembración de árbol’ de un grafo. Lo anterior con el objetivo de proveer las bases teóricas para analizar la factibilidad de resolver ciertos problemas computacionales en algunas familias de grafos. Particularmente, se hace énfasis en la ‘desmembración de árbol’. Se discuten algunos conceptos de conectividad derivados de esta estructura y se estudia una de sus representaciones alternas. Dicha representación es más restrictiva, pero facilita el diseño de algoritmos de programación dinámica que se ejecuten sobre ésta. Asimismo, se presenta brevemente a la ‘complejidad paramétrica’ como una forma de lidiar con problemas NP-difíciles y que comprende a los conceptos presentados en el capítulo.

- El Capítulo 5 propone un algoritmo ‘FPT’ de programación dinámica que calcula un conjunto 2-packing máximo sobre una desmembración de árbol agradable del grafo 1-outerplanar. El conjunto encontrado (en la desmembración) también es un conjunto 2-packing máximo en el grafo 1-outerplanar. Para diseñar este algoritmo, se describen ecuaciones por cada tipo de nodo de la desmembración. La demostración formal del algoritmo se hace mediante la validación de las ecuaciones propuestas. Finalmente, se presenta el análisis de complejidad del algoritmo propuesto, concluyendo que es lineal bajo ciertas suposiciones.

- El Capítulo 6 resume el trabajo de investigación realizado y provee algunas observaciones finales relacionadas al trabajo de este documento. Asimismo, se discuten brevemente algunos problemas no resueltos y las posibles rutas que puede tomar el trabajo futuro sobre esta línea de investigación.

Capítulo 2. Conjunto 2-packing máximo en un cactus

2.1. Terminología y notación

En este documento, se llama *vértice marcado* al vértice que pertenece a un conjunto 2-packing. Un *componente conectado* C de G es un subgrafo maximal de G en el que existe un camino entre cada par de vértices u, v de C. Un *vértice de corte* x de G es aquel cuya eliminación de G, junto con todas las aristas incidentes a x, aumenta el número de componentes conectados. Un *bloque* o *componente biconectado* de G es un subgrafo maximal de G sin vértices de corte. Para el grafo cactus, cada bloque es un ciclo simple o un 'clique' de tamaño dos. Por lo tanto, en el contexto de este trabajo, se hace referencia a los bloques como *bloque cíclico* o *bloque clique*, respectivamente. Un *clique* de tamaño q en un grafo G es un subgrafo completo de G con q vértices.

Después de identificar los bloques de un grafo G, es posible construir un 'árbol de bloques.' En un *árbol de bloques*, T_B, cada vértice representa un bloque de G, y existe una arista entre algún par de vértices si los bloques representados comparten un vértice, cuidando el no generar ciclos. Se puede elegir arbitrariamente un bloque de T_B como un bloque raíz y enraizar a T_B como se explica en la sección 2.2. Enraizar a T_B genera una relación ancestro-descendiente entre los bloques. En este documento, N denota el número de bloques en el grafo G

Se supone que cada vértice del cactus de entrada contiene un identificador único extraído del conjunto $\{1, 2, \ldots, n\}$. Cada vértice v_x, además de su identificador único, x, tiene una etiqueta de bloque única. La *etiqueta de bloque*, $v_{i,j}$, del vértice v_x consiste en el número de postorden, i, del bloque que contiene el vértice v_x y una etiqueta local j, que identifica cada vértice dentro del bloque. Hay que tener en cuenta que v_x puede tener más de una etiqueta de bloque. A continuación, se definen las etiquetas locales de los vértices en cada bloque del cactus.

Si $B_i \neq B_N$ es un bloque clique, entonces el vértice $v_{i,0}$ es el vértice de B_i más cercano al bloque raíz, B_N. El vértice $v_{i,0}$ es el *vértice origen* del bloque B_i. El vértice $v_{i,1}$ es el vértice restante. Si $B_i \neq B_N$ es un bloque cíclico, entonces $v_{i,0}, v_{i,1}, \ldots, v_{i,|B_i|-1}$

son los vértices del ciclo. El vértice $v_{i,0}$ es el vértice de B_i más cercano al bloque B_N. Los otros vértices de B_i se numeran secuencialmente comenzando en el vértice vecino de $v_{i,0}$, dentro de B_i, con el identificador más bajo. Para el bloque clique o cíclico B_N, el vértice $v_{N,0}$ es el que tiene el identificador más pequeño en B_N.

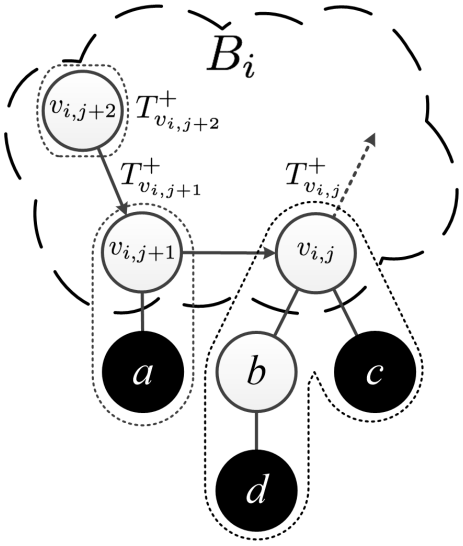
Sea $C_{v_{i,j}}$ el conjunto de bloques hijos de B_i conectados a $v_{i,j}$. Sea C_{B_i} el conjunto de bloques hijos de B_i (es decir, $C_{B_i} = \bigcup_{\forall j} C_{v_{i,j}}$). Sea $C^+_{v_{i,j}}$ la unión de $C_{v_{i,j}}$ y todos los bloques descendientes de $C_{v_{i,j}}$. Sea $C^+_{B_i}$ el conjunto de todos los bloques descendientes de B_i. Sea $T^+_{v_{i,j}}$ el grafo inducido por todos los vértices de $C^+_{v_{i,j}}$.

El algoritmo PROCESS-CYCLIC-BLOCK, descrito en la sección 2.3, elimina una arista de un bloque cíclico B_i, sea $e_{i,h} \in E_{B_i}$ dicha arista, y genera el árbol t^h_i. Este algoritmo realiza dicha acción como parte del algoritmo MAXIMUM-2-PACK-UNICYCLE. También se supone que el vértice de origen $v_{i,0}$ es la raíz de t^h_i. De forma similar, el vértice de origen $v_{i,0}$ es la raíz del árbol para un bloque cíclico B_i.

Casi todos los algoritmos descritos en este documento suponen que cada vértice $v_{i,j} \in V_{B_i}$ tiene una tabla $\mathcal{T}_{v_{i,j}}$. Si $j \neq 0$, la tabla tiene tres filas y dos columnas; de lo contrario, tiene cuatro filas y tres columnas. El contenido y la función de esta tabla es similar a la tabla M utilizada por el algoritmo de Mjelde (Mjelde, 2004). Para hacer referencia al elemento del renglón r y columna c de $\mathcal{T}_{v_{i,j}}$, se usa la notación $\mathcal{T}_{v_{i,j}}[r][c]$. Cada elemento $\mathcal{T}_{v_{i,j}}[r][c]$ tiene tres campos llamados 'key', 'set' y 'dist'. Para hacer referencia a cada campo, se usa $\mathcal{T}_{v_{i,j}}[r][c].key$, $\mathcal{T}_{v_{i,j}}[r][c].set$, y $\mathcal{T}_{v_{i,j}}[r][c].dist$. Por ejemplo, en la Figura 2(b), la tabla $\mathcal{T}_{v_{i,j}}[0][0]$ contiene $\{2, (c, d), 1\}$ que corresponden a los campos *key*, *set* y *dist*, respectivamente.

El campo *key* del elemento $\mathcal{T}_{v_{i,j}}[r][c]$ contiene un número entero que representa el número de vértices marcados en un 'subgrafo específico', tal que el vértice marcado más cercano a $v_{i,j}$ está a una distancia igual o mayor que r en dicho 'subgrafo específico', donde $0 \leq r \leq 3$. El campo *set* del elemento $\mathcal{T}_{v_{i,j}}[r][c]$ almacena una lista que contiene los identificadores de todos los vértices marcados en un 'subgrafo específico'. El campo *dist* del elemento $\mathcal{T}_{v_{i,j}}[r][c]$ contiene un número entero que representa la distancia entre el vértice $v_{i,j}$ y el vértice marcado más cercano en un 'subgrafo específico'.

Cada una de las tres columnas c corresponden a un 'subgrafo específico', para



(a) Un grafo con un bloque cíclico B_i. Cada círculo resaltado representa un vértice marcado en el grafo

$\mathcal{T}_{v_{i,j}}$

r \ c	0	1
0	$\{2,(c,d),1\}$	$\{3,(a,c,d),1\}$
1	$\{2,(c,d),1\}$	$\{3,(a,c,d),1\}$
2	$\{1,(d),2\}$	$\{2,(a,d),2\}$

(b) Tabla $\mathcal{T}_{v_{i,j}}$. Cada elemento de la tabla almacena tres campos $\{key, set, dist\}$. La columna 0 está asociada al subárbol $T^+_{v_{i,j}}$ y la columna 1 a $T^+_{v_{i,j}} \cup T^+_{v_{i,j+1}} \cup T^+_{v_{i,j+2}}$

Figura 2. Un ejemplo que explica el significado de la tabla $\mathcal{T}_{v_{i,j}}$.

$0 \leq c \leq 2$, como se muestra a continuación.

- El *subgrafo asociado con la columna 0* de $\mathcal{T}_{v_{i,j}}$ es $T^+_{v_{i,j}}$. Cuando $T^+_{v_{i,j}}$ está vacío, $\mathcal{T}_{v_{i,j}}[0][0]$ se establece inicialmente como $\{1,(v_{i,j}),0\}$ y los elementos de las filas restantes en la columna 0 se establecen en $\{0,(\emptyset),\infty\}$. Esta información inicial de $\mathcal{T}_{v_{i,j}}[r][0]$ se llama *configuración nula*, para toda r.
- El *subgrafo asociado con la columna 1* de $\mathcal{T}_{v_{i,j}}$ es el grafo inducido por todos los vértices de $C^+_{v_{i,j}} \cup \left(\bigcup_{\forall k} C^+_{v_{i,k}}\right)$, donde $v_{i,k}$ es un descendiente de $v_{i,j}$ en el árbol t^h_i.
- El *subgrafo asociado con la columna 2* de $\mathcal{T}_{v_{i,0}}$ es el grafo inducido por todos los vértices de $B_i \cup C^+_{B_i}$.

Ejemplo 1. Considere el grafo y la tabla $\mathcal{T}_{v_{i,j}}$ de la Figura 2. Los vértices $v_{i,j}$, $v_{i,j+1}$, $v_{i,j+2} \in B_i$, donde $v_{i,j+1}$ y $v_{i,j+2}$ son descendientes de $v_{i,j}$ en t^h_i. Los vértices a, b, c, y d pertenecen a los bloques que son descendientes del bloque B_i. Los elementos de $\mathcal{T}_{v_{i,j}}[r][0]$ almacenan $\{1,(d),2\}$, $\{2,(c,d),1\}$, $\{2,(c,d),1\}$ para $r = 2, 1$, y 0, respectivamente. Los elementos de $\mathcal{T}_{v_{i,j}}[r][1]$ almacenan $\{2,(a,d),2\}$, $\{3,(a,c,d),1\}$, $\{3,(a,c,d),1\}$ para $r = 2, 1$, y 0, respectivamente.

Se definen dos operadores, ‘suma’ y ‘resta’. La *suma* de dos elementos, denotado por “$\bullet$”, es la suma algebraica de los valores del campo *key* de ambos elementos, la concatenación de las listas del campo *set* de ambos elementos y, el mínimo de los valores del campo *dist* de ambos elementos. De forma semejante, la *resta* de dos elementos de la tabla, denotado por “$\circ$”, es la siguiente. Para el campo *key*, la resta algebraica en lugar de la suma. Para el campo *set*, el borrado de la segunda lista de la primera en lugar de la concatenación de las listas. Para el campo *dist*, la distancia entre $v_{i,j}$ y el vértice marcado más cercano en la lista resultante del campo *set*.

El *marcado* $M^h_{i,j}[c] = \left(S^h_{i,j}[c], d^h_{i,j}[c]\right)$ para el subgrafo asociado con la columna c, considerando al vértice $v_{i,j}$ como origen, es un par ordenado que consiste de dos parámetros. El primero es un conjunto de vértices marcados, $S^h_{i,j}[c]$, en el subgrafo asociado con la columna c. El segundo es un parámetro, $d^h_{i,j}[c]$, que representa la distancia entre $v_{i,j}$ y el vértice marcado más cercano en $S^h_{i,j}[c]$, llamado $v^{\circledcirc}_{i,j}$.

Para el vértice $v_{i,j}$, se busca el marcado $M^h_{i,j}[c]$ de mayor cardinalidad. Este marcado se almacena en el renglón 0 de $\mathcal{T}_{v_{i,j}}$, donde $S^h_{i,j}[c] = \mathcal{T}_{v_{i,j}}[0][c].set$ y $d^h_{i,j}[c] = \mathcal{T}_{v_{i,j}}[0][c].dist$.

Cuando el subgrafo asociado con la columna c no se refiere al árbol de expansión de algún bloque cíclico B_i, se omite el superíndice “h”.

La cardinalidad de un marcado $M^h_{i,j}[c]$ es mayor que la cardinalidad de un marcado $\left(M^h_{i,j}[c]\right)'$, denotado por $\left|M^h_{i,j}[c]\right| > \left|\left(M^h_{i,j}[c]\right)'\right|$, si $\left|S^h_{i,j}[c]\right| > \left|\left(S^h_{i,j}[c]\right)'\right|$. En caso de empate, $\left|M^h_{i,j}[c]\right| > \left|\left(M^h_{i,j}[c]\right)'\right|$ si $d^h_{i,j}[c] > \left(d^h_{i,j}[c]\right)'$. Si el empate persiste, $\left|M^h_{i,j}[c]\right| > \left|\left(M^h_{i,j}[c]\right)'\right|$ si el índice del bloque que contiene el vértice marcado más cercano a $v_{i,j}$ en $S^h_{i,j}[c]$ es más pequeño que el índice del bloque que contiene al correspondiente vértice marcado más cercano en $\left(S^h_{i,j}[c]\right)'$. Dados dos elementos, $\mathcal{T}_{v_x}[r][c]$ y $\mathcal{T}_{v_y}[r][c]$, la función $\max\left(\mathcal{T}_{v_x}[r][c], \mathcal{T}_{v_y}[r][c]\right)$ escoge el elemento en el renglón r y columna c que tiene el marcado con mayor cardinalidad.

Dos árboles enraizados $T = (V_T, E_T)$ y $T' = (V_{T'}, E_{T'})$ enraizados en r y r', respectivamente, son isomorfos si existe una biyección $f : V_T \to V_{T'}$ tal que para toda $(u, v) \in E_T$ si y sólo si $(f(u), f(v)) \in E_{T'}$ y $f(r) = r'$. Para este trabajo, se fija el grafo de entrada para facilitar la comparación entre distintos métodos de marcado cuando se ejecutan sobre árboles isomorfos.

2.2. Algoritmos para el procesamiento de bloques

Esta sección describe el procedimiento de marcado de vértices en el bloque B_i, donde B_i es un bloque cíclico o un árbol de expansión generado de un bloque cíclico. Primero, se describe el algoritmo ROOTED-BLOCK-TREE-POSTORDER. Este procedimiento encuentra un árbol de bloques en el grafo y etiqueta sus bloques de acuerdo a su numeración postorden. Posteriormente, se describe el algoritmo INITIALIZE-BLOCK-TABLES. Este procedimiento incorpora en el bloque actual la información de marcado de los bloques previamente procesados.

Finalmente, se describen los algoritmos PROCESS-CLIQUE-BLOCK y MAXIMUM-2-PACK-TREE. Estos procedimientos llevan a cabo el marcado de un bloque clique y de un árbol generado a partir de un bloque cíclico, respectivamente. El marcado generado por estos algoritmos representa un conjunto 2-packing máximo en el grafo inducido por los vértices del conjunto $\left\{B_i \cup C^+_{B_i}\right\}$.

Los algoritmos descritos en esta sección son fundamentales para describir los algoritmos MAXIMUM-2-PACK-UNICYCLE y MAXIMUM-2-PACK-CACTUS que se presentan en las Secciones 2.3 y 2.5, respectivamente.

2.2.1. ROOTED-BLOCK-TREE-POSTORDER

El algoritmo ROOTED-BLOCK-TREE-POSTORDER, mostrado en el Pseudocódigo 1, consiste de una colección de algoritmos básicos cuya ejecución es necesaria antes de ejecutar los algoritmos que se describen en las Secciones 2.3 y 2.5. La entrada para este algoritmo es un cactus conectado, no dirigido $K = (V_K, E_K)$ de orden n. La salida es un árbol de bloques T_B, enraizado en algún bloque arbitrario, y una etiqueta en cada bloque que representa su numeración postorden.

La línea 2 del Pseudocódigo 1 identifica los bloques del grafo de entrada mediante un algoritmo basado en búsqueda en profundidad (DFS) en $O(n)$ u.t. (Hopcroft y Tarjan, 1973). La línea 3 selecciona un bloque arbitrario B_x de K como bloque raíz en $O(1)$ u.t. La línea 4 usa el algoritmo descrito en (Papamanthou y Tollis, 2008) para generar un árbol de bloques enraizado T_B, tomando B_x como el bloque raíz, en $O(n)$ u. t. Finalmen-

Pseudocódigo 1: ROOTED-BLOCK-TREE-POSTORDER(K)

Entrada: Un cactus conectado, no dirigido K.
Salida : Un árbol de bloques T_B enraizado en algún bloque arbitrario y una etiqueta en cada bloque que representa su numeración postorden.

```
1 begin
2     Identificar los bloques de K mediante el algoritmo DFS;
3     Elegir un bloque arbitrario B_x como el bloque raíz;
4     Aplicar un algoritmo DFS iniciando en B_x para generar un árbol enraizado de bloques T_B;
5     Etiquetar cada bloque de T_B usando su numeración en postorden;
6     return T_B;
7 end
```

te, la línea 5 calcula el número postorden de cada bloque en $O(n)$ u.t. (Cormen *et al.*, 2009). Sean $\{B_1, B_2, \ldots, B_N\}$ el conjunto de bloques de T_B ordenados de acuerdo a su número postorden. Note que B_N es la raíz de T_B. El tiempo de ejecución total de ROOTED-BLOCK-TREE-POSTORDER es $O(n)$ u.t.

2.2.2. INITIALIZE-BLOCK-TABLES

El algoritmo INITIALIZE-BLOCK-TABLES, mostrado en el Pseudocódigo 2, incorpora en cada vértice $v_{i,j}$ la información de todos los vértices marcados en $C^+_{v_{i,j}}$. Este algoritmo inserta esta información en la columna 0 de $\mathcal{T}_{v_{i,j}}$. La entrada para este algoritmo es la tabla $\mathcal{T}_{v_{s,0}}$ de cada bloque $B_s \in C_{B_i}$. Su salida es $\mathcal{T}_{v_{i,j}}[r][0]$, para toda r y j.

La lógica detrás de INITIALIZE-BLOCK-TABLES se basa en las ecuaciones 1 - 5, las cuales son variaciones de las descitas en el trabajo de Mjelde (2004). La ecuación 1 maneja el caso cuando $C^+_{v_{i,j}} = \emptyset$. Las ecuaciones 2 - 5 manejan el caso cuando $C^+_{v_{i,j}} \neq \emptyset$.

$$\mathcal{T}_{v_{i,j}}[r][0] \leftarrow \text{configuración nula}, \forall r \text{ y } \forall j \tag{1}$$

$$\mathcal{T}_{v_{i,j}}[2][0] \leftarrow \alpha_1 \leftarrow \sum_{\forall B_s \in C_{v_{i,j}}} \mathcal{T}_{v_{s,0}}[2][2] \tag{2}$$

$$\mathcal{T}_{v_{i,j}}[1][0] \leftarrow \max \begin{cases} \mathcal{T}_{v_{i,j}}[2][0] \\ \max_{\forall B_s \in C_{v_{i,j}}} (\alpha_1 \circ \mathcal{T}_{v_{s,0}}[2][2] \bullet \mathcal{T}_{v_{s,0}}[1][2]) \end{cases} \tag{3}$$

Pseudocódigo 2: INITIALIZE-BLOCK-TABLES(B_i, C_{B_i})

Entrada: Para cada vértice $v_{i,j} \in B_i$, $\mathcal{T}_{v_{s,0}}[r][2]$ de cada bloque $B_s \in C_{v_{i,j}}$, $\forall r$.
Salida : $\mathcal{T}_{v_{i,j}}[r][0], \forall r$ y $\forall j$.

```
1  begin
2      foreach v_{i,j} ∈ B_i do
3          if |C_{v_{i,j}}| = 0 then
4              Asignar a T_{v_{i,j}}[r][0] la configuración nula ∀r;
5          else
6              α_1 ← {0, (∅), ∞};
7              foreach v_{s,0} ∈ C_{v_{i,j}} do
8                  α_1 ← α_1 • T_{v_{s,0}}[2][2];
9              end
10             T_{v_{i,j}}[2][0] ← α_1;
11             T_{v_{i,j}}[1][0] ← max_{∀B_s∈C_{v_{i,j}}}(α_1 ∘ T_{v_{s,0}}[2][2] • T_{v_{s,0}}[1][2]);
12             T_{v_{i,j}}[1][0] ← max(T_{v_{i,j}}[2][0], T_{v_{i,j}}[1][0]);
13             α_0 ← {0, (∅), ∞};
14             foreach v_{s,0} ∈ C_{v_{i,j}} do
15                 α_0 ← α_0 • T_{v_{s,0}}[3][2] ;
16             end
17             T_{v_{i,j}}[0][0] ← α_0 • {1, (v_{i,j}), 0};
18             T_{v_{i,j}}[0][0] ← max(T_{v_{i,j}}[1][0], T_{v_{i,j}}[0][0]);
19         end
20     end
21     return T_{v_{i,j}}[r][0], ∀r and ∀j;
22 end
```

$$\alpha_0 \leftarrow \sum_{\forall B_s \in C_{v_{i,j}}} \mathcal{T}_{v_{s,0}}[3][2] \tag{4}$$

$$\mathcal{T}_{v_{i,j}}[0][0] \leftarrow \max \begin{cases} \mathcal{T}_{v_{i,j}}[1][0] \\ \alpha_0 \bullet \{1, (v_{i,j}), 0\} \end{cases} \tag{5}$$

Las líneas 3 y 4 del Pseudocódigo 2 consideran el caso cuando $C^+_{v_{i,j}} = \emptyset$.

Para este caso, la columna 0 de $\mathcal{T}_{v_{i,j}}[r][0]$ recibe la configuración nula (ecuación 1).

Las líneas 5 - 19 manejan el caso cuando $C^+_{v_{i,j}} \neq \emptyset$. Para tal caso, las líneas 6 - 10 calculan la ecuación 2. Las líneas 11 - 12 calculan la ecuación 3. Las líneas 13 - 16 calculan la ecuación 4. Finalmente, las líneas 17 - 18 calculan la ecuación 5.

Para cada columna, INITIALIZE-BLOCK-TABLES primero calcula el valor de $r = 2$, luego para $r = 1$, y finalmente para $r = 0$. El tiempo de ejecución de este algoritmo es $O(|B_i| + |C_{B_i}|)$ para el bloque B_i.

2.2.3. PROCESS-CLIQUE-BLOCK

Pseudocódigo 3: PROCESS-CLIQUE-BLOCK(B_i).

```
   Entrada: Un clique de tamaño 2 del bloque B_i.
   Salida  : Un marcado M_{i,0}[1] de T_{v_{i,0}}.
 1 begin
 2     for r ← 2 down to 0 do
 3         T_{v_{i,1}}[r][1] ← T_{v_{i,1}}[r][0];
 4     end
 5     T_{v_{i,0}}[3][1] ← T_{v_{i,0}}[3][0] • T_{v_{i,1}}[2][1] ;
 6     T_{v_{i,0}}[2][1] ← T_{v_{i,0}}[2][0] • T_{v_{i,1}}[1][1];
 7     T_{v_{i,0}}[1][1] ← max(T_{v_{i,0}}[1][0] • T_{v_{i,1}}[1][1], T_{v_{i,0}}[2][0] • T_{v_{i,1}}[0][1]);
 8     T_{v_{i,0}}[1][1] ← max(T_{v_{i,0}}[1][1], T_{v_{i,0}}[2][1]);
 9     T_{v_{i,0}}[0][1] ← max(T_{v_{i,0}}[0][0] • T_{v_{i,1}}[2][1], T_{v_{i,0}}[1][1]);
10     return T_{v_{i,0}}[r][1], ∀r;
11 end
```

El algoritmo PROCESS-CLIQUE-BLOCK, mostrado en el Pseudocódigo 3, marca todos los vértices del bloque clique B_i. La entrada para este algoritmo es el árbol t_i del bloque clique B_i. Su salida es $M_{i,0}[1]$. Este marcado representa un conjunto 2-packing máximo para el grafo inducido por los vértices de $B_i \cup C^+_{B_i}$, suponiendo que todos los bloques en $C^+_{B_i}$ son cliques.

La lógica detrás de PROCESS-CLIQUE-BLOCK se basa en el conjunto de ecuaciones 6 - 10, que son variantes de las descritas en el trabajo de Mjelde (2004). La ecuación 6 establece los valores para la columna 1 de cualquier vértice hoja del árbol de entrada. La ecuación 7 establece los valores para la columna 1 y renglón 3 del vértice origen del árbol de entrada. Las ecuaciones 8 - 10 calculan los Renglones 2, 1, y 0 de la columna 1 para todos los vértices internos del árbol de entrada, respectivamente.

$$\mathcal{T}_{v_{i,j}}[r][1] \underset{\forall r}{\leftarrow} \mathcal{T}_{v_{i,j}}[r][0] \tag{6}$$

$$\mathcal{T}_{v_{i,0}}[3][1] \leftarrow \mathcal{T}_{v_{i,0}}[3][0] \bullet \begin{cases} \mathcal{T}_{v_{i,1}}[2][1], & si\ |B_i| = 2 \\ \mathcal{T}_{v_{i,1}}[2][1], & si\ |B_i| \geq 3 \wedge h = |B_i| - 1 \\ \mathcal{T}_{v_{i,|B_i|-1}}[2][1], & si\ |B_i| \geq 3 \wedge h = 0 \\ \mathcal{T}_{v_{i,|B_i|-1}}[2][1] \bullet \mathcal{T}_{v_{i,1}}[2][1], & si\ |B_i| \geq 3 \wedge \\ & (0 < h < |B_i| - 1) \end{cases} \tag{7}$$

$$\mathcal{T}_{v_{i,j}}[2][1] \leftarrow \mathcal{T}_{v_{i,j}}[2][0] \bullet \mathcal{T}_{v_{i,j+1}}[1][1] \tag{8}$$

$$\mathcal{T}_{v_{i,j}}[1][1] \leftarrow \max \begin{cases} \max \begin{cases} \mathcal{T}_{v_{i,j}}[1][0] \bullet \mathcal{T}_{v_{i,j+1}}[1][1] \\ \mathcal{T}_{v_{i,j}}[2][0] \bullet \mathcal{T}_{v_{i,j+1}}[0][1] \end{cases} \\ \mathcal{T}_{v_{i,j}}[2][1] \end{cases} \tag{9}$$

$$\mathcal{T}_{v_{i,j}}[0][1] \leftarrow \max \begin{cases} \mathcal{T}_{v_{i,j}}[0][0] \bullet \mathcal{T}_{v_{i,j+1}}[2][1] \\ \mathcal{T}_{v_{i,j}}[1][1] \end{cases} \tag{10}$$

Las líneas 2 - 4 del Pseudocódigo 3 implementan la ecuación 6. La línea 5 implemente la ecuación 7. La línea 6 implementa la ecuación 8. Las líneas 7 - 8 implementan la ecuación 9. Finalmente, la línea 9 implementa la ecuación 10. El tiempo total de ejecución de este algoritmo es $O(1)$ u.t.

Lema 2.1 *El marcado $M_{i,0}[1]$ calculado por los algoritmos* INITIALIZE-BLOCK-TABLES *y* PROCESS-CLIQUE-BLOCK *en el bloque clique B_i, suponiendo que todos los bloques anteriores son cliques y tienen un marcado óptimo, representa un conjunto 2-packing máximo en el grafo inducido por los vértices de $B_i \cup C^+_{B_i}$.*

Demostración. Suponga que primero se ejecutó ROOTED-BLOCK-TREE-POSTORDER en el grafo de entrada. También, suponga que los bloques de $C^+_{B_i}$ son cliques y tienen un marcado óptimo.

Caso 1. Cuando $C^+_{B_i} = \emptyset$. INITIALIZE-BLOCK-TABLES se ejecuta en B_i, y $\mathcal{T}_{v_{i,j}}[r][0]$ asume la configuración nula para toda j. Después, PROCESS-CLIQUE-BLOCK se ejecuta en B_i y marca sólo el vértice $v_{i,1}$. El procedimiento anterior marca dicho bloque clique.

Caso 2. Cuando $C^+_{B_i} \neq \emptyset$. Cuando INITIALIZE-BLOCK-TABLES se ejecuta en B_i, establece los valores iniciales de $\mathcal{T}_{v_{i,j}}[r][0]$ para toda j. Este procedimiento inicializa correctamente la columna 0 puesto que implementa la inicialización provista por el algoritmo de Mjelde. Finalmente, PROCESS-CLIQUE-BLOCK marca correctamente los vértices del bloque B_i, considerando el marcado de $C^+_{B_i}$, puesto que implementa el marcado del procedimiento del algoritmo de Mjelde. ■

Corolario 2.1 *Sea el árbol T el grafo de entrada. Sea T_B el bloque del árbol T generado por* ROOTED-BLOCK-TREE-POSTORDER. *Sea $M_{N,0}[1]$ el marcado calculado en T_B por* INITIALIZE-BLOCK-TABLES *y* PROCESS-CLIQUE-BLOCK *en $B_1, B_2, \ldots, B_N$, suponiendo $v_{N,0}$ como el vértice origen. Sea $\mathcal{M}_{N,0}[1]$ el marcado calculado por el algoritmo de Mjelde en el árbol T, suponiendo $v_{N,0}$ como el vértice origen. Entonces, $M_{N,0}[1]$ es igual a $\mathcal{M}_{N,0}[1]$ y por lo tanto un conjunto 2-packing máximo en T.*

Demostración. Por inducción en el número postorden k del bloque.

Caso base. Para $k = 1$, este caso es el mismo que el Caso 1 del Lema 2.1.

Hipótesis inductiva. Suponga que el Corolario 2.1 se mantiene para $1 \leq k \leq i-1$.

Paso inductivo. Para $k = i$, este caso es el mismo que el Caso 2 del Lema 2.1. Por lo tanto, el marcado $M_{N,0}[1]$ debe ser igual a $\mathcal{M}_{N,0}[1]$ y también un conjunto 2-packing máximo. ■

Pseudocódigo 4: MAXIMUM-2-PACK-TREE$\left(t_i^h, h\right)$.

Entrada: El árbol de expansión t_i^h de B_i. El índice h de la arista eliminada de B_i.
Salida : Un árbol actualizado t_i^h y un marcado $M_{i,0}^h[1]$.

```
1  begin
2      if h > 0 then
3          for r ← 2 down to 0 do
4              T_{v_{i,h}}[r][1] ← T_{v_{i,h}}[r][0];
5          end
6          for k = h − 1 down to 1 do
7              T_{v_{i,k}}[2][1] ← T_{v_{i,k}}[2][0] • T_{v_{i,k+1}}[1][1];
8              T_{v_{i,k}}[1][1] ← max(T_{v_{i,k}}[1][0] • T_{v_{i,k+1}}[1][1], T_{v_{i,k}}[2][0] • T_{v_{i,k+1}}[0][1]);
9              T_{v_{i,k}}[1][1] ← max(T_{v_{i,k}}[1][1], T_{v_{i,k}}[2][1]);
10             T_{v_{i,k}}[0][1] ← max(T_{v_{i,k}}[0][0] • T_{v_{i,k+1}}[2][1], T_{v_{i,k}}[1][1]);
11         end
12     end
13     if h < |B_i| − 1 then
14         for r ← 2 down to 0 do
15             T_{v_{i,h+1}}[r][1] ← T_{v_{i,h+1}}[r][0];
16         end
17         for k = h + 2 to |B_i| − 1 do
18             T_{v_{i,k}}[2][1] ← T_{v_{i,k}}[2][0] • T_{v_{i,k−1}}[1][1];
19             T_{v_{i,k}}[1][1] ← max(T_{v_{i,k}}[1][0] • T_{v_{i,k−1}}[1][1], T_{v_{i,k}}[2][0] • T_{v_{i,k−1}}[0][1]);
20             T_{v_{i,k}}[1][1] ← max(T_{v_{i,k}}[1][1], T_{v_{i,k}}[2][1]);
21             T_{v_{i,k}}[0][1] ← max(T_{v_{i,k}}[0][0] • T_{v_{i,k−1}}[2][1], T_{v_{i,k}}[1][1]);
22         end
23     end
24     T_{v_{i,0}}[r][1] ← PROCESS-SOURCE-VERTEX(t_i^h, h);
25     return t_i^h;
26 end
```

2.2.4. MAXIMUM-2-PACK-TREE

El algoritmo MAXIMUM-2-PACK-TREE, mostrado en el Peudocódigo 4, realiza el marcado de $t_i^h = B_i \setminus e_{i,h}$, donde B_i es un bloque cíclico. La entrada para este algoritmo es el árbol de expansión t_i^h. Su salida es un marcado $M_{i,0}^h[1]$ que representa un conjunto 2-packing máximo en el grafo inducido por los vértices de $\left(B_i \cup C_{B_i}^+\right) \setminus e_{i,h}$.

Este algoritmo supone lo siguiente. Primero, que se ejecutó ROOTED-BLOCK-TREE-POSTORDER en el grafo de entrada. Segundo, que los bloques de $C_{B_i}^+$ son cliques y están marcados de forma óptima. Finalmente, que ejecutó INITIALIZE-BLOCK-TABLES en B_i.

MAXIMUM-2-PACK-TREE implementa las ecuaciones 6 - 10. Las líneas 2 - 12 realizan el marcado de los vértices $v_{i,h}, v_{i,h-1}, \ldots, v_{i,1}$. De forma semejante, las líneas 13 - 23 realizan el marcado de los vértices $v_{i,h+1}, v_{i,h+2}, \ldots, v_{i,|B_i|-1}$. Finalmente, la línea 24 procesa el vértice $v_{i,0}$ mediante la ejecución de PROCESS-SOURCE-VERTEX. El tiempo de ejecución total de este algoritmo es $O(|B_i|)$ u.t.

El algoritmo PROCESS-SOURCE-VERTEX, mostrado en el Pseudocódigo 5, realiza el marcado del vértice $v_{i,0}$. La entrada para este algoritmo es el árbol t_i^h y el índice h de la arista $e_{i,h}$. La salida es un marcado $M_{i,0}^h[1]$.

El algoritmo comprende tres casos. El primero ocurre cuando t_i^h sólo tiene la "rama izquierda" (líneas 3 - 9). El segundo caso es cuando t_i^h sólo tiene la "rama derecha" (líneas 10 - 16). El tercer caso sucede cuando t_i^h tiene ambas ramas (líneas 17 - 23). Este algoritmo implementa las ecuaciones 7 - 10, y su tiempo de ejecución es $O(1)$ u.t.

Sea el uniciclo U el grafo de entrada. Suponga que ROOTED-BLOCK-TREE-POSTORDER se ejecuta en U y que el bloque cíclico es el bloque raíz (es decir, B_N es el bloque cícliclo).

Lema 2.2 *Sea el uniciclo U el grafo de entrada. Sea T_B el árbol de bloques de U generado por ROOTED-BLOCK-TREE-POSTORDER. Sea $M_{N,0}^h[1]$ el marcado creado por INITIALIZE-BLOCK-TABLES y PROCESS-CLIQUE-BLOCK para $B_1, B_2, \ldots, B_{N-1}$, en ese orden, e INITIALIZE-BLOCK-TABLES y MAXIMUM-2-PACK-TREE in $t_N^h = B_N \setminus e_{N,h}$, para alguna h, suponiendo $v_{N,0}$ como el vértice origen. Sea $\mathcal{M}_{N,0}^h[1]$ el marcado calculado por el algoritmo de Mjelde*

Pseudocódigo 5: PROCESS-SOURCE-VERTEX$\left(t_i^h, h\right)$.

Entrada: El árbol de expansión t_i^h de B_i. El índice h de la arista eliminada de B_i.
Salida : Un marcado $M_{i,0}^h[1]$.

begin
  $d \leftarrow |B_i| - 1$;
  if $h = 0$ **then**
    $\mathcal{T}_{v_{i,0}}[3][1] \leftarrow \mathcal{T}_{v_{i,0}}[3][0] \bullet \mathcal{T}_{v_{i,d}}[2][1]$;
    $\mathcal{T}_{v_{i,0}}[2][1] \leftarrow \mathcal{T}_{v_{i,0}}[2][0] \bullet \mathcal{T}_{v_{i,d}}[1][1]$;
    $\mathcal{T}_{v_{i,0}}[1][1] \leftarrow \max(\mathcal{T}_{v_{i,0}}[1][0] \bullet \mathcal{T}_{v_{i,d}}[1][1], \mathcal{T}_{v_{i,0}}[2][0] \bullet \mathcal{T}_{v_{i,d}}[0][1])$;
    $\mathcal{T}_{v_{i,0}}[1][1] \leftarrow \max(\mathcal{T}_{v_{i,0}}[1][1], \mathcal{T}_{v_{i,0}}[2][1])$;
    $\mathcal{T}_{v_{i,0}}[0][1] \leftarrow \max(\mathcal{T}_{v_{i,0}}[0][0] \bullet \mathcal{T}_{v_{i,d}}[2][1], \mathcal{T}_{v_{i,0}}[1][1])$;
  end
  if $h = d$ **then**
    $\mathcal{T}_{v_{i,0}}[3][1] \leftarrow \mathcal{T}_{v_{i,0}}[3][0] \bullet \mathcal{T}_{v_{i,1}}[2][1]$;
    $\mathcal{T}_{v_{i,0}}[2][1] \leftarrow \mathcal{T}_{v_{i,0}}[2][0] \bullet \mathcal{T}_{v_{i,1}}[1][1]$;
    $\mathcal{T}_{v_{i,0}}[1][1] \leftarrow \max(\mathcal{T}_{v_{i,0}}[1][0] \bullet \mathcal{T}_{v_{i,1}}[1][1], \mathcal{T}_{v_{i,0}}[2][0] \bullet \mathcal{T}_{v_{i,1}}[0][1])$;
    $\mathcal{T}_{v_{i,0}}[1][1] \leftarrow \max(\mathcal{T}_{v_{i,0}}[1][1], \mathcal{T}_{v_{i,0}}[2][1])$;
    $\mathcal{T}_{v_{i,0}}[0][1] \leftarrow \max(\mathcal{T}_{v_{i,0}}[0][0] \bullet \mathcal{T}_{v_{i,1}}[2][1], \mathcal{T}_{v_{i,0}}[1][1])$;
  end
  if $h \neq 0 \wedge h \neq d$ **then**
    $\mathcal{T}_{v_{i,0}}[3][1] \leftarrow \mathcal{T}_{v_{i,0}}[3][0] \bullet \mathcal{T}_{v_{i,d}}[2][1] \bullet \mathcal{T}_{v_{i,1}}[2][1]$;
    $\mathcal{T}_{v_{i,0}}[2][1] \leftarrow \mathcal{T}_{v_{i,d}}[1][1] \bullet \mathcal{T}_{v_{i,1}}[1][1]$;
    $\mathcal{T}_{v_{i,0}}[1][1] \leftarrow \max(\mathcal{T}_{v_{i,d}}[0][1] \bullet \mathcal{T}_{v_{i,1}}[1][1], \mathcal{T}_{v_{i,d}}[1][1] \bullet \mathcal{T}_{v_{i,1}}[0][1])$;
    $\mathcal{T}_{v_{i,0}}[1][1] \leftarrow \max(\mathcal{T}_{v_{i,0}}[1][1], \mathcal{T}_{v_{i,0}}[2][1])$;
    $\mathcal{T}_{v_{i,0}}[0][1] \leftarrow \max(\mathcal{T}_{v_{i,d}}[2][1] \bullet \mathcal{T}_{v_{i,1}}[2][1] \bullet \mathcal{T}_{v_{i,0}}[0][0], \mathcal{T}_{v_{i,0}}[1][1])$;
  end
  return $\mathcal{T}_{v_{i,0}}[r][1], \forall r$;
end

(2004), en t_N^h, suponiendo $v_{N,0}$ como el vértice origen. Entonces, $M_{N,0}^h[1]$ es igual a $\mathcal{M}_{N,0}^h[1]$ y por lo tanto un conjunto 2-packing máximo en t_N^h.

Demostración. Existen dos casos.

Caso 1. Suponga que después de ejecutar ROOTED-BLOCK-TREE-POSTORDER en U, $N = 1$ (es decir, U es un ciclo). INITIALIZE-BLOCK-TABLES se ejecuta en B_N, y $\mathcal{T}_{v_{N,j}}[r][0]$ recibe la configuración nula para toda j. Posteriormente, MAXIMUM-2-PACK-TREE se ejecuta en B_N y realiza el marcado de t_N^h, para alguna h, mediante alguna implementación del algoritmo de Mjelde. Por lo tanto, $M_{N,0}^h[1]$ es el marcado correcto para t_N^h.

Case 2. Suponga que después de ejecutar ROOTED-BLOCK-TREE-POSTORDER en U, $N > 1$. INITIALIZE-BLOCK-TABLES, cuando se ejecuta en B_N, asigna los valores iniciales de $\mathcal{T}_{v_{N,j}}[r][0]$, para toda j. Este procedimiento inicializa correctamenta la columna 0 puesto que implementa la inicialización presentada por el algoritmo de Mjelde. Finalmente, MAXIMUM-2-PACK-TREE realiza el marcado de t_N^h, para alguna h, considerando los vértices marcado de $C_{B_N}^+$, puesto que también implementa el procedimiento del algoritmo

de Mjelde. ■

Sea B_N el bloque cíclico del uniciclo U. Suponga que se ejecuta ROOTED-BLOCK-TREE-POSTORDER en el grafo U. También suponga que se ejecutó INITIALIZE-BLOCK-TABLES y PROCESS-CLIQUE-BLOCK en los bloques $B_1, B_2, \ldots, B_{N-1}$, e INITIALIZE-BLOCK-TABLES en B_N. Sea $t_N^h = B_N \backslash e_{N,h}$, donde $v_{N,0}$ es la raíz de dicho árbol. Sea $v_{N,0}, v_{N,1}, \ldots v_{N,h}$ los vértices de una rama de t_N^h.

Suponga que MAXIMUM-2-PACK-TREE se ejecutó en t_N^h y produjo el marcado $M_{N,0}^h[1]$. Suponga que el vértice $v_{N,j}$ está marcado en $M_{N,0}^h[1]$. Suponga que existe un procedimiento alternativo similar a MAXIMUM-2-PACK-TREE que se ejecuta en t_N^h y genera el marcado $\left(M_{N,0}^h[1]\right)^A$. Al marcado creado por el procedimiento alternativo se señala como "$(\,)^A$". Este método alternativo trabaja igual que MAXIMUM-2-PACK-TREE, excepto que intencionalmente omite el marcado del vértice $v_{N,j}$.

Lema 2.3 *Sea $M_{N,0}^h[1]$ un marcado para $U \backslash e_{N,h}$ generado por los algoritmos propuestos y $\left(M_{N,0}^h[1]\right)^A$ el marcado alternativo para $U \setminus e_{N,h}$ así como está definido arriba. Entonces, $\left|M_{N,0}^h[1]\right| \geq \left(M_{N,0}^h[1]\right)^A$.*

Demostración. Por contradicción. Suponga que después de que MAXIMUM-2-PACK-TREE y el procedimiento alternativo calcularon el marcado para B_N, $\left|M_{N,0}^h[1]\right| < \left|\left(M_{N,0}^h[1]\right)^A\right|$. Observe que ambos procedimientos trabajan de la misma manera hasta el marcado del vértice $v_{N,j+1}$; por lo tanto, $\left|M_{N,j+1}^h[1]\right| = \left|\left(M_{N,j+1}^h[1]\right)^A\right|$.

Entonces, MAXIMUM-2-PACK-TREE marca $v_{N,j}$ y el procedimiento alternativo no; de aquí que $\left|M_{N,j}^h[1]\right| > \left|\left(M_{N,j}^h[1]\right)^A\right|$.

Por la suposición inicial, $\left|M_{N,0}^h[1]\right| < \left|\left(M_{N,0}^h[1]\right)^A\right|$, debe existir un vértice $v_{N,x}$, para alguna $x < j$, tal que $\left|M_{N,x+1}^h[1]\right| > \left|\left(M_{N,x+1}^h[1]\right)^A\right|$ y $\left|M_{N,x}^h[1]\right| < \left|\left(M_{N,x}^h[1]\right)^A\right|$. Se presentan dos casos.

- Cuando $\left|S_{N,x+1}^h[1]\right| > \left|\left(S_{N,x+1}^h[1]\right)^A\right|$, para satisfacer que $\left|M_{N,x}^h[1]\right| < \left|\left(M_{N,x}^h[1]\right)^A\right|$, el procedimiento alternativo debe marcar $v_{N,x}$ y MAXIMUM-2-PACK-TREE no lo debe marcar. De forma semejante, $\left(d_{N,x}^h[1]\right)^A$ debe ser mayor que $d_{N,x}^h[1]$; no obstante, $\left(d_{N,x}^h[1]\right)^A = 0$ y $d_{N,x}^h[1] > 0$. Existe una contradicción.

- Cuando $\left|S^h_{N,x+1}[1]\right| = \left|\left(S^h_{N,x+1}[1]\right)^A\right|$ y $d^h_{N,x+1}[1] > \left(d^h_{N,x+1}[1]\right)^A$, para satisfacer $\left|M^h_{N,x}[1]\right| < \left|\left(M^h_{N,x}[1]\right)^A\right|$, el procedimiento alternativo debe marcar $v_{N,x}$ mientras que MAXIMUM-2-PACK-TREE no lo marca; no obstante, MAXIMUM-2-PACK-TREE también debe marcar $v_{N,x}$ puesto que $d^h_{N,x}[1] > \left(d^h_{N,x}[1]\right)^A$. Por otro lado, si el procedimiento de marcado alternativo y MAXIMUM-2-PACK-TREE no marcan $v_{N,x}$, entonces $d^h_{N,x}[1] > \left(d^h_{N,x}[1]\right)^A$ y existe una contradicción.

■

El Lema 2.3 implica que cualquier marcado que intencionalmente omite el marcado de un vértice v_x, que debería estar marcado, no puede tener cardinalidad mayor que un marcado que incluye a v_x.

2.3. Conjunto 2-packing máximo en un grafo uniciclo

Esta sección describe el algoritmo MAXIMUM-2-PACK-UNICYCLE, mostrado en el Pseudocódigo 6. El componente principal de este algoritmo es el procedimiento PROCESS-CYCLIC-BLOCK que marca los vértices de un bloque cíclico en el grafo. Este método también incluye a los algoritmos ADAPT-FOR-UNICYCLE y CORRECT-CONFLICT. Estos algoritmos detectan y corrigen un 'conflicto' entre vértices marcados. Un *conflicto* entre dos vértices marcados ocurre en $M^h_{N,0}[1]$ después de remplazar la arista $e_{i,h}$ en $U \setminus e_{i,h}$ cuando la distancia entre esos vértices es menor que o igual a dos. Finalmente, se demuestra que MAXIMUM-2-PACK-UNICYCLE es correcto.

2.3.1. Descripción de MAXIMUM-2-PACK-UNICYCLE

La entrada del algoritmo MAXIMUM-2-PACK-UNICYCLE es un grafo unicíclo no dirigido U. La salida es un marcado $M_{N,0}[2]$ que representa un conjunto 2-packing máximo en U.

La línea 2 del Pseudocódigo 6 calcula el árbol de bloques enraizado T_B y etiqueta sus bloques de acuerdo a su numeración postorden. Suponga que el bloque cíclico B_N

Pseudocódigo 6: MAXIMUM-2-PACK-UNICYCLE(U)

```
   Entrada: Un grafo uniciclo no dirigido U.
   Salida : Un marcado M_{N,0}[2] de U que representa un conjunto 2-packing máximo.
1  begin
2      T_B ←ROOTED-BLOCK-TREE-POSTORDER(U);
3      for i ← 1 to N − 1 do
4          T_{v_{i,j}}[r][0] ←INITIALIZE-BLOCK-TABLES(B_i, C_{B_i}), ∀j;
5          T_{v_{i,0}}[r][1] ←PROCESS-CLIQUE-BLOCK(B_i);
6          T_{v_{i,0}}[r][2] ← T_{v_{i,0}}[r][1];
7      end
8      T_{v_{N,j}}[r][0] ←INITIALIZE-BLOCK-TABLES(B_N, C_{B_N}), ∀j;
9      T_{v_{N,0}}[r][2] ←PROCESS-CYCLIC-BLOCK(B_N);
10     U ← DISTRIBUTE-MAXIMUM-2-PACK-SOLUTION(U, T_{v_{N,0}}[0][2]);
11     return U;
12 end
```

es la raíz de T_B. Las líneas 3 - 7 procesan de forma iterativa los bloques $B_1, B_2, \ldots, B_{N-1}$ mediante la ejecución de INITIALIZE-BLOCK-TABLES y PROCESS-CLIQUE-BLOCK. Por lo tanto, $\mathcal{T}_{v_{s,0}}[r][2]$ está disponible para cada bloque $B_s \in C_{B_N}$.

La línea 8 inicializa $\mathcal{T}_{v_{N,j}}[r][0]$ con la información de $C_{v_{N,j}}$ (si es que existe), para toda j. Posteriormente, mediante PROCESS-CYCLIC-BLOCK, la línea 9 marca los vértices del bloque B_N. Los vértices marcados de U es un conjunto 2-packing máximo. Finalmente, en la línea 10, el algoritmo DISTRIBUTE-MAXIMUM-2-PACK-SOLUTION (no se muestra) distribuye el marcado final, almacenado en $\mathcal{T}_{v_{N,0}}[0][2]$, hacia todos los vértices marcados en U a través de un recorrido sobre el árbol de expansión de U.

Pseudocódigo 7: PROCESS-CYCLIC-BLOCK(B_i)

```
   Entrada: Un bloque cíclico B_i cuyo vértice origen es v_{i,0}.
   Salida : Un marcado M_{i,0}[2].
1  begin
2      for h = 0 to |B_i| − 1 do
3          t_i^h ← B_i \ e_{i,h};
4          t_i^h ← MAXIMUM-2-PACK-TREE(t_i^h, h);
5          T^h_{v_{i,0}}[r][1] ← ADAPT-FOR-UNICYCLE(t_i^h, h);
6      end
7      max_sol ← T^0_{v_{i,0}}[r][1];
8      for h = 1 to |B_i| − 1 do
9          max_sol ← max(max_sol, T^h_{v_{i,0}}[r][1]), ∀r;
10     end
11     T_{v_{i,0}}[r][2] ← max_sol;
12     return T_{v_{i,0}}[r][2], ∀r;
13 end
```

El algoritmo PROCESS-CYCLIC-BLOCK, mostrado en el Pseudocódigo 7, marca los vértices del bloque B_i. El ciclo en las líneas 2 - 6 calcula de forma iterativa una familia de

árboles de expansión de B_i, denotada por $t_i^h = B_i \setminus e_{i,h}$, para $0 \leq h \leq |B_i| - 1$. Después la línea 4 marca los vértices de t_i^h, para toda h, mediante MAXIMUM-2-PACK-TREE. Finalmente, mediante ADAPT-FOR-UNICYCLE, la línea 5 corrige (si es necesario) el marcado de t_i^h después de remplazar la arista $e_{i,h}$. Las líneas 7 - 10 eligen el mejor marcado entre los $|B_i|$ marcados calculados por el ciclo previo.

Pseudocódigo 8: ADAPT-FOR-UNICYCLE$\left(t_i^h, h\right)$.

Entrada: El árbol de expansión t_i^h de B_i. El índice h de la arista eliminada de B_i.
Salida : Un marcado $M_{i,0}^h[1]$.

```
1  begin
2      v'_f ← v_{i,h−1 mod |B_i|};
3      v'_g ← v_{i,h};
4      v_g ← v_{i,(h+1) mod |B_i|};
5      v_f ← v_{i,(h+2) mod |B_i|};
6      if (v_g ∧ v'_h) ∈ T_{v_{i,0}}[0][1].set | v'_h ∈ C_{v'_g} ∧ dist(v'_g, v'_h) = 1 then
7          T_{v_{i,0}}[r][1] ← CORRECT-CONFLICT(t_i^h, h, v_g, v'_h);
8      else
9          if (v_h ∧ v'_g) ∈ T_{v_{i,0}}[0][1].set | v_h ∈ C_{v_g} ∧ dist(v_g, v_h) = 1 then
10             T_{v_{i,0}}[r][1] ← CORRECT-CONFLICT(t_i^h, h, v_h, v'_g);
11         else
12             if (v_g ∧ v'_g) ∈ T_{v_{i,0}}[0][1].set then
13                 T_{v_{i,0}}[r][1] ← CORRECT-CONFLICT(t_i^h, h, v_g, v'_g);
14             else
15                 if (v_g ∧ v'_f) ∈ T_{v_{i,0}}[0][1].set then
16                     T_{v_{i,0}}[r][1] ← CORRECT-CONFLICT(t_i^h, h, v_g, v'_f);
17                 else
18                     if (v_f ∧ v'_g) ∈ T_{v_{i,0}}[0][1].set then
19                         T_{v_{i,0}}[r][1] ← CORRECT-CONFLICT(t_i^h, h, v_f, v'_g);
20                     end
21                 end
22             end
23         end
24     end
25     return T_{v_{i,0}}[r][1], ∀r;
26 end
```

El algoritmo ADAPT-FOR-UNICYCLE, mostrado en el Pseudocódigo 8, corrige el marcado generado por MAXIMUM-2-PACK-TREE para el árbol t_i^h para convertirlo en un marcado libre de conflictos para B_i. Este procedimiento regresa la arista $e_{i,h}$ a t_i^h y revisa si existe o no un conflicto entre dos vértices marcados (vea la Figura 3). Cuando el algoritmo identifica uno de estos casos, ejecuta el algoritmo CORRECT-CONFLICT para corregirlo. Si el algoritmo no detecta conflicto alguno, regresa como salida el marcado de entrada; de otra forma, regresa un marcado libre de conflicto.

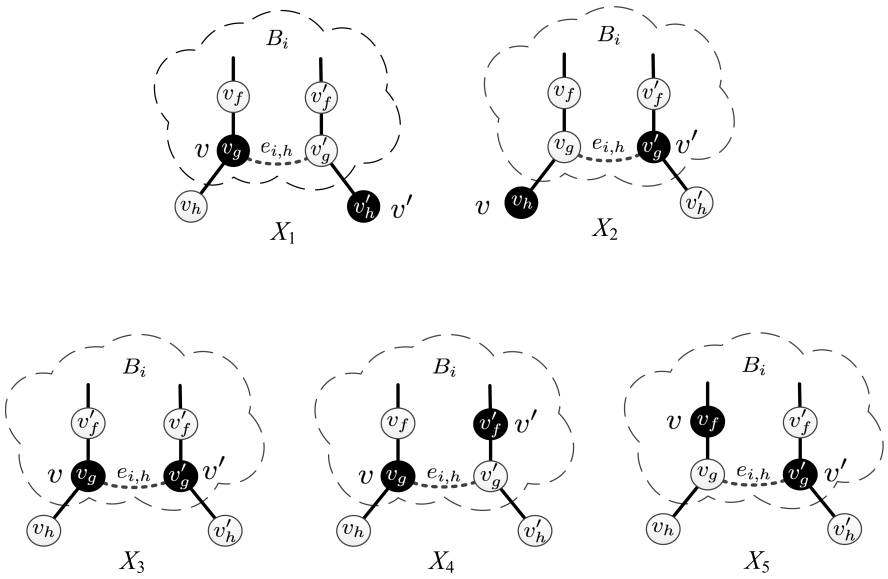


Figura 3. Conflictos entre los vértices marcados detectados por ADAPT-FOR-UNICYCLE. Las líneas punteadas alargadas indican un componente cíclico B_i. La arista $e_{i,h}$ genera un conflicto entre dos vértices marcados después de que se regresa al árbol t_i^h.

La Figura 3 muestra los cinco posibles conflictos, etiquetados con X_q, para $1 \leq q \leq 5$, que pueden ocurrir en un marcado dado. Por ejemplo, en la imagen X_3 de la Figura 3, después de remplazar la arista $e_{i,h}$, los vértices extremos de la arista $e_{i,h}$ están marcados y por lo tanto producen un marcado conflictivo para U.

La Figura 4 muestra el conjunto de todos los posibles marcados libres de conflicto, etiquetados con S_p, para $1 \leq p \leq 9$, que tienen al menos un vértices marcado en el área vecinal de la arista $e_{i,h}$. ADAPT-FOR-UNICYCLE genera uno de estos tipos de marcados como salida. Observe que para el caso en que existen dos vértices marcados, la distancia entre ellos es al menos de tres aristas.

El algoritmo CORRECT-CONFLICT, mostrado en el Pseudocódigo 9, remarca el árbol de entrada y encuentra un marcado libre de conflicto para $U \setminus e_{i,h}$. Para realizar este procedimiento, se generan dos marcados alternativos (almacenados en $\mathcal{T}^1_{v_{i,0}}[r][1]$ y $\mathcal{T}^2_{v_{i,0}}[r][1]$). Para el primer marcado, CORRECT-CONFLICT elimina el vértice v del marcado (línea 5 u 8). Posteriormente, actualiza el resto de las entradas si es necesario (línea 6 o 9). Luego remarca los vértices de t_i^h (línea 11), y valida si este marcado es libre de conflictos (línea 12). Si el marcado no es libre de conflictos, el algoritmo aplica recursivamente el mismo procedimiento para obtener un marcado libre de conflicto. CORRECT-CONFLICT sigue un procedimiento similar para calcular el segundo marcado, excepto que ahora remueve el vértice v' en lugar de v (líneas 13 - 19). Finalmente, el

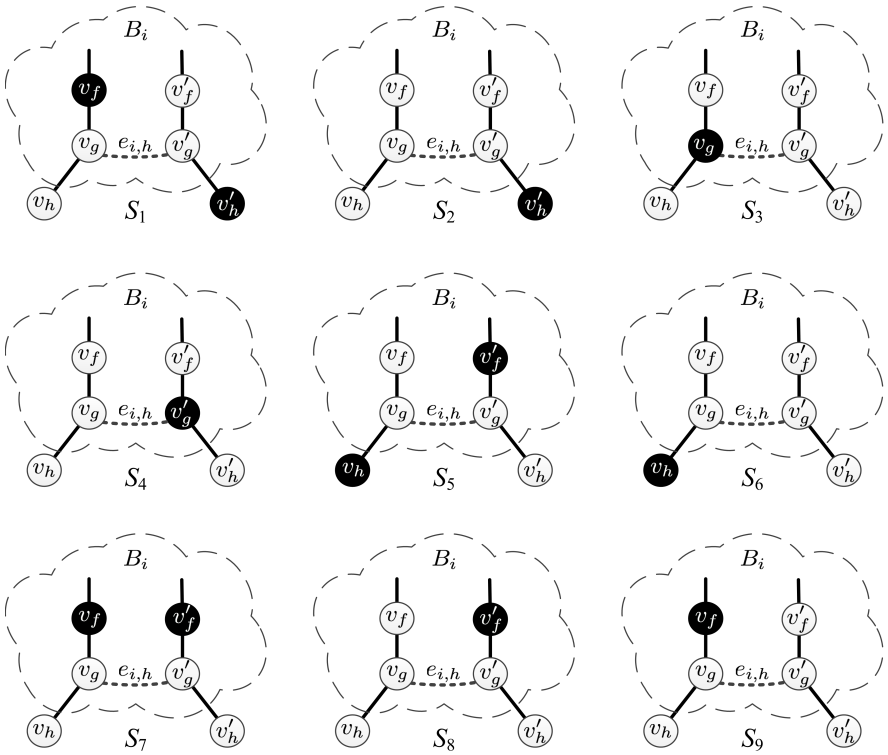


Figura 4. Marcados libres de conflicto generados por el procedimiento CORRECT-CONFLICT. Las líneas punteadas alargadas indican un componente cíclico B_i. La arista $e_{i,h}$ es la arista que el algoritmo regresa al árbol t_i^h.

algoritmo elige y regresa el mejor marcado (líneas 22 - 23).

En la Figura 5, cada círculo con la etiqueta X_q o S_p representa al conjunto de todos los marcados que tienen un submarcado como el mostrado en las figuras 3 o 4, respectivamente. En cada arista de la Figura 5, existe un conjunto de cálculos que CORRECT-CONFLICT debe realizar para moverse de un marcado a otro. Por ejemplo, la arista (X_2, X_3), etiquetada con "$\circ\, v_h \bullet v_g$", indica que el algoritmo remueve a v_h del marcado e inserta v_g.

2.3.2. Demostración de que MAXIMUM-2-PACK-UNICYCLE es correcto

Lema 2.4 *Sean $M_{N,0}[2]$ y $M^h_{N,0}[1]$ conjuntos 2-packing máximos de U y del árbol de expansión $U \setminus e_{N,h}$, respectivamente. Suponga que $\left|M^h_{N,0}[1]\right|$ es la cardinalidad más pequeña de todos los conjuntos 2-packing máximos de todos los árboles de expansión de U. Entonces, $|M_{N,0}[2]| \leq \left|M^h_{N,0}[1]\right|$.*

Pseudocódigo 9: CORRECT-CONFLICT$\left(t_i^h, h, v, v'\right)$.

Entrada: Un árbol de expansión t_i^h de B_i, el índice h, y dos vértices conflictivos v y v'.
Salida : Un marcado libre de conflictos $M_{i,0}^h[1]$.

begin
  $t^1 \leftarrow t_i^h$;
  $t^2 \leftarrow t_i^h$;
  if $v \in t^1$ **then**
    $\mathcal{T}_v^1[0][0] \leftarrow \mathcal{T}_v^1[0][0] \circ \{1, (v), 0\}$;
    $\mathcal{T}_v^1[0][0] \leftarrow \max\left(\mathcal{T}_v^1[0][0], \mathcal{T}_v^1[1][0]\right)$;
  else
    $\mathcal{T}_{p(v)}^1[1][0] \leftarrow \mathcal{T}_{p(v)}^1[2][0]$;
    $\mathcal{T}_{p(v)}^1[0][0] \leftarrow \max\left(\mathcal{T}_{p(v)}^1[1][0], \mathcal{T}_{p(v)}^1[0][0]\right)$;
  end
  $t^1 \leftarrow$MAXIMUM-2-PACK-TREE$\left(t^1, h\right)$;
  $\mathcal{T}_{v_{i,0}}^1[r][1] \leftarrow$ ADAPT-FOR-UNICYCLE$\left(t^1, h\right)$;
  if $v' \in t^2$ **then**
    $\mathcal{T}_{v'}^2[0][0] \leftarrow \mathcal{T}_{v'}^2[0][0] \circ \{1, (v'), 0\}$;
    $\mathcal{T}_{v'}^2[0][0] \leftarrow \max\left(\mathcal{T}_{v'}^2[0][0], \mathcal{T}_{v'}^2[1][0]\right)$;
  else
    $\mathcal{T}_{p(v')}^2[1][0] \leftarrow \mathcal{T}_{p(v')}^2[2][0]$;
    $\mathcal{T}_{p(v')}^2[0][0] \leftarrow \max\left(\mathcal{T}_{p(v')}^2[1][0], \mathcal{T}_{p(v')}^2[0][0]\right)$;
  end
  $t^2 \leftarrow$ MAXIMUM-2-PACK-TREE$\left(t^2, h\right)$;
  $\mathcal{T}_{v_{i,0}}^2[r][1] \leftarrow$ ADAPT-FOR-UNICYCLE$\left(t^2, h\right)$;
  $\mathcal{T}_{v_{i,0}}[r][1] \leftarrow \max\left(\mathcal{T}_{v_{i,0}}^1[r][1], \mathcal{T}_{v_{i,0}}^2[r][1]\right)$;
  return $\mathcal{T}_{v_{i,0}}[r][1], \forall r$;
end

Demostración. Por contradicción. Suponga que $|M_{N,0}[2]| > \left|M_{N,0}^h[1]\right|$. Construya el árbol de expansión $U \setminus e_{N,h}$. Observe que $M_{N,0}[2]$ también es un conjunto 2-packing máximo de $U \setminus e_{N,h}$, lo cual es una contradicción puesto que $M_{N,0}^h[1]$ es un conjunto 2-packing máximo de $U \setminus e_{N,h}$. ■

El Lema 2.4 establece que la cardinalidad de cualquier conjunto 2-packing máximo de cualquier árbol de U es una cota superior a la cardinalidad de un conjunto 2-packing máximo de U.

Lema 2.5 *Sea $M_{N,0}^h[1]$ un marcado libre de conflictos que representa un conjunto 2-packing máximo para algún árbol $U \setminus e_{N,h}$. Entonces, dicho marcado también es un conjunto 2-packing máximo para U.*

Demostración. Por contradicción. Suponga que el marcado $M_{N,0}^h[1]$ no es un conjunto

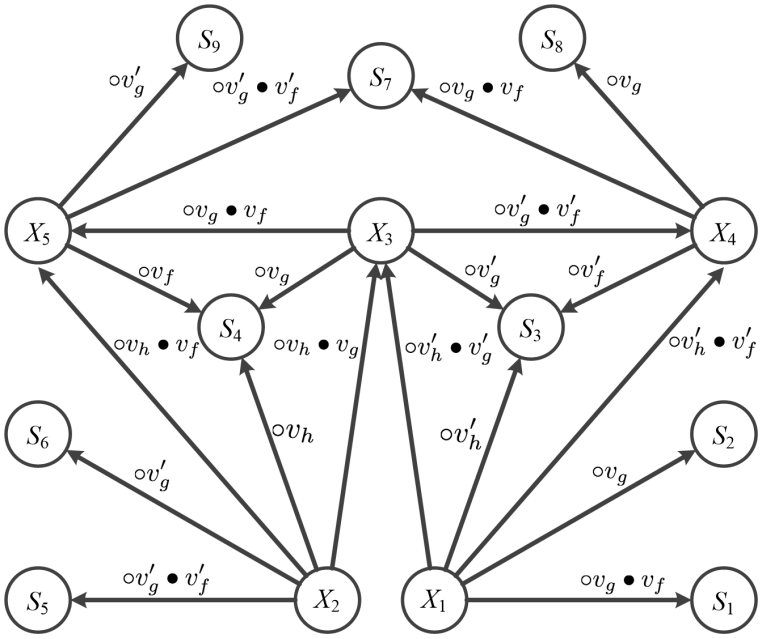


Figura 5. Los círculos X_q son marcados conflictivos de entrada para el algoritmo CORRECT-CONFLICT. Las aristas en el grafo representan transiciones hacia posibles marcados de salida generados por CORRECT-CONFLICT. Observe que este algoritmo puede generar tanto marcados conflictivos como marcados libres de conflictos.

2-packing máximo para U. Por lo tanto, existe otro conjunto 2-packing en U, digamos $M_{N,0}$, tal que $|M_{N,0}[2]| > \left|M^h_{N,0}[1]\right|$. Note que $M_{N,0}[2]$ también es un conjunto 2-packing de $U \setminus e_{N,h}$, lo cual es una contradicción puesto que $M^h_{N,0}[1]$ es un conjunto 2-packing máximo de $U \setminus e_{N,h}$. ■

El Lema 2.5 provee de una estrategia para calcular un conjunto 2-packing máximo de U. Lo que se necesita es encontrar un marcado libre de conflictos que represente un conjunto 2-packing máximo en algún árbol de expansión de U. Puesto que tal conjunto pudiera no existir, una acción alternativa es encontrar el mejor marcado libre de conflictos para cada árbol de expansión de U. Finalmente, escoger el árbol con la mayor cardinalidad de vértices marcados.

Lema 2.6 *Dado un marcado conflictivo $M^h_{i,0}[1]$ de algún $U \setminus e_{i,h}$ como entrada, ADAPT-FOR-UNICYCLE genera a lo más diez marcados de $U \setminus e_{i,h}$ para encontrar un marcado libre de conflictos.*

Demostración. La Figura 6 muestra uno de los peores casos que ocurren cuando el marcado de entrada para CORRECT-CONFLICT es X_2, y este algoritmo genera los marcados X_3 y X_5. Posteriormente, este algoritmo genera X_4 y X_5 cuando la entrada es

X_3. Finalmente, cualquier par de marcados generados por este algoritmo es libre de conflictos cuando su entrada es X_4 o X_5. El número total de marcados generados es diez. Un comportamiento general ocurre para el caso cuando el marcado de entrada es X_1. ■

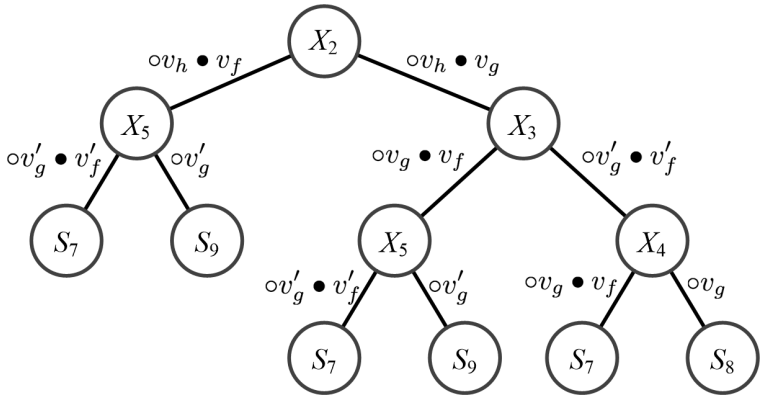


Figura 6. Posibles configuraciones a las que se puede llegar partiendo de X_2.

Lema 2.7 *Dado un marcado $M^h_{i,0}[1]$ de $U \setminus e_{i,h}$ como entrada, ADAPT-FOR-UNICYCLE correctamente determina si éste es un marcado libre de conflicto. Cuando existe un conflicto, éste genera un marcado alternativo libre de conflicto; si no existe conflicto, regresa el marcado de entrada.*

Demostración. El marcado de entrada puede tener hasta cinco tipos diferentes de conflicto, así como se muestra en la Figura 3. ADAPT-FOR-UNICYCLE detecta estos conflictos mediante las líneas 6 - 24 del Pseudocódigo 8. Si no hay conflictos, este procedimiento regresa el marcado de entrada.

Cuando ADAPT-FOR-UNICYCLE detecta un conflicto, éste genera dos marcados alternativos, t^1 y t^2, mediante CORRECT-CONFLICT. Con base en el marcado de entrada, este procedimiento primero remueve el vértice conflictivo, v, del marcado t^1 (líneas 5 - 6 u 8 - 9). CORRECT-CONFLICT maneja dos casos, puesto que el vértice conflictivo, v, puede estar en el ciclo (Configuraciones X_1, X_3, X_4 de la Figura 3) o estar a distancia uno del ciclo (Configuración X_2 de la Figura 3). Después de eliminar el vértice conflictivo v del marcado, la línea 11 del algoritmo marca de nuevo a t^h_i sin incluir a v. Puesto que la línea 11 utiliza el algoritmo de Mjelde (2004), una de las ramas de t^h_i (la rama que contiene a v') mantiene el mismo marcado. De forma semejante, la rama de t^h_i que contiene a v recibe un marcado óptimo, excepto por la omisión del marcado de v.

CORRECT-CONFLICT lleva a cabo un procedimiento similar para generar t^2, pero ahora evita marcar el segundo vértice conflictivo, v', pero manteniendo el marcado de v (líneas 13 - 20). Finalmente, si el marcado de t^1 y t^2 están libres de conflicto, la línea 22 selecciona el marcado con la cardinalidad más grande; de otra forma, genera un marcado libre de conflicto aplicando de forma recursiva ADAPT-FOR-UNICYCLE a t^1 o t^2 (líneas 12 o 21). En el peor de los casos, es necesario producir diez marcados para obtener un marcado libre de conflictos, así como se muestra en el Lema 2.6. ■

Lema 2.8 *El marcado $M^h_{i,0}[1]$ generado por ADAPT-FOR-UNICYCLE es un marcado libre de conflicto que representa un conjunto 2-packing máximo de la más alta cardinalidad para $U \setminus e_{i,h}$.*

Demostración. Suponga sin pérdida de generalidad que t^h_i tiene dos ramas enraizadas en $v_{i,0}$. Sean $t^{h\leftarrow v}_i$ y $t^{h\leftarrow v'}_i$ las ramas de t^h_i que tienen a los vértices conflictivos v y v', respectivamente. La rama $t^{h\leftarrow v'}_i$ consiste del conjunto de aristas y vértices entre $v_{i,1}$ y $v_{i,h}$. De forma semejante, la rama $t^{h\leftarrow v}_i$ consiste del conjunto de aristas y vértices entre $v_{i,|B_i-1|}$ y $v_{i,h+1}$.

Primero, suponga que CORRECT-CONFLICT está procesando la eliminación de v. Ahora, considere al subgrafo formado por la unión de $t^{h\leftarrow v'}_i$ y $\bigcup_{\forall k} T^+_{v_{i,k}}$, $v_{i,k} \in t^{h\leftarrow v'}_i$. Este subgrafo es un árbol y CORRECT-CONFLICT (línea 11 del Pseudocódigo 9) genera un marcado que representa un conjunto 2-packing máximo para este subgrafo, por el Corolario 2.1.

De forma semejante, por el Corolario 2.1, todos los marcados de los árboles $T^+_{v_{i,k}}$, para todo $v_{i,k} \in t^{h\leftarrow v}_i$ son también conjuntos 2-packing máximos. En este punto, la única parte que el algoritmo necesita remarcar son los vértices de $t^{h\leftarrow v}_i$. Observe que las líneas 4 - 9 del Pseudocódigo 9 intencionalmente remueven a v del marcado. Posteriormente, la línea 11 termina el marcado de los vértices restantes de $t^{h\leftarrow v}_i$.

Note que este nuevo marcado, llamado $\left(M^h_{i,0}[1]\right)^1$, con dicha restricción también es un conjunto 2-packing para $U \setminus e_{i,h}$. Siguiendo un procedimiento similar, el cual remueve al vértice conflictivo v' del marcado, se obtiene $\left(M^h_{i,0}[1]\right)^2$, que también es un conjunto 2-packing para $U \setminus e_{i,h}$. Se puede demostrar mediante un procedimiento

similar, que cuando t_i^h consiste de sólo una rama, los marcados generados también son conjuntos 2-packing para $U \setminus e_{i,h}$.

En caso de que algunos de los marcados generados, $\left(M_{i,j}^h[1]\right)^1$ o $\left(M_{i,j}^h[1]\right)^2$, tengan conflicto, el algoritmo calcula de forma recursiva un marcado libre de conflicto, por el Lema 2.7. Finalmente, las líneas 22 - 23 del Pseudocódigo 9 regresan el mejor marcado entre $\left(M_{i,0}^h[1]\right)^1$ y $\left(M_{i,0}^h[1]\right)^2$.

Observe que pueden existir otros marcados libres de conflicto para $U \setminus e_{i,h}$ que representen conjuntos 2-packing, pero por el Lema 2.3, estos marcados alternativos no son mejores que aquel que intencionalmente omite el menor número de vértices marcados. Por lo tanto, el marcado generado por ADAPT-FOR-UNICYCLE es libre de conflicto y representa un conjunto 2-packing para $U \setminus e_{i,h}$ con la más alta cardinalidad. ■

Lema 2.9 *Sea U un grafo unicíclo conectado no dirigido. El marcado $M_{N,0}[2]$ generado por MAXIMUM-2-PACK-UNICYCLE es un conjunto 2-packing máximo para U.*

Demostración. Por el Corolario 2.1, el marcado creado por el algoritmo ROOTED-BLOCK-TREE-POSTORDER en U y por INITIALIZE-BLOCK-TABLES y PROCESS-CLIQUE-BLOCK en los bloques $B_1, B_2, \ldots, B_{N-1}$ (líneas 2 - 7 del Pseudocódigo 6) es un conjunto 2-packing máximo para cada árbol $T^+_{v_{N,j}}$, para todo $v_{N,j}$. Después, INITIALIZE-BLOCK-TABLES (línea 8) incorpora los marcados de los bloques previos a las tablas de los vértices de B_N. Posteriormente, la línea 9 ejecuta PROCESS-CYCLIC-BLOCK en B_N.

PROCESS-CYCLIC-BLOCK primero genera todos los árboles de expansión de $U \setminus e_{N,h} = t_N^h$, para toda h, y los enraiza en el vértice $v_{N,0}$ (líneas 2 - 3 del Pseudocode 7). Después, la línea 4 marca cada uno de los árboles de expansión mediante MAXIMUM-2-PACK-TREE. Por el Lema 2.2, cada uno de estos marcados representa un conjunto 2-packing máximo para dichos árboles.

Si existe un marcado libre de conflicto para algún árbol de expansión, por el Lema 2.5, el marcado resultante es un conjunto 2-packing máximo para U; de otra forma, la línea 5 genera un marcado libre de conflicto que representa un conjunto 2-packing. Por el Lema 2.8, ADAPT-FOR-UNICYCLE calcula el mejor marcado libre de conflicto para el árbol de expansión t_N^h. Finalmente, las líneas 7 - 10 seleccionan el mejor

marcado libre de conflicto entre todos los árboles de expansión, lo cual es el marcado resultante para U. ■

2.3.3. Análisis de complejidad de MAXIMUM-2-PACK-UNICYCLE

Primero, se realiza el análisis de complejidad de PROCESS-CYCLIC-BLOCK. La parte más costosa de este algoritmo (vea el Pseudocódigo 7) es el ciclo for de las líneas 2 - 6, que consiste de $O(|B_N|)$ iteraciones. En este ciclo, ADAPT-FOR-UNICYCLE (Pseudocódigo 8), en el peor de los casos, ejecuta MAXIMUM-2-PACK-TREE a lo más diez veces por el Lema 2.6. Consecuentemente, ADAPT-FOR-UNICYCLE corre en $O(|B_N|)$ u.t. Por lo tanto, este ciclo y el algoritmo completo requiere de $O\left(|B_N|^2\right)$ u.t.

Ahora, se realiza el análisis de complejidad de MAXIMUM-2-PACK-UNICYCLE. El procedimiento ROOTED-BLOCK-TREE-POSTORDER (línea 2) requiere $O(n)$ u.t. La parte más costosa del ciclo for de las líneas 3 - 7 es INITIALIZE-BLOCK-TABLES-LIST, el cual requiere $O\left(|B_i| + \left|C_{B_i}\right|\right) = O(n - |B_N|)$ u.t. Puesto que $N - 1 = n - |B_N|$, este ciclo requiere $O\left((n - |B_N|)^2\right)$ u.t. La línea más costosa del resto del Pseudocódigo 6 es la ejecución de PROCESS-CYCLIC-BLOCK (línea 9), la cual requiere $O\left(|B_N|^2\right)$ u.t. Por lo tanto, MAXIMUM-2-PACK-UNICYCLE se ejecuta en $O\left(n^2\right)$ u.t.

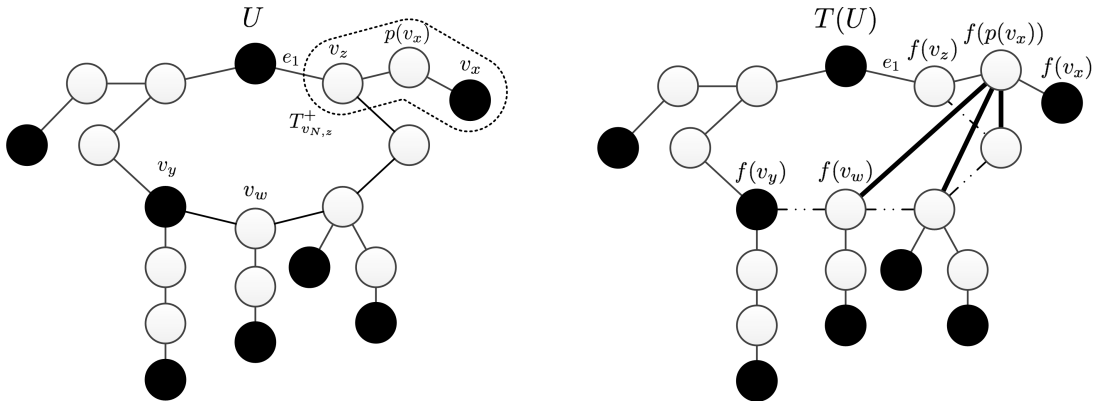
2.4. Transformación de uniciclo a árbol

El algoritmo UNICYCLE-TO-TREE, mostrado en el Pseudocódigo 10, transforma un grafo uniciclo U a un árbol $T(U)$. Esta transformación no es tan sólo la eliminación de una arista de U, puesto que $T(U)$ tiene que satisfacer algunos requerimientos, como se explica a continuación. La entrada para este algoritmo es el grafo uniciclo $U = (V_U, E_U)$ con un marcado $M_{N,0}[2]$ que representa un conjunto 2-packing máximo para U. Suponga que $v_{N,0}$ es el vértice origen de U y que pertenece al bloque cíclico. La salida de este algoritmo es el árbol $T(U) = \left(V_{T(U)}, E_{T(U)}\right)$, tal que existe una función biyectiva $f : V_U \to V_{T(U)}$. Además, el marcado $M_{N,0}[1]$ de $T(U)$ representa un conjunto 2-packing máximo, y éste es igual al marcado $\mathcal{M}_{N,0}[1]$ del algoritmo de Mjelde (2004), cuando se le aplica a $T(U)$, suponiendo $f(v_{N,0})$ como el vértice origen.

Pseudocódigo 10: UNICYCLE-TO-TREE(U)

Entrada: Un grafo uniciclo no dirigido U enraizado en $v_{N,0}$ del bloque cíclico B_N y un marcado $M_{N,0}[2]$ de U que representa un conjunto 2-packing máximo.
Salida : Un árbol $T(U)$ con un marcado $M_{N,0}[2] = M_{N,0}[1] = \mathcal{M}_{N,0}[1]$, el cual representa un conjunto 2-packing máximo.

```
1 begin
2     Aplicar BFS(U, v_{N,0}) e identificar v_x, p(v_x), v_z, v_y, la arista e_1;
3     T(U) ← U;
4     Borre todas las aristas de P_1 en T(U);
5     Agregue una arista entre cada vértice f(v_{N,m}) a f(p(v_x)), tal que v_{N,m} ∈ P_1 en U, v_{N,m} ≠ v_y, y
        v_{N,m} ≠ v_z;
6     return T(U);
7 end
```



(a) Un grafo uniciclo U con su marcado $M_{N,0}[2]$

(b) El árbol transformado, $T(U)$, con su marcado $M_{N,0}[1]$

Figura 7. Transformación de un grafo uniciclo a un árbol.

2.4.1. Descripción de la transformación

Sea U un grafo uniciclo y B_N su bloque cíclico correspondiente. Suponga que $M_{N,0}[2]$ representa un conjunto 2-packing máximo para U calculado con el algoritmo MAXIMUM-2-PACK-UNICYCLE cuando $v_{N,0}$ es el vértice origen. El proceso para crear el árbol $T(U)$ de U es como sigue.

La línea 2 del Pseudocódigo 10 identifica los vértices $v_x, p(v_x), v_z, v_y$, y la arista e_1. El vértice v_x es el vértice marcado más cercano a $v_{N,0}$ en $U \setminus T^+_{v_{N,0}}$. El algoritmo identifica v_x mediante BFS en U, suponiendo que $v_{N,0}$ es el vértice origen. Si existe más de un vértice v_x que satisface este criterio, v_x es el vértice cuyo identificador único es el más bajo. Sea $p(v_x)$ el padre de v_x en el árbol BFS. Sea v_z el primer vértice en el camino de v_x a $v_{N,0}$ que pertenece al bloque cíclico B_N. Observe que v_z puede ser v_x, $p(v_x)$ o $p(p(v_x))$. Sea e_1 la primera arista en el camino de $v_{N,0}$ a v_x en el árbol BFS. Sea P el camino de v_z a $v_{N,0}$ en el ciclo B_N que no pasa por la arista e_1. Sea

$v_y \neq v_z$ el primer vértice marcado en P. Si dicho vértice no existe, entonces $v_y \leftarrow v_{N,0}$. Sea $P_1 \subseteq P$ el camino de v_z a v_y. La línea 3 crea una copia de U y le asigna el nombre de $T(U)$. Posteriormente, la línea 4 borra todas las aristas de P_1 en $T(U)$. Finalmente, la línea 5 agrega una arista por cada vértice $f(v_{N,m})$ a $f(p(v_x))$ en $T(U)$, tal que $v_{N,m} \in P_1$ en U, $v_{N,m} \neq v_y$, y $v_{N,m} \neq v_z$.

Ejemplo 2. Considere el uniciclo U de la Figura 7(a). En esta figura, los vértices resaltados representan a $S_{v_{N,0}}[2]$. Las líneas punteadas denotan a $T^+_{v_{N,z}}$. La Figura 7(b) es el árbol resultante, $T(U)$, después de la ejecución de UNICYCLE-TO-TREE en U con el conjunto $S_{v_{N,0}}[2]$. En la Figura 7(b), los segmentos de líneas representan las aristas eliminadas de U, mientras que las líneas gruesas representan a las nuevas aristas agregadas a $T(U)$. Finalmente, observe que cada vértice en U le corresponde un vértice en $T(U)$ y los marcados $M_{v_{N,0}}[2]$ y $M_{v_{N,0}}[1]$ son iguales.

2.4.2. Demostración de que UNICYCLE-TO-TREE es correcto

Sea $U = (V_U, E_U)$ un grafo uniciclo que contiene un bloque cíclico B_N. Sea $M_{N,0}[2]$ un marcado de U calculado por MAXIMUM-2-PACK-UNICYCLE que representa un conjunto 2-packing máximo, suponiento $v_{N,0}$ como el vértice origen.

Lema 2.10 *El marcado $M_{N,0}[2]$ de U es igual al marcado $M_{N,0}[1]$ de $T(U)$ generado por UNICYCLE-TO-TREE.*

Demostración. Suponga, sin pérdida de generalidad, que $T^+_{v_{N,0}} = v_{N,0}$. Se demuestra que ningún vértice cambia su marcado en cada uno de los dos subárboles enraizados en $f(v_{N,0})$ de $T(U)$.

Caso 1. Sea $v_{N,m}$ cualquier vértice de P_1 en U, excepto v_z y v_y. Sea $v^{\odot}_m$ el vértice marcado (si existe) más cercano a $v_{N,m}$ en $T^+_{v_{N,m}}$. Observe que $v_{N,m} \neq v^{\odot}_m$, puesto que $v_{N,m} \notin S_{N,0}[2]$.

El marcado $M_{N,m}[0]$ es el mismo en U y $T(U)$ puesto que dicho subgrafo es el mismo. Observe que existe un camino de dos aristas entre $f(v_{N,m})$ y $f(v_x)$ en $T(U)$, y que existe un camino de a lo más tres aristas entre $f(v_{N,m})$ y $f\left(v^{\odot}_m\right)$. De aquí que,

el camino de $f(v_x)$ a $f\left(v_m^{\circledcirc}\right)$ tenga a lo más cinco aristas. Puesto que $f(v_x)$ y $f\left(v_m^{\circledcirc}\right)$ pertenecen a $S_{N,0}[2]$, ningún otro vértice en dicho camino puede cambiar su marcado en $T(U)$. Este argumento se mantiene para todo vértice $f(v_{N,m})$.

De forma semejante, note que v_x es un vértice marcado en U y $T(U)$, y que la distancia entre $v_m^{\circledcirc}$ y v_x es al menos tres en $T(U)$. Luego, el marcado de $T^+_{v_{N,z}}$ en U es el mismo que el marcado en $T(U)$. En otras palabras, el marcado de cualquier vértice $v_m^{\circledcirc}$ no altera el marcado de ningún vértice $f(v_u)$, donde $v(u) \in T^+_{v_{N,z}}$ en U.

Note que el camino más corto entre $v_{N,0}$ y v_x en U es a lo más tres. Dicho camino también existe en $T(U)$ entre $f(v_{N,0})$ y $f(v_x)$. Puesto que v_x y $f(v_x)$ son vértices marcados en U y $T(U)$, respectivamente, entonces el marcado en ambos caminos es el mismo. Por lo tanto, los vértices marcados en ambos subárboles enraizados en $v_{N,0}$ en U y $f(v_{N,0})$ en $T(U)$ son los mismos.

Caso 2. Suponga que $f(v_y) \neq f(v_{N,0})$. Sea P_2 el camino de v_y a $v_{N,0}$ que no pasa por la arista e_1 en U. El camino P_2 también existe en $T(U)$ entre $f(v_y)$ y $f(v_{N,0})$. Observe que el subgrafo inducido por todos los vértices en $P_2 \cup C^+_{v_{N,k}}$, para todo $v_{N,k} \in P_2$ en U es el mismo para el subgrafo inducido por sus vértices correspondientes en $T(U)$. Sea $v_w \neq v_z$ el vecino de v_y en P_1 (vea la Figura 7). Puesto que v_y y $f(v_y)$ son vértices marcados en U y $T(U)$, respectivamente, entonces la eliminación de la arista entre $f(v_w)$ y $f(v_y)$ no altera el marcado de $f(v_y)$ ni el marcado de los vecinos de $f(v_y)$ a distancia menor o igual que dos en $T(U)$. Por lo tanto, el marcado de los vértices en este subárbol es el mismo para U y $T(U)$.

Cuando $f(v_y) = f(v_{N,0})$, no hay nada por demostrar puesto que hay tan solo un subárbol enraizado en $f(v_{N,0})$ en $T(U)$, y el Caso 1 maneja este caso. Cuando $T^+_{v_{N,0}} \neq v_{N,0}$, dicho subgrafo es el mismo en U y $T(U)$ y sus marcados $M_{N,0}[0]$ son iguales.

Note que puesto que ningún vértice cambia su marcado en los subárboles enraizados en $f(v_{N,0})$ de $T(U)$, entonces $f(v_{N,0})$ no puede cambiar su marcado en $T(U)$. ■

Lema 2.11 *El marcado $M_{N,0}[1]$ de $T(U)$ generado por UNICYCLE-TO-TREE, suponiendo $f(v_{N,0})$ como el vértice origen, representa un conjunto 2-packing máximo y es igual al marcado $\mathcal{M}_{N,0}[1]$ calculado en $T(U)$ por el algoritmo de Mjelde (2004), considerando*

$f(v_{N,0})$ *como el origen.*

Demostración. Suponga, sin pérdida de generalidad, que $T^+_{f(v_{N,0})} = f(v_{N,0})$ y $f(v_{N,0}) \neq f(v_y)$. Bajo estas suposiciones, el árbol $T(U)$ tiene dos ramas conectadas a $f(v_{N,0})$. Sea $T(U)^{\leftarrow f(v_y)}$ y $T(U)^{\leftarrow f(v_z)}$ las ramas de $T(U)$ que tienen a los vértices $f(v_y)$ y $f(v_z)$, respectivamente. Ahora, se demuestra que los marcados de cada rama generado por UNICYCLE-TO-TREE y por el algoritmo de Mjelde son iguales.

Caso 1. Rama $T(U)^{\leftarrow f(v_y)}$. Puesto que $f(v_y)$ es un vértice marcado en $T(U)$, el vértice $f(v_w)$ no impone ninguna restricción en el marcado de $f(v_y)$. Puesto que el subgrafo inducido por los vértices de $T(U)^{\leftarrow f(v_y)}$ también existe en U y MAXIMUM-2-PACK-UNICYCLE es una implementación del algoritmo de Mjelde, ambos marcados para $T(U)^{\leftarrow f(v_y)}$ generados por ambos procedimientos son iguales.

Caso 2. Rama $T(U)^{\leftarrow f(v_z)}$. Sea $f(v_{N,m})$ un vértice arbitrario del bloque B_N en la rama $T(U)^{\leftarrow f(v_z)}$. Observe que cada subgrafo inducido por los vértices de $T^+_{f(v_{N,m})}$ en $T(U)$ también existen en U, excepto por $T^+_{f(v_z)}$. Note que el marcado de estos subgrafos en U y $T(U)$ son el mismo, y que éstos usan una implementación del algoritmo de Mjelde. Por lo tanto, el marcado producido por UNICYCLE-TO-TREE y del algoritmo de Mjelde para estros subgrafos es el mismo. Ahora, se demuestra que el marcado en el subgrafo inducido por los vértices de $T^+_{f(v_z)}$ es el mismo en ambos procedimientos. Observe que todos los vértices $v_{N,m} \in P_1$ en U, donde $v_{N,m} \neq v_y$ y $v_{N,m} \neq v_z$, no están marcados en U ni en sus vértices correspondientes en $T(U)$.

Asimismo, note que cada uno de estos vértices se conecta a $f(p(v_x))$ en $T(U)$, el cual no es un vértice marcado. Puesto que la distancia entre cada $f\left(v^{\circledast}_m\right)$ a $f(p(v_x))$ es al menos dos, entonces ningún vértice en $T^+_{f(v_{N,m})}$ impone una restricción en el marcado de algún vértices de $T^+_{f(v_z)}$. Por lo tanto, el marcado para $T(U)^{\leftarrow v_z}$ es el mismo para ambos procedimientos.

Cuando $f(v_{N,0}) = f(v_y)$ existe sólo una rama conectada a $f(v_{N,0})$, y el Caso 1 del Lema 2.11 es verdadero para dicha rama.

Cuando $T_{f(v_{N,0})} \neq f(v_{N,0})$ (es decir, existen otras ramas conectadas a $f(v_{N,0})$), note que el subgrafo inducido por los vértices de dichas ramas existen en U y $T(U)$. Asimismo, observe que el marcado de aquellos vértices en U (el cual es el mismo en $T(U)$)

usó una variación del algoritmo de Mjelde. Por lo tanto, el marcado de dichas ramas es igual para ambos procedimientos.

Observe que puesto que ningún vértice cambia su marcado en $T(U)$ en ambos subárboles enraizados en $f(v_{N,0})$, entonces $f(v_{N,0})$ no puede cambiar su marcado en $T(U)$. ■

El algoritmo UNICYCLE-TO-TREE corre en $O(n)$ u.t.; no obstante, únicamente se emplea como componente para la demostración de que el algoritmo MAXIMUM-2-PACK-CACTUS es correcto.

2.5. Conjunto 2-packing máximo en un grafo cactus

Esta sección describe el algoritmo MAXIMUM-2-PACK-CACTUS, que se muestra en el Pseudocódigo 11, la demostración de que es correcto y su análisis de complejidad.

2.5.1. Descripción de MAXIMUM-2-PACK-CACTUS

La entrada para este algoritmo es un cactus $K = (V_K, E_K)$. La salida es un conjunto de vértices marcados, $S_{N,0}[2]$, que representa un conjunto 2-packing máximo para K.

La línea 2 del Pseudocódigo 11 calcula el árbol de bloques enraizado T_B y etiqueta sus bloques de acuerdo a su numeración postorden. El ciclo de las líneas 3 - 12 procesa de forma secuencial cada bloque. Este procedimiento obtiene $M_{i,0}[2]$, para cada bloque i. Las líneas 5 - 8 manejan el caso para un bloque clique y las líneas 9 - 11 para un bloque cíclico. Después de procesar el bloque B_N, el vértice $v_{N,0}$ almacena $M_{N,0}[2]$. Finalmente, en la línea 13, el algoritmo DISTRIBUTE-MAXIMUM-2-PACK-SOLUTION (no se muestra) distribuye el marcado final a todos los vértices de K.

2.5.2. Demostración de que MAXIMUM-2-PACK-CACTUS es correcto

Teorema 2.1 *Sea K un grafo cactus conectado no dirigido. El conjunto $S_{N,0}[2]$ generado por MAXIMUM-2-PACK-CACTUS es un conjunto 2-packing máximo para K.*

Pseudocódigo 11: MAXIMUM-2-PACK-CACTUS(K)

```
   Entrada: Un grafo cactus conectado K = (V_K, E_K), donde |V_K| ≥ 2.
   Salida : Un conjunto S_{N,0}[2] ⊆ V_K que representa un conjunto 2-packing máximo para K.
 1 begin
 2     T_B ←ROOTED-BLOCK-TREE-POSTORDER(K);
 3     for i ← 1 to N do
 4         T_{v_{i,j}}[r][0] ← INITIALIZE-BLOCK-TABLES(B_i, C_{B_i}), ∀j;
 5         if B_i is a clique block then
 6             T_{v_{i,0}}[r][1] ← PROCESS-CLIQUE-BLOCK(B_i);
 7             T_{v_{i,0}}[r][2] ← T_{v_{i,0}}[r][1]
 8         end
 9         if B_i is a cyclic block then
10             T_{v_{i,0}}[r][2] ← PROCESS-CYCLIC-BLOCK(B_i);
11         end
12     end
13     K ← DISTRIBUTE-MAXIMUM-2-PACK-SOLUTION(K, T_{v_{N,0}}[0][2]);
14     return K;
15 end
```

Demostración. Por inducción en el número de bloque i.

Inicialmente, ROOTED-BLOCK-TREE-POSTORDER construye un árbol de bloques enraizado T_B de K y etiqueta sus bloques de acuerdo a su numeración en postorden.

Caso base. Cuando $i = 1$, MAXIMUM-2-PACK-CACTUS procesa el bloque B_1 (el bloque hoja más a la izquierda de T_B). El bloque B_1 puede ser un clique o un ciclo.

- Cuando B_1 es un bloque clique. Por el Lema 2.1, INITIALIZE-BLOCK-TABLES y PROCESS-CLIQUE-BLOCK marcan correctamente los vértices de B_1.
- Cuando B_1 es un bloque cíclico. El ejecutar INITIALIZE-BLOCK-TABLES y PROCESS-CYCLIC-BLOCK es equivalente al algoritmo MAXIMUM-2-PACK-UNICYCLE cuando la entrada es un ciclo. Por el Lema 2.9, este procedimiento marca de forma correcta los vértices de B_1.

Hipótesis inductiva. Suponga que el Teorema 2.1 marca de forma correcta los bloques desde B_1 a B_{i-1} y que cada bloque cíclico de T_B usa la transformación UNICYCLE-TO-TREE.

Paso inductivo. Ahora se muestra que el Teorema 2.1 marca de forma correcta el bloque B_i. Por la hipótesis inductiva y por los Lemas 2.10 y 2.11, se pueden ver los bloques desde B_1 a B_{i-1} como un conjunto de árboles con sus vértices marcados de

forma óptima, de los cuales algunos de ellos están conectados a algún vértice de B_i. Existen dos casos.

- Cuando B_i es un bloque clique. Por el Lema 2.1, INITIALIZE-BLOCK-TABLES y PROCESS-CLIQUE-BLOCK marcan de forma óptima los vértices de B_i.
- Cuando B_i es un bloque cíclico. El grafo inducido por los vértices de $B_i \cup C^+_{B_i}$ forman un uniciclo. Por el Lema 2.9, INITIALIZE-BLOCK-TABLES y PROCESS-CYCLIC-BLOCK marcan de forma óptima los vértices de B_i. Finalmente, por los Lemas 2.10 y 2.11, se puede ver que el grafo inducido por los vértices de $B_i \cup C^+_{B_i}$ como un árbol con el mismo número conjunto de vértices marcados.

■

2.5.3. Análisis de complejidad de MAXIMUM-2-PACK-CACTUS

El tiempo de ejecución de MAXIMUM-2-PACK-CACTUS depende de el número de vértices en el bloque cíclico. De hecho, entre más grande el tamaño del bloque cíclico más grande es su tiempo de ejecución. Por el contrario, cuando no hay ciclos en el cactus, el algoritmo se ejecuta más rápido, en $O(n)$ u.t.

En el Pseudocódigo 11, la línea 2 requiere $O(n)$ u.t. La línea 4 requiere $O\left(|B_i| + \left|C_{B_i}\right|\right)$ u.t. La línea 6 requiere tiempo constante. La línea 10 requiere $O\left(|B_i|^2\right)$ u.t. Finalmente, la línea 13 requiere $O(n)$ u.t. Puesto que N es $O(n)$, el algoritmo MAXIMUM-2-PACK-CACTUS se ejecunta en $O\left(n^2\right)$ u.t.

Capítulo 3. Algoritmo distribuido para el conjunto 2-packing maximal en grafos Halin

3.1. Introducción

Sea $H = (V_H, E_H)$ un grafo Halin, como se define en el Capítulo 1. Esta sección presenta el algoritmo M2H (**M**AXIMAL_**2**-PACKING-SET_**H**ALIN) que encuentra un conjunto 2-packing maximal en grafos Halin no geométricos en $O(n)$ u.t., donde $n = |V_H|$. M2H combina una versión distribuida del algoritmo secuencial de Eppstein (Eppstein, 2016) con una variación del algoritmo de (Trejo-Sánchez y Fernández-Zepeda, 2014). Dado un grafo no dirigido G, el algoritmo de Eppstein determina si G es un grafo Halin o no. Si G es Halin, este algoritmo encuentra las aristas que pertenecen a alguna cara externa (la cara externa no es necesariamente única para los grafos Halin). A diferencia del algoritmo descrito en (Trejo-Sánchez y Fernández-Zepeda, 2014), M2H no requiere que el grafo de entrada sea geométrico. Incluso, la entrada para (Trejo-Sánchez y Fernández-Zepeda, 2014) es 1-outerplanar; mientras que, la entrada para M2H es 2-outerplanar.

3.2. Conjunto 2-packing en grafos Halin

3.2.1. Terminología y notación

Sea $\Gamma = (V_\Gamma, E_\Gamma)$ un 'subgrafo triangular' de H, tal que $\{u, v, w\} \subset V_H$ y $\{(u, v), (v, w), (w, u)\} \subset E_H$ (vea la Figura 8(b)). Sea $\Lambda = (V_\Lambda, E_\Lambda)$ un 'subgrafo doble triangular' de H tal que $\{a, u, v, w\} \subset V_H$ y $\{(a, u), (a, v), (a, w), (u, v), (v, w)\} \subset E_H$ (vea la Figura 9). Sea $\triangle^H$ ($\blacktriangle^H$) el conjunto de todos los subgrafos triangulares (doble triangulares) de H y las versiones contraidas de H (i.e., aquellos grafos generados a partir de H después de la ejecución del algoritmo de Eppstein).

Dado $\Gamma \in \{\triangle^H \cup \blacktriangle^H\}$, cualquier arista $(x, y) \in E_H$ tal que $x \in V_\Gamma$ y $y \notin V_\Gamma$ es una *periarista* y y es un *perivértice* de Γ. En la Figura 9(b), las aristas $(u, u'), (v, v')$, y

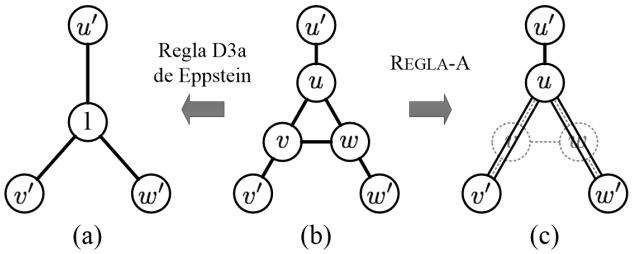


Figura 8. Transformación de subgrafos triangulares. (a) Grafo resultante después de aplicar la regla D3a de Eppstein. (b) Subgrafo triangular original. (c) Subgrafo resultante después de la ejecución de la REGLA-A propuesta en este trabajo. Las líneas paralelas, "||", representan a las superaristas.

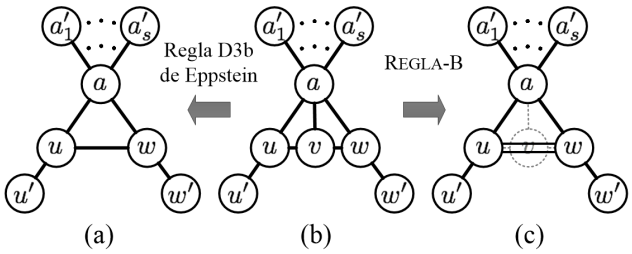


Figura 9. Transformación de subgrafos doble triangulares. a) Subgrafo resultante después de aplicar la regla D3b de Eppstein. b) Subgrafo doble triangular original. c) Subgrafo resultante después de la ejecución de la REGLA-B propuesta en este trabajo. Las líneas paralelas, "||", representan a las superaristas.

(w, w') son periaristas, y los vértices $\{u', v', w'\}$ son perivértices. Dos subgrafos $\Gamma_i, \Gamma_j \in \{\triangle^H \cup \not\triangle^H\}$ son *vecinos* si estos comparter al menos una periarista. Una *superarista* de x_i a x_j es un camino $\{x_i, x_{i+1}, \ldots, x_j\} \in V_H$, tal que $(x_k, x_{k+1}) \in E_H$, para toda k en $i \leq k < j$, y sólo x_i y x_j están 'activos'.

3.2.2. Encontrando una cara externa en los grafos Halin

M2H se basa principalmente en las reglas D3a y D3b de Eppstein (2016) para encontrar una cara externa en el grafo Halin. Aquí se implementan estas reglas en un ambiente distribuido y reciben el nombre de REGLA-A y REGLA-B, respectivamente. M2H primero identifica los subgrafos triangulares y doblemente triangulares en H para ejecutar estas reglas. Para reducir sobrecostos en tiempo, sólo los vértices con grado tres llevan a cabo esta identificación. Los subgrafos triangulares y doble triangulares siempre tienen al menos una arista en la cara externa. M2H también incorpora un procedimiento para evitar la ejecución de estas reglas en dos subgrafos vecinos $\Gamma_i, \Gamma_j \in \{\triangle^H \cup \not\triangle^H\}$. Además, M2H incluye un procedimiento para reducir el tamaño de las superaristas largas.

Selección de ápice. Un *ápice* para el subgrafo $\Gamma \in \{\triangle^H \cup \mathbb{A}^H\}$ es un vértice $x \in V_\Gamma$ que toma el valor de representante de Γ. Las siguientes reglas definen el ápice.

- **Caso 1.** Si $\Gamma \in \mathbb{A}^H$, su ápice es el vértice $x \in V_\Gamma$ con el grado más grande.
- **Caso 2.** Si $\Gamma \in \triangle^H$ y Γ no tiene ningún vecino $\Gamma' \in \{\triangle^H \cup \mathbb{A}^H\}$, el ápice de Γ es el vértice $x \in V_\Gamma$ de menor identificador.
- **Caso 3.** Si $\Gamma \in \triangle^H$ y Γ comparte una o dos periaristas con algún $\Gamma' \in \{\triangle^H \cup \mathbb{A}^H\}$, el ápice de Γ es el vértice $x \in V_\Gamma$ de menor identificador tal que x no es un extremo de alguna periarista compartida.
- **Caso 4.** Si $\Gamma \in \triangle^H$ y Γ comparte tres periaristas con algún $\Gamma' \in \{\triangle^H \cup \mathbb{A}^H\}$, el ápice de Γ es el vértice $x \in V_\Gamma$ con el menor identificador.

Regla-A. La regla D3a de Eppstein reemplaza un subgrafo triangular Γ por un 'supervértice', manteniendo las conexiones existentes hacia las periaristas (vea la Figura 8(a)). En la Regla-A, el ápice u permanece 'activo', mientras que los otros dos vértices en Γ, sean v y w, y las aristas $(u, v), (v, v'), (u, w), (w, w')$ y (v, w) permanecen 'inactivas' (vea la Figura 8(c)). Los vértices y aristas *inactivos* no son parte del grafo contraído, pero permiten la propagación de mensajes en el sistema distribuido subyaciente. La Regla-A también crea las superaristas (u, v, v') y (u, w, w') en los grafos contraídos de H.

Regla-B. La regla D3b de Eppstein transforma un subgrafo doble triangular con ápice en a en un grafo triangular con ápice en a, así como se muestra en la Figura 9(a). En la Regla-B, el vértice v y las aristas $(v, u), (v, a)$ y (v, w) se inactivan. La Regla-B también crea la superarista (u, v, w), como se ve en la Figura 9(c).

El algoritmo de Eppstein contrae subgrafos triangulares y doble triangulares de H hasta que alcanza un grafo K_4 o ninguna otra contracción es posible. Finalmente, mediante un procedimiento de expansión, este algoritmo determina una cara externa de H. M2H ejecuta un proceso de contracción similar, pero en forma distribuida. Éste aplica la Regla-A o la Regla-B, de forma simultánea, a cada $\Gamma \in \{\triangle^H \cup \mathbb{A}^H\}$, siempre y cuando estos subgrafos no sean vecinos. Este procedimiento también termina cuando

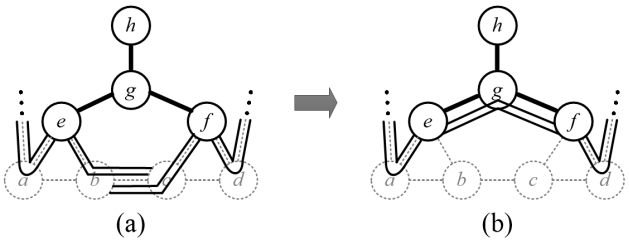


Figura 10. Reducción de superaristas. (a) Subgrafo con una superarista larga (e, b, c, f). (b) Subgrafo resultante después de la ejecución del procedimiento de reducción de superaristas. La superarista es (e, g, f). Las líneas parallelas, "||", representan las superaristas.

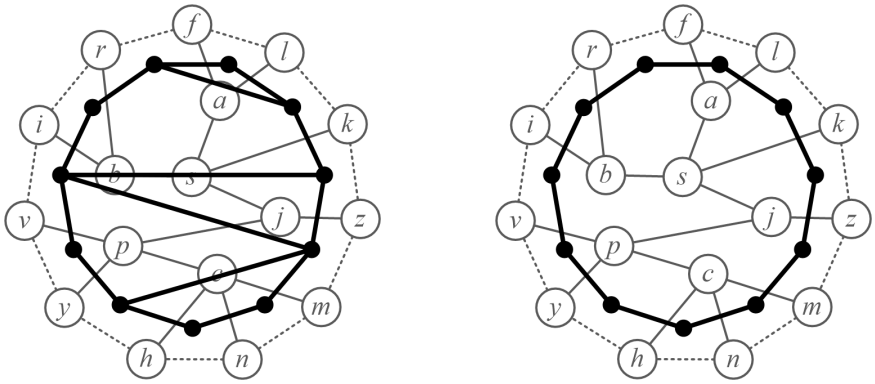
el grafo resultante es K_4. Finalmente, M2H ejecuta un procedimiento de expansión similar al de Eppstein.

Procedimientos freeze y unfreeze. M2H inhibe la contracción simultanea de dos subgrafos vecinos triangulares o doble triangulares para evitar la creación de un número excesivo de superaristas largas. Sean $\Gamma_i, \Gamma_j \in \{\triangle^H \cup \mathbb{\Delta}^H\}$ dos subgrafos vecinos con ápices en u_i y u_j, respectivamente, los procedimientos freeze y unfreeze trabajan de la siguiente manera:

- Si $\Gamma_i \in \triangle^H$ y $\Gamma_j \in \mathbb{\Delta}^H$, entonces Γ_i inhibe la ejecución de cualquier Regla-A o Regla-B hasta que reciba un mensaje de *unfreeze* del vértice u_j o un ancestro de u_j. Dicho vértice genera el mensaje de unfreeze cuando no pertenece a ningún otro subgrafo triangular o doble triangular.
- Si $\Gamma_i, \Gamma_j \in \triangle^H$ o $\Gamma_i, \Gamma_j \in \mathbb{\Delta}^H$ y $u_i < u_j$, entonces Γ_j se congela hasta que reciba un mensaje de unfreeze.

Reducción de superaristas. M2H reemplaza superaristas largas por cortas para reducir el retraso de propagación de mensajes. Sea g un ápice y e, f dos de sus vértices vecinos en su triángulo. Si existen dos super aristas $(e, \ldots, c)$ y $(f, \ldots, b)$ compartiendo una arista (i.e., existe un triángulo entre los vértices g, e, f), entonces el algoritmo cambia la superarista (e, b, c, f) por (e, g, f), como se ve en la Figura 10.

Cada vez que M2H ejecuta la Regla-A o Regla-B, el número de vértices en el subgrafo contraído se reduce al menos en una unidad. M2H ejecuta cada una de estas reglas en tiempo constante. Por lo tanto, M2H requiere $O(n)$ u.t. para reducir H a K_4. El procedimiento de expansión invierte el procedimiento de reducción y completa la



(a) Un grafo Halin H y su grafo dual débil $H^{w(*)}$, representado en negritas, el cual es un grafo outerplanar

(b) La cara externa del grafo dual débil, el cual es un ciclo

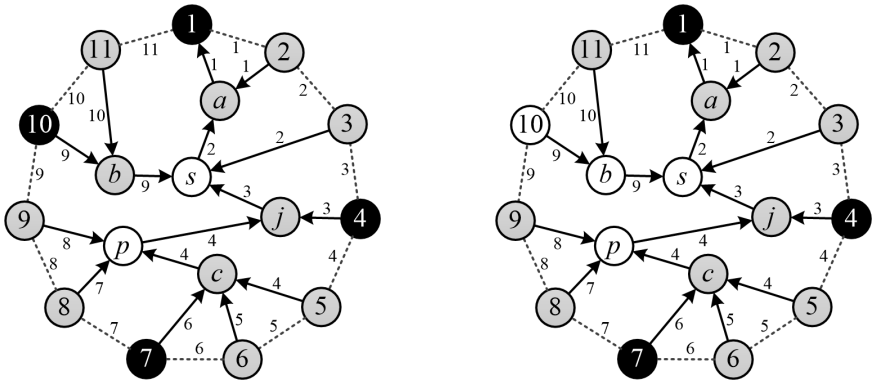
Figura 11. Grafo Halin con su grafo dual débil y cara externa.

identificación de una de las caras externas. M2H requiere $O(n)$ u. t. para encontrar una cara externa de H.

3.2.3. Descomposición de orejas en grafos Halin

Sea G un grafo plano. Entonces $G^* = (V_{G^*}, E_{G^*})$ es el *grafo dual* de G, donde V_{G^*} es el conjunto de caras de G y $(x, y) \in E_{G^*}$ si las caras x y y comparten al menos una arista en G. *El grafo dual débil* $G^{w(*)}$ es el subgrafo inducido $G^*[V_{G^*} - x]$, donde x representa la cara externa de G. El procedimiento de (Trejo-Sánchez y Fernández-Zepeda, 2014) para calcular la descomposición de orejas de un grafo outerplanar sigue un recorrido sobre su grafo dual débil, el cual es un árbol. El grafo dual débil para los grafos Halin es un grafo outerplanar (vea la Figura 11(a)). M2H sigue un recorrido sobre la cara externa de dicho grafo para calcular una descomposición de orejas en el grafo Halin (vea la Figura 11(b)).

Después de encontrar una cara externa de H, M2H elige un líder de forma óptima, entre los vértices de la cara externa, mediante el uso del algoritmo de Awerbuch (1987). Posteriormente, M2H asigna una numeración secuencial del conjunto $\{1, \ldots, |E_C|\}$ a todos los vértices de la cara externa, iniciando en el líder y continuando con su vecino de menor índice. Después de etiquetar los vértices, M2H enraiza el árbol T de H en el líder en $O(n)$ u. t. (vea las aristas con flechas en los grafos de la



(a) Un coloreo con una inconsistencia

(b) El mismo grafo con su coloreo corregido

Figura 12. Encontrando un conjunto 2-packing maximal en el grafo Halin. (los vértices *red, blue*$_1$ y *blue*$_2$ se muestran en *negro, gris* y *blanco*, respectivamente). El número al costado de cada arista representa su número de oreja.

Figura 12).

Para identificar las orejas de H, M2H usa un conjunto de tokens τ_i, donde $1 \leq i \leq |E_C|$. Cada arista que propaga el token τ_i pertenece a la oreja i (vea las etiquetas de las aristas en los grafos de la Figura 12). De forma semejante, cada vértice que recibe el token τ_i se marca a sí mismo como visitado. M2H repite el siguiente procedimiento para cada vértice i de la cara externa, iniciando en el líder (i.e., $i = 1$). El vértice i crea el token τ_i y lo transmite hacia el vértice $i+1$. Luego, el vértice $i+1$ retransmite τ_i hacia la raíz del árbol, siguiendo sólo las aristas del árbol, hasta que τ_i llega a un vértice previamente visitado. Posteriormente, τ_i regresa al vértice $i+1$, siguiendo el camino inverso sobre el árbol. La Figura 11 muestra un ejemplo de este procedimiento. Puesto que cada token recorre cada arista a lo más dos veces, este procedimiento toma $O(n)$ u. t.

3.2.4. Identificación de un conjunto 2-packing maximal

M2H assigna a cada vértice un color del conjunto $\{red, blue_1, blue_2\}$, de forma semejante a la subfase *red/blue* de (Trejo-Sánchez y Fernández-Zepeda, 2014). El algoritmo inicialmente asigna el color rojo al líder. M2H simultáneamente ejecuta el procedimiento de descomposición de orejas con la identificación del conjunto 2-packing set.

Durante la propagación del token τ_i, éste cuenta el número de vértices en la oreja. Este procedimiento se asemeja a la subfase de *counting* de (Trejo-Sánchez y Fernández-Zepeda, 2014). Cuando τ_i encuentra el primer vértice visitado, τ_i guarda el color de dicho vértice. Al momento que τ_i viaja de regreso, éste informa a cada vértice de los colores del primer y último vértice en la oreja, el número de vértices en el camino, y la posición de cada vértice en el camino. Con esta información, cada vértice en el camino determina su color. Este procedimiento se asemeja a la subfase *red/blue* de (Trejo-Sánchez y Fernández-Zepeda, 2014) puesto que usa el mismo conjunto de reglas.

Observe que con este procedimiento, la última oreja consiste sólo de la arista $(|E_C|, 1)$. Si en este último paso, el token τ_{E_C} detecta una inconsistencia entre los colores de los vértices E_C y 1, el vértice 1 inicia una fase de 'recoloreo' pero siguiendo la dirección opuesta en el ciclo externo. Durante este proceso, los vértices pueden únicamente recolorearse de *red* a $blue_1$ o $blue_2$, y sólo de $blue_1$ a $blue_2$. El procedimiento se detiene cuando ya no se necesita ningún cambio de color durante el procesamiento de una nueva oreja. M2H genera una descomposición de orejas de H e identifica un conjunto 2-packing maximal en $O(n)$ u.t. La complejidad total de M2H es $O(n)$ u. t.

3.2.5. Demostración de que el algoritmo es correcto

M2H sólo implementa variaciones del algoritmo de Eppstein (2016) y Trejo-Sánchez y Fernández-Zepeda (2014). La demostración de que M2H es correcto viene directamente de las demostraciones de dichos algoritmos.

Capítulo 4. Desmembración de grafos

4.1. Introducción

Este capítulo discute los conceptos de grafo ‘desmembrable’, ‘k-árbol parcial’, ‘lógica monádica de segundo orden’, el ‘teorema de Courcel’, ‘complejidad paramétrica’ y ‘anchura y desmembración de árbol’. Lo anterior con el objetivo de proporcionar las bases teóricas que permitan analizar la factibilidad de resolver ciertos problema computacionales NP-difíciles en algunas familias de grafos en tiempo polinomial. En este sentido, la literatura proporciona diversos teoremas que listan las características que el grafo y el problema deben cumplir. No obstante, generalmente las demostraciones de los teoremas son pruebas existenciales y no constructivas.

4.1.1. Grafos desmembrables

Una forma común de expresar un problema computacional en el área de análisis y diseño de algoritmos para grafos es la siguiente: “dado un grafo G extraído de una clase Γ, encontrar entre todos los subgrafos posibles H de G que satisfacen cierta propiedad P_G, i.e., un subgrafo óptimo (ya sea maximizar o minimizar un conjunto de vértices, encontrar un peso máximo, etc.) (Bern *et al.*, 1987)”. Existen otras formas de llamar a los grafos desmembrables; por ejemplo: grafos triangulados (Berge, 1985), recursivos (Hsieh, 2005), de circuito rígido (Dirac, 1961), cordales (Gavril, 1972), o grafos k-regulares (Mahajan y Peters, 1994). No obstante, en este capítulo se utiliza la definición de Bern *et al.* (1987).

Una familia de grafos es *desmembrable* (decomposable) si su construcción involucra un conjunto finito de reglas de composición recursivas que satisfacen las siguientes propiedades:

- **D1** Las reglas definen un conjunto finito de grafos primitivos o grafos base.
- **D2** Cada grafo contiene un conjunto (de tamaño constante) ordenado (posiblemente vacío) de vértices ‘terminales’.

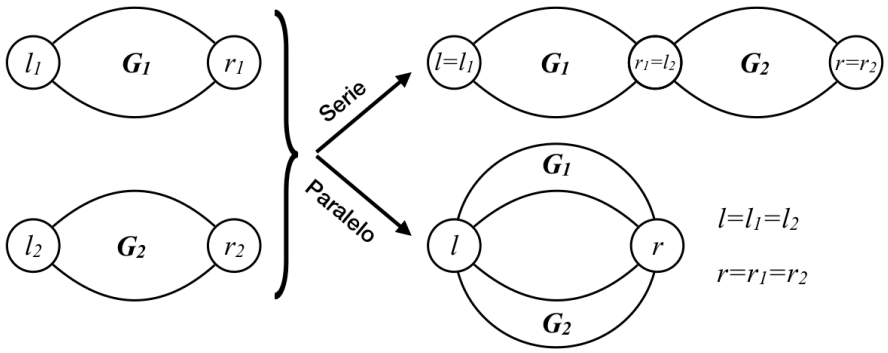


Figura 13. Una composición serie y paralelo de los grafos serie-paralelo G_1 y G_2.

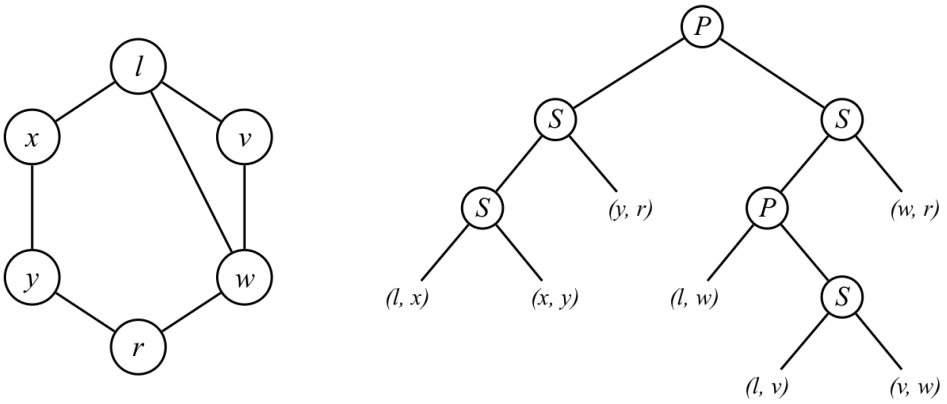
- **D3** Las operaciones binarias de composición son finitas, se ejecutan en los vértices terminales para unirlos añadiendo aristas o sólo para identificarlos.

La propiedad **D1** proporciona el caso base para la construcción del grafo desmembrable; mientras que las propiedades **D2** y **D3** son las instrucciones para crear nuevos grafos. Algunos grafos desmembrables son: árboles enraizados, grafos serie-paralelo, ancho de banda k, 1-outerplanar y Halin (Bern *et al.*, 1987; Arnborg *et al.*, 1991; Hsieh, 2005), entre otros.

Sea Γ una clase de grafos desmembrables. El *árbol parse* T_G del grafo $G \in \Gamma$ es un árbol en el cual las hojas corresponden al grafo base que permite la construcción de G (**D1**). Además, cada vértice de $v \in T_G$ representa el resultado de aplicar una operación de composición (**D2** y **D3**) al subárbol enraizado en v (Hsieh, 2005).

La Figura 13 presenta un ejemplo de cómo generar un grafo serie-paralelo, $G \in \Gamma$, a partir de reglas de composición. La notación empleada es la siguiente: $G = (V_G, E_G, Q)$ es un grafo con V_G su conjunto de vértices, E_G sus aristas y $Q \subset V_G$ el conjunto de vértices terminales. Sean $G_1 = (V_1, E_1, (l_1, r_1))$ y $G_2 = (V_2, E_2, (l_2, r_2))$ dos grafos serie-paralelo con vértices terminales (l_1, r_1) y (l_2, r_2), entonces (Hsieh, 2005):

- El grafo que se obtiene al emparejar a r_1 y l_2 es un grafo serie-paralelo, con terminales izquierdo y derecho l_1 y r_2, respectivamente. El grafo resultante es la *composición serie* de G_1 y G_2.
- La *composición paralela* de G_1 y G_2 se obtiene al emparejar a l_1 y l_2 y también a r_1 y r_2. Las nuevas terminales son $l = l_1 = l_2$ y $r = r_1 = r_2$.



(a) Un grafo serie-paralelo que también es un grafo 1-outerplanar

(b) Árbol parse del grafo del inciso (a).

Figura 14. Creación de un grafo 1-outerplanar mediante las reglas de composición del grafo serie-paralelo.

Bern *et al.* (1987) mencionan que a un par de grafos 1-outerplanares se les puede ver como grafos serie-paralelo, donde uno de los grafos en la composición es un grafo primitivo. La Figura 14(a) presenta un grafo serie-paralelo que también es un grafo 1-outerplanar. La Figura 14(b) muestra un árbol parse del grafo en el inciso (a). Las letras *S* y *P* denotan las composiciones en serie y paralelo, respectivamente. Observe que en este caso, el grafo primitivo consiste de un clique de tamaño dos, y éste se encuentra en las hojas del árbol parse.

Las propiedades de los grafos desmembrables han sido estudiadas en (Bern *et al.*, 1987; Arnborg y Proskurowski, 1989; He, 1991; Arnborg *et al.*, 1991) y en (Mahajan y Peters, 1994); asímismo, presentan soluciones a algunos problemas computacionales en esta clase de grafos mediante algoritmos secuenciales o paralelos.

Un problema pertenece a la clase de *problemas de optimización de subgrafo* si el objetivo es encontrar un subgrafo $G' \subseteq G$ que satisfaga alguna propiedad P_G de optimización sobre G. Algunos ejemplos de esta clase de problemas son encontrar un conjunto independiente máximo en G, el problema de matching máximo y el conjunto dominante (Hsieh, 2005).

Bern *et al.* (1987) proponen un algoritmo de programación dinámica que usa 'tablas multiplicación' para resolver algunos problemas de optimización de subgrafo. El algoritmo recorre los nodos del árbol parse T_G, iniciando en las hojas para terminar en la raíz, y asocia a cada vértice en el nodo un 'representante óptimo.' Sea v un vérti-

ce que pertenece a un nodo hoja, entonces el *representante óptimo* para v es el par (B_i, H), donde B_i es un grafo primitivo y H es conjunto de vértices óptimo en B_i. Ahora suponga que v es un vértice en algún nodo interno de T_G; el representante óptimo en el subgrafo G_v es (G_v, H), tal que H es un conjunto de vértices óptimo compuesto por alguna función que involucra las reglas de composición del árbol parse y a los representantes óptimos de los descendientes de v. Jamieson (2007) generaliza este enfoque para problemas de optimización que involucran conjuntos de aristas.

Teorema 4.1 *Bern* et al. *(1987). Suponga que* Γ *es una clase de grafos desmembrables para la cual se puede encontrar un árbol parse de tamaño lineal. Suponga que* P_G *es una propiedad regular computable de subconjuntos de vértices con respecto de* Γ*. Entonces existe un algoritmo de tiempo lineal que encuentra un subconjunto óptimo (ya sea de cardinalidad máxima o mínima) que satisface la propiedad* P_G *en el grafo parse de* Γ*.*

El Teorema 4.1 proporciona información sobre los problemas de optimización de subgrafo que se pueden resolver eficientemente. No obstante, así como lo señalan Mahajan y Peters (1994), el teorema tiene una desventaja, no proporciona alguna garantía de cómo saber si P_G es regular con respecto a Γ. Para completar esta parte, Mahajan y Peters (1994) introducen el término propiedad *uniformemente regular* como aquella propiedad *local* que se puede determinar al examinar un vecindario acotado de los vértices en el grafo, y que es regular para todo grafo en Γ. En particular, la siguiente definición sirve para el objetivo de este trabajo:

Un par de vértices en el grafo son *j-independientes*, para $j \geq 1$, si el camino más corto entre éstos tiene más de j aristas. El par (G, H) es j-independiente si los vértices en H son j-independientes en G.

Corolario 4.1 *(Mahajan y Peters, 1994). j-independencia es una propiedad de subgrafo uniformemente regular para cualquier* $j \geq 1$*.*

Teorema 4.1 *(Mahajan y Peters, 1994). Existe un procedimiento eficiente para calcular las tablas de multiplicación para cualquier propiedad local de subgrafo para cualquier grafo en* Γ*.*

Observación 4.1 *Sea $G = (V_G, E_G)$ un grafo 1-outerplanar y P_G un conjunto 2-packing sobre V_G. Entonces, con base en el trabajo de Bern* et al. *(1987) y Mahajan y Peters (1994), se tiene lo siguiente:*

- *El grafo G es un grafo desmembrable.*
- *Es posible obtener un árbol parse de tamaño lineal de G.*
- *P_G es una propiedad uniformemente regular; por lo tanto, regular en Γ.*
- *Las tablas para P_G se pueden calcular eficientemente.*

Entonces, por el Teorema de 4.1, existe un algoritmo de tiempo lineal para encontrar un conjunto 2-packing máximo en el grafo 1-outerplanar.

Adicionalmente, todo grafo desmembrable está acotado por un valor fijo de 'anchura de árbol', sea k dicho valor, entonces todo grafo con 'anchura de árbol' a lo más k es un grafo 'k-árbol parcial' (Wimer, 1987; van Leeuwen, 1990).

4.2. *k*-árbol parcial

Un grafo es *k-árbol* (k-tree) si es un grafo completo de k vértices (K_k), o tiene un vértice v de grado k tal que su vecindario abierto está completamente conectado, y al quitar v, así como sus aristas incidentes, el grafo resultante es un k-árbol. Un grafo *k-árbol parcial* (partial k-tree) es un subgrafo (no necesariamente inducido) de un k-árbol. Los k-árbol son grafos maximales con cierta 'anchura de árbol', tal que si se agregan más aristas también se incrementa la 'anchura de árbol' (Nešetřil, 2008).

Dentro de la jerarquía de los k-árbol parciales se encuentran los siguientes grafos:

- **1-árbol**: árbol (Patil, 1986)
- **2-árbol**: grafo serie-paralelo y 1-outerplanar maximal (Hwang *et al.*, 1992)
- **3-árbol**: redes apolonias y grafos cordales maximales (Frieze y Tsourakakis, 2011).

Tabla 1. Algunos problemas que se pueden resolver en grafos k-árbol parciales. De acuerdo a Arnborg y Proskurowski (1989). Su metodología es factible para los valores de k de la segunda columna.

Problema	**Valor**
Conjunto independiente	9 a 13
Conjunto dominante	4 a 8
Grafo k-colorable	7 a 8
Circuito Hamiltoniano	4 a 7
Confiabilidad de la red	3 a 8

Arnborg y Proskurowski (1989) presentan una metodología para diseñar algoritmos para grafos en tiempo lineal cuando el grafo se puede empotrar en un k-árbol para un valor fijo de k. Esta metodología supone que el empotramiento del grafo se proporciona como parte de la entrada del problema; si no es así, la complejidad es polinomial. La Tabla 1 presenta algunos problemas que encuentran solución en tiempo lineal en grafos k-árbol parciales. No obstante, la complejidad en k suele ser exponencial, como en el caso del problema del conjunto independiente que es $(k+1)2^{k+1}$.

Observación 4.2 *Sea $G = (V_G, E_G)$ un grafo 1-outerplanar, entonces G también es un grafo 2-árbol parcial. Con base en el trabajo de Arnborg y Proskurowski (1989) es posible encontrar eficientemente algún conjunto de vértices óptimo en G. No obstante, no es claro si es posible calcular un conjunto 2-packing máximo.*

4.3. Lógica monádica de segundo orden

La 'Lógica Mónádica de Segundo Orden, LMSO' es una forma sistemática de construir algoritmos de programación dinámica para resolver problemas expresados en LMSO para grafos de 'anchura de árbol' acotada.

La *Lógica Monádica de Segundo Orden, LMSO* es una extensión de la 'lógica monádica de primer orden, *LMPO*' que permite cuantificar no sólo sobre objetos, sino conjuntos de objetos. De forma general, la *LMPO* está compuesta por:

- Conectores lógicos: **y** $\wedge$, **o** $\vee$, **negación** $\neg$, **implica** $\Longrightarrow$ y **sí y sólo sí** $\Longleftrightarrow$
- Variables individuales: $x, y, z, x_1, x_2, \ldots$

- Cuantificadores existencial ∃ y universal ∀
- Predicados

La *LMSO* se caracteriza por contener a la primera y admitir conjuntos de variables: $X, Y, Z, \ldots$, pruebas de pertenencia $\in$ y cuantificar sobre conjuntos de variables. Adicionalmente, algunos autores incluyen el término *Logica Monádica Extendida de Segundo Orden, LMESO*, para expresar que la LMSO también permite medir el tamaño de los conjuntos (Kneis y Langer, 2009); por lo general no se hace distinción entre la LMSO y la LMESO.

En 1990, Courcelle (1990) demostró que todo problema definido en LMSO se puede resolver, incluso en tiempo lineal, en grafos de 'anchura de árbol' acotada. La demostración consiste en la construcción de un autómata determinista; no obstante, su construcción se considera complicada en términos prácticos (Frick y Grohe, 2004). Para demostraciones alternativas vea (Arnborg *et al.*, 1991; Kneis y Langer, 2009).

El problema del 3-*coloreo* pregunta si dado un grafo no dirigido, es posible colorear sus vértices con tres colores, tal que los vértices extremos de cualquier arista tengan distinto color. Se utiliza este problema para ejemplificar su definición en LMSO:

Problema del 3-coloreo: Existe un partición $W_1 \cup W_2 \cup W_3 = V_G$ en tres colores:

$$\begin{aligned} \textbf{3colorable} = \exists W_1, W_2, W_3 \subseteq V_G : \textbf{partición}(W_1, W_2, W_3) \wedge \\ \textbf{indp}(W_1) \wedge \textbf{indp}(W_2) \wedge \textbf{indp}(W_3). \end{aligned} \tag{11}$$

Donde:

$$\begin{aligned} \textbf{partición}(W_1, W_2, W_3) = \forall v \in V_G : (v \in W_1 \wedge v \notin W_2 \wedge v \notin W_3) \vee \\ (v \notin W_1 \wedge v \in W_2 \wedge v \notin W_3) \vee \\ (v \notin W_1 \wedge v \notin W_2 \wedge v \in W_3) \end{aligned} \tag{12}$$

$$\textbf{indp}(W) = \forall u, v \in W : (u, v) \notin E_G \tag{13}$$

Para expresar que los dos vértices extremos de la arista $(v, w) \in E_G$ no deben tener el mismo color:

$$\begin{aligned} \forall v, w \in V_G : (v, w) \in E_G \Rightarrow \\ (\neg(v \in W_1 \wedge w \in W_1) \wedge \neg(v \in W_2 \wedge w \in W_2) \wedge \neg(v \in W_3 \wedge w \in W_3)) \end{aligned} \tag{14}$$

Un *conjunto dominante* para un grafo $G = (V_G, E_G)$ es un conjunto $S \subseteq V_G$ tal que para cada vértice $x \in V_G$, $x \in S$ o x es adyacente a un vértice en S. Lo anterior se expresa de la siguiente forma:

$$\textbf{Dom}(S) = \forall x \in V_G, \exists y \in V_G : (x \in S) \vee ((y \in S) \wedge (x, y) \in E_G) \tag{15}$$

Ahora se procede a definir el problema del conjunto 2-packing como un partición de vértices de dos bloques $(\hat{S}, \bar{S}) \subseteq V_G$. El conjunto $\hat{S}$ denota a los vértices seleccionados que forman un conjunto 2-packing y $\bar{S}$ al resto de los vértices. Además, la distancia entre cada par de vértices en $\hat{S}$ es al menos tres aristas. Entonces, el conjunto 2-packing se expresa como sigue:

$$\textbf{2-packing}\left(\hat{S}, \bar{S}\right) : \textbf{partición}\left(\hat{S}, \bar{S}\right) \wedge \textbf{dist3}^{+}\left(\hat{S}\right) \tag{16}$$

Donde cualquier vértice pertenece al conjunto $\hat{S}$ o $\bar{S}$, pero no a ambos:

$$\begin{aligned} \textbf{partición}\left(\hat{S}, \bar{S}\right) = \forall v \in V_G : \left(v \in \hat{S} \wedge v \notin \hat{S}\right) \vee \\ \left(v \notin \hat{S} \wedge v \in \hat{S}\right) \end{aligned} \tag{17}$$

La distancia entre cualquier par de vértices en $\hat{S}$ es al menos tres aristas:

$$\textbf{dist3}^{+}\left(\hat{S}\right) = \forall u, v \in \hat{S}, u \neq v : ((u, v) \notin E_G) \wedge ((\mathbf{N}(u), v) \notin E_G) \tag{18}$$

La definición de $\textbf{dist3}^{+}\left(\hat{S}\right)$ se apoya de la definición del 'vecindario abierto' de un vértice. El *vecindario abierto* $\mathbf{N}(u)$ del vértice $u \in V_G$ es el conjunto de vértices $X \subset V_G$ adyacentes a u:

$$\mathbf{N}(u) = \forall u \in V_G, \exists X \subset V_G : \forall x \in X, (u, x) \in E_G \tag{19}$$

Teorema 4.2 *(Courcelle, 1990) Suponga que φ es una fórmula en LMSO y G es un grafo de n vértices. Suponga además que se conoce una 'desmembración' de árbol de G de 'anchura' k. Entonces, existe un algoritmo que verifica si φ se satisface en G en tiempo $f(|\varphi|, k)n$ para alguna función computable f.*

Posteriormente, Kneis y Langer (2009) generalizan el Teorema 4.2 para aplicarlo a problemas de optimización.

Teorema 4.3 *(Kneis y Langer, 2009) Dado un grafo G de orden n con anchura de árbol w (w constante), una fórmula opt $U \subseteq V$ $\varphi(U)$ en LMSO, donde φ es una fórmula en lógica de primer orden con vocabulario (adj, U) y opt $\in \{min, max\}$, la fórmula φ regresa el tamaño del conjunto óptimo U en el cual $\varphi(U)$ se satisface.*

La Tabla 2 presenta algunas propiedades expresables en LMSO. La notación que se emplea en la columna de *Clasificación* corresponde a (Garey y Johnson, 1979). Los prefijos *GT* y *ND* significan teoría de grafos y diseño de redes, respectivamente. El número a la derecha del prefijo es el número del problema en (Garey y Johnson, 1979).

Observación 4.3 *Con base en los Teoremas 4.2 y 4.3 se puede completar la Observación 4.2, puesto que ahora se sabe que es posible calcular la propiedad P_G. No obstante, el Teorema 4.2 requiere también la 'anchura de árbol' y 'una desmembración de árbol'. La sección 4.4 presenta estos conceptos.*

Tabla 2. Algunas propiedades expresables en LMSO Arnborg *et al.* (1991).

Problema	**Clasificación**
Cubrimiento de vértices	GT1
Conjunto dominante	GT2
Retroalimentación con conjunto de vértices	GT7
Retroalimentación con conjunto de aristas	GT8
Empatamiento mínimo maximal	GT10
Partición en cliques	GT15
Clique	GT19
Conjunto independiente	GT20
Subrafo completo bipartito balanceado	GT24
Grafo bipartito	GT25
Completar el ciclo de Hamilton	GT34
Árbol de expansión con el máximo número de hojas	ND2
Camino más corto con peso restringido	ND30
Trayectorias separadas de longitud máxima fija	ND42

4.4. Anchura de árbol y desmembración de árbol

Downey y Fellows (2013) presentan una forma de lidiar con problemas NP-difícil bajo la observación de que, en algunos casos específicos, existen ‘parámetros’ adicionales asociados a la complejidad del problema. A esta clase de análisis se le conoce como ‘complejidad paramétrica’. La *complejidad paramétrica* mide la complejidad de un algoritmo en términos de la longitud del problema n y de un parámetro numérico que no necesariamente depende de n (Fomin y Kratsch, 2010). Si el parámetro adicional es pequeño, existe la posibilidad de desarrollar un algoritmo que resuelva ese caso específico en tiempo polinomial.

Dentro de la clase de complejidad paramétrica se dice que un problema es *Fixed Parameter Tractable* (*FPT*) con respecto a un parámetro k, si existe una solución que se ejecute en $f(k)n^{O(1)}$ u.t., donde f es una función que depende de k y es independiente al tamaño del problema. Entre los parámetros existentes se encuentra la ‘anchura de árbol’. Este parámetro mide qué tan cercana es la estructura de un grafo cualquiera a un árbol (Robertson y Seymour, 1984). Cuando la anchura de árbol es acotada por una constante, diversos problemas NP-difícil se convierten en FPT y éstos, generalmente, llegan a tener solución (incluso en tiempo lineal) mediante algoritmos de programación dinámica.

A la ‘anchura de árbol’ y la ‘desmembración de árbol’ los han estudiado autores como Arnborg y Proskurowski (1989) que abordan estos conceptos desde el punto de vista de los grafos k-árbol parciales, contextos diferentes como en (Bertele y Brioschi, 1972), y bajo otro nombre como ‘funciones-S’ por Halin (1976). No obstante, el presente trabajo se adhiere a las definiciones de Robertson y Seymour (1984).

La definición de ‘anchura de árbol’ se auxilia de la ‘desmembración de árbol’ de un grafo. Una *desmembración de árbol* de un grafo $G = (V_G, E_G)$ es un par $\mathcal{T} = \left(T, \{X_t\}_{t \in V_T}\right)$ (Cygan *et al.*, 2015), donde $T = (V_T, E_T)$ es un árbol en el que cada nodo $t \in V_T$ se le asigna un subconjunto de vértices $X_t \subseteq V_G$, llamado bolsa (o parte (Diestel, 2018)), tal que las siguientes tres condiciones se cumplen:

- **T1** Cada vértice de V_G se encuentra en al menos una bolsa; i.e., $\bigcup_{t \in V_T} X_t = V_G$.
- **T2** Para cada arista $(u, v) \in E_G$, existe un nodo $t \in V_T$ tal que X_t contiene a los vértices u y v.
- **T3** Para cada $u \in V_G$, el conjunto de vértices $T_u = \{t \in V_T\} : u \in X_t$ induce un subárbol conectado con el conjunto de nodos correspondientes a las bolsas que contienen al vértice u.

La *anchura* de una desmembración de árbol es la cardinalidad de la bolsa de mayor tamaño menos un elemento. Por ejemplo, la anchura de árbol de un clique de n vértices es $n-1$. Restar un elemento permite que la anchura de la desmembración de árbol del grafo árbol sea de 1 (Flum y Grohe, 2006). Lo anterior porque las bolsas de su desmembración contiene cliques de tamaño 2.

$$\text{Anchura de desmembración de árbol} = \max_{t \in V_T} |X_t| - 1 \tag{20}$$

La *anchura de árbol* de G, denotado por $tw(G)$ es la anchura mínima posible de todas las desmembraciones de árbol de G. Para facilitar la notación, considere que $k \leftarrow tw(G)$. La Tabla 3, extraída de (Downey y Fellows, 2013), presenta algunas familias de grafos cuya anchura de árbol se conoce.

Tabla 3. Anchura de árbol para algunas familias de grafos

Familias de grafos	**Anchura acotada de árbol**
Árbol	1
Casi árbol (k)	$k+1$
k-árbol parcial	k
Ancho de banda k	k
Ancho de corte k	k
Plano de radio k	$3k$
Serie-paralelo	2
1-outerplanar	2
Halin	3
k-outerplanar	$3k-1$
Grafo cordal con clique máximo de tamaño k	$k-1$
Camino no dirigido con clique máximo de tamaño k	$k-1$
Camino dirigido con clique máximo de tamaño k	$k-1$
Grafo de intervalos con clique máximo de tamaño k	$k-1$
Grafo de intervalos propios con clique máximo de tamaño k	$k-1$
Grafo de arcos circulares con clique máximo de tamaño k	$2k-1$
Grafo de arcos circulares propios con clique máximo de tamaño k	$2k-2$

A partir de este momento, se sigue la convención de Cygan *et al.* (2015) para diferenciar entre los elementos de un grafo y aquellos de una desmembración de árbol. Si $v \in V_G$, entonces v es un vértice; si $t \in V_T$, t es un nodo. Adicionalmente, se expresa una desmembración de árbol de G como $\mathcal{T}_G$.

4.4.1. Motivación

Si se conoce la desmembración de árbol de un grafo $G = (V_G, E_G)$ y su anchura de árbol es a lo más una constante k, entonces diversos problemas intratables en el grafo general se pueden resolver en tiempo polinomial e incluso lineal. Algunos de estos problemas son el conjunto independiente, conjunto dominante, cubrimiento de vértices, circuito Hamiltoniano, árbol de Steiner, entre otros. Lo mismo aplica incluso para sistemas estadísticos y sistemas que trabajan bajo incertidumbre (Lauritzen y Spiegelhalter, 1990), o sistemas expertos. Por ejemplo, van der Gaag (1990) expone que algunos de los sistemas expertos más eficientes son aquellos cuya anchura de árbol es pequeña.

Asimismo, la anchura de árbol tiene relación con otras áreas; por ejemplo, en la

factorización de Cholesky. Dicha factorización describe cómo descomponer una matriz simétrica como el producto de una matriz superior y la transpuesta de la matriz triangular inferior. Es posible modelar este sistema mediante un grafo y acotar el tamaño máximo de estas matrices mediante la anchura de árbol del grafo (Bodlaender *et al.*, 1995).

En el área de la teoría de la evolución los científicos buscan un árbol filogenético entre especies que represente las relaciones evolutivas entre organismos. En estos árboles, un par de especies están muy relacionadas si tienen un ancestro común reciente, y menos relacionadas si tienen un ancestro común menos reciente. En trabajos como (Agarwala y Fernández-Baca, 1993; Bodlaender *et al.*, 1992) y (Kannan y Warnow, 1994) modelan el problema como: dado un grafo $G = (V_G, E_G)$ con un coloreo de vértices, encontrar una asignación de aristas en G tal que el grafo resultante sea cordal sin unir vértices del mismo color. El problema se resuelve mediante una desmembración de árbol $\mathcal{T}_G$ donde los pares de vértices en las bolsas tienen diferente color.

Otras áreas de aplicación son el diseño de circuitos VLSI (Deo *et al.*, 1987), procesamiento del lenguaje natural (Kornai y Tuza, 1992) y esquemas de enrutamiento en redes (Dourisboure y Gavoille, 2002; Fraigniaud, 2005).

4.4.2. Cálculo de una desmembración de árbol

Determinar si un grafo G tiene anchura de árbol a lo más k es un problema NP-completo (Arnborg *et al.*, 1987) incluso si se restringe a los grafos planos. La Tabla 4 presenta algunos resultados conocidos referentes a lo complejo de determinar la anchura de árbol de G (Bodlaender, 1993).

Bodlaender (1996) presentó un algoritmo FPT (que se ejecuta en tiempo lineal cuando el valor de k está acotado por una constante) que decide si la anchura de árbol de el grafo de entrada G es a lo más k; de ser así, el algoritmo además regresa una desmembración de árbol de G de anchura a lo más k. No obstante dicho algoritmo contiene ‘constantes muy grandes ocultas’ que imposibilitan su implementación y uso con propósitos prácticos (Niedermeier, 2006).

Tabla 4. Complejidad de encontrar la anchura de árbol de algunas familias de grafos

Familias de grafo	**Complejidad**
Grafo de grado acotado	NP-completo
Árboles y bosques	Lineal
Grafos serie-paralelo	Lineal
Grafo 1-outerplanar	Lineal
Grafo Halin	Lineal
Grafo r-outerplanar	Lineal
Grafo plano	Problema abierto
Grafo cordal	Polinomial
Grafo cordal tipo estrella	Polinomial
Grafo cordal tipo r-estrella	Polinomial
Grafo co-cordal	Polinomial
Grafo split	Polinomial
Grafo bipartito	NP-completo
Grafo de permutación	Polinomial
Grafo de permutación circular	Polinomial
Grafo cocomparable	NP-completo
Cografos	Polinomial
Grafo bipartito cordal	Polinomial
Grafo de intervalos	Polinomial
Grafo de arcos circulares y circular	Polinomial

También, debido al trabajo de Bodlaender (1988) se sabe que la anchura del grafo r-outerplanar es a lo más $3r-1$. Posteriormente, Katsikarelis (2013) complementa dicho resultado con una implementación algorítmica que encuentra una desmembración de árbol para el grafo r-outerplanar. La complejidad del algoritmo es $O(rn)$ en tiempo y espacio. Cuando el grafo de entrada es 1-outerplanar, la anchura de árbol del grafo es a lo más 2. Siguiendo el algoritmo de Katsikarelis (2013), el procedimiento para encontrar una desmembración de árbol para el grafo 1-outerplanar es como sigue:

- **Fase 1:** Sean $v, x, y \in V_G$, tal que v es de grado 2 y tanto x como y son vértices vecinos de v.
  - Remueva v de V_G así como sus aristas incidentes.
  - Si no existe la arista $(x, y) \in E_G$, agréguela.
  - Maneje de forma semejante a los vértices de grado 1.
  - Repita este proceso hasta que sólo quede una arista $(v, w) \in E_G$ en G.
- **Fase 2:** Para construir $\mathcal{T}_G$:

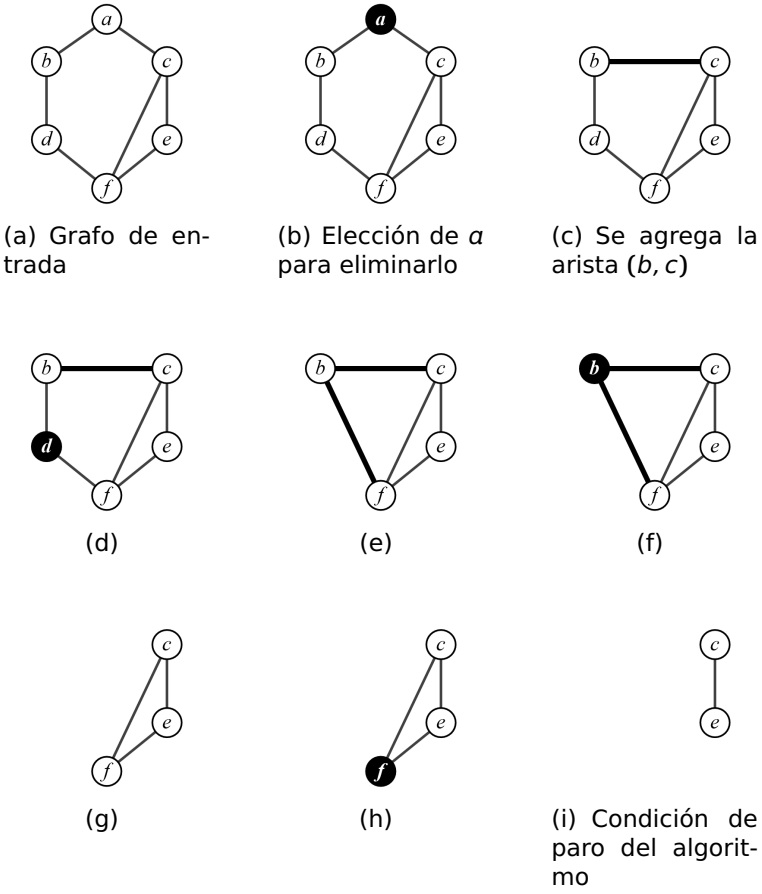


Figura 15. Primera fase del algoritmo de Katsikarelis (2013). El algoritmo elige al vértice resaltado v, en cada figura, para ser removido. Las aristas en negritas se agregan para unir vértices vecinos de v.

- Agregue $(v, w) \in E_G$ y sus vértices extremos como nodo a $\mathcal{T}_G$.
- Por cada vértice u eliminado en la **Fase 1**, agregue un nodo t a $\mathcal{T}_G$ cuyo contenido sea u y sus vecinos.
- Agregue una arista entre t y el nodo agregado previamente a $\mathcal{T}_G$ que contiene a los vecinos de u.

Del procedimiento de Katsikarelis (2013), note que: puesto que G es 1-outerplanar, siempre existe un vértice de grado 1, o grado 2; además, el subgrafo resultante después de eliminar un vértice y sus aristas incidentes es 1-outerplanar. Por este motivo, el procedimiento se puede repetir hasta que sólo quede una arista con sus vértices extremos. Por construcción, la anchura de $\mathcal{T}_G$ es a lo más 2.

Las figuras 15(a) - 15(i) muestran un ejemplo de la ejecución de la **Fase 1** del algoritmo de Katsikarelis (2013). La Figura 15(a) presenta al grafo de entrada. El algoritmo selecciona al vértice a para eliminarlo de V_G (Figura 15(b)). Posteriormente,

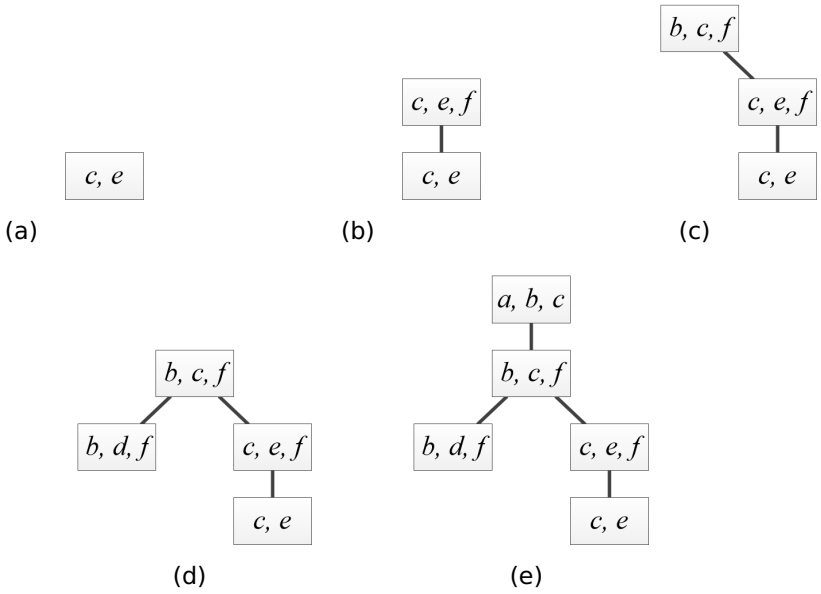


Figura 16. Segunda fase del algoritmo de Katsikarelis (2013).

en la Figura 15(c) se agrega la arista (b, c) por ser vecinos del vértice eliminado y no estar previamente unidos (véase la arista negrita). Un comportamiento similar se ve en las figuras 15(d) - 15(h). Note que en la Figura 15(f) los vecinos del vértice b ya se encuentran unidos por una arista, por ello no se agrega ninguna arista adicional. El algoritmo se detiene cuando $|V_G| = 2$; es decir, G es un clique de tamaño 2, vea la Figura 15(i).

La **Fase 2** del algoritmo de Katsikarelis (2013) se presenta en las figuras 16(a) - 16(e). La Figura 16(a) inicia la creación de $\mathcal{T}_G$ con el primer nodo $\{c, e\}$ correspondiente al subgrafo de la Figura 15(i). La Figura 16(b) muestra la unión del nodo $\{c, e, f\}$ (subgrafo de la Figura 15(h)) con el nodo $\{c, e\}$. En la Figura 16(c) se agrega el nodo $\{b, c, f\}$ (vértice resaltado y sus dos vecinos en el subgrafo de la Figura 15(f)) con $\{c, e, f\}$, ésto se debe a que la intersección entre sus elementos es mayor que con el resto del grafo. Se observa un comportamiento semejante en las figuras 16(d) y 16(e).

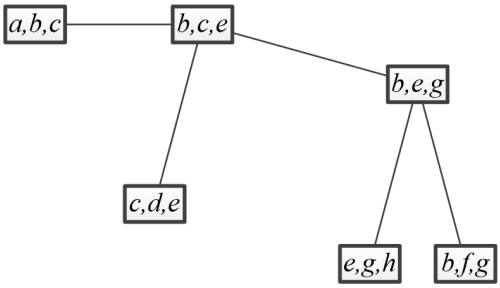
4.4.3. Desmembración agradable de árbol

Existe más de una estructura de desmembración de árbol; posiblemente la más general es la de Cygan *et al.* (2015). No obstante, la desmembración de árbol puede tener restricciones adicionales; por ejemplo, que sus partes induzcan subgrafos conectados, que tengan r-paths entre ciertos subgrafos (Diestel y Müller, 2018), o que la desmembración sea 'agradable'. Particularmente a nosotros nos interesa la última desmembración. Una desmembración de árbol es *agradable* (nice) si es un árbol binario que satisface las siguientes condiciones.

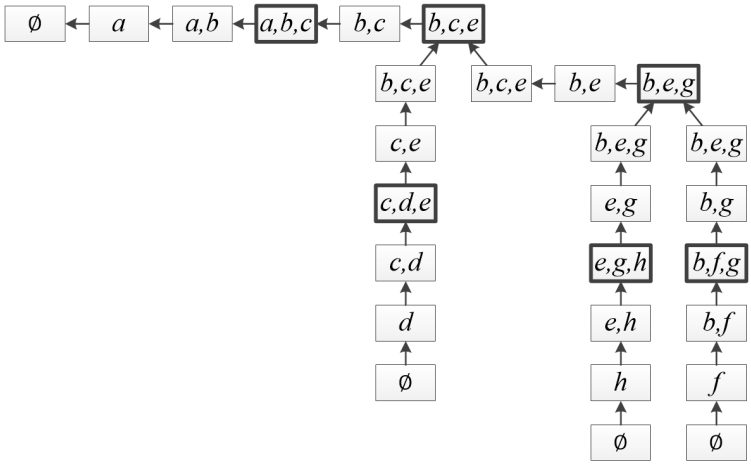
- **P1** $\mathcal{T}_G$ está enraizado en algún nodo $r \in T$.
- **P2** $X_r = \{\emptyset\}$ y para todo nodo hoja $l \in T$, $X_l = \{\emptyset\}$ (*nodo hoja*)
- **P3** Cada nodo $t \in T$ tienen a lo más dos descendientes.
- **P4** Si t tiene dos hijos: t_1 y t_2, entonces $X_t = X_{t_1} = X_{t_2}$ (*nodo unión*)
- **P5** Si t tiene sólo un hijo: t_1, entonces una de las siguientes condiciones es verdad:
  - $X_t \supset X_{t_1}$ y $|X_t| \leftarrow \left|X_{t_1}\right| + 1$ (*nodo introductor*)
  - $X_t \subset X_{t_1}$ y $|X_t| \leftarrow \left|X_{t_1}\right| - 1$ (*nodo eliminador*)

La desmembración de árbol agradable se expresa de la siguiente forma (T, X) para diferenciarlo de cualquier otra desmembración de árbol $\mathcal{T}_G$, vea la Figura 17. Observe que las Propiedades **P1** a **P5** también indican los cuatro tipos de nodos presentes en la desmembración de árbol agradable. El primero es el nodo *hoja* cuya bolsa X_t contiene el conjunto vacío, propiedad **P2**. El nodo *unión* es aquél que tiene exactamente dos descendientes, propiedades **P3** y **P4**. El nodo *introductor* es aquel que agrega exactamente un vértice (a su bolsa) que no se encuentra en la bolsa de su nodo descendiente, propiedad **P5**. De forma semejante, la bolsa de un nodo *eliminador* olvida uno de los vértices presentes en la bolsa de su nodo descendiente, propiedad **P5**.

Observe la rama más derecha del grafo de la Figura 17(b). El nodo de contenido $\{\emptyset\}$ es un nodo hoja. El nodo $\{f\}$ es un nodo introductor puesto que introduce o agrega al vértice f que no se encuentra presente en el nodo hoja. Asimismo, el nodo



(a) Desmembración de árbol $\mathcal{T}_G$



(b) Desmembración de árbol agradable (T, X) del árbol $\mathcal{T}_G$

Figura 17. Desmembración de árbol y desmembración de árbol agradable.

$\{b, f\}$ también es introductor puesto que en este nodo se agrega el vértice b que no está en el nodo f. De forma semejante, el nodo $\{b, f, g\}$ también es introductor. El nodo $\{b, g\}$ es un nodo eliminador, puesto que contiene los mismos vértices del nodo $\{b, f, g\}$ excepto por el vértice f que se removió. El nodo $\{b, e, g\}$, que se encuentra resaltado, es un nodo unión. Recuerde que el nodo unión tiene exactamente dos hijos y el contenido de las bolsas de los hijos y del padre es el mismo.

Ahora se presenta un procedimiento para convertir una desmembración de árbol $\mathcal{T}_G$ en una desmembración de árbol agradable (T, X).

Para satisfacer las propiedades **P1** y **P2** realice una copia de la desmembración de árbol $(T', X') \leftarrow (T, X)$. Por cada hoja $l \in T'$, agregue un nuevo nodo l' $(X_{t'} \leftarrow \{\varnothing\})$ y una arista conectando l con l'. Seleccione de forma arbitraria un nodo hoja como raíz, sea

t_r dicho nodo, y enraice el resto del árbol en t_r, vea la Figura 18.

Sean $t \in T$ y $\{t_1, t_2, \ldots, t_i, t_q\}$ para $q > 2$, $1 \leq i \leq q$ sus descendientes. La propiedad **P3** se satisface generando un nuevo nodo t' cuya bolsa contenga los elementos de t, i.e., $X_{t'} \leftarrow X_t$. Agregue una arista entre t y t' de tal forma que t sea el padre de t'. Adicionalmente, t' es el padre de cada t_i para $i > 2$. Note que ahora t tiene dos hijos: t_1 y t'. Repita este procedimiento si t' tiene más de dos descendientes, vea la Figura 19.

La propiedad **P4** dice que si un nodo tiene exactamente dos descendientes, entonces el contenido de las bolsas de los descendientes debe ser igual a la del nodo padre. Sea $t \in T$ un nodo con dos descendientes t_1 y t_2. Genere dos nuevos nodos t'_1 y t'_2. Asigne a las bolsas de dichos nodos el contenido de X_t; i.e., $X_{t'_1} \leftarrow X_{t'_2} \leftarrow X_t$. Agregue y elimine las aristas que sean necesarias para satisfacer que t sea el padre de t'_1 y t'_2; que t'_1 sea el padre de t_1 y de forma semejante t'_2 sea el padre de t_2, vea la Figura 20.

Ahora se discute el procedimiento para satisfacer la propiedad **P5**, vea la Figura 21. Sea $t \in T$ un nodo y t_1 su descendiente

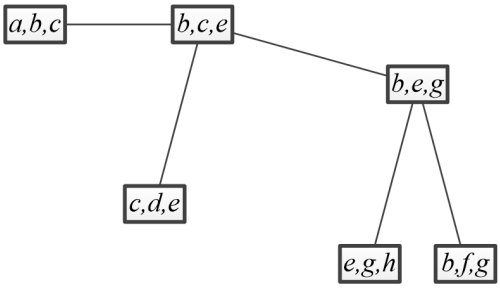
- **Caso 1:** Suponga que $X_t \subset X_{t_1}$, tal que $|X_t| + c = \left|X_{t_1}\right|$ para una constante $c > 1$. Genere un nuevo nodo t'. Asigne a t como padre de t' y a t' como padre de t_1. Sea u algún vértice que está en X_{t_1} pero que no se encuentra en X_t, asigne a $X_{t'}$ el contenido de la bolsa $X_t \cup \{u\}$. Repita el procedimiento en t o sus descendientes según sea necesario.

  Algo semejante sucede cuando $X_t \supset X_{t_1}$ y $|X_t| = \left|X_{t_1}\right| + c$, pero en esta ocasión $u \in X_t \setminus X_{t_1}$.

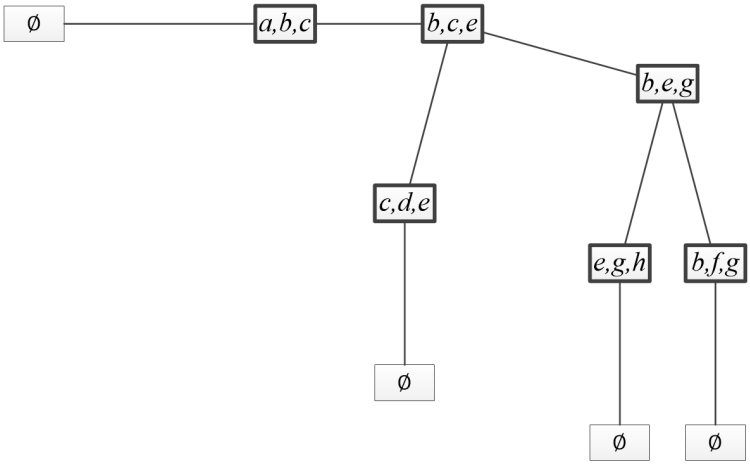
- **Caso 2:** $|X_t| = \left|X_{t_1}\right|$. Genere un nodo t', asigne a t como padre de t' y a t' como padre de t_1. El contenido de la bolsa de t' es $X_t \cap X_{t_1}$.

- **Caso 3:** El nodo t_1 es una hoja. Genere un nuevo nodo t' que sea descendiente t, asigne a t' como padre de t_1. El contenido de la bolsa de t' es $X_t \setminus v$. Repita este procedimiento hasta que la bolsa del padre de t_1 contenga un sólo vértice. Suponga que la desmembración de árbol corresponde a un grafo 1-outerplanar, y que p es el nodo padre de t, entonces:

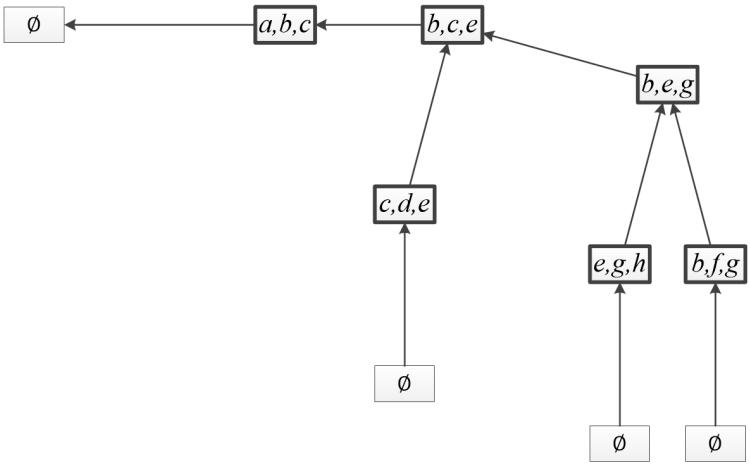
  - Si $X_t = 3$, agregue dos nuevos nodos t' y t'' de tal forma que t sea el padre de t', t' el de t'' y t'' el de t_1. El contenido de las bolsas es el siguiente:



(a) Desmembración de árbol $\mathcal{T}_G$

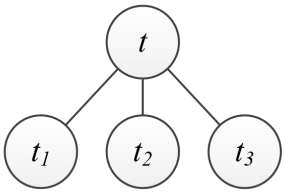


(b) Creación de nuevos nodos hojas y asignación de bolsas vacías

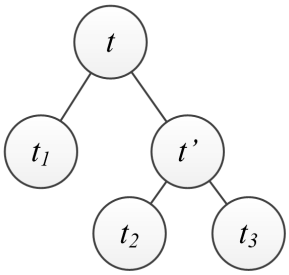


(c) Selección de una raíz y enraizamiento del árbol

Figura 18. Satisfaciendo las propiedades **P1** y **P2** de la desmembración de árbol agradable.

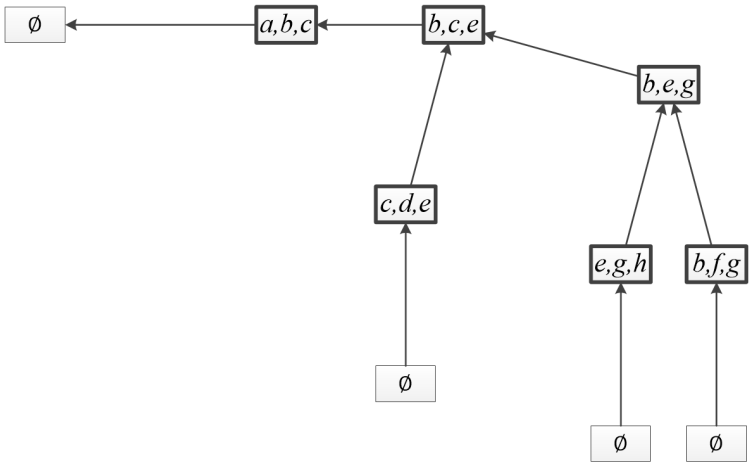


(a) Desmembración de árbol con un nodo t con tres descendientes

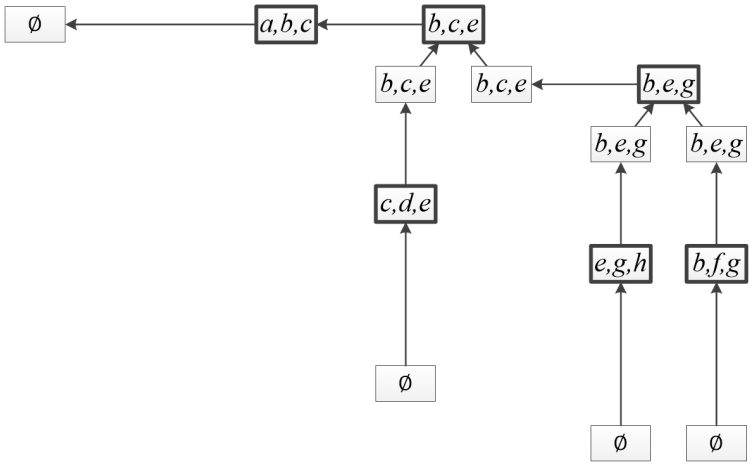


(b) Desmembración de árbol con el nodo adicional t'

Figura 19. Propiedad **P3** de la desmembración de árbol agradable.



(a) Una desmembración de árbol $\mathcal{T}_G$, los nodos $\{b, c, e\}$ y $\{b, e, g\}$ tienen dos descendientes



(b) Grafo $\mathcal{T}_G$, a los nodos $\{b, c, e\}$ y $\{b, e, g\}$ se les agregaron dos descendientes

Figura 20. Propiedad **P4** de la desmembración de árbol agradable.

- $X_{t_1} = X_{t_2} \leftarrow X_t \setminus X_p$
- $X_{t_2} \leftarrow X_{t_2} \cup v$, $v \in X_p$.

- Si $|X_t| = 2$, sea t' un nuevo nodo descendiente de t, asigne $X_{t_1} \leftarrow v$, donde $v \in X_t$.

El siguiente lema, concerniente a la desmembración de árbol agradable, se extrajo de (Cygan *et al.*, 2015).

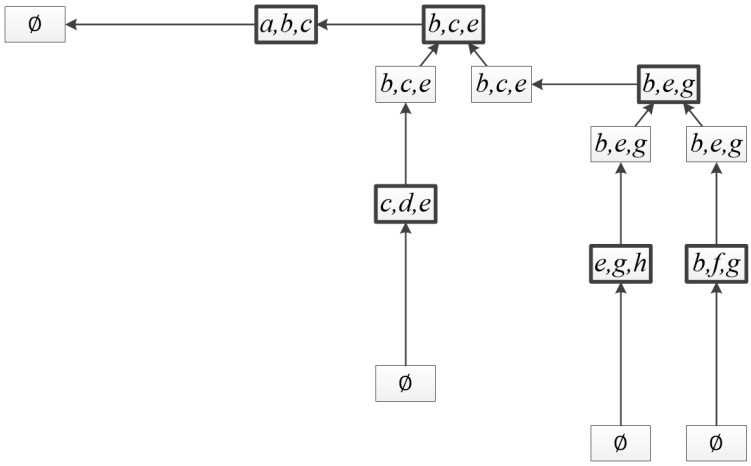
Lema 4.1 *Si un grafo G admite una desmembración de árbol cuya anchura es a lo más k, entonces éste también admite una desmembración de árbol agradable de anchura a lo más k. Además, dada una desmembración de árbol $\mathcal{T}_G$ de G de anchura a lo más k, uno puede calcular una desmembración de árbol agradable de G en tiempo $O\left(k^2 max(|V_T|, |V_G|)\right)$ de anchura a lo más k que tiene a lo más $O(k|V_G|)$ nodos.*

Es posible completar la última parte del Lema 4.1 mediante el Lema 4.2, extraído de la tesis de Kloks (1994).

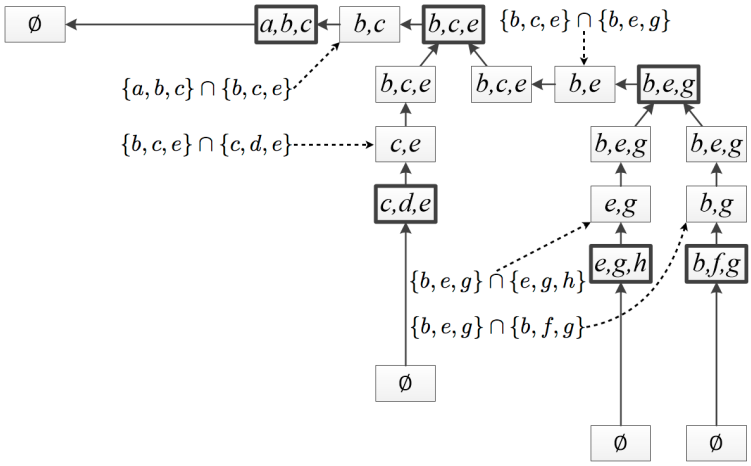
Lema 4.2 *Todo grafo G con anchura de árbol k tiene una desmembración de árbol agradable de anchura k. Además, si n es el número de vértices de G, entonces existe una desmembración de árbol agradable con a lo más 4n vértices.*

4.5. Conclusión

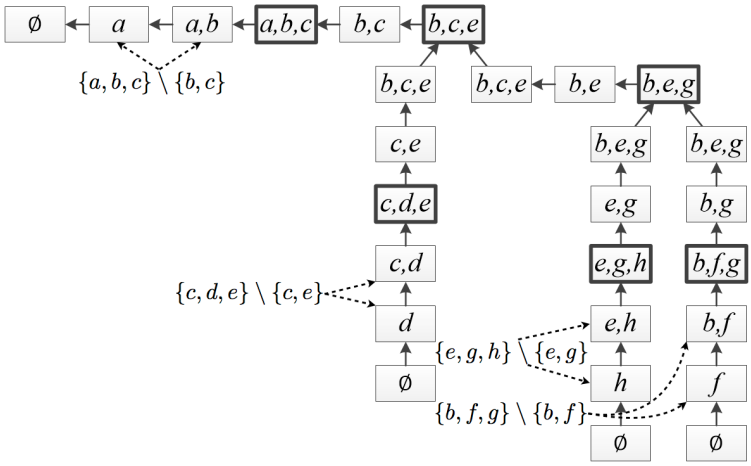
Los conceptos de grafo desmembrable y propiedad regular fueron presentados por Bern *et al.* (1987). Independientemente Arnborg y Proskurowski (1989) introdujeron el concepto de *k*-árbol parciales. La literatura relacionada a estos conceptos permite caracterizar a las familias de grafos Γ y problemas de optimización de subgrafo que pueden resolverse eficientemente en Γ. No obstante, diversos autores señalan la falta de garantías para los problemas. Para subsanar esta parte, el trabajo de Mahajan y Peters (1994) extiende el trabajo de Bern *et al.* (1987) definiendo las propiedades uniformemente regulares. De forma independiente, Courcelle (1990) define a la lógica monádica de segundo orden como un lenguaje para descripciones lógicas que permite



(a) Una desmembración de árbol $\mathcal{T}_G$



(b) Creación de nodos intermedios t' cuando $|X_t| = |X_{t_1}|$



(c) Creación de nodos intermedios cuando t_1 es un nodo hoja o raíz.

Figura 21. Propiedad **P5** de la desmembración de árbol agradable.

construir algoritmos de programación dinámica en desmembraciones de árbol de un grafo.

La intersección de todos los conceptos anteriores es el par anchura de árbol y desmembración de árbol. Por lo que, con base en lo presentado en este capítulo, se puede llegar al siguiente lema:

Observación 4.4 *Sea G un grafo que pertenece a una familia de grafos desmembrables* Γ*, donde se conoce una desmembración de árbol de G de anchura de árbol a lo más k (para k constante), y* P_G *un problema de optimización de sugbrafo de G (propiedad regular) tal que* P_G *es expresable en LMSO. Entonces, debe existir un algoritmo que calcule un óptimo* P_G *para G en* $O(n)$ *u.t.*

Ahora se presenta cómo el problema de encontrar el conjunto 2-packing máximo en el grafo 1-outerplanar cumple con las características de la Observación 4.4. Sea G un grafo 1-outerplanar y P_G el problema del conjunto 2-packing máximo, entonces:

- G pertenece a una familia de grafos desmembrables (Bern *et al.*, 1987; Hsieh, 2005)
- Tanto la desmembración como la anchura de árbol de G son conocidas (Downey y Fellows, 2013; Bodlaender, 1988; Katsikarelis, 2013; Cygan *et al.*, 2015)
- P_G es un problema de optimización de subgrafo (propiedad regular) Mahajan y Peters (1994); Hsieh (2005)
- P_G es expresable en LMSO, ecuaciones 16 - 19.

Entonces, debe de existir un algoritmo que calcule un conjunto 2-packing máximo en tiempo polinomial en el grafo 1-outerplanar. El Capítulo 5 presenta un algoritmo tabular para realizar esta tarea.

Capítulo 5. Encontrando un conjunto 2-packing máximo en un grafo 1-outerplanar

Este capítulo presenta un algoritmo de programación dinámica que calcula un conjunto 2-packing máximo $\hat{S}$ en una desmembración de árbol agradable del grafo 1-outerplanar G. Donde $\hat{S}$ también corresponde a un conjunto máximo en G. Encontrar un conjunto 2-packing máximo en un grafo arbitrario es un problema NP-difícil y en algunos casos FPT cuando el parámetro a fijar es la anchura de árbol. No obstante, es posible desarrollar algoritmos lineales cuando se trabaja sobre grafos cuya anchura de árbol es acotada.

5.1. Descripción del algoritmo propuesto

Se utilizan tablas para incrementar el tamaño del conjunto 2-packing en la desmembración de árbol, iniciando desde las hojas hasta llegar al nodo raíz. A partir de este momento se considera que los nodos de cualquier desmembración de árbol a la que se haga alusión conocen su número postorden (o índice) i.

Para cualquier nodo t_i, sea V_{t_i} la unión de todas las bolsas presentes en el subárbol de T enraizado en t_i, incluyendo X_{t_i}. Dado un subconjunto $S \subseteq X_{t_i}$ lo que se busca es una extensión $\hat{S} \supseteq S$ tal que $\hat{S} \subseteq V_{t_i}$, $\hat{S} \cap X_{t_i} = S$ y que $\hat{S}$ forme un conjunto 2-packing.

Para cada nodo t_i y cada $S \subseteq X_{t_i}$, sea la tabla $c[t_i, S]$ la que aloja la cardinalidad máxima del conjunto $\hat{S}$, donde:

- La tabla $c[t_i, S]$ tiene $2^{|X_{t_i}|}$ columnas, una por cada elemento del conjunto potencia de X_t.
- Tres filas:
  - Cada celda del primer renglón contiene algún conjunto S del conjunto potencia de X_{t_i}.
  - Las celdas de la segunda fila, guardan la cardinalidad del conjunto 2-packing máximo $\hat{S}$ cuando S forma parte de la solución $\hat{S}$. Si $S = \{\emptyset\}$, entonces la

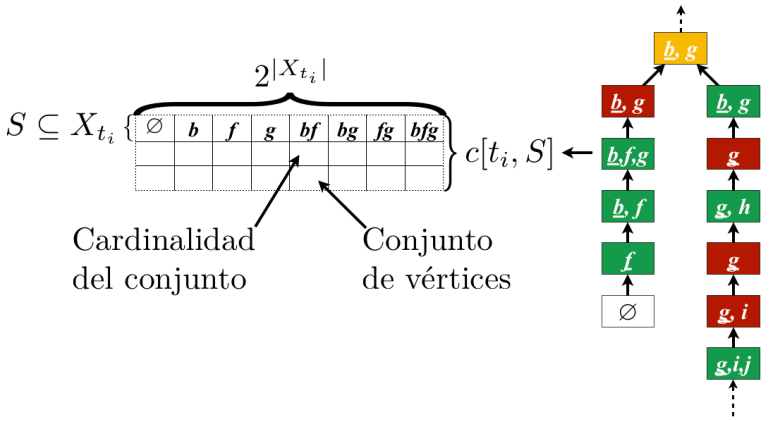


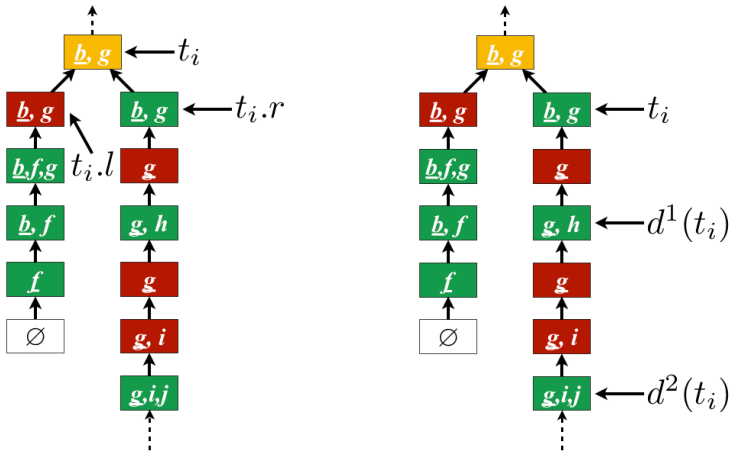
Figura 22. La tabla $c[t_i, S]$ y un subgrafo de alguna desmembración de árbol.

celda guarda un conjunto 2-packing que considera que $\hat{S}$ no contiene ningún elemento de X_{t_i}.

- Las celdas de la tercera fila contienen los vértices que conforman a $\hat{S}$ con respecto de S.

En el ejemplo de la Figura 22 se muestra un subgrafo de alguna desmembración agradable. Los nodos sin color (o color blanco) son nodos hojas. En color verde están los nodos introductores, en rojo los nodos eliminadores y en amarillo un nodo unión. A partir de este momento se usa esta convención de colores. Adicionalmente, la Figura 22 presenta el nodo $t_i, X_{t_i} = \{b, f, g\}$ con su tabla $c[t_i, S]$. El número de columnas para la tabla está dado por $2^{|X_{t_i}|}$, i.e., la cardinalidad del conjunto potencia de los elementos su bolsa. Recuerde que la anchura de árbol para el grafo 1-outerplanar es a lo más dos. Es decir, el tamaño de la bolsa más grande en la desmembración de árbol del grafo 1-outerplanar contiene a lo más 3 elementos. Por lo tanto, el número de columnas de la tabla $c[t_i, S]$ es a lo más ocho para cualquier nodo t_i de la desmembración de árbol del grafo 1-outerplanar. Cuando el conjunto de vértices en S no forma un conjunto 2-packing, entonces se asigna a las entradas de $c[t, S]$ el valor de $-\infty$.

Cuando se haga mención a un nodo unión t_i, por $t_i.l$ se refiere al nodo hijo de t_i con menor índice postorden, y por $t_i.r$ al nodo hijo de mayor índice. Una consideración adicional es la función $d^q(t_i)$, ésta regresa el par (t_j, S_j) donde t_j es el q-ésimo descendiente de t_i ($j < i$) que no es nodo eliminandor y $S_j \subseteq X_{t_j}$. Por simplicidad $d^q(t_i)$ puede referir solamente al nodo t_j cuando sea necesario. Si t_i es un nodo unión, la función $d^q(t_i)$ debe aplicarse individualmente a $t_i.l$ y $t_i.r$. Vea las figuras 23(a) y 23(b).



(a) Identificación de los nodos izquierdo y derecho de un nodo unión

(b) Ejemplo de la función $d^q(t_i)$

Figura 23. Identificación de nodos en la desmembración de árbol.

Ahora se presenta cómo calcular los valores de la tabla $c[t, S]$. Puesto que la desmembración de árbol que se utiliza es agradable, entonces existen pocas formas en la cual un nodo t y los vértices de su bolsa X_t pueden comunicarse con sus nodos descendientes. Los valores de la tabla $c[t, S]$ se calculan de forma recursiva a partir de las soluciones de sus descendientes. El caso base es cuando t_i es un nodo hoja, el resto de los casos cuando t_i es un nodo intermedio. Se presentan cuatro fórmulas que calculan los valores de $c[t, S]$, una por cada tipo de nodo de la desmembración agradable. Posteriormente se muestra un ejemplo dónde se aplican las fórmulas y, finalmente, se analiza formalmente el procedimiento presentado.

Nodo hoja: Si t_i es un nodo hoja $X_{t_i} = \{\emptyset\}$, entonces el único valor que su tabla puede tener es cero. Vea la ecuación 21.

$$c[t_i, \{\emptyset\}] \leftarrow 0 \tag{21}$$

Nodo introductor: Suponga que t_i es un nodo introductor y t_{i-1} es su hijo tal que $X_{t_i} \leftarrow X_{t_{i-1}} \cup \{v\}$ para algún vértice $v \notin X_{t_{i-1}}$. Se consideran dos casos: cuando el vértice $v \notin S$ y cuando $v \in S$. Vea la ecuación 22.

$$c[t_i, S] \leftarrow \begin{cases} c[t_{i-1}, S] & \text{si } v \notin S \\ \max \begin{cases} \max(1, c[t_{i-1}, \{\varnothing \cup v\}]) \\ \max \left(f\left(d^1(t_i)\right), \begin{cases} f\left(d^2(t_i)\right) & \text{si } d^1(t_i) \text{ es introductor} \\ f\left(d^1(t_i.l)\right), f\left(d^1(t_i.r)\right) & \text{si } d^1(t_i) \text{ es unión} \end{cases} \right) \end{cases} & \text{si } v \in S \end{cases} \tag{22}$$

Nodo eliminador: Sea t_i un nodo eliminador y t_{i-1} su descendiente, entonces $X_{t_i} = X_{t_{i-1}} \setminus \{w\}$, para algún vértice $w \in X_{t_{i-1}}$. Se consideran dos casos: cuando $w \notin S$ y cuando el vértice w se añade a S. Vea la ecuación 23.

$$\underset{w \in t_{i-1}, w \notin t_i}{c[t_i, S]} \leftarrow \max \begin{cases} c[t_{i-1}, S] \\ c[t_{i-1}, S \cup \{w\}] \end{cases} \tag{23}$$

Nodo unión: Cuando t_i es un nodo unión se consideran dos casos: cuando $S \neq \{\varnothing\}$ y cuando $S = \{\varnothing\}$. Vea la ecuación 24.

$$c[t_i, S] \leftarrow \begin{cases} c[t.l, S] + c[t.r, S] - S & \text{si } S \neq \{\varnothing\} \\ \max \begin{cases} c[t.l, \{\varnothing\}] + f\left(d^1(t.r)\right) & \text{distancia 1-2} \\ f\left(d^1(t.l)\right) + c[t.r, \{\varnothing\}] & \text{distancia 2-1} \\ f\left(d^1(t.l)\right) + f\left(d^1(t.r)\right) & \text{distancia 2-2} \end{cases} & \text{si } S = \{\varnothing\} \end{cases} \tag{24}$$

Sean $X, Y \subseteq V_G$ tal que $X \cup Y = V_G$. El subconjunto $X \cap Y$ *separa* a X de Y si todos los caminos de cualquier vértice en X hacia cualquier vértice en Y contienen un vértice de S (Flum y Grohe, 2006). El *separador* de esta separación es $S = X \cap Y$ (Cygan *et al.*, 2015). Note que al remover S, el grafo se descompone en dos o más componentes conectados. Ésto es semejante a la definición de vértice de corte de la sección 2.1. La importancia de los separadores radica en que permiten que algunos algoritmos (para desmembración de árbol) se ejecuten de forma más eficiente e incluso paralela Downey y Fellows (2013). En el presente documento se emplea una versión ‘debil’ de separadores como medio de comunicación entre las bolsas de una desmembración de

árbol, a ésta se le llama *vértices de articulación*. En la Figura 23, los vértices subrayados en cada nodo son vértices de articulación y La subsección 5.1.1 muestra cómo calcularlos. Los vértices de articulación en conjunto con la función max{} permiten discernir entre conjuntos 2-packing de igual tamaño.

Sea t_i un nodo de (T, X) con vértice de articulación $a \in X_{t_i}$. Sean $\hat{S}_1$ y $\hat{S}_2$ un par de soluciones calculadas por alguna de las ecuaciones 22 - 24 para la entrada $s \in S$ de la tabla $c[t_i, s]$. En caso de empate entre $\hat{S}_1$ y $\hat{S}_2$ $\left(\left|\hat{S}_1\right| = \left|\hat{S}_2\right|\right)$, la función $\max\{\hat{S}_1, \hat{S}_2\}$ regresa la solución $\hat{S}_1$ si la distancia entre los vértices de $\hat{S}_1$ al vértice de articulación a es mayor que las distancias entre los vértices de $\hat{S}_2$ y a. En caso de que el empate persista, la función max{} regresa la primera solución calculada.

El problema computacional a resolver es: Dado un grafo 1-outerplanar conexo $G = (V_G, E_G)$, encontrar un subconjunto de vértices $\hat{S} \subseteq V_G$, tal que:

- $\forall u, v \in \hat{S}$, la $distancia(u, v) \geq 3$, i.e., $\hat{S}$ es un 2-packing.
- $\hat{S}$ es un conjunto máximo.

Para resolver este problema computacional el autor de este documento se apoya de la desmembración de árbol del grafo 1-outerplanar. El procedimiento general a seguir es el siguiente:

1. Obtener una desmembración de árbol $\mathcal{T}_G$ de G mediante el algoritmo de Katsikarelis (2013).
2. Generar una desmembración de árbol agradable (T, X) de $\mathcal{T}_G$ mediante el procedimiento descrito por Cygan *et al.* (2015).
3. Asignar a cada nodo $t \in T$ su número postorden.
4. Aplicar un algoritmo de programación dinámica (siguiendo el recorrido postorden en t_i) que use las ecuaciones 21 - 24 para encontrar un conjunto 2-packing máximo en (T, X).
    - El conjunto de mayor tamaño se obtiene del nodo raíz de (T, X).
5. Reconstruir en G la respuesta encontrada en (T, X).

5.1.1. Pseudocódigos

Los pseudocódigos 12 - 15 presentan el algoritmo. Las rutinas se denotan en tipo de letra VERSALITAS. Los vértices, variables e índices se presentan en letras minúsculas (por ejemplo, v) y los conjuntos en mayúsculas (R), excepto cuando se toman elementos de un conjunto ($s \in S$). Los vértices de articulación de un nodo se denotan con "$.a$", por ejemplo $t_i.a$. Si el nodo t_i es un nodo unión entonces $t_i.l$ y $t_i.r$ denotan sus nodos hijos izquierdo y derecho, respectivamente. Finalmente, se supone que la numeración postorden del árbol (T, X) inicia en $i = 1$ hasta $i = N = |T|$.

La entrada del Pseudocódigo 12 es un grafo 1-outerplanar conexo, la salida es un conjunto 2-packing en G. La línea 2 crea una desmembración de árbol del grafo G mediante el algoritmo de Katsikarelis (2013). La línea 3 selecciona un nodo cualquiera como raíz para $\mathcal{T}_G$. La línea 4 calcula una desmembración agradable de árbol (T, X) mediante el procedimiento de Cygan *et al.* (2015). Note en la línea 5 que el nodo raíz cambia al generar (T, X). En la línea 6 se le asigna a cada nodo t su número postorden. Las líneas 7 y 8 le indican a cada nodo t_i cuáles vértices $a \subseteq X_{t_i}$ son de articulación. La rutina de la línea 9 calcula la tabla $c[t_i, S]$ para cada nodo mediante las ecuaciones 21 - 24. La rutina RECONSTRUYEM2PO de la línea 10 (no se muestra) distribuye la solución encontrada en $c[t_N, \{\emptyset\}]$ sobre los vértices de G. Finalmente, la línea 39 regresa el grafo G.

La rutina del Pseudocódigo 13 le indica a cada nodo t_i cuáles son sus vértices de articulación. La entrada es un nodo t_i que conoce sus vértices de articulación.

El Pseudocódigo 14, CALCULAM2PO, calcula la tabla $c[t_i, S]$ para cada nodo de la desmembración de árbol agradable (líneas 2 - 38). Las líneas 4 - 6 aplican la ecuación 21 a los nodos hoja. Las líneas 7 - 9 ejecutan la ecuación 22 (correspondiente al Pseudocódigo 15) cuando t_i es introductor. Las líneas 10 - 15 aplican la ecuación 23 si t_i es eliminador. Las líneas 16 - 37 ejecutan la ecuación 24 cuando t_i es un nodo unión. Finalmente, la línea 39 regresa (T, X), note que el conjunto 2-packing máximo está alojado en $c[t_N, \{\emptyset\}]$.

La rutina del Pseudocódigo 15 es parte de las instrucciones del Pseudocódigo 14. Las líneas 1 - 31 calculan la ecuación 22 para cuando t_i es un nodo introductor.

Pseudocódigo 12: MAXIMUM-2-PACK-OUTERPLANAR($G = (V_G, E_G)$).

Entrada: Un grafo 1-outerplanar $G = (V_G, E_G)$.
Salida : Un conjunto de vértices resaltados que representan un 2-packing máximo en G.

```
1  begin
2      T_G ← TREE-DECOMPOSITION(G)
3      Seleccione arbitrariamente un nodo de T_G como raíz, sea t_r dicho nodo
4      (T, X) ← NICE-TREE-DECOMPOSITION(T_G, t_r)
5      Sea t_N el nuevo nodo raíz
6      (T, X) ← POSTORDEN-TRAVERSAL((T, X), t_N)
7      t_{N-1}.a ← X_{t_{N-1}}
8      VÉRTICES-ARTICULACIÓN(T_{N-1}, X_{T_{N-1}})
9      (T, X) ← CALCULAM2PO((T, X))
10     G ← RECONSTRUYEM2PO(G, t_N)
11     return G
12 end
```

Pseudocódigo 13: VÉRTICES-ARTICULACIÓN(t_i).

Entrada: Un nodo t_i perteneciente a una desmembración de árbol agradable junto con un conjunto de vértices de articulación $t_i.a$.
Salida : Todos los nodos de (T, X) tienen sus vértices de articulación definidos.

```
1  begin
2      if t_i es nodo unión then
3          t_i.l.a ← t_i.a
4          t_i.r.a ← t_i.a
5          VÉRTICES-ARTICULACIÓN(t_i.l)
6          VÉRTICES-ARTICULACIÓN(t_i.r)
7          return
8      end
9      t_j ← t_{i-1}
10     if t_j es nodo hoja then
11         return
12     end
13     if t_i.a ∩ X_{t_j} ≠ ∅ then
14         t_j.a ← t_i.a ∩ X_{t_j}
15     else
16         t_j.a ← X_{t_i} \ t_i.a
17     end
18     VÉRTICES-ARTICULACIÓN(t_j)
19     return
20 end
```

Pseudocódigo 14: CALCULAM2PO((T, X)).

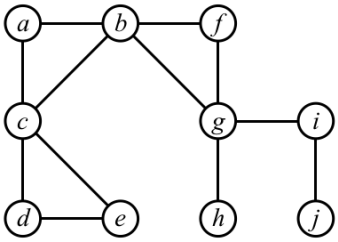
Entrada: Una desmembración de árbol agradable (T, X).
Salida : El grafo (T, X) con un conjunto de vértices almacenados en $c[t_N, \{\emptyset\}] \in t_N$ que representan un 2-packing máximo.

```
1  begin
2      for 1 ≤ i ≤ N do
3          S ← CONJUNTO-POTENCIA(X_{t_i})
4          if t_i es hoja then
5              c[t_i, {∅}] ← 0
6          end
7          if t_i es introductor then
8              t_i ← INTRODUCTOR(t_i)
9          end
10         if t_i es eliminador then
11             w ← X_{t_{i-1}} \ X_{t_i}
12             foreach s ∈ S do
13                 c[t_i, s] ← max(c[t_{i-1}, s], c[t_{i-1}, s ∪ w])
14             end
15         end
16         if t_i es join then
17             t1.l ← d^1(t_i.l)
18             S1.l ← CONJUNTO-POTENCIA(X_{t1.l})
19             t1.r ← d^1(t_i.r)
20             S1.r ← CONJUNTO-POTENCIA(X_{t1.r})
21             foreach s ∈ S, s ≠ {∅} do
22                 c[t_i, s] ← c[t_i.l, s] + c[t_i.r, s] − s
23             end
24             Ŝ ← {0, {∅}}
25             foreach s1 ∈ S1.r do
26                 Ŝ ← max(Ŝ, (c[t_i.l, {∅}] + c[t1.r, s1]))
27             end
28             foreach s1 ∈ S1.l do
29                 Ŝ ← max(Ŝ, (c[t1.l, s1] + c[t_i.r, {∅}]))
30             end
31             foreach l ∈ S1.l do
32                 foreach r ∈ S1.r do
33                     Ŝ ← max(Ŝ, (c[t1.l, l] + c[t1.r, r]))
34                 end
35             end
36             c[t_i, {∅}] ← Ŝ
37         end
38     end
39     return (T, X)
40 end
```

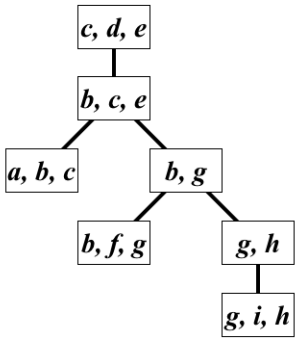
Pseudocódigo 15: INTRODUCTOR(t_i).

Entrada: Un nodo introductor $t_i \in (T, X)$
Salida : El nodo t_i con todos los valores $\hat{S}$ de $c[t_i, S]$ calculados.

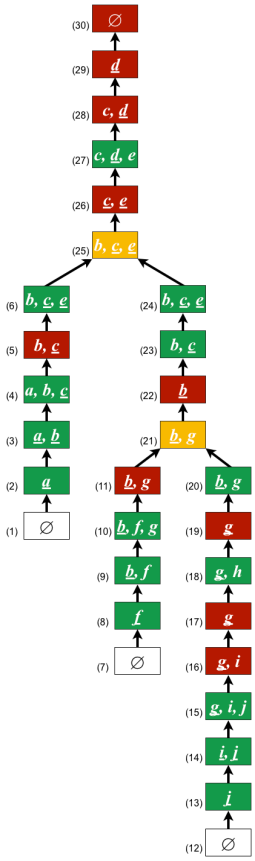
```
1  begin
2      v ← X_{t_i} \ X_{t_{i-1}}
3      t1 ← d^1(t_i)
4      t2 ← d^2(t_i)
5      S1 ← CONJUNTO-POTENCIA(X_{t1})
6      S2 ← CONJUNTO-POTENCIA(X_{t2})
7      S_{i-1} ← CONJUNTO-POTENCIA(X_{t_{i-1}})
8      foreach s ∈ S do
9          if s ∈ S_{i-1} then
10             c[t_i, s] ← c[t_{i-1}, s]
11         else
12             Ŝ ← max(1, c[t_{i-1}, {∅ ∪ v}])
13             foreach s1 ∈ S1 do
14                 Ŝ ← max(Ŝ, ({v} ∪ c[t1, s1]))
15             end
16             if t1 es introductor then
17                 foreach s2 ∈ S2 do
18                     Ŝ ← max(Ŝ, ({v} ∪ c[t2, s2]))
19                 end
20             end
21             if t1 es unión then
22                 foreach s1 ∈ S1 do
23                     Ŝ ← max(Ŝ, ({v} ∪ c[t1.l, s1]))
24                     Ŝ ← max(Ŝ, ({v} ∪ c[t1.r, s1]))
25                 end
26             end
27             c[t_i, s] ← Ŝ
28         end
29     end
30     return t_i
31 end
```



(a) Grafo $G = (V_G, E_G)$ de entrada



(b) Una desmembración de árbol $\mathcal{T}_G$ del grafo de entrada



(c) Desmembración agradable de árbol (T, X) del grafo de entrada

Figura 24. Grafo para ejemplificar el Pseudocódigo 12

5.1.2. Ejemplo de ejecución

Este apartado presenta un ejemplo de la ejecución del algoritmo propuesto. La Figura 24(a) muestra el grafo 1-outerplanar de entrada $G = (V_G, E_G)$. La Figura 24(b) presenta una posible desmembración de árbol $\mathcal{T}_G$ de G. En la Figura 24(c) está su desmembración de árbol agradable (T, X); el número entre paréntesis que se encuentra a la izquierda de cada nodo corresponde a su numeración postorden. La Tabla 5 presenta las tablas $c[t_i, S]$ correspondientes al grafo de la Figura 24(c). Note que se conserva el código de colores para cada nodo y la numeración postorden. Adicionalmente, en color verde se resalta la celda que contiene el vértice agregado en cada nodo introductor. La tablas $c[t_i, S]$ de los nodos eliminadores tienen una celda extra que señala el vértice olvidado con respecto a t_{i-1}.

Observe las tablas $c[t_i, S]$ para $12 \leq i \leq 20$ en la Tabla 6. Para calcular el conte-

Tabla 5. Tablas $c[t_i, S]$ del grafo (T, X) de la Figura 24(c)

(30)

∅	***d***
3	
$a+h+j$	

(29)

∅	***d***	***c***
3	3	
$a+h+j$	$d+h+j$	

(28)

∅	***c***	***d***	***cd***	***e***
3	3	3	$-\infty$	
$a+h+j$	$c+h+j$	$d+h+j$		

(27)

∅	***c***	***d***	***e***	***cd***	***ce***	***de***	***ced***
3	3	3	3	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$a+h+j$	$c+h+j$	$d+h+j$	$e+h+j$				

(26)

∅	***c***	***e***	***ce***	***b***
3	3	3	$-\infty$	
$a+h+j$	$c+h+j$	$e+h+j$		

(25)

∅	***b***	***c***	***e***	***bc***	***be***	***ce***	***bce***
3	2	3	3	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$a+h+j$	$b+j$	$c+h+j$	$e+h+j$				

(6)

∅	***b***	***c***	***e***	***bc***	***be***	***ce***	***bce***
1	1	1	1	$-\infty$	$-\infty$	$-\infty$	$-\infty$
a	b	c	e				

(5)

∅	***b***	***c***	***bc***	***a***
1	1	1	$-\infty$	
a	b	c		

(4)

∅	***a***	***b***	***c***	***ab***	***ac***	***bc***	***abc***
0	1	1	1	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	a	b	c				

(3)

∅	***a***	***b***	***ab***
0	1	1	
	a	b	

(2)

∅	***a***
0	1
	a

(1)

∅
0

(24)

∅	***b***	***c***	***e***	***bc***	***be***	***ce***	***bce***
2	2	3	3	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$h+j$	$b+j$	$c+h+j$	$e+h+j$				

(23)

∅	***b***	***c***	***bc***
2	2	3	$-\infty$
$h+j$	$b+j$	$c+h+j$	

(22)

∅	***b***	***g***
2	2	
$h+j$	$b+j$	

(21)

∅	***b***	***g***	***bg***
2	2	1	$-\infty$
$h+j$	$b+j$	g	

(11)

∅	***b***	***g***	***bg***	***f***
1	1	1	$-\infty$	
f	b	g		

(20)

∅	***b***	***g***	***bg***
2	2	1	$-\infty$
$h+j$	$b+j$	g	

(10)

∅	***b***	***f***	***g***	***bf***	***bg***	***fg***	***bfg***
0	1	1	1	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	b	f	g				

(19)

∅	***g***	***h***
2	1	
$h+j$	g	

(9)

∅	***b***	***f***	***bf***
0	1	1	$-\infty$
	b	f	

(18)

∅	***g***	***h***	***gh***
1	1	2	$-\infty$
j	g	$h+j$	

(8)

∅	***f***
0	1
	f

(17)

∅	***g***	***i***
1	1	
j	g	

(7)

∅
0

(16)

∅	***g***	***i***	***gi***	***j***
1	1	1	$-\infty$	
j	g	i		

(15)

∅	***g***	***i***	***j***	***gi***	***gj***	***ij***	***gij***
0	1	1	1	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	g	i	j				

(14)

∅	***i***	***j***	***ij***
0	1	1	$-\infty$
	i	j	

(13)

∅	***j***
0	1

(12)

∅
0

nido de $c[t_{12}, S]$ se utiliza la ecuación 21. Como t_{12} es un nodo hoja, el contenido de $c[t_{12}, \{\emptyset\}]$ es cero. La tabla $c[t_{19}, S]$ corresponde a un nodo eliminador, para llenar sus celdas se emplea la ecuación 23 que toma información de la tabla $c[t_{18}, S]$. La ecuación 25 muestra cómo calcular los valores de la tabla $c[t_{19}, S]$ cuando $S = \{\emptyset\}$, se conserva el máximo de las dos opciones.

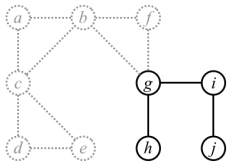
$$c[t_{19}, \{\emptyset\}] \leftarrow \max \begin{cases} c[t_{18}, \{\emptyset\}] = 1, \{j\} & \text{opción 1} \\ c[t_{18}, \{\emptyset \cup h\}] = 2, \{h + j\} & \text{opción 2} \end{cases} \tag{25}$$

La ecuación 26 presenta cómo calcular la entrada de la tabla $c[t_{19}, S]$ cuando $S = \{g\}$. Note que cuando se evalúa $S = \{g, h\}$ (opción 2) el resultado es $-\infty$ porque los vértices g, h no forman un conjunto 2-packing.

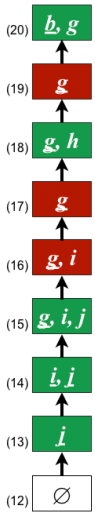
$$c[t_{19}, \{g\}] \leftarrow \max \begin{cases} c[t_{18}, \{g\}] = 1, \{g\} & \text{opción 1} \\ c[t_{18}, \{gh\}] = -\infty & \text{opción 2} \end{cases} \tag{26}$$

Ahora observe la tabla $c[t_{20}, S]$ en la Tabla 6. Se usa la ecuación 22 por tratarse de un nodo introductor. Note que los valores de $c[t_{20}, S]$ cuando $S = \{\emptyset\}$ y $S = \{g\}$ se pueden tomar de $c[t_{19}, S]$; ésto se indica en la ecuación 22 en la párte de $v \notin S$. Cuando $S = \{bg\}$, los vértices b, g no forman un conjunto 2-packing, por lo tanto $c[t_{20}, \{bg\}] \leftarrow -\infty$. El único valor que falta calcular es cuando $S = \{b\}$; ésto corresponde a la ecuación 22 en la parte de $v \in S$. Las ecuaciones 27 - 29 muestran cómo se calcula $c[t_{20}, \{b\}]$. La ecuación 27 muestra el procedimiento general, las funciones $d^1(t_{20})$ y $d^2(t_{20})$ apuntan hacia los nodos t_{18} y t_{15}, respectivamente. La ecuación 28 muestra de forma expandida los cálculos de la ecuación 27. Finalmente, $c[t_{20}, \{b\}]$ toma el valor de 2, vea la ecuación 29.

$$c[t_{20}, \{b\}] \leftarrow \max \begin{cases} \max\left(1, c[t_{19}, \{\emptyset \cup b\}]\right) \\ \max\left(f\left(d^1(t_{20})\right), f\left(d^2(t_{20})\right)\right) \end{cases} \tag{27}$$



(a) El subgrafo resaltado de G corresponde al subgrafo de (T, X) en la Figura de abajo



(b) Subgrafo de de (T, X)

Figura 25. Subgrafos para ejemplificar el uso de las ecuaciones 21 - 23.

Tabla 6. Tablas $c[t_i, S]$ del subgrafo de la Figura 25(b).

(20)

∅	***b***	***g***	***bg***
2	2	1	$-\infty$
$h+j$	$b+j$	g	

(19)

∅	***g***	***h***
2	1	
$h+j$	g	

(18)

∅	***g***	***h***	***gh***
1	1	2	$-\infty$
j	g	$h+j$	

(17)

∅	***g***	***i***
1	1	
j	g	

(16)

∅	***g***	***i***	***gi***	***j***
1	1	1	$-\infty$	
j	g	i		

(15)

∅	***g***	***i***	***j***	***gi***	***gj***	***ij***	***gij***
0	1	1	1	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	g	i	j				

(14)

∅	***i***	***j***	***ij***
0	1	1	$-\infty$
	i	j	

(13)

∅	***j***
0	1

(12)

∅
0

$$c[t_{20},\{b\}] \leftarrow \max\left\{\begin{array}{l}\max\left(1,(c[t_{19},\{\varnothing\cup b\}]=-\infty)\right)\\ \max\left(\begin{array}{l}\max\left(\begin{array}{l}c[t_{18},\{\varnothing\cup b\}]=2\\ c[t_{18},\{bg\}]=-\infty\\ c[t_{18},\{bh\}]=-\infty\\ c[t_{18},\{bgh\}]=-\infty\end{array}\right),\max\left(\begin{array}{l}c[t_{15},\{\varnothing\cup b\}]=1\\ c[t_{15},\{bg\}]=-\infty\\ c[t_{15},\{bi\}]=-\infty\\ c[t_{15},\{bj\}]=2\\ c[t_{15},\{bgi\}]=-\infty\\ c[t_{15},\{bgj\}]=-\infty\\ c[t_{15},\{bij\}]=-\infty\\ c[t_{15},\{bgij\}]=-\infty\end{array}\right)\end{array}\right)\end{array}\right. \tag{28}$$

$$c[t_{20},\{b\}] \leftarrow 2,\{b,j\} \tag{29}$$

Para ejemplificar el uso de la ecuación 24 se usa la Figura 26 y de la Tabla 7. El propósito de la ecuación 24 es unir las respuestas parciales de los subárboles enraizados en el hijo izquierdo y derecho de t_i. Cuando $S \neq \{\varnothing\}$, el valor de la tabla $c[t_{25},S]$ une las soluciones de $c[t_6,S]$ y $c[t_{24},S]$ y elimina los elementos repetidos. Si $S=\{\varnothing\}$, entonces se revisan los descendientes del nodo. Se utiliza el algoritmo de Mjelde (2004) para saber cuáles nodos descendientes revisar. Recuerde que el algoritmo de Mjelde (2004) calcula un conjunto k-packing máximo en el grafo árbol; suponiendo un árbol binario y $k=2$, el algoritmo indica que deben de examinarse el nodo descendiente izquierdo a distancia 1 con el nodo descendiente derecho a distancia 2 (distancias 1-2), así como los nodos a distancias 2-1 y 2-2. Vea la Figura 26, la línea sólida del nodo t_6 al nodo t_{21} representa la distancia 1-2. Las líneas segmentadas entre los nodos t_4 y t_{24} son las distancias 2-1. La línea punteada entre los nodos t_4 y t_{21} indica los nodos a distancia 2-2. Vea las ecuaciones 30 - 32. Finalmente, $c[t_{25},\{\varnothing\}] \leftarrow 3,\{a,h,j\}$.

$$c[t_6,\{\varnothing\}] + \begin{cases} c[t_{21},\{\varnothing\}] & = 3,\{a,h,j\} \\ c[t_{21},\{g\}] & = -\infty, \end{cases} \tag{30}$$

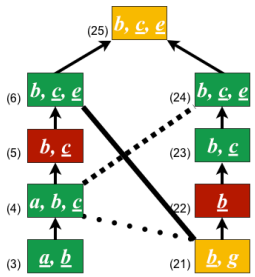


Figura 26. Subgrafo para ejemplificar el uso de la ecuación 24.

Tabla 7. Tablas $c[t_i, S]$ correspondientes al subgrafo de la Figura 26.

(25)

∅	**b**	**c**	**e**	**bc**	**be**	**ce**	**bce**
3	2	3	3	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$a+h+j$	$b+j$	$c+h+j$	$e+h+j$				

(6)

∅	**b**	**c**	**e**	**bc**	**be**	**ce**	**bce**
1	1	1	1	$-\infty$	$-\infty$	$-\infty$	$-\infty$
a	b	c	e				

(24)

∅	**b**	**c**	**e**	**bc**	**be**	**ce**	**bce**
2	2	3	3	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$h+j$	$b+j$	$c+h+j$	$e+h+j$				

(5)

∅	**b**	**c**	**bc**	**a**
1	1	1	$-\infty$	
a	b	c		

(23)

∅	**b**	**c**	**bc**
2	2	3	$-\infty$
$h+j$	$b+j$	$c+h+j$	

(4)

∅	**a**	**b**	**c**	**ab**	**ac**	**bc**	**abc**
0	1	1	1	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	a	b	c				

(22)

∅	**b**	**g**
2	2	
$h+j$	$b+j$	

(3)

∅	**a**	**b**	**ab**
0	1	1	
	a	b	

(21)

∅	**b**	**g**	**bg**
2	2	1	$-\infty$
$h+j$	$b+j$	g	

$$\max\left\{\begin{array}{l} c[t_4, \{\emptyset\}] \\ c[t_4, \{a\}] \end{array}\right\} + c[t_{24}, \{\emptyset\}] = \left\{\begin{array}{l} 2, \{h, j\} \\ 3, \{a, h, j\} \end{array}\right. \tag{31}$$

$$\begin{aligned} c[t_4, \{\emptyset\}] + \begin{cases} c[t_{21}, \{\emptyset\}] = 2, \{h, j\} \\ c[t_{21}, \{g\}] = \{g\} \end{cases} \\ c[t_4, \{a\}] + \begin{cases} c[t_{21}, \{\emptyset\}] = 3, \{a, h, j\} \\ c[t_{21}, \{g\}] = -\infty \end{cases} \end{aligned} \tag{32}$$

5.2. Demostración de que el algoritmo es correcto

Ahora se analizan las ecuaciones 21 - 24 con el objetivo de demostrar que el algoritmo propuesto es correcto. Recuerde que las ecuaciones están definidas sobre una desmembración de árbol agradable (T, X) y que para calcular las entradas de la tabla $c[t_i, S]$ sólo se necesita conocer las soluciones de los nodos descendientes de t_i.

Lema 5.1 *La ecuación 21 es verdadera si t_i es un nodo hoja.*

Demostración. Dado que t_i no tiene descendientes (por ser nodo hoja), el único valor que $c[t_i, S]$ puede tener es cero; i.e., $c[t_i, \{\emptyset\}] \leftarrow 0$. ■

Lema 5.2 *Sea t_i un nodo introductor, entonces la ecuación 22 es verdadera.*

Demostración. Puesto que t_i es un nodo introductor, la bolsa X_{t_i} debe tener un vértice más que la bolsa del nodo t_{i-1}. Sea v dicho vértice. Existen dos casos, $v \notin S$ y $v \in S$.

- **Caso 1:** el vértice v **no** se considera en el conjunto S. Entonces todos los posibles conjuntos S que se generan en t_{i-1} y t_i son iguales. De aquí que todas las soluciones $\hat{S}$ calculadas en $c[t_{i-1}, S]$ también existe en $c[t_i, S]$; por lo tanto, $c[t_i, S] \leftarrow c[t_{i-1}, S]$.
- **Caso 2:** el vértice v **sí** se considera en el conjunto S. Sea $\hat{S}$ la solución donde el máximo se obtiene en $c[t_i, S]$. Entonces $\hat{S} \setminus \{v\}$ es una solución calculada en algún descendiente de t_i que se encuentre a distancia 3 o más de v, sea t_x dicho descendiente. Entonces $\hat{S} \setminus \{v\}$ debe estar considerado en $c[t_x, S \setminus \{v\}]$, lo cual implica que: $c[t_x, S \setminus \{v\}] \geq \hat{S} \setminus \{v\} = \left|\hat{S}\right| - 1 = c[t_i, S] \setminus \{v\}$. Consecuentemente:

$$c[t_i, S] = c[t_x, S \setminus \{v\}] \cup v \tag{33}$$

  Ahora considere que $\hat{S}_x$ es la solución donde el máximo se obtiene de $c[t_x, S \setminus \{v\}]$. Puesto que se supone que S es un conjunto 2-packing, entonces v no tiene ningún vecino a distancia 2 o menos en $S \setminus \{v\} = \hat{S}_x \cap X_x$. Además, el vértice v no tiene ningún vecino a distancia menor o igual que 2 en $V_{t_x} \setminus X_{t_x}$; i.e., los vértices en $X_{t_{x-1}}$ están a distancia mayor o igual que 3 de v. De aquí que v no tenga ningún vecino a distancia 2 o menos en $\hat{S}_x$ lo cual significa que $\hat{S}_x \cup \{v\}$ es un conjunto 2-packing. Puesto que este conjunto intersecta con X_{t_i} exactamente en S, esto se considera en $c[t_i, S]$ por lo tanto:

$$c[t_i, S] = \hat{S}_x \cup \{v\} = |\hat{S}| + 1 = c[t_x, S \setminus \{v\}] \cup \{v\} \tag{34}$$

Note que para conocer la identidad del nodo t_x se hace uso de la función $d^q(t_i)$. En conjunto, las ecuaciones 33 y 34 demuestran el caso cuando $v \in S$.

■

Lema 5.3 *Sea t_i un nodo eliminador, entonces la ecuación 23 es verdadera.*

Demostración. Si t_i es un nodo eliminador, entonces X_{t_i} tienen un vértice menos que $X_{t_{i-1}}$, sea w dicho nodo. Se consideran dos casos:

- **Caso 1:** el vértice w **no** pertenece a $\hat{S}$. Entonces $\hat{S}$ es una solución considerada en $c[t_{i-1}, S]$; por lo tanto, $c[t_i, S] \leftarrow c[t_{i-1}, S]$.
- **Caso 2:** el vértice w **sí** pertenece a $\hat{S}$. Entonces, $\hat{S}$ es una solución considerada en $c[t_{i-1}, S \cup \{w\}]$; por lo tanto, $c[t_i, S] \leftarrow c[t_{i-1}, S]$.

Observe que ambas alternativas están consideradas en la ecuación 23. ■

Lema 5.4 *Sea t_i un nodo unión, entonces la ecuación 24 es correcta.*

Demostración. Si t_i es un nodo unión, entonces: $X_{t_i} = X_{t_l} = X_{t_r}$. Se consideran dos casos:

- **Caso 1:** el conjunto S **no** contiene a $\{\emptyset\}$. Sea $\hat{S}$ la solución en $c[t_i, S]$. Sean $\hat{S}_l = \hat{S}_l \cap V_{t_l}$ y $\hat{S}_r = \hat{S} \cap V_{t_r}$. Note que $\hat{S}_l$ es un conjunto 2-packing y que $\hat{S}_l \cap X_{t_i} = S$ y ésto está considerado en $c[t_l, S]$, algo semejante ocurre con $\hat{S}_r$. De aquí que:

$$c[t_i, S] \leftarrow \hat{S} = \left|\hat{S}_l\right| + \left|\hat{S}_r\right| - |S| = c[t_l, S] \cup c[t_r, S] \setminus S \tag{35}$$

- **Caso 2:** el conjunto S **sí** contiene a $\{\emptyset\}$. A diferencia del caso anterior no se sabe cuál es el contenido de $\hat{S}_l$ ni de $\hat{S}_r$, por lo tanto hay que revisar las combinaciones de vértices que formen un conjunto 2-packing. Las combinaciones a revisar son los vértices a distancia 1-2, 2-1 y 2-2. La razón y demostración de revisar estos vértices se toma del algoritmo de Mjelde (2004).

El **Caso 1** y **Caso 2** completa la demostración. ■

5.3. Análisis de complejidad del algoritmo

A continuación se presenta el análisis de complejidad del algoritmo propuesto.

Teorema 5.1 *Sea $G = (V_G, E_G)$ un grafo 1-outerplanar de orden n, de anchura de árbol a lo más k y (T, X) su desmembración de árbol agradable. Entonces, encontrar un conjunto 2-packing máximo para G en (T, X), usando las ecuaciones 21 - 24 y las tablas $c[t_i, S]$, requiere $O\left(2^{k+1}k^c n\right)$ u.t., para c constante.*

Demostración. Observe que la anchura de árbol para G es a lo más k; por lo tanto, el tamaño de la bolsa para cada nodo t_i es $X_{t_i} \leq k+1$. Por cada nodo t_i se llena la tabla $c[t_i, S]$ de tamaño $2^{|X_{t_i}|} \leq 2^{k+1}$. Las operaciones para llenar una entrada de $c[t_i, S]$ requieren $O(k)$ u.t; no obstante, también es necesario revisar que S sea un conjunto 2-packing y revisar el vecindario de algunos vértices. Puesto que G es un grafo de anchura de árbol a lo más k, es posible construir una estructura de datos en tiempo $O(k^c n)$ (para c constante) que permita revisar adyacencias en $O(k)$ u.t. (Cygan *et al.* (2015)).

Entonces, por cada nodo t_i se requieren $O\left(2^{k+1}k^c\right)$ u.t. para calcular la tabla $c[t_i, S]$; puesto que el orden de (T, X) es a lo más $4n$ (Lema 4.2). El problema de encontrar un conjunto 2-packing máximo en (T, X) para G se puede resolver en $O\left(2^{k+1}k^c n\right)$ u.t. ■

Teorema 5.2 *Sea $G = (V_G, E_G)$ un grafo 1-outerplanar de orden n, de anchura de árbol a lo más k y (T, X) su desmembración de árbol agradable. Entonces, encontrar un conjunto 2-packing máximo para G en (T, X) requiere $O(n)$ u.t., para c constante.*

Demostración. Por el Teorema 5.1 encontrar un conjunto 2-packing máximo en (T, X) para el grafo 1-outerplanar toma $O\left(2^{k+1}k^c n\right)$ u.t. Observe que c es una constante y que para el grafo 1-outerplanar k también está limitado por una constante, particularmente $k \leq 2$. De aquí que la complejidad de encontrar un conjunto 2-packing máximo para el grafo 1-outerplanar dado (T, X) mediante el Teorema 5.1 es: $O(n)$ u. t. ■

Teorema 5.3 *El algoritmo del Pseudocódigo 12 encuentra un conjunto 2-packing máximo en el grafo 1-outerplanar en $O(n)$ u. t.*

Demostración. Sea $G = (V_G, E_G)$ el grafo de entrada el Pseudocódigo 12, tal que G es 1-outerplanar y $n \leftarrow |V_G|$, entonces:

- Línea 2: calcular $\mathcal{T}_G$ mediante el algoritmo de Katsikarelis (2013) requiere $O(n)$ u.t.
- Línea 3: seleccionar un nodo raíz para $\mathcal{T}_G$ toma $O(1)$ u.t.
- Línea 4: crear el grafo (T, X) mediante el procedimiento de Cygan *et al.* (2015) requiere $O(n)$ u.t. cuando G es 1-outerplanar.
- Línea 5: seleccionar la raíz para (T, X) es $O(1)$ u. t.
- Línea 6: realizar un recorrido postorden en (T, X) requiere $O(n)$ u.t. (puesto que el orden de (T, X) está acotado por $4n$, Lema 4.2)
- Líneas 7 y 8: es un recorrido sobre (T, X) indicando los vértices de articulación, $O(n)$ u.t.
- Línea 9: por el Teorema 5.2 la complejidad es $O(n)$ u.t.
- Línea 10: la rutina ReconstruyeM2PO (no se muestra) distribuye la solución encontrada en $c[t_N, \{\emptyset\}]$ sobre los vértices de G en $O(n)$ u.t.

La complejidad total del algoritmo del Pseudocódigo 12 es $O(n)$ u.t. ■

Capítulo 6. Conclusiones

Este capítulo resume las aportaciones del presente trabajo de investigación, provee de información adicional sobre los conceptos y temas tratados, y discute las posibles rutas que puede tomar el trabajo futuro en esta línea de investigación.

6.1. Resumen

La tesis se enfoca en encontrar un conjunto 2-packing máximo en los grafos 1 y 2-outerplanares, para ello el trabajo se divide en cuatro fases. La primera fase para resolver este problema fue a través del algoritmo MAXIMUM-2-PACK-CACTUS que encuentra un conjunto 2-packing máximo en el grafo cactus en $O\left(n^2\right)$ u.t. A partir del cactus de entrada, el algoritmo calcula un árbol de componentes biconectados enraizado. Luego procesa secuencialmente cada bloque de acuerdo con su número postorden.

En cada bloque, el algoritmo MAXIMUM-2-PACK-CACTUS marca los vértices que deben pertenecer al conjunto 2-packing máximo. El primer acercamiento para resolver este problema fue encontrar un conjunto 2-packing máximo en un grafo uniciclo. Para dicho problema, el reto mayor es generar un conjunto de soluciones factibles con base en los mejores marcados de todos los árboles de expansión del uniciclo. Una vez que se encuentra este conjunto, se elige el mejor marcado. El mecanismo para transformar un grafo uniciclo y su marcado en un árbol con un marcado equivalente, mediante UNICYCLE-TO-TREE, es muy útil para demostrar por inducción que el algoritmo MAXIMUM-2-PACK-CACTUS es correcto.

Hasta donde el autor tiene conocimiento, el algoritmo MAXIMUM-2-PACK-CACTUS mejora los resultados actuales en la literatura para este problema y clase de grafos. No obstante, su tiempo de ejecución es elevado (comparado con las cotas conjeturadas en la literatura), además de que la implementación del algoritmo es complicada.

En la segunda fase de trabajo se calcula un conjunto 2-packing maximal en un grafo 2-outerplanar mediante el algoritmo distribuido M2H (**M**AXIMAL_**2**-PACKING-SET_**H**ALIN). Este algoritmo encuentra un conjunto 2-packing maximal en un grafo Halin no geométrico y no dirigido en $O(n)$ pasos. Uno de los retos en esta parte fue el transformar

el algoritmo de Trejo-Sánchez y Fernández-Zepeda (2014) para poderlo ejecutar en un grafo no geométrico y sobre una topología más general para el que se diseñó. Para ello es necesario previamente encontrar una de las caras externas del grafo, y asignar a las aristas una numeración mediante un proceso que se asemeja a la descomposición de orejas.

Para encontrar una de las caras externas del grafo Halin se utiliza una adaptación distribuida del algoritmo secuencial de Eppstein (2016). Este algoritmo aplica reglas de reducción sobre el grafo de entrada para reconocer si el grafo es Halin. Posteriormente, mediante una fase de expansión, es posible identificar una de las caras externas del grafo. M2H combina una versión distribuida del algoritmo de Eppstein (2016) y la transformación de Trejo-Sánchez y Fernández-Zepeda (2014) para encontrar un conjunto 2-packing maximal en el grafo Halin en un entorno distribuido.

La tercera fase presenta algunas metodologías para resolver algunos problemas de grafos que cumplen con ciertas propiedades en clases restringidas de grafos, así como las bases teóricas que apuntan hacia la existencia de algoritmos polinomiales para ciertos problemas NP-difíciles en grafos de ciertas familias. Las bases teóricas y las metodologías tienen su intersección en la desmembración de grafos. Particularmente, esta fase presta atención a la desmembración de árbol de un grafo y a la anchura de árbol como parámetros que en cierto sentido indican la conectividad del grafo.

Asimismo, se expone que algunos problemas NP-difíciles con anchura de árbol acotada tienden a tener solución en tiempo polinomial e incluso lineal. Se recopilan las propiedades que tanto el grafo como el problema deben de cumplir para analizar la factibilidad de resolver el problema en tiempo polinomial. Finalmente, se muestra cómo tanto el grafo 1-outerplanar y el problema del 2-packing máximo en este grafo cumplen con las propiedades requeridas.

La cuarta fase y contribución principal de esta tesis es un algoritmo de programación dinámica que calcula un conjunto 2-packing máximo en el grafo 1-outerplanar no geométrico en $O(n)$ u.t. Este algoritmo utiliza la desmembración de árbol agradable del grafo 1-outerplanar y sobre ésta encuentra un conjunto 2-packing máximo que también lo es para el grafo 1-outerplanar. El algoritmo propuesto es FPT puesto que se utiliza la anchura de árbol como parámetro sobre el cual se realizan las operaciones

combinatorias; por lo general, en este parámetro la combinatoria explota en un valor exponencial. No obstante, se concluye que la complejidad del algoritmo propuesto es polinomial bajo la observación de que la combinatoria que se realiza está acotada por una constante pequeña.

6.2. Conclusiones y trabajo futuro

La desmembración de árbol es una metodología poderosa para resolver ciertos problemas en grafos donde la anchura de árbol es acotada. Por lo tanto, se convierte en un concepto central en el diseño de algoritmos para grafos. El Teorema de Courcelle (1990) también es muy útil para discernir sobre cuáles problemas se pueden resolver mediante algún algoritmo de programación dinámica que contemple la anchura y la desmembración de árbol como parámetros adicionales.

Respecto al trabajo realizado en esta tesis, las ideas presentadas pueden generalizarse a otros grafos de anchura acotada donde la desmembración de árbol se conozca. Por ejemplo, el algoritmo propuesto del grafo 1-outerplanar puede adaptarse al grafo Halin. El impacto más substancial sería el incremento de la complejidad computacional. No obstante, con base en la evidencia teórica, el tiempo de ejecución debe mantenerse polinomial.

Precisamente, uno de los retos a resolver cuando se usan estas metodologías de desmembración de grafos es el de encontrar una desmembración óptima o al menos una aproximación buena, esto con el objetivo de evitar que las operaciones combinatorias sean demasiado grandes. Si se encuentra una buena desmembración, el proceso en general es describir un algoritmo de programación dinámica que aproveche las propiedades de conectividad de la desmembración y así calcular el valor óptimo de interés. Si no se encuentra dicha desmembración, debe de existir algún obstáculo combinatorio empotrado en el grafo que impide desmembrarlo. De aquí que se deban buscar otras técnicas de desmembración que permitan solucionar el problema o ayuden a decidir si es un caso que no se puede resolver en tiempo polinomial.

En esta tesis se trabajó con la desmembración de árbol de un grafo; no obstante, existe más de un tipo de desmembraciones. Algunas de ellas son más restrictivas y

piden que, por ejemplo, los separadores sean ‘balanceados.’ Otras desmembraciones interesantes son ‘rankwidth’ y ‘branchwidth’ para desmembrar grafos densos y no densos, respectivamente.

Finalmente, también resulta interesante el adaptar los algoritmos propuestos en este trabajo para solucionar el problema del conjunto k-packing máximo en los grafos 1-outerplanar y Halin.

Literatura citada

Agarwala, R. y Fernández-Baca, D. (1993). A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. *SIAM J. Comput.*, **23**: 1216–1224.

Arnborg, S. y Proskurowski, A. (1989). Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete Appl. Math.*, **23**(1): 11–24.

Arnborg, S., Corneil, D. G., y Proskurowski, A. (1987). Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, **8**(2): 277–284.

Arnborg, S., Lagergren, J., y Seese, D. (1991). Easy problems for tree-decomposable graphs. *Journal of Algorithms*, **12**(2): 308 – 340.

Awerbuch, B. (1987). Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. En: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, New York, NY, USA. ACM, STOC '87, pp. 230–240.

Ben-Moshe, B., Bhattacharya, B., y Shi, Q. (2005). Efficient algorithms for the weighted 2-center problem in a cactus graph. En: X. Deng y D.-Z. Du (eds.), *Algorithms and Computation*, Vol. 3827 de *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 693–703.

Berge, C. (1985). *Graphs and Hypergraphs*. Elsevier Science Ltd. Oxford, UK, UK.

Bern, M., Lawler, E., y Wong, A. (1987). Linear-time computation of optimal subgraphs of decomposable graphs. *Journal of Algorithms*, **8**(2): 216 – 235.

Bertele, U. y Brioschi, F. (1972). *Nonserial Dynamic Programming*. Academic Press, Inc. Orlando, FL, USA.

Bodlaender, H., Gilbert, J., Hafsteinsson, H., y Kloks, T. (1995). Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, **18**(2): 238 – 255.

Bodlaender, H. L. (1988). *Planar graphs with bounded treewidth*, Vol. 88. Unknown Publisher.

Bodlaender, H. L. (1993). A tourist guide through treewidth. *Acta Cybern.*, **11**(1-2): 1–21.

Bodlaender, H. L. (1996). A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, **25**(6): 1305–1317.

Bodlaender, H. L., Fellows, M. R., y Warnow, T. (1992). Two strikes against perfect phylogeny. En: *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, London, UK, UK. Springer-Verlag, ICALP '92, pp. 273–283.

Bollobás, B. (2013). *Modern graph theory*, Vol. 184. Springer Science & Business Media.

Brause, C., Lê, N. C., y Schiermeyer, I. (2015). The maximum independent set problem in subclasses of subcubic graphs. *Discrete Mathematics*, **338**(10): 1766 – 1778. Seventh Czech-Slovak International Symposium on Graph Theory, Combinatorics, Algorithms and Applications, Košice 2013.

Brehaut, W. M. (1977). An efficient outerplanarity algorithm. En: *Congr. Numer. XIX*. Proceedings of the 8th Southeastern Conference on Combinatorics, Graph Theory, and Computing, pp. 99 – 113.

Butenko, S. (2003). *Maximum Independent Set and Related Problems, with Applications*. Tesis de doctorado, University of Florida, Gainesville, FL, USA.

Castro, A., Klavar, S., Mollard, M., y Rho, Y. (2011). On the domination number and the 2-packing number of fibonacci cubes and lucas cubes. *Comput. Math. Appl.*, **61**(9): 2655–2660.

Chávez, E., Dobrev, S., Kranakis, E., Opatrny, J., Stacho, L., y Urrutia, J. (2004). Route discovery with constant memory in oriented planar geometric networks. En: S. E. Nikoletseas y J. D. P. Rolim (eds.), *Algorithmic Aspects of Wireless Sensor Networks*, Berlin, Heidelberg. Springer Berlin Heidelberg, pp. 147–156.

Chekuri, C., Gupta, A., Newman, I., Rabinovich, Y., y Sinclair, A. (2003). Embedding k-outerplanar graphs into $\ell 1$. En: *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, Philadelphia, PA, USA. SIAM, SODA '03, pp. 527–536.

Chellali, M., Favaron, O., Hansberg, A., y Volkmann, L. (2012). k-domination and k-independence in graphs: A survey. *Graphs and Combinatorics*, **28**(1): 1–55.

Cockayne, E., Goodman, S., y Hedetniemi, S. (1975). A linear algorithm for the domination number of a tree. *Inform Process. Lett.*, **4**(2): 41 – 44.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., y Clifford, S. (2009). *Introduction to Algorithms*. The MIT Press. p. 1312.

Cornuéjols, G., Naddef, D., y Pulleyblank, W. R. (1983). Halin graphs and the travelling salesman problem. *Mathematical Programming*, **26**(3): 287–294.

Courcelle, B. (1990). The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, **85**(1): 12 – 75.

Cygan, M., Fomin, F. V., Kowalik, Ł., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., y Saurabh, S. (2015). *Parameterized algorithms*, Vol. 3. Springer.

Deo, N., Krishnamoorthy, M. S., y Langston, M. A. (1987). Exact and approximate solutions for the gate matrix layout problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **6**(1): 79–84.

Diestel, R. (2018). *Graph theory*. Springer Publishing Company, Incorporated.

Diestel, R. y Müller, M. (2018). Connected tree-width. *Combinatorica*, **38**(2): 381–398.

Dijkstra, E. W. (1974). Self-stabilizing systems in spite of distributed control. *Commun. ACM*, **17**(11): 643–644.

Dirac, G. A. (1961). On rigid circuit graphs. En: *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*. Springer, Vol. 25, pp. 71–76.

Dourisboure, Y. y Gavoille, C. (2002). Improved compact routing scheme for chordal graphs. En: *Proceedings of the 16th International Conference on Distributed Computing*, Berlin, Heidelberg. Springer-Verlag, DISC '02, pp. 252–264.

Downey, R. G. y Fellows, M. R. (2013). *Fundamentals of parameterized complexity*, Vol. 4. Springer.

Eppstein, D. (2016). Simple recognition of halin graphs and their generalizations. *JGAA*, **20**(2): 323 – 346.

Fleischner, H., Sabidussi, G., y Sarvanov, V. I. (2010). Maximum independent sets in 3- and 4-regular hamiltonian graphs. *Discrete Mathematics*, **310**(20): 2742 – 2749. Graph Theory — Dedicated to Carsten Thomassen on his 60th Birthday.

Flum, J. y Grohe, M. (2006). *Parameterized complexity theory*. Springer Science & Business Media.

Fomin, F. V. y Kratsch, D. (2010). *Exact Exponential Algorithms*. Springer-Verlag, primera edición. Berlin, Heidelberg.

Fraigniaud, P. (2005). Greedy routing in tree-decomposed graphs. En: G. S. Brodal y S. Leonardi (eds.), *Algorithms – ESA 2005*, Berlin, Heidelberg. Springer Berlin Heidelberg, pp. 791–802.

Frick, M. y Grohe, M. (2004). The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, **130**(1): 3 – 31. Papers presented at the 2002 IEEE Symposium on Logic in Computer Science (LICS).

Frieze, A. M. y Tsourakakis, C. E. (2011). High degree vertices, eigenvalues and diameter of random apollonian networks. *CoRR*, **abs/1104.5259**.

Gairing, M., Geist, R. M., Hedetniemi, S. T., y Kristiansen, P. (2004). A self-stabilizing algorithm for maximal 2-packing. *Nordic J. of Computing*, **11**(1): 1–11.

Garey, M. R. y Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co. New York, NY, USA.

Gavril, F. (1972). Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, **1**(2): 180–187.

Gavril, P. (1975). An algorithm for testing chordality of graphs. *Information Processing Letters*, **3**(4): 110 – 112.

Ghosh, S. (2014). *Distributed Systems: An Algorithmic Approach, Second Edition*. Chapman & Hall/CRC, segunda edición.

Goddard, W., Hedetniemi, S. T., Jacobs, D. P., y Trevisan, V. (2008). Distance- k knowledge in self-stabilizing algorithms. *Theor. Comput. Sci.*, **399**(1-2): 118–127.

Hale, W. K. (1980). Frequency assignment: Theory and applications. En: *IEEE journals and magazines*. Proceedings of the IEEE, Vol. 68, pp. 1497–1514.

Halin, R. (1971). Studies on minimally n-connected graphs. En: D. J. A. Welsh (ed.), *Combinatorial Mathematics and its Applications*. Published by Academic Press Inc, pp. 129–136.

Halin, R. (1976). S-functions for graphs. *Journal of Geometry*, **8**(1): 171–186.

Harary, F. (1969). *Graph theory*. Addison-Wesley, Reading, MA.

Haynes, T., Hedetniemi, S., y Slater, P. (1998). *Domination in Graphs: Advanced Topics*. Chapman and Hall/CRC Pure and Applied Mathematics Series. Marcel Dekker, Incorporated. p. 520.

He, X. (1991). Efficient parallel algorithms for series parallel graphs. *Journal of Algorithms*, **12**(3): 409 – 430.

Hedetniemi, S., Laskar, R., y Pfaff, J. (1986). A linear algorithm for finding a minimum dominating set in a cactus. *Discrete Appl. Math.*, **13**(2a3): 287 – 292.

Hochbaum, D. S. y Shmoys, D. B. (1985). A best possible heuristic for the k-center problem. *Math. Oper. Res.*, **10**: 180–184.

Hopcroft, J. y Tarjan, R. (1973). Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, **16**(6): 372–378.

Hopcroft, J. y Tarjan, R. (1974). Efficient planarity testing. *J. ACM*, **21**(4): 549–568.

Hsieh, S.-Y. (2005). Efficiently parallelizable problems on a class of decomposable graphs. *Journal of Computer and System Sciences*, **70**(1): 140 – 156.

Hwang, F. K., Richards, D. S., y Winter, P. (1992). Chapter 5 polynomially solvable cases. En: F. K. Hwang, D. S. Richards, y P. Winter (eds.), *The Steiner Tree Problem*, Vol. 53 de *Annals of Discrete Mathematics*. Elsevier, pp. 177 – 188.

Imrich, W., Klavzar, S., y Rall, D. F. (2008). *Topics in Graph Theory: Graphs and Their Cartesian Product*. AK Peters Ltd. p. 219.

Jamieson, A. (2007). *Linear-time algorithms for edge-based problems*. Tesis de doctorado, Clemson University, Clemson, SC, USA.

Jiménez, R. y Estrada, M. (2014). *Introducción a los algoritmos distribuidos*. Universidad Autónoma Metropolitana. p. 251.

Kannan, S. K. y Warnow, T. J. (1994). Inferring evolutionary history from dna sequences. *SIAM Journal on Computing*, **23**(4): 713–737.

Kariv, O. y Hakimi, S. L. (1979). An algorithmic approach to network location problems. I: The *p*-centers. *SIAM J. on Appl. Math.*, **37**(3): pp. 513–538.

Karthick, T. (2016). Weighted independent sets in a subclass of p6-free graphs. *Discrete Mathematics*, **339**(4): 1412 – 1418.

Karthick, T. y Maffray, F. (2017). Maximum weight independent sets in classes related to claw-free graphs. *Discrete Applied Mathematics*, **216**: 233 – 239. Special Graph Classes and Algorithms — in Honor of Professor Andreas Brandstädt on the Occasion of His 65th Birthday.

Katsikarelis, I. (2013). Computing bounded-width tree and branch decompositions of k-outerplanar graphs. *arXiv preprint arXiv:1301.5896*.

Keil, J. M., Mitchell, J. S., Pradhan, D., y Vatshelle, M. (2017). An algorithm for the maximum weight independent set problem on outerstring graphs. *Computational Geometry*, **60**: 19 – 25. The Twenty-Seventh Canadian Conference on Computational Geometry August 2015.

Kloks, T. (1994). *Treewidth: computations and approximations*, Vol. 842. Springer Science & Business Media.

Kneis, J. y Langer, A. (2009). A practical approach to courcelle's theorem. *Electronic Notes in Theoretical Computer Science*, **251**: 65 – 81. Proceedings of the International Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2008).

Koontz, W. (1980). Economic evaluation of loop feeder relief alternatives. *Bell System Technical Journal*, **59**(3): 277–293.

Kornai, A. y Tuza, Z. (1992). Narrowness, pathwidth, and their application in natural language processing. *Discrete Appl. Math.*, **36**(1): 87–92.

Kshemkalyani, A. D. y Singhal, M. (2008). *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, primera edición. New York, NY, USA.

Kuratowski, C. (1930). Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, **15**(1): 271–283.

Laskar, R., Pfaff, J., Hedetniemi, S., y Hedetniemi, S. (1984). On the algorithmic complexity of total domination. *SIAM Journal on Algebraic Discrete Methods*, **5**(3): 420–425.

Lauritzen, S. L. y Spiegelhalter, D. J. (1990). Local computations with probabilities on graphical structures and their application to expert systems. En: G. Shafer y J. Pearl (eds.), *Readings in Uncertain Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 415–448.

Lozin, V. (2017). From matchings to independent sets. *Discrete Applied Mathematics*, **231**: 4 – 14. Algorithmic Graph Theory on the Adriatic Coast.

Mahajan, S. y Peters, J. G. (1994). Regularity and locality in k-terminal graphs. *Discrete Applied Mathematics*, **54**(2): 229 – 250.

Manne, F. y Mjelde, M. (2006). A memory efficient self-stabilizing algorithm for maximal k-packing. En: A. Datta y M. Gradinariu (eds.), *Stabilization, Safety, and Security of Distributed Systems*, Vol. 4280 de *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 428–439.

Meir, A. y Moon, J. W. (1975). Relations between packing and covering numbers of a tree. *Pacific J. of Math.*, **61**(1): 225–233.

Michael, D. y Battiston, S. (2009). From graph theory to models of economic networks. a tutorial. En: A. Naimzada, S. Stefani, y A. Torriero (eds.), *Networks, Topology and Dynamics*, Vol. 613 de *Lecture Notes in Economics and Mathematical Systems*. Springer Berlin Heidelberg, pp. 23–63.

Mitchell, S. L. (1979). Linear algorithms to recognize outerplanar and maximal outerplanar graphs. *Information Processing Letters*, **9**(5): 229 – 232.

Mjelde, M. (2004). *K-packings and K-domination on tree graphs*. Tesis de maestría, Department of Informatics, University of Bergen, Norway.

Mosca, R. (2017). A sufficient condition to extend polynomial results for the maximum independent set problem. *Discrete Applied Mathematics*, **216**(P1): 281–289.

Murat, C. y Paschos, V. T. (2002). A priori optimization for the probabilistic maximum independent set problem. *Theoretical Computer Science*, **270**(1): 561 – 590.

Nešetřil, J. (2008). Structural properties of sparse graphs. *Electronic Notes in Discrete Mathematics*, **31**: 247 – 251. The International Conference on Topological and Geometric Graph Theory.

Niedermeier, R. (2006). *Invitation to fixed-parameter algorithms*. Oxford.

Orlovich, Y., Blazewicz, J., Dolgui, A., Finke, G., y Gordon, V. (2011). On the complexity of the independent set problem in triangle graphs. *Discrete Mathematics*, **311**(16): 1670 – 1680.

Pach, J. (2013). The beginnings of geometric graph theory. En: L. Lovász, I. Z. Ruzsa, y V. T. Sós (eds.), *Erdős Centennial*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 465–484.

Papamanthou, C. y Tollis, I. G. (2008). Algorithms for computing a parameterized st-orientation. *Theor. Comput. Sci.*, **408**(2–3): 224 – 240. Excursions in Algorithmics: A Collection of Papers in Honor of Franco P. Preparata.

Paten, B., Diekhans, M., Earl, D., St. John, J., Ma, J., Suh, B., y Haussler, D. (2010). Cactus graphs for genome comparisons. En: B. Berger (ed.), *Research in Computational Molecular Biology*, Vol. 6044 de *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 410–425.

Patil, H. (1986). On the structure of k-trees. *Journal of Combinatorics, Information and System Sciences*, **11**(2-4): 57–64.

Robertson, N. y Seymour, P. (1984). Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, **36**(1): 49 – 64.

Sakai, S., Togasaki, M., y Yamazaki, K. (2003). A note on greedy algorithms for the maximum weighted independent set problem. *Discrete Applied Mathematics*, **126**(2): 313 – 322.

Shi, Z. (2012). A self-stabilizing algorithm to maximal 2-packing with improved complexity. *Inform Process. Lett.*, **112**(13): 525–531.

Tanenbaum, A. S. y Steen, M. v. (2006). *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA.

Trejo-Sánchez, J. A. y Fernández-Zepeda, J. A. (2012). A self-stabilizing algorithm for the maximal 2-packing in a cactus graph. En: *IPDPS Workshops*. IEEE Computer Society, pp. 863–871.

Trejo-Sánchez, J. A. y Fernández-Zepeda, J. A. (2014). Distributed algorithm for the maximal 2-packing in geometric outerplanar graphs. *J. Parall. Distr. Com.*, **74**(3): 2193 – 2202.

Turau, V. (2012). Efficient transformation of distance-2 self-stabilizing algorithms. *J. Parall. Distr. Com.*, **72**(4): 603 – 612.

Valiente, G. (2002). *Algorithms on Trees and Graphs*. Springer-Verlag. Berlin, Heidelberg.

van der Gaag, L. C. (1990). *Probability-based models for plausible reasoning*. Tesis de doctorado, University of Amsterdam.

van Leeuwen, J. (1990). Chapter 10 - graph algorithms. En: J. van Leeuwen (ed.), *Algorithms and Complexity*. Elsevier, Amsterdam, Handbook of Theoretical Computer Science, pp. 525 – 631.

Wimer, T. V. (1987). *Linear Algorithms on K-terminal Graphs*. Tesis de doctorado, Clemson University, Clemson, SC, USA. AAI8803914.

Winter, P. (1987). Steiner problem in halin networks. *Discrete Applied Mathematics*, **17**(3): 281–294.

Xiao, M. y Nagamochi, H. (2016). An exact algorithm for maximum independent set in degree-5 graphs. *Discrete Applied Mathematics*, **199**: 137 – 155. Sixth Workshop on Graph Classes, Optimization, and Width Parameters, Santorini, Greece, October 2013.

Xu, X., Ma, J., y Wang, H. (2006). An improved simulated annealing algorithm for the maximum independent set problem. En: D.-S. Huang, K. Li, y G. W. Irwin (eds.), *Intelligent Computing*, Berlin, Heidelberg. Springer Berlin Heidelberg, pp. 822–831.

Zmazek, B. y Žerovnik, J. (2004). The obnoxious center problem on weighted cactus graphs. *Discrete Appl. Math.*, **136**(2-3): 377–386.