

Tesis defendida por
Joel Antonio Trejo Sánchez
y aprobada por el siguiente comité

Dr. José Alberto Fernández Zepeda
Director del Comité

Dr. Carlos Alberto Brizuela Rodríguez
Miembro del Comité

Dr. Edgar Leonel Chávez González
Miembro del Comité

Dr. Hugo Homero Hidalgo Silva
Miembro del Comité

Dr. José Antonio García Macías
Coordinador del Programa de
Posgrado en Ciencias de la Computación

Dr. Jesús Favela Vara
Director de Estudios de Posgrado

Febrero de 2014

**CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE
EDUCACIÓN SUPERIOR DE ENSENADA, BAJA
CALIFORNIA**



**Programa de Posgrado en Ciencias
en Ciencias de la Computación**

Diseño de Algoritmos Auto-estabilizantes para el Problema del Conjunto
Independiente Fuerte

Tesis

para cubrir parcialmente los requisitos necesarios para obtener el grado de

Doctor en Ciencias

Presenta:

Joel Antonio Trejo Sánchez

Ensenada, Baja California, México

2014

Resumen de la tesis de Joel Antonio Trejo Sánchez, presentada como requisito parcial para la obtención del grado de Doctor en Ciencias en Ciencias de la Computación.

Diseño de Algoritmos Auto-estabilizantes para el Problema del Conjunto Independiente Fuerte

Resumen aprobado por:

Dr. José Alberto Fernández Zepeda

Director de Tesis

Esta tesis se enfoca en el estudio del problema del conjunto independiente fuerte en algoritmos distribuidos y en algoritmos auto-estabilizantes. Se presenta un algoritmo auto-estabilizante para elección de líder para un grafo arbitrario. Este algoritmo es óptimo dado que encuentra un líder en $O(\text{diam})$ rondas, donde diam es el diámetro del grafo de entrada. Dicho algoritmo es un componente fundamental de los algoritmos de la tesis.

En segundo lugar, se presenta un algoritmo auto-estabilizante que calcula un conjunto independiente fuerte máximo en un anillo. Posteriormente, se extiende dicho algoritmo para calcular un conjunto independiente fuerte maximal en un cactus. Ambos algoritmos mejoran la complejidad temporal de los resultados previos cuando el grafo de entrada es un anillo o un cactus. Finalmente, se presenta un algoritmo distribuido que calcula un conjunto independiente fuerte maximal en un grafo outerplanar geométrico. La complejidad temporal de este algoritmo es $O(n)$ rondas, donde n es el número de vértices en el grafo. Este algoritmo también calcula un conjunto independiente fuerte máximo cuando el grafo de entrada es un anillo. La dificultad principal en diseñar estos algoritmos es que los procesadores sólo tienen una vista local del sistema y necesitan leer información a una distancia dos. Todos los algoritmos que se presentan en la presente tesis son determinísticos.

Palabras Clave: **Auto-estabilización, Complejidad computacional, Conjunto independiente fuerte**

Abstract of the thesis presented by Joel Antonio Trejo Sánchez, in partial fulfillment of the requirements of the degree of Doctor in Sciences in Computer Sciences.

Design of Self-stabilizing Algorithms for the 2-Packing Set Problem

Abstract approved by:

Dr. José Alberto Fernández Zepeda

Thesis director

This thesis focuses on studying the 2-packing set problem in distributed and self-stabilizing algorithms. It presents a simple leader election self-stabilizing algorithm for a graph with arbitrary topology. This algorithm is optimal since it finds a leader in $O(\text{diam})$ rounds, where diam is the diameter of the input graph. This algorithm is a fundamental component of the algorithms in the thesis.

Second, we present a self-stabilizing algorithm that computes a maximum 2-packing set in a ring. Next, we extend such an algorithm to also compute a maximal 2-packing set in a cactus graph. Both algorithms outperform the time complexity of previous results when the input graph is a ring or a cactus. Finally, we present a distributed algorithm that computes a maximal 2-packing set in a geometric outerplanar graph. The time complexity of this algorithm is $O(n)$ rounds, where n is the number of vertices in the graph. This algorithm also computes a maximum 2-packing set when the input graph is a ring. The main difficulty in designing these algorithms is that processors only have a local view of the system and they need to read information at distance-two. All the algorithms presented in this thesis are deterministic.

Keywords: **Self-stabilization, Computer complexity, 2-packing set**

Dedicatoria

A mi esposa

A mis hijos

A mis padres

A mi abuelita

Agradecimientos

A mi director de tesis, el Dr. José Alberto Fernández Zepeda, por ser un guía constante en mi formación científica. Gracias por su confianza para colaborar conmigo y por brindarme su invaluable apoyo.

A los miembros de mi comité de tesis: el Dr. Carlos A. Brizuela Rodríguez, el Dr. Hugo Homero Hidalgo Silva y al Dr. Edgar L. Chávez González. Gracias por sus valiosos comentarios y sugerencias durante el desarrollo de mi tesis.

A todos mis compañeros del departamento de Ciencias de la Computación, en especial a los del laboratorio de algoritmos. Gracias por su amistad.

A todos mis amigos del Cuerpo Académico en Ciencias de la Computación y Comunicaciones. Al Dr. Julio César Ramírez Pacheco, al Dr. Luis Rizo Domínguez, al M. en C. Francisco Manzano Pinzón. Excelente compañeros de trabajo, pero sobretodo invaluable amigos.

Al único y sabio Dios, mi Salvador, por ser mi esperanza y mi fortaleza.

A mi familia. En especial a mi esposa, mis hijos, mis padres y mis hermanos quienes fueron mi principal motivación en los momentos más difíciles. A Nelda, a mis tíos, primos y demás familiares.

Al Centro de Investigación Científica y de Educación Superior de Ensenada.

A la Universidad del Caribe.

Al Consejo Nacional de Ciencia y Tecnología

Al PROMEP

Contenido

	Página
Resumen en español	ii
Resumen en inglés	iii
Dedicatoria	iv
Agradecimientos	v
Lista de Figuras	viii
Lista de Tablas	ix
1 Introducción	1
1.1 Auto-estabilización en sistemas distribuidos	4
1.1.1 Conceptos básicos	7
1.2 Preliminares	15
1.2.1 Elección de líder	15
1.2.2 Conjuntos independientes	17
1.2.3 Conjuntos dominantes	20
1.2.4 Conjunto independiente fuerte	21
2 Algoritmo auto-estabilizante para elección de líder	26
2.1 Introducción	26
2.2 Trabajo previo	27
2.3 Especificación del modelo	29
2.4 Algoritmo de elección de líder	30
2.4.1 Algoritmo ELECCION-LIDER	35
2.5 Convergencia del algoritmo ELECCION-LIDER	37
3 Algoritmos auto-estabilizantes para el CIF	42
3.1 Introducción	42
3.2 Algoritmo CIF-ANILLO	43
3.2.1 Fase de optimización	44
3.3 Algoritmo CIF-CACTUS	46
3.4 Convergencia de la fase de optimización	49
4 Algoritmo Distribuido para el CIF Maximal en Grafos Outerplanares Geométricos	61
4.1 Introducción	61
4.2 Preliminares	67
4.3 Algoritmo CIF-OUTERPLANAR	71
4.3.1 Fase de exploración	72

	Página
4.3.2 Fase de coloreado	75
4.3.3 Fase de coloreado extendida	81
4.4 Corrección del algoritmo	83
4.4.1 Corrección y análisis de la fase de exploración	83
4.4.2 Corrección y análisis de la fase de coloreado	92
Conclusiones	95
Referencias bibliográficas	99

Lista de Figuras

Figura		Página
1	Anillo con 3 vértices, cada uno con una variable binaria.	12
2	Diagrama de transición de estados posibles en el sistema del ejemplo. .	14
3	Ilustración de un conjunto independiente	18
4	Ilustración de un conjunto dominante	20
5	Ilustración de un conjunto independiente fuerte	22
6	Grafo con topología de anillo	29
7	Contaminación infinita con un líder ficticio	34
8	Ejemplo de la regla REINICIA	36
9	Ejemplo de la regla UNIR	36
10	Grafo cactus	47
11	Conjuntos de cerradura de tamaño uno y dos.	50
12	Grafo dual débil $T^*(G)$ del grafo outerplanar G	68
13	Diagrama auxiliar para la demostración del Lema 7.	84
14	Diagrama auxiliar para la demostración del Lema 9.	86
15	Diagrama auxiliar para la demostración del Lema 11	90
16	Diagrama auxiliar para la demostración del Lema 12.	91

Lista de Tablas

Tabla		Página
1	Representación de los estados globales del grafo.	13
2	Tabla de comparación de algoritmos auto-estabilizantes para un CIF en un grafo.	25

Capítulo 1. Introducción

Un sistema distribuido se puede ver como una colección de procesadores separados físicamente, conectados entre sí mediante una red de comunicación, y con el propósito de cooperar para alcanzar un fin. Cada procesador posee sus propios recursos de hardware y software que el usuario percibe como un solo sistema. Una de las propiedades de los sistemas distribuidos es la extensibilidad, la cual se refiere a incrementar el número de procesadores en el sistema distribuido sin que esto afecte su rendimiento y funcionalidad. Al incrementar el número de procesadores en el sistema, también aumenta la probabilidad de que ocurra una falla en alguno de ellos. Es por eso que se deben crear sistemas distribuidos tolerantes a fallas que garanticen el buen funcionamiento del sistema.

La auto-estabilización es una propiedad relacionada con la tolerancia a fallas en sistemas distribuidos, con un número cada vez mayor de aplicaciones. Dicha propiedad es muy útil cuando los procesadores en dichos sistemas distribuidos son propensos a la ocurrencia de fallas transitorias (aquellas fallas que alteran el estado de algún procesador pero no su comportamiento). Dijkstra fue el primero en introducir el concepto de auto-estabilización (Dijkstra, 1974), como aquel sistema en el que independientemente de la configuración inicial del sistema, se garantiza que converja a una configuración estable en un número finito de pasos. Existen varios algoritmos auto-estabilizantes para problemas relacionados con teoría de grafos.

Esta tesis se enfoca principalmente a aquellos problemas de teoría de grafos relacionados con la asignación de recursos, exclusión mutua, entre otros. En general son problemas en los que se requiere encontrar un conjunto independiente. Un *conjunto independiente* I en un grafo G , es un conjunto de vértices, tal que cualquier par de

vértices de I , están separados por al menos dos aristas. Un *conjunto independiente fuerte* o CIF es un conjunto independiente con más restricciones; en particular, un *CIF* (Haynes *et al.*, 2004), (Hochbaum y Shmoys, 1985), (Cameron y Edmonds, 2005) es un conjunto de vértices, S , tal que la longitud de la trayectoria más corta entre cualquier par de vértices de S es al menos tres. De forma más general, un conjunto *k-packing*, L , es un conjunto de vértices tal que la longitud de la trayectoria más corta entre cualquier par de vértices en L es al menos $k+1$ (en este sentido, el conjunto independiente también se conoce como el conjunto 1-packing). Butenko (2003) describe aplicaciones para los conjuntos independientes en diversas áreas, tales como tratamiento de información, transmisión de señales, teoría de clasificación, economía, calendarización, e ingeniería biomédica.

En general, el problema del CIF (también conocido como 2-packing) consiste en determinar el conjunto S de cardinalidad máxima de vértices, tales que la distancia entre cualquier par de vértices de S es al menos 3. Este problema pertenece a la clase NP-difícil (Hochbaum y Shmoys, 1985). Un problema más simple es encontrar el CIF maximal. Un CIF S es maximal, si no existe un conjunto independiente fuerte S' tal que $S \subset S'$.

Encontrar un CIF es importante y es la continuación del estudio de conjuntos independientes. Una aplicación del problema del CIF se da en la asignación de frecuencias (Hale, 1980), en donde se requiere evitar la interferencia entre canales. El algoritmo que resuelve el CIF es un componente fundamental en algoritmos que requieren garantizar la exclusión mutua entre vértices con vecinos traslapados, como es el caso del trabajo descrito en (Gairing *et al.*, 2003). Se hace notar que aunque algunos problemas para grafos pueden resolverse fácilmente bajo el esquema secuencial utilizando algoritmos voraces, estos mismos problemas requieren una estrategia mucho

más ingeniosa en el caso de algoritmos auto-estabilizantes.

En el presente trabajo de tesis, se estudia la factibilidad de diseñar algoritmos distribuidos auto-estabilizantes para el problema del CIF que mejoren el tiempo de convergencia de los algoritmos existentes en la literatura. Entre las estrategias que se propusieron está la de resolver el problema del CIF en tiempo polinomial para algunos grafos. El primero que se escogió fue el anillo, ya que tiene una estructura simétrica y es simple. Además, se definió un algoritmo auto-estabilizante para encontrar un CIF maximal para ciertas topologías más generales que los árboles, *e.g.* el cactus. Finalmente se extiende el resultado anterior hacia otras topologías más generales, en especial los grafos outerplanares. Para este último caso se diseñó un algoritmo distribuido que calcula un CIF maximal en un grafo outerplanar geométrico. En un grafo geométrico, cada vértice conoce sus coordenadas geométricas y las de sus vecinos.

El objetivo principal de esta investigación es profundizar en el estudio del problema del CIF en el contexto de los algoritmos distribuidos y de los algoritmos auto-estabilizantes. Para alcanzar este objetivo se proponen los siguientes objetivos específicos.

1. Identificar aquellas topologías donde se puede simplificar el cálculo del CIF (maximal o máximo).
2. Definir la especificación global del sistema, que permita identificar si el CIF en una topología dada es legal, esto es, definir el predicado global que todos los vértices deben cumplir para decir que el sistema está en un estado válido.
3. Diseñar un algoritmo auto-estabilizante para el problema del CIF máximo en el anillo

4. Diseñar un algoritmo auto-estabilizante para el problema del CIF maximal en el grafo cactus.
5. Diseñar un algoritmo distribuido para el problema del CIF maximal en un grafo outerplanar geométrico.

A continuación, se presenta formalmente la auto-estabilización en el contexto de los sistemas distribuidos. Posteriormente se describen algunos conceptos relacionados con los conjuntos independientes.

1.1 Auto-estabilización en sistemas distribuidos

Existen principalmente dos modelos de comunicación en los sistemas distribuidos: el modelo de paso de mensajes y el modelo de memoria compartida (Attiya y Welch, 2004).

En el *modelo de paso de mensajes*, los procesadores se comunican a través del envío de mensajes a través de canales de comunicación donde cada canal proporciona una vía de comunicación entre un par de procesadores. El patrón de las conexiones proporcionada por los canales de comunicación describe la topología del sistema. La topología se representa generalmente mediante un grafo en el cual cada vértice representa a un procesador. Existe una arista entre dos procesadores, si y solo si existe un canal de comunicación entre dichos procesadores. El estado de un procesador P_i consiste de los valores actuales de los registros en dicho procesador y los mensajes de entrada hacia dicho procesador.

En el *modelo de memoria compartida* se supone la existencia de una memoria común que almacena registros compartidos entre los procesadores vecinos. En el modelo de memoria compartida, los procesadores se comunican mediante el uso de dichos registros

compartidos. Cada procesador puede escribir en un conjunto de registros y puede leer de un conjunto de registros (posiblemente distinto al primero). El estado de un procesador P_i consiste de los valores actuales de los registros locales en dicho procesador y de los registros compartidos $r_{i,j}$ entre un procesador P_i y un procesador P_j para $i \neq j$.

Una configuración del sistema distribuido se define por un vector $C = (q_0, q_1, \dots, q_{n-1})$ donde q_i es el estado del procesador P_i . Se modela un sistema distribuido como un grafo $G = (V, E)$ donde V es el conjunto de procesadores (vértices) y E representa al conjunto de canales de comunicación (aristas). Un algoritmo distribuido consiste de un programa local en cada procesador P_i que considera el estado actual de P_i y, después de realizar ciertas acciones en P_i , puede enviar mensajes (escribir en los registros compartidos para el modelo de memoria compartida) a sus procesadores vecinos.

En un instante dado, los procesadores ejecutan pasos atómicos. Un paso atómico (o simplemente paso) consiste de una operación interna y una operación de comunicación: un envío o recepción de mensajes en el modelo de paso de mensajes y una escritura o lectura en el modelo de memoria compartida. Un sistema distribuido es *asíncrono* si no existe una cota en cuanto al tiempo que le toma a un procesador realizar un paso. Es conveniente utilizar rondas para poder medir el número de pasos que ocurren en la ejecución de un algoritmo distribuido. La primera *ronda asíncrona* (Dolev, 2000) en una ejecución E es el prefijo más corto E' de E tal que cada procesador ejecute al menos un paso en E' . Sea E'' el sufijo de E que sigue a E' , $E = E'E''$. La segunda ronda de E es la primera ronda de E'' . En un sistema distribuido asíncrono, los algoritmos se orientan a eventos, *i.e.*, los procesadores no pueden acceder a un reloj global para decidir en que ronda se encuentran. Los mensajes enviados por un procesador v a un vecino u llegan en un tiempo finito, pero impredecible.

En contraste, un sistema distribuido es *síncrono* si cada paso de un procesador se

ejecuta de acuerdo a un “reloj global”. La ejecución se particiona en rondas. En cada ronda, cada procesador realiza al menos un paso. Más precisamente, cada procesador mantiene un reloj local, cuyas rondas (pulsos) deben satisfacer la siguiente propiedad: un mensaje enviado por un procesador v a un vecino u en la ronda r (en el caso del modelo de memoria compartida puede verse como una escritura en el registro $r_{v,u}$) debe arribar a u (debe leerlo u en el caso del modelo de memoria compartida) antes de la ronda $r + 1$ de u . Aunque este modelo no es muy realista para sistemas distribuidos prácticos, es muy conveniente para el diseño de algoritmos.

Existen dos medidas principales para medir el rendimiento de los algoritmos distribuidos. La *complejidad de comunicación*, que es el máximo número de mensajes totales enviados durante la ejecución de un algoritmo. La *complejidad temporal*, que es el máximo número de rondas o de movimientos requeridos para completar la ejecución del algoritmo.

En alguna ocasión Leslie Lamport dijo: “. . .un sistema distribuido es aquel sistema en el que la falla de un procesador del que no tenías idea de su existencia puede hacer que tu propio procesador falle. . .” Ghosh (2006), p.3. De allí la importancia de definir un mecanismo de recuperación de fallas en sistemas distribuidos. Esta tesis se enfoca a un mecanismo de fallas no enmascarables denominado auto-estabilización.

Se dice que un sistema es *auto-estabilizante*, si independientemente de su estado inicial, converge a un estado legal en un número finito de pasos. Por lo tanto, un sistema auto-estabilizante no requiere inicializarse y puede recuperarse de fallas transitorias (aquellas fallas que afectan el estado de los nodos, pero que no afectan el comportamiento del sistema) de forma automática. A continuación se describen algunos conceptos básicos relacionados con la auto-estabilización.

1.1.1 Conceptos básicos

El primero en definir formalmente los sistemas auto-estabilizantes fue Dijkstra (1974), como aquel sistema, en el que independientemente del estado en el que se encuentre, llegará a un estado legal en un número finito de pasos.

Un sistema auto-estabilizante está compuesto de varios procesadores, cada uno de los cuales tiene su propio *estado local*, es decir, el valor que tienen asignadas las variables de ese procesador. Schneider (1993) define el *estado global* del sistema como la unión de los estados locales de sus procesadores. El comportamiento de dichos sistemas lo define su conjunto de estados, la relación de transición entre ellos, y un criterio justo para la función de transición. De esta forma, Schneider define la auto-estabilización de un sistema S con respecto a una condición P , si satisface las siguientes dos propiedades:

- **Convergencia:** Independientemente del estado global en el que se encuentre S , se garantiza que se llegará a un estado global que satisface P , en un número finito de pasos.
- **Cerradura:** Una vez que el sistema S cumple con P , S seguirá cumpliendo P .

Los sistemas auto-estabilizantes tienen la propiedad de recuperarse de aquellas fallas que pueden cambiar el estado del sistema, pero no su comportamiento, a estas fallas se les conoce como fallas transitorias. Una *falla transitoria* (Schneider, 1993) puede cambiar el estado global del sistema al corromper el estado local de un procesador, o bien, al corromper los canales de comunicación en el modelo de paso de mensajes.

Para garantizar la recuperación de fallas transitorias, cada procesador ejecuta repetidamente una pieza de código. Cada código consiste de un conjunto de reglas de la forma: (etiqueta)[guardia]: <programa>; un *guardia* (Antonoiu y Srimani, 1997) es una expresión booleana de las variables que un procesador puede leer, es decir, sus propias

variables y las variables de sus vecinos. Si el guardia de una regla es verdadero, se dice que esa regla está habilitada. Cuando existe al menos una regla habilitada en un procesador, se dice que el procesador está *habilitado*. La ejecución de una regla en algún procesador, cambia el estado en el que se encuentra dicho procesador. Cuando existen varios procesadores habilitados, debe definirse alguna manera de seleccionar cuál de éstos debe ejecutarse. Cuando Dijkstra (1974) definió la auto-estabilización, él utilizó en su algoritmo un esquema de calendarización central, es decir, solo una regla podía ejecutarse a la vez; sin embargo, hoy en día existen diferentes esquemas de calendarización. A continuación se describen los principales.

Calendarizadores

Dijkstra (1974) introdujo el concepto del *calendarizador central*. Este calendarizador selecciona sólo uno de los vértices habilitados. Este esquema tuvo serias críticas debido a que no representa el comportamiento real de un sistema distribuido.

Shukla *et al.* (1994) hacen la distinción de tres diferentes esquemas de calendarización: el *calendarizador central*, el cual lo definen igual que Dijkstra; el *máximo paralelismo*, también conocido como calendarizador distribuido síncrono, en el que todos los vértices habilitados deben ejecutarse; y el *paralelismo restringido*, también conocido como distribuido, en el que cuando existen al menos dos vértices privilegiados, sólo un subconjunto propio de éstos puede ejecutarse. Se dice que el calendarizador es *injusto* (adversario) si la única forma en que seleccione un vértice dado es que sea el único vértice privilegiado.

Cabe mencionar que no todos los algoritmos auto-estabilizantes soportan todos los esquemas; por ejemplo, hay algoritmos auto-estabilizantes para el calendarizador central, pero que no son auto-estabilizantes para otros calendarizadores. A continuación se

describe el modelo de computación que se utiliza en los algoritmos auto-estabilizantes presentados en el presente documento.

Modelo de computación

Un sistema distribuido se puede modelar por medio de un grafo $G = (V, E)$, en donde cada vértice $v \in V$ representa un procesador. Cada procesador puede comunicarse con un subconjunto de procesadores a los cuales se les denomina *vecinos* y la comunicación entre cada par de vecinos se representa mediante una arista $e \in E$. Para el modelo de computación se utiliza el término vértice en lugar del término procesador. El *vecindario abierto* $N(v) \in V$ de un vértice v es el subconjunto de vecinos de v , mientras que el vecindario cerrado de un vértice v , $N[v]$, es igual a $\{N(v) \cup v\}$.

La comunicación entre los procesadores (vértices) vecinos se realiza a través del paso de mensajes o de memoria compartida (Dolev, 2000). El modelo más utilizado es el modelo de paso de mensajes. Los algoritmos auto-estabilizantes de los Capítulos II y III utilizan el modelo de memoria compartida. El algoritmo distribuido presentado en el Capítulo IV utiliza el modelo de paso de mensajes.

La principal dificultad de los sistemas auto-estabilizantes consiste en demostrar las propiedades de convergencia y cerradura. Analizar el tiempo de convergencia de un algoritmo auto-estabilizante incluye las mismas dificultades para analizar el tiempo de ejecución de los algoritmos distribuidos.

Análisis de complejidad temporal de algoritmos auto-estabilizantes

Los algoritmos auto-estabilizantes al ser algoritmos distribuidos, incrementan la dificultad de analizar la complejidad de los mismos. Para medir la complejidad de los algoritmos auto-estabilizantes, se debe considerar el tiempo de convergencia, es decir,

el número de pasos que el algoritmo requiere para llegar a un estado estable. Demostrar que un algoritmo auto-estabilizante converge es un reto interesante. Existen algunas técnicas para demostrar la convergencia de un algoritmo auto-estabilizante.

Técnicas de demostración

A continuación se describen algunas de las principales técnicas para demostrar la convergencia de un algoritmo auto-estabilizante. Las técnicas de demostración dan una idea de la dificultad de diseñar un algoritmo auto-estabilizante.

- **Funciones variantes:** Las funciones variantes son la técnica más utilizada para demostrar la convergencia en los algoritmos auto-estabilizantes; esta técnica la propuso (Kessels, 1988). La idea es definir una función sobre el conjunto de estados posibles y demostrar que la función decrece monotónicamente cada vez que algún procesador ejecuta un paso del algoritmo. Una vez que la función alcanza un umbral, se dice que el sistema está en un estado estable.
- **Escaleras de convergencia:** Las escaleras de convergencia (Dolev, 2000) se basan en que dadas las configuraciones A_1, A_2, \dots, A_k , donde $k > 1$ y $1 \leq i < k$, A_{i+1} es un refinamiento de la configuración A_i . La idea es que la configuración A_{i+1} se cumple solo después de que todas las configuraciones A_1, \dots, A_i se hayan cumplido, de tal forma que cuando se cumple A_k quiere decir que se ha llegado a un estado estable.
- **Acoplamiento:** La técnica de acoplamiento descrita en (Fribourg *et al.*, 2002) se basa en que dado un algoritmo auto-estabilizante A , existe una función potencial ρ que mide la distancia “vertical” de cualquier estado arbitrario al conjunto

de estados legales λ , de tal forma que la distancia entre ambos estados decrece monotónicamente en cada paso de A .

- **Otras técnicas:** Existen otras técnicas para demostrar la convergencia de un sistema auto-estabilizante. Por ejemplo, en (Theel, 2001) se utiliza la teoría de control para demostrar que un algoritmo converge; por su parte Berard *et al.* (2000) proponen utilizar lógicas de primer orden para demostrar la convergencia de los sistemas auto-estabilizantes.

Convenciones de pseudocódigo

En un algoritmo auto-estabilizante, el segmento de código que ejecuta cada vértice consiste de un conjunto de reglas que se aplican a un vértice como se muestra a continuación.

- $regla_1$
- ...
- $regla_n$

La habilitación de una regla en un vértice, implica la ejecución de ciertas acciones para asignar nuevos valores a las variables del vértice. En el siguiente apartado se describe un algoritmo auto-estabilizante para encontrar un líder. Este algoritmo es muy sencillo pero ejemplifica claramente los conceptos definidos hasta ahora.

Ejemplo

Sea $G = (V, E)$ un anillo no dirigido de 3 vértices en donde cada vértice tiene una variable binaria denominada *líder* (como se ilustra en la Figura 1). El valor dentro de

cada vértice representa al identificador de dicho vértice. El objetivo del algoritmo es que todos los vértices excepto uno de ellos, tenga su variable *líder* en 0 y el vértice que tenga 1 se elige como líder. En la Tabla 1 se pueden observar los ocho posibles estados globales. De los ocho estados, tres de ellos son estados estables (los marcados en negritas).

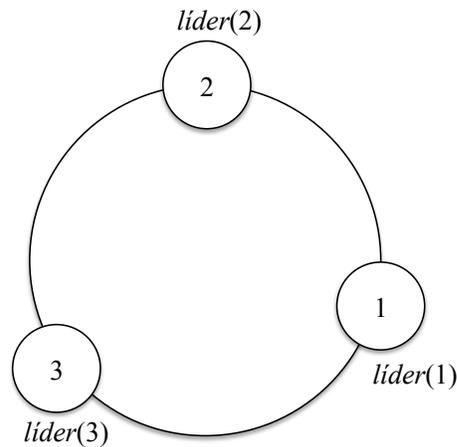


Figura 1. Anillo con 3 vértices, cada uno con una variable binaria.

El algoritmo auto-estabilizante debe ser capaz de, a partir de cualquier configuración inicial del grafo, llegar a una configuración legal en un número finito de pasos. El algoritmo consiste de dos reglas que se aplican a cada uno de los vértices del anillo. Cuando no hay ninguna regla habilitada, tampoco hay vértices habilitados (y viceversa). Para este ejemplo, cuando esto ocurre, quiere decir que el sistema está en un estado estable.

- *regla*₁: Si $\text{líder}(v) = 0 \wedge \forall u \in N(v) \text{líder}(u) = 0$
 \rightarrow hacer $\text{líder}(v) = 1$

Tabla 1. Representación de los estados globales del grafo. Los estados globales válidos se representan en negritas

$líder(1)$	$líder(2)$	$líder(3)$
0	0	0
0	0	1
0	1	0
1	0	0
1	1	0
1	0	1
0	1	1
1	1	1

- *regla₂*: Si $líder(v) = 1 \wedge \exists u \in N(v)$ tal que $líder(u) = 1$
 \rightarrow hacer $líder(v) = 0$

La primera regla dice que si un vértice tiene el valor de su variable líder igual a cero y todos sus vecinos tienen el valor de la variable líder en cero, entonces este vértice debe cambiar el valor de su variable líder a uno. La segunda regla indica que si existen dos o más vértices que tengan el valor de la variable líder en uno, al menos uno de ellos (el de menor identificador por el uso del calendarizador propuesto) debe cambiar el valor de su variable líder a cero. Este algoritmo considera un esquema de calendarizador central en el cual, si existen más de un procesador habilitado, el calendarizador escoge el procesador con menor índice. La Figura 2 presenta un diagrama de transición de los estados posibles del sistema. Existen 8 estados posibles, de los cuales tres son estados estables. Note que la trayectoria más larga, de cualquier estado al estado estable, es dos.

Aquí no se presenta una demostración analítica, pero del ejemplo es claro que no importa en estado en que inicie el sistema, éste siempre llegará a un estado estable.

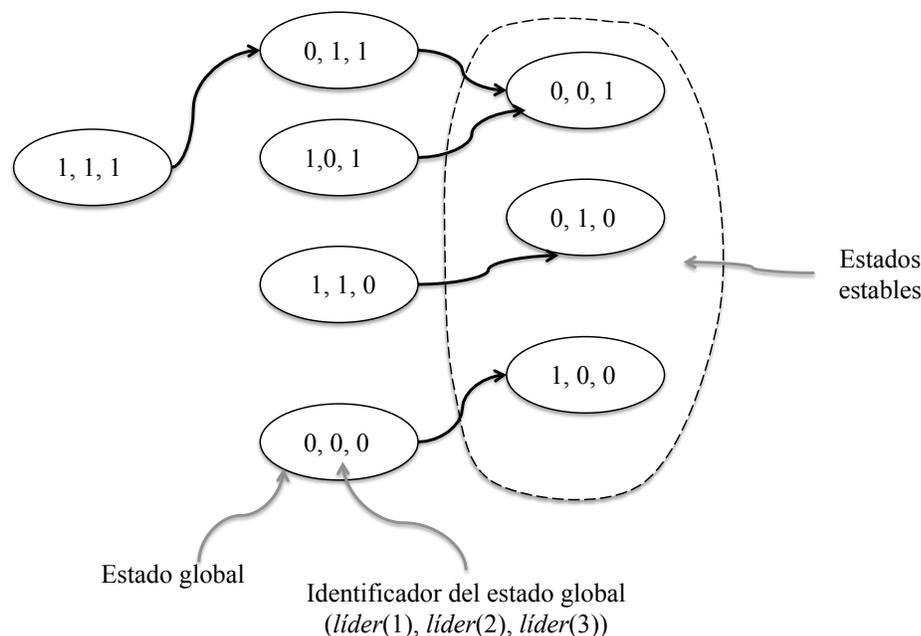


Figura 2. Diagrama de transición de estados posibles en el sistema del ejemplo.

Áreas de aplicación

La auto-estabilización se ha aplicado a diversas áreas y en diversos problemas, algunos ejemplos son: exclusión mutua (Datta *et al.*, 2000), sistemas multiagentes (Funk y Zinnikus, 2002), asignación de recursos (Dolev y Yagel, 2005), sistemas operativos (Brukman *et al.*, 2007), entre otros.

Los algoritmos auto-estabilizantes que se presentan en este trabajo se enfocan principalmente al área de asignación de recursos. A continuación se describen los principales problemas que se analizan en esta tesis. El objetivo principal de esta investigación es profundizar en el estudio del problema del CIF en el contexto de los algoritmos distribuidos y de los algoritmos auto-estabilizantes.

Para alcanzar este objetivo se proponen los siguientes objetivos específicos.

1.2 Preliminares

Ahora se describen varios problemas relacionados con grafos en el contexto de los sistemas distribuidos. Principalmente se enfatizan en aquellos problemas de exclusión mutua y de asignación de recursos.

Muchos de los algoritmos presentados en esta tesis requieren de un vértice especial que coordine sus acciones. El problema de elección de líder, es un problema fundamental en cómputo distribuido. En el caso particular de los algoritmos aquí propuestos, el algoritmo de elección de líder es un componente fundamental en los algoritmos de esta tesis. A continuación se describe el problema de elección de líder.

1.2.1 Elección de líder

El problema de elección de líder consiste en que dado un grupo de procesadores (o vértices), éstos deben seleccionar uno de ellos como líder. La existencia de un líder puede simplificar la coordinación en un sistema distribuido. A continuación se describen los principales algoritmos distribuidos de elección de líder. Primero se describen los principales algoritmos distribuidos no auto-estabilizante para el problema de elección de líder.

Awerbuch (1987) presenta un algoritmo (no auto-estabilizante) para la elección de líder en una red asíncrona. Este algoritmo se basa en la construcción de un árbol de expansión mínimo para encontrar al líder. Awerbuch demuestra que se requieren al menos $\Omega(m + n \log n)$ mensajes y $\Omega(n)$ rondas en una red general, donde m y n representan el número de aristas y el número de vértices del grafo, respectivamente.

Dado que un algoritmo auto-estabilizante puede iniciar en un estado arbitrario, pueden existir vértices que identifican como líder un valor que no pertenece al valor de

un identificador válido de algún vértice del grafo. Cuando ocurre la situación anterior, se dice que estos vértices identifican como líder a un *líder ficticio*. En el contexto auto-estabilizante una de las principales dificultades que debe resolverse es la eliminación de líderes ficticios. Afek y Bremner (1997) proponen un algoritmo auto-estabilizante que en $O(n)$ rondas encuentra un líder global en una red distribuida dinámica, donde n es el tamaño del grafo.

La idea del algoritmo de Afek y Bremner se basa en construir árboles enraizados en líderes locales. El árbol T con el mejor líder (aquel con el menor identificador) recluta aquellos vértices que son vecinos a un vértice hoja de T ; es decir, dichos vecinos que en una ronda i no pertenecen a T , en la ronda $i + 1$ ya pertenecen a T .

Este algoritmo utiliza una técnica denominada *suplemento de energía*, en la que sólo los líderes legales (no ficticios) generan energía. La idea es que la ronda i un vértice x que colinda con un vértice hoja de un árbol T , recibe un mensaje de solicitud para añadirse al árbol T . El vértice x se añade a T , sólo si en la ronda $i + 1$, el vértice recibe un mensaje de confirmación para unirse a T , de otra forma x no se añade a T . Note que esta técnica es válida únicamente para el esquema de calendarización síncrono. Un calendarizador adversario podría evitar el envío del mensaje de confirmación en la ronda $i + 1$. Este algoritmo converge a una configuración estable bajo la suposición del calendarizador síncrono después de $O(n)$ rondas. Afek y Bremner (1997) no afirman que su algoritmo funcione bajo el esquema de calendarización injusto.

Por su parte, Datta *et al.* (2011) proponen un algoritmo auto-estabilizante que resuelve el problema de elección de líder en $O(n)$ rondas bajo cualquier calendarizador. La idea del algoritmo de Datta es similar a la de Afek y Bremner. La estrategia consiste en crear árboles de expansión enraizados en líderes locales. Al final sólo queda el árbol enraizado en el líder global.

Dado que el algoritmo puede iniciar en un estado arbitrario, existe la posibilidad de que el árbol T se encuentre enraizado a un líder ficticio. Datta denomina a estos árboles como *árboles falsos*. Para eliminar los árboles falsos, utiliza una técnica denominada *ondas de color*. A diferencia del esquema de suplemento de energía propuesto por Afek y Bremner, la técnica de Datta garantiza que el algoritmo converja bajo la suposición de un calendarizador adversario. La idea de esta técnica es utilizar una variable booleana color que impide, mediante una especie de bloqueo, que aquellos vértices que no tienen un padre válido en T puedan reclutar a sus vecinos.

Este algoritmo es silencioso, *i.e.* una vez que el sistema converge a una configuración estable, ninguna regla se habilita para su ejecución en algún vértice del grafo. Este algoritmo converge a una configuración estable bajo la suposición del calendarizador adversario después de $O(n)$ rondas.

Muchos problemas relacionados con grafos tienen que ver con la asignación de recursos, ruteo de mensajes, exclusión mutua, entre otros. Los conjuntos independientes se utilizan para resolver este tipo de problemas.

Algunos problemas relacionados con conjuntos independientes son el conjunto dominante, el coloreado de vértices y el conjunto independiente fuerte. Las versiones de optimización para estos problemas pertenecen a la clase NP-difícil cuando la entrada es un grafo general; sin embargo, se sabe que para ciertas topologías de grafos, algunos de estos problemas se pueden resolver en tiempo polinomial. Ahora se describen formalmente estos problemas y se presentan algunos algoritmos.

1.2.2 Conjuntos independientes

Dado un grafo $G = (V, E)$, se dice que un subconjunto $I \subseteq V$ es un conjunto independiente de G si para cualquier par de vértices $v_i, v_j \in I$ se cumple que v_i y v_j no

son adyacentes (Diestel, 2005). Además se dice que un conjunto independiente I es maximal si no existe un subconjunto independiente I' tal que $I \subset I'$. Es importante mencionar que el concepto de conjunto independiente maximal es diferente a un conjunto independiente de cardinalidad máxima (máximo). Un conjunto independiente es *máximo* cuando I es el conjunto independiente de mayor cardinalidad en G . Encontrar un conjunto independiente máximo es un problema NP-difícil. Un problema más sencillo es encontrar un conjunto independiente maximal. En la Figura 3a se ilustra un conjunto independiente máximo de tamaño 3 y en la Figura 3b un conjunto independiente maximal de tamaño 4. Los vértices que pertenecen al conjunto independiente se representan sombreados. Un conjunto independiente de cardinalidad máxima es siempre maximal, lo contrario, no necesariamente se cumple.

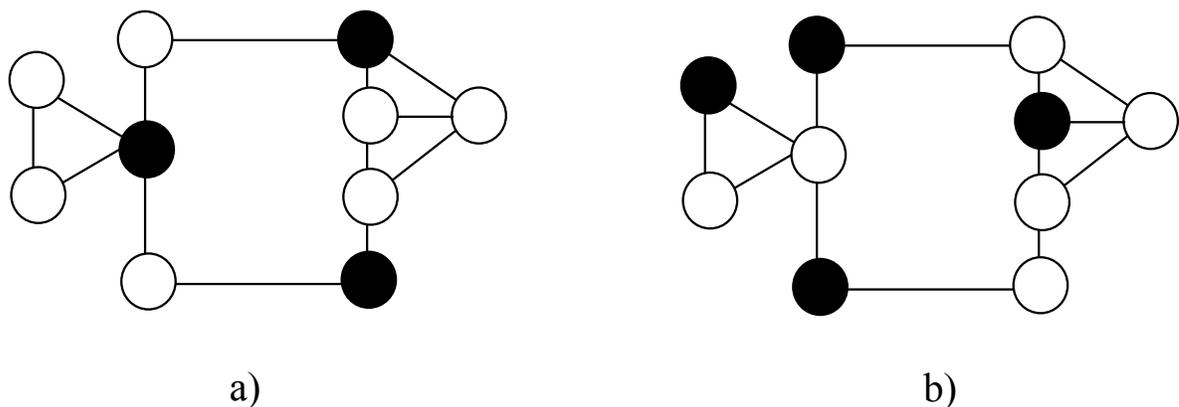


Figura 3. Los vértices en negro representan el conjunto independiente; a) un conjunto independiente maximal; b) un conjunto independiente máximo.

En el Capítulo IV se presentan los algoritmos distribuidos más representativos para encontrar un conjunto independiente maximal (MIS por sus siglas en inglés). Muchos de estos algoritmos pueden correr en orden sublineal pero requieren como entrada el tamaño n del grafo. El algoritmo distribuido que se presenta en el Capítulo IV no requiere conocer el número de vértices del grafo.

En el contexto auto-estabilizante, los algoritmos más rápidos para calcular el MIS

en grafos generales tardan al menos $\Omega(n)$ rondas. Hedetniemi *et al.* (2003) presentan un algoritmo auto-estabilizante que encuentra un MIS en grafos arbitrarios.

Este algoritmo utiliza una estrategia voraz para calcular el MIS. La estrategia consiste en incluir en el MIS todo aquel vértice que no tiene un vecino en el MIS. Si dos vértices vecinos pertenecen al MIS, se utiliza un criterio de desempate, para que uno de ellos deje de pertenecer al MIS. Este algoritmo supone un calendarizador central. Después de $O(n)$ rondas el algoritmo calcula el MIS, considerando un esquema de calendarizador central.

Shi *et al.* (2004) presentan un algoritmo auto-estabilizante para encontrar un conjunto independiente maximal en árboles anónimos y en cierto tipo de anillos anónimos en $O(n^2)$ rondas. La idea es tomar ventaja de las propiedades de un MIS en un árbol de tamaño n para diseñar el algoritmo. La idea del algoritmo consiste en obtener el MIS más grande posible. Este algoritmo converge a un MIS en $O(n^2)$ rondas, considerando un calendarizador adversario.

Por su parte, Turau (2007) describe un algoritmo auto-estabilizante considerando el calendarizador es adversario, y garantiza que, en $O(n)$ rondas, encuentra un conjunto independiente maximal. El algoritmo que propone Turau utiliza tres estados: *Entra*, *Sale* y *Espera*. El estado *Entra* indica que el vértice es parte del MIS. El estado *Sale* indica que el vértice no es parte del MIS. El estado *Espera* indica que un vértice puede cambiar al estado *Entra*, si es que dicho vértice no tiene un vecino con el mismo estado pero con identificador menor.

Los conjuntos dominantes se relacionan con los conjuntos independientes y se describen en la siguiente sección.

1.2.3 Conjuntos dominantes

Dado un grafo $G = (V, E)$, donde $|V| = n$, un subconjunto $D \subset V$ es un conjunto dominante, si cada vértice que no está en D es adyacente al menos a un vértice que está en D , en otras palabras $|D \cap N[v]| \geq 1$ para todo vértice v de G . Se dice que un conjunto dominante D es minimal, si no existe un conjunto dominante D' tal que $D' \subset D$. El problema de encontrar el conjunto dominante minimal es distinto al problema de encontrar el conjunto dominante mínimo (aquel de menor cardinalidad). La Figura 4a muestra un conjunto dominante minimal de tamaño 5 y la Figura 4b muestra un conjunto dominante de cardinalidad mínima de tamaño 3.

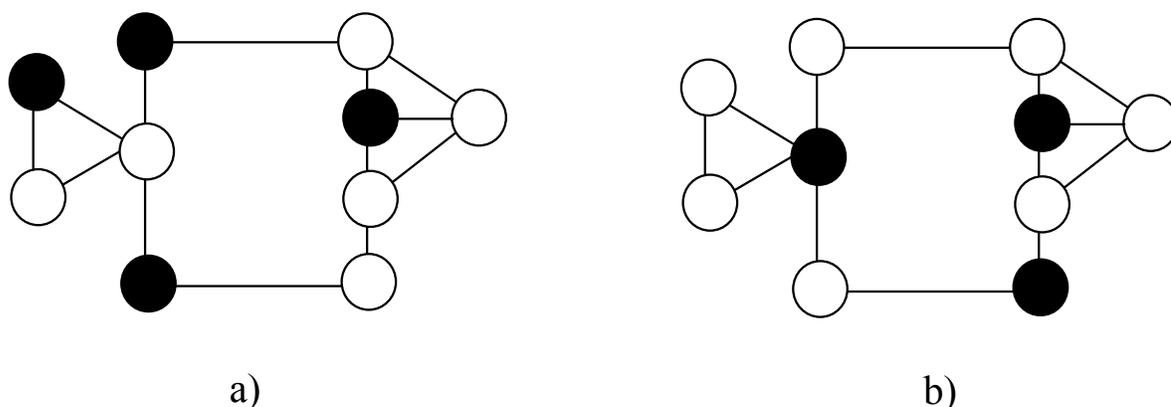


Figura 4. Los vértices en negro representan el conjunto dominante; a) un conjunto dominante minimal; b) un conjunto dominante mínimo.

El problema de encontrar un CIF está relacionado con el problema de encontrar un conjunto dominante. Haynes *et al.* (1998) demostraron que el problema de encontrar un conjunto dominante mínimo es el dual, en el sentido de programación lineal, al problema de encontrar un CIF máximo. Existen algunos algoritmos auto-estabilizantes que tratan el problema de los conjuntos dominantes.

Hedetniemi *et al.* (2003) presentan un algoritmo que en $O(n^2)$ rondas encuentra un conjunto dominante minimal para un grafo general; en ese mismo trabajo se presenta

un algoritmo que encuentra un conjunto independiente maximal en $O(n)$ rondas para un grafo general. Por otra parte, Jain y Gupta (2005) describen un algoritmo auto-estabilizante que en $O(n^2)$ rondas encuentra un conjunto dominante conectado; es decir, un conjunto dominante en el que el subgrafo generado por los elementos del conjunto dominante es un grafo conectado.

Al conjunto independiente en el cual la distancia entre dos vértices pertenecientes a dicho conjunto es mayor a dos, se le conoce como conjunto independiente fuerte o como 2-packing (Hochbaum y Shmoys, 1985). La versión más general de este problema es encontrar el conjunto de máxima cardinalidad. Este problema tiene aplicaciones en varias áreas tales como asignación de recursos y exclusión mutua. En la siguiente sección se describe formalmente este problema.

1.2.4 Conjunto independiente fuerte

Dado un grafo no dirigido $G = (V, E)$, un conjunto independiente fuerte o CIF es aquel conjunto independiente $S \subset V$ tal que para cualquier par de vértices v_i y $v_j \in S$, la distancia entre v_i y v_j es mayor a 2 (otra forma de verlo es que la trayectoria más corta que conecta el vértice v_i al v_j tiene al menos tres aristas). Si no existe otro conjunto S' tal que $S \subset S'$, se dice que el CIF es maximal. Es importante mencionar que el concepto de CIF maximal es diferente a un CIF de cardinalidad máxima. Un CIF es *máximo* cuando S es el conjunto independiente de mayor cardinalidad en G . Encontrar un CIF máximo es un problema NP-difícil (Hochbaum y Shmoys, 1985). En la Figura 5 los vértices sombreados representan un CIF. La Figura 5a muestra un CIF maximal y la Figura 5b muestra un CIF máximo.

Encontrar un CIF en un grafo es muy útil en aplicaciones que requieren exclusión mutua en los vértices a una distancia dos. Un ejemplo es el problema de asignación

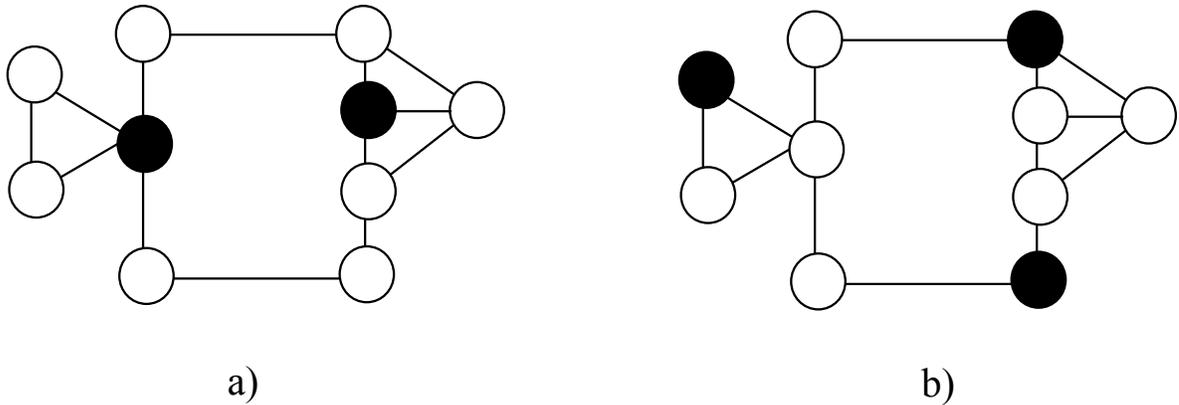


Figura 5. Los vértices en negro representan a un CIF; a) se ilustra un CIF maximal; b) se ilustra un CIF máximo.

de frecuencias (Hale, 1980), en el cual la asignación debe evitar la interferencia en canales compartidos. Encontrar un CIF maximal es muy útil también como una subrutina en algoritmos que resuelven problemas más complejos y que requieren asegurar la exclusión mutua de aquellos vértices con vecinos que se superponen (Gairing *et al.*, 2003; Hedetniemi *et al.*, 2012).

En general, se dice que un conjunto independiente L es un conjunto independiente a una distancia mayor a k (conocido también como k -packing) si para cualquier par de vértices $v_i, v_j \in L$ se cumple que $|v_i - v_j| > k$ (Haynes *et al.*, 1998).

Algunos autores proponen algoritmos auto-estabilizantes para encontrar un CIF. Gairing *et al.* (2004) describen un algoritmo auto-estabilizante que encuentra un CIF maximal en un grafo arbitrario en un número exponencial de movimientos. La estrategia de este algoritmo consiste en utilizar apuntadores para conocer información a una distancia dos. La estrategia de este algoritmo consiste en construir árboles de expansión cuyas raíces son los elementos del CIF. El algoritmo de Gairing *et al.* consiste de seis reglas. Un vértice x se incluye en el CIF cuando ninguno de sus vecinos pertenece al CIF y todos esos vecinos apuntan a x . Un vértice x que no se encuentra en el CIF, corrige su apuntador y apunta a su vecino y perteneciente al CIF. Un vértice x se remueve

del CIF, cuando x tiene un vecino de menor identificador que también está en el CIF o bien existe un vértice a una distancia dos de x que también se encuentra en el CIF. Este algoritmo considera el calendarizador adversario y converge a un CIF maximal en un número exponencial de movimientos.

Manne y Mjelde (2006) presentan un algoritmo auto-estabilizante que permite encontrar un CIF maximal con la misma complejidad temporal que el algoritmo propuesto por Gairing *et al.* (2004), pero reduciendo la complejidad espacial. Recientemente, Shi (2012) presenta un algoritmo auto-estabilizante para calcular un CIF maximal en un grafo arbitrario. El algoritmo de Shi se basa en el algoritmo de Gairing *et al.*. Cada vértice puede tener uno de tres posibles valores: 0, 1 ó 2 que indican la distancia a la que se encuentra un vértice en el CIF. La idea del algoritmo consiste en particionar los vértices del grafo, de tal forma que todos los vértices pertenezcan a un árbol cuya raíz pertenece al CIF. Si un vértice x es una raíz y existe otra raíz y que está a una distancia menor de tres, entonces x deja el árbol si el vértice y tiene menor identificador que x . La diferencia con el algoritmo de Gairing *et al.*, es que el algoritmo de Shi considera el valor de los identificadores como un criterio de desempate. Este algoritmo converge a un CIF maximal para un grafo arbitrario en $O(nm)$ rondas suponiendo el calendarizador adversario, donde m es el número de aristas. Converge a $O(n^2)$ rondas considerando el calendarizador síncrono.

Turau (2012) presenta una metodología que permite consultar información a una distancia dos en un sistema auto-estabilizante. Utilizando dicha metodología es posible diseñar un algoritmo auto-estabilizante para calcular un CIF maximal en un grafo arbitrario en $\Omega(n^2)$ rondas.

En cuanto al problema de encontrar el CIF máximo, Mjelde (2004) utiliza programación dinámica para diseñar un algoritmo auto-estabilizante, que lo resuelve en $O(n^3)$

en un árbol con identificadores únicos. El algoritmo de Mjelde consiste de dos fases. En la primera fase, cada vértice x calcula la cardinalidad del CIF máximo del subárbol enraizado en x . El algoritmo empieza a recopilar información desde las hojas del árbol hasta la raíz. Al finalizar la primera fase, la raíz del árbol tiene información requerida para iniciar la segunda fase. Dicha fase inicia desde la raíz hasta las hojas del árbol. Se utiliza la información obtenida durante la primera fase para determinar aquellos vértices que pertenecen al CIF máximo.

La Tabla 2 resume algunos algoritmos auto-estabilizantes para encontrar un CIF y se incluyen los diseñados en esta tesis.

En esta tesis se presentan varios algoritmos para calcular el CIF maximal en ciertas topologías. En el Capítulo III se presenta un algoritmo auto-estabilizante para encontrar el CIF maximal en un grafo cactus en $O(diam)$ rondas, donde $diam$ es el diámetro del cactus. Dicho algoritmo encuentra un CIF máximo cuando el cactus es un anillo. En el Capítulo IV se presenta un algoritmo distribuido que encuentra un CIF maximal en un grafo outerplanar geométrico.

El resto de esta tesis se divide como sigue. En el Capítulo II se presenta un algoritmo auto-estabilizante para la elección de líder. Dicho algoritmo es óptimo en tiempo. En el Capítulo III se presenta un algoritmo auto-estabilizante que encuentra un CIF máximo en un anillo y un CIF maximal cuando el grafo de entrada es un cactus. El Capítulo IV presenta un algoritmo distribuido que encuentra un CIF maximal en un grafo outerplanar geométrico. Finalmente, en el Capítulo V se presentan las conclusiones y el trabajo futuro.

Tabla 2. Tabla de comparación de algoritmos auto-estabilizantes para un CIF en un grafo.

Autor	Tipo	Topología	Complejidad Temporal
Gairing <i>et al.</i> (2004)	Maximal	Grafo general	Número exponencial de movimientos
Mjelde (2004)	Cardinalidad máxima	Árbol en-raizado	$O(n^3)$ rondas
Manne y Mjelde (2006)	Maximal	Grafo general	Número exponencial de movimientos. Óptimo espacialmente
Trejo-Sánchez y Fernández-Zepeda (2012)	Máximo	Anillo	$O(diam)$ rondas
Trejo-Sánchez y Fernández-Zepeda (2012)	Maximal	Cactus	$O(diam)$ rondas
Trejo-Sánchez y Fernández-Zepeda (2014)	Maximal no auto-estabilizante	Outerplanar geométrico	$O(n)$ rondas

Capítulo 2. Algoritmo auto-estabilizante para elección de líder

En este capítulo se presenta un algoritmo auto-estabilizante para la elección de líder. Aunque para describir este algoritmo de elección de líder se considera principalmente el caso del anillo, este algoritmo funciona para un grafo arbitrario. Un anillo es un grafo regular simétrico que tiene solamente dos caras y en el que cada vértice tiene grado dos. El anillo tiene importantes aplicaciones en redes de telecomunicaciones, asignación de recursos, entre otros. El tiempo de ejecución de este algoritmo es proporcional al diámetro del grafo.

A pesar de que la idea del algoritmo es muy simple, fue muy útil como una fase de aprendizaje de la tesis. Muchas de las dificultades que se encontraron durante el diseño de este algoritmo, también se presentan en los algoritmos del siguiente capítulo.

2.1 Introducción

Este capítulo describe un algoritmo para elección de líder. En dicho algoritmo, un subproblema fundamental es la eliminación de líderes ficticios.

Aunque existen algoritmos auto-estabilizantes eficientes para elección de líder, el diseño de este algoritmo permitió comprender algunos conceptos fundamentales en el contexto auto-estabilizante. Además el algoritmo diseñado es óptimo.

La idea del algoritmo es ir creando árboles de expansión iniciando desde un líder local que representa la raíz de dicho árbol. Cuando dos árboles colisionan aquel cuya raíz tiene el menor identificador absorbe al otro. Después de $O(diam)$ rondas, donde $diam$ es el diámetro del grafo, solo queda un árbol cuya raíz es el líder del grafo. Una particularidad del algoritmo es el de identificar aquellos vértices que se encuentran en

un “cierre de cadena”. El concepto de “cierre de cadena” es indispensable para la ejecución de los algoritmos del siguiente capítulo.

La Sección 2.2 describe el trabajo previo. La Sección 2.3 describe una especificación del modelo. La Sección 2.4 describe el algoritmo. La Sección 2.5 presenta la demostración de que el algoritmo es correcto y la complejidad temporal.

2.2 Trabajo previo

Dentro del contexto auto-estabilizante, existen diversos algoritmos auto-estabilizantes para la elección de líder en grafos con identificadores únicos. Arora y Gouda (1994) proponen un algoritmo auto-estabilizante para la elección de líder en un grafo. Ellos utilizan el modelo de memoria compartida. Su algoritmo tiene una complejidad temporal de $O(n)$ rondas y utiliza $O(\log n)$ espacio en memoria por procesador, donde n es el número de vértices en el grafo. El algoritmo de Arora y Gouda (1994) consiste de tres fases. En la primera fase, se encuentra al líder. Este líder es la raíz de un árbol de expansión que construyen en la segunda fase. Finalmente, la tercera fase, envía un mensaje de difusión que deja el grafo en un estado inicial para todos los vértices.

Por su parte Dolev y Herman (1995) presentan un algoritmo auto-estabilizante para elección de líder. Ellos utilizan el modelo de memoria compartida y el tiempo de convergencia de su algoritmo es $O(diam)$ rondas, donde $diam$ es el diámetro de la red, pero utiliza $O(n \log n)$ espacio de memoria por procesador. La principal ventaja que ofrece el algoritmo de Dolev y Herman (1995) es que una vez que el sistema está estable, en caso de que la topología cambie, el sistema converge nuevamente a una configuración estable en $O(1)$ rondas adicionales.

Un algoritmo que sigue una idea muy similar a la que se propone en este capítulo

es el de Afek y Bremner (1997). Su idea es crear árboles de expansión enraizados en un líder local y sólo aquel árbol cuya raíz es el líder global, prevalece al final de la ejecución del algoritmo. Para la eliminación de líderes ficticios se utiliza un concepto denominado “suministro de energía”. Esta técnica garantiza que desaparezcan aquellos árboles cuya raíz no sea un identificador válido del grafo. Este algoritmo es correcto para el calendarizador distribuido síncrono, pero no es claro si converge bajo un esquema de calendarizador adversario. El algoritmo utiliza un modelo de memoria compartida, converge en $O(n)$ rondas y utiliza $O(\log n)$ espacio de memoria por procesador.

Recientemente, Datta *et al.* (2011) propusieron un algoritmo auto-estabilizante para elección de líder. La idea es similar al algoritmo de Afek y Bremner (1997), pero para la eliminación de líderes ficticios se utiliza un concepto denominado “ondas de color”. Esta técnica garantiza la desaparición de aquellos árboles cuya raíz es un líder ficticio. La técnica de “ondas de color” permite la convergencia del algoritmo bajo un esquema de calendarización adversario. El algoritmo utiliza un modelo de memoria compartida, el tiempo de convergencia es $O(n)$ rondas, y utiliza $O(\log n)$ espacio de memoria por procesador.

El algoritmo que se propone en este capítulo utiliza un modelo de memoria compartida, cuyo tiempo de convergencia es $O(diam)$ rondas y cada procesador utiliza $O(\log n)$ espacio en memoria. Dicho algoritmo es óptimo bajo el calendarizador síncrono, pero no converge con el calendarizador adversario. Como se mencionó al inicio de este capítulo, el diseño de este algoritmo se pensó como una fase de entrenamiento. Este algoritmo se utiliza como un algoritmo auxiliar en los algoritmos del siguiente capítulo.

2.3 Especificación del modelo

En esta sección se generaliza el modelo que se describe en la introducción de este documento. Aunque el algoritmo es válido para topologías generales, para facilitar la descripción del algoritmo, se considera la topología de un anillo no dirigido $G = (V, E)$ en el cuál existen $|V| = n$ procesadores (vértices) y $|E| = m$ aristas. Un anillo tiene grado dos y el grafo contiene únicamente dos caras (la cara interna y la cara externa). La Figura 6 muestra un anillo de cinco vértices. Se puede ver que cada vértice de V tiene un identificador único y tiene grado dos.

Cualquier vértice $x \in V$ puede leer las variables en $N[x]$, pero sólo puede escribir en sus propias variables.

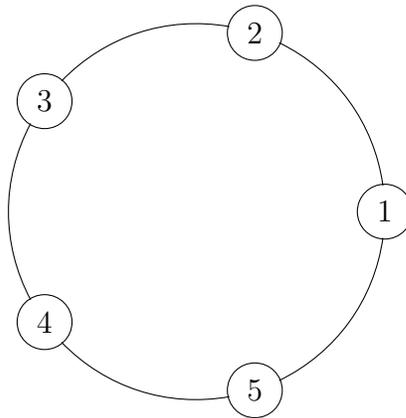


Figura 6. Grafo con topología de anillo con cinco vértices.

Cada vértice x tiene las siguientes variables:

- $x.ldr$ toma un valor entero no negativo. Esta variable almacena el identificador del vértice que x reconoce como el líder.
- $x.dst$ toma un valor entero no negativo. Esta variable almacena la distancia medida en aristas entre el vértice x y su líder.

- $x.padre$ toma un valor entero no negativo. Esta variable apunta a un elemento del vecindario cerrado $N[x]$ e indica la dirección hacia el líder.
- $x.fsn$ toma un valor Booleano. Esta variable se utiliza para generar un retraso
- $x.cls$ toma un valor entero no negativo y apunta a un elemento del vecindario cerrado $N[x]$. También indica si x está en un “cierre de la cadena”.
- $x.key$ es una tupla que consiste de las variables $x.ldr$ y $x.dst$, *i.e.*, $x.key = (x.ldr, x.dst)$. Dado un par de vértices $x, y \in V$, se dice que $y.key < x.key$ si $y.ldr < x.ldr$ o si $y.ldr = x.ldr$ y $y.dst < x.dst$.

El algoritmo que se propone considera el calendarizador distribuido síncrono. Es decir, si en la ronda i existen k vértices habilitados para ejecutar alguna regla, el calendarizador síncrono privilegia esos k vértices para su ejecución en dicha ronda.

2.4 Algoritmo de elección de líder

En esta sección se describe el algoritmo de elección de líder. Para facilitar la notación de las reglas del algoritmo, se definen las siguientes funciones:

- La función Booleana $safe_ldr(x)$ es verdadera cuando $x.ldr = x \wedge x.dst = 0 \wedge x.padre = x \wedge x.cls = x$, en caso contrario $safe_ldr(x)$ es falsa. Un vértice x es un *líder seguro* si la función $safe_ldr(x)$ es verdadera.
- La función Booleana $safe_par(x)$ es verdadera cuando $x.ldr < x$ y existe un vértice $y \in N(x)$ (el padre de x) tal que $x.ldr = y.ldr$, $x.dst = y.dst$ y $x.padre = y$; esta función es falsa en caso contrario.

- La función Booleana $safe(x)$ es verdadera cuando cualquiera de las funciones $safe_ldr(x)$ o $safe_par(x)$ es verdadera; esta función es falsa en caso contrario.
- La función $min_key(x) = \min \{y.key | y \in N[x]\}$. En caso de empate, la función $best_ldr(x)$ sirve como criterio de desempate.
- La función $bst_ldr(x) = \min \{y | y \in N[x] \wedge y.key = min_key(x)\}$.
- La función Booleana $sane(x)$ es verdadera si las siguientes condiciones son verdaderas:
 1. $safe(x)$ es verdadera.
 2. $x.padre = bst_ldr(x)$.
 3. $x.fsn = 0$.
 4. $y.ldr$ es el mismo para toda $y \in N[x]$.

Un vértice x es *cuerto* si $sane(x)$ es verdadero.

- La función $opt(x)$ regresa el “oponente” de x , en caso de que x tenga un oponente; en otro caso, $opt(x)$ regresa x . Un vértice x está en un *cierre de cadena* si las siguientes condiciones son verdaderas:
 1. $sane(x) = true$.
 2. El vértice x tiene un único vecino $y \in N(x)$ tal que $(y.dst = x.dst) \cup (y.dst = x.dst + 1 \wedge y.padre \neq x) \cup (y.dst = x.dst - 1 \wedge x.padre \neq y)$.

El vértice y se conoce como el *oponente* de x . En la Figura 6, dado que el vértice 1 es el líder, los vértices con identificador 3 y 4 están en un cierre de cadena. El vértice 3(4) es el oponente del vértice 4(3).

- La función $mst(x)$ es verdadera si las siguientes condiciones son verdaderas:
 1. $sane(x) = true$.
 2. $x.cls = opt(x)$.
 3. Para cada $y \in N(x)$, excepto para el padre y para el oponente de x , las variables $y.padre = x$ y $y.dst = x.dst + 1$.

Un vértice x es un *maestro* si $mst(x) = true$.

A continuación se presentan algunos conceptos básicos para facilitar la comprensión del algoritmo.

Una *cadena con líder* es el subgrafo inducido conectado maximal $G' = (V', E')$ del anillo G , tal que para cada vértice $x \in V$, la variable $x.ldr = r$ (donde el valor r no necesariamente es un elemento de V'); si el vértice $r \notin V'$, se dice que r es un líder ficticio (Datta *et al.* (2011) también utiliza el concepto de líder ficticio).

Dado un anillo G' con líder, se dice que una arista $e = (u, v)$, donde $e \in E'$, es una *arista verdadera* si $u.padre = v$ o si $v.padre = u$. Cualquier otra arista es una *pseudoarista*.

Dada una cadena G' con líder, un par de vértices $u, v \in V'$ están *líder-conectados*, si existe un camino desde u hasta v que consiste únicamente de aristas verdaderas.

Una cadena G' con líder es una *cadena segura*, si se cumplen las siguientes condiciones:

1. La función $safe_ldr(r)$ es verdadera únicamente para un vértice $r \in V'$ (el líder de la cadena segura).
2. La variable $x.ldr = r, \forall x \in V'$.

3. La función $safe_par(x)$ es verdadera para cada vértice $x \in V'$, excepto para el líder r .
4. Cada vértice $x \in \{V' - \{r\}\}$ es líder-conectado con el vértice r .

Dada una cadena G' con líder, una *cadena con líder ficticio* es un subgrafo inducido conectado $G'' = (V'', E'')$ de G' tal que cada vértice $x \in V''$ no es líder-conectado a $x.ldr$.

El algoritmo debe garantizar que después de determinado número de rondas ya no existan cadenas con líderes ficticios. Sea l el vértice con menor identificador en el anillo y sea $C_{l'}$ una cadena con líder ficticio l' . Si $l < l'$, la cadena con líder ficticio desaparecerá cuando colinde con la cadena con líder l . En caso de que $l' < l$, en la ronda i alguno de los vértices en $C_{l'}$ detecta que no es un vértice seguro y se reinicia. Sin embargo, a su vez en esa misma ronda, algún vértice que no se encuentre en $C_{l'}$ y colinda con dicha cadena se añadiría a $C_{l'}$ y este proceso podría continuar hasta el infinito. Este proceso infinito se ilustra en la Figura 7. En la Figura 7a, los vértices 10, 13 y 25 pertenecen a $C_{l'}$. En la Figura 7b, el vértice 10 ya no pertenece a $C_{l'}$, pero ya se agregó el vértice 9 a dicha cadena. En la Figura 7c, el vértice 13 ya no pertenece a $C_{l'}$, pero ya se agregó el vértice 30 a dicha cadena. En la Figura 7d el vértice 25 ya no pertenecé a $C_{l'}$, pero ya se agregó el vértice 14 a dicha cadena. Este proceso puede continuar indefinidamente y la cadena con líder ficticio $C_{l'}$ prevalece.

Si se conoce una cota superior del número de vértices en el anillo, es posible remover cualquier cadena con líder ficticio. Cuando dicha cadena tenga una distancia igual a la cota, se puede evitar que crezca más. La técnica que utiliza el algoritmo propuesto en este capítulo no requiere conocer el número de vértices del anillo. La idea es que el proceso de limpiar aquellos vértices de la cadena $C_{l'}$, sea dos veces más rápido que el

seguro maestro. Este algoritmo es fundamental para los algoritmos propuestos en el Capítulo III.

A continuación se describen las reglas del algoritmo ELECCION-LIDER

2.4.1 Algoritmo ELECCION-LIDER

El algoritmo ELECCION-LIDER consiste de cuatro reglas: REINICIA, UNIR, ESPERAR y CERRAR. A continuación se definen cada una de estas reglas.

Regla REINICIA. Se dice que un vértice x está *contaminado* si no es un vértice seguro. La regla REINICIA está habilitada para todos los vértices contaminados. La Figura 8 muestra el caso cuando la función $safe(x)$ es falsa para el vértice x , cuando $x = 8$. En la Figura 8a el vértice con identificador 8 tiene como líder al vértice con identificador 3 y una distancia a dicho líder igual a 0, por lo que la función $safe_ldr(8)$ es falsa. En la Figura 8b la función $safe_par(8)$ es falsa dado que no existe un vecino v de 8 que tenga como variable $v.ldr = 3$.

REINICIA: **if** $\neg safe(x)$ **then**
 {
 $x.ldr \leftarrow x$
 $x.dst \leftarrow 0$
 $x.padre \leftarrow x$
 $x.fsn \leftarrow 1$
 $x.cls \leftarrow x$
 }

Regla UNIR. Se dice que un vértice seguro x está disponible para concatenarse a su vecino y , cuando $y.key < p.key$ (p es el padre actual de x ; y es el nuevo padre de x después de la concatenación). La regla UNIR está habilitada para aquellos vértices que



Figura 8. Ejemplo de la aplicación de la regla REINICIA; a) La función $safe_ldr(8)$ es falsa para el vértice 8; b) La función $safe_par(8)$ es falsa para el vértice 8.

están disponibles para concatenarse. La Figura 9 muestra el caso en que el vértice 9, está disponible para concatenarse con el vértice 5.

UNIR: **if** ($safe(x) \wedge x.fsn = 0 \wedge \exists y \in N(x), bst_ldr(x) = y \wedge x.padre \neq y$
 $\wedge y.fsn = 0)$ **then**
 {
 $x.ldr \leftarrow y.ldr$
 $x.dst \leftarrow y.dst + 1$
 $x.padre \leftarrow y$
 $x.fsn \leftarrow 1$
 $x.cls \leftarrow x$
 }

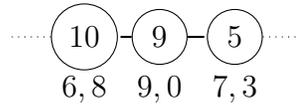


Figura 9. Vértice 9 está disponible para concatenarse con el vértice 5

Regla ESPERAR. Esta regla garantiza que el vértice seguro x , recientemente concatenado a y , no permita que sus vecinos se concatenen a dicho vértice en la siguiente ronda. Esta regla asegura que aquellas cadenas con líderes ficticios se “limpien” más rápido de lo que crecen.

ESPERAR: **if** ($safe(x) \wedge x.fsn = 1$) **then**
 {
 $x.fsn \leftarrow 0$
 }

Regla CERRAR. Esta regla actualiza la variable $x.cls$ cuando el vértice x es cuerdo; es decir, cuando $sane(x)$ es verdadero. Esta regla garantiza que los vértices en un cierre de cadena indiquen el valor de su oponente.

CERRAR: **if** ($sane(x) \wedge x.cls \neq opt(x)$) **then**
 {
 $x.cls \leftarrow opt(x)$
 }

Ahora se presenta la demostración de convergencia y el análisis de complejidad del algoritmo ELECCION-LIDER.

2.5 Convergencia del algoritmo ELECCION-LIDER

En esta sección se demuestra que el algoritmo ELECCION-LIDER converge a un estado legal en un número de rondas finito. Se supone que no hay fallas transitorias durante la ejecución de dicho algoritmo.

Para demostrar la convergencia del algoritmo ELECCION-LIDER, primero se demuestra que el número de líderes ficticios en el anillo G decrece gradualmente hasta llegar a cero en a lo más $O(diam)$ rondas, donde $diam$ es el diámetro del anillo.

En esta sección se utiliza la notación $x^\lambda.var$ para representar el valor de la variable var del vértice x en la ronda λ ; donde $x \in V$, la variable var pertenece al conjunto $\{key, fsn, cls, par\}$, y $\lambda \geq 0$ (la ronda 0 representa las condiciones iniciales de G).

Sea $G' = (V', E')$ una cadena con líder. La *altura* h' de G' es el valor más largo de entre todos los caminos más cortos entre algún vértice $y \in V'$ y el vértice $x \in V'$ con el menor valor en su variable $x.dst$. Para cualquier arista (i, j) de estos caminos, $i, j \in V'$.

Sea $F^\lambda = (V_{F^\lambda}, E_{F^\lambda})$ la cadena con mayor altura que tiene un líder ficticio en el anillo G en la ronda λ . Sea x_{minF^λ} el vértice $y \in V_{F^\lambda}$ tal que $y^\lambda.dst$ es el menor valor entre todos los vértices en V_{F^λ} .

Lema 1. La altura de la cadena $F^{\lambda-2}$ decrece en al menos una unidad después de dos rondas; *i.e.* $h_{F^\lambda} < h_{F^{\lambda-2}}$.

Demostración. Se demuestra por contradicción. Suponga que $h_{F^\lambda} \geq h_{F^{\lambda-2}}$ y que $y = x_{minF^{\lambda-2}}$.

Primero, se calcula el decremento de la altura producida por dos rondas consecutivas. Posteriormente, se muestra que el decremento total de la altura es mayor que el incremento total generado en el mismo número de rondas.

Por la regla REINICIA, y se remueve de $F^{\lambda-2}$ en la ronda $\lambda-1$ (note que en la ronda $\lambda-1$, $y \neq x_{minF^{\lambda-2}}$. Esta operación produce una nueva cadena con líder ficticio cuya altura $h_{F^{\lambda-1}} \leq h_{F^{\lambda-2}} - 1$ (note que y no puede concatenarse a alguna subcadena con líder, ficticio o no ficticio, en la ronda λ dado que $y^{\lambda-1}.fsn = 1$). Siguiendo el mismo argumento, en la ronda λ , el algoritmo produce una nueva cadena F^λ cuya altura es $h_{F^\lambda} \leq h_{F^{\lambda-1}} - 1 \leq h_{F^{\lambda-2}} - 2$. Por lo tanto, el decremento total de la altura después de dos rondas es al menos dos.

Por la regla UNIR, algún vértice que se concatene a $F^{\lambda-2}$ produce una nueva cadena $F^{\lambda-1}$ cuya altura es $h_{F^{\lambda-1}} - 1 \leq h_{F^{\lambda-2}} + 1$. Note que en la ronda λ , ningún vértice se concatena al vértice z dado que $z_{\lambda-1}.fsn = 1$, por lo tanto $h_{F^\lambda} \leq h_{F^{\lambda-1}} - 1$. De esta forma, el incremento total de la altura es a lo más uno. Por lo tanto, la altura decrementa al menos una unidad cada dos rondas, lo cual es una contradicción a la

hipótesis inicial de que $h_{F^\lambda} \geq h_{F^{\lambda-2}}$. \square

Corolario 1. Después de $O(diam)$ rondas, el anillo G no tiene líderes ficticios.

Demostración. Por el Lema 1, el algoritmo requiere lo más de $2h_{F^\lambda}$ rondas para reducir la altura de la cadena F^λ a cero. Dado que F^λ es la cadena de mayor altura, de entre todas las cadenas con líder ficticio, cualquier otra de dichas cadenas con líder ficticio también desaparece simultáneamente. Note que $h_F^\lambda \leq diam$ para cualquier λ . Por lo tanto, después de $O(diam)$ rondas, el anillo G no tiene líderes ficticios. \square

Corolario 2. Si no existen líderes ficticios en el anillo G , la función $safe(x)$ es verdadera $\forall x \in V$.

Demostración. La demostración se sigue directamente de la definición de $safe(x)$. \square

Lema 2. Si no existen líderes ficticios en el anillo G , el algoritmo ELECCION-LIDER genera el anillo global seguro $G^s = (V^s, E^s)$ después de $O(diam)$ rondas.

Demostración. Se demuestra por inducción en el diámetro, $diam$, del anillo. Sea r el identificador más pequeño de todos los vértices de V .

Caso base. Por el Corolario 2, la función $safe_ldr(r)$ es verdadera y $r \in V^s$. Entonces r ejecuta la regla ESPERAR en la siguiente ronda, permitiendo que se incorporen nuevos vértices a G^s . (Note que r nunca se elimina de G^s , dado que ninguna regla modifica $r.key$).

Hipótesis de inducción. Se supone que todos los vértices $y \in V$ tales que $dist(r, y)$ es a lo más d , están concatenadas a G^s después de $2d + 1$ rondas. Ninguno de los vértices y se remueven de G^s y están configurados para que otros vértices se concatenen con ellos.

Paso inductivo. Se demuestra que todos los vértices $x \in V$ a una distancia $d + 1$ de r , se concatenan con G^s en dos rondas adicionales. Estos vértices x no se remueven

de G^s y están configurados para permitir que otros vértices se concatenen a ellos. Sea un vértice x a una distancia $d + 1$ de r y supongase que $x \notin V^s$. Por el Corolario 2, la función $safte(x)$ es verdadera. Por la hipótesis de inducción, existe un vértice $y \in N(x)$ a una distancia d de r , tal que $best_ldr(x) = y$. Por lo tanto, en una ronda, la regla UNIR concatena x a G^s . En una ronda adicional, la regla ESPERAR configura la variable $x.fsn = 0$, permitiendo que cualquier vecino a una distancia $d + 2$ se concatene con x en la ronda siguiente. Note que la regla CERRAR es la única regla del algoritmo que puede privilegiar al vértice x ; esta regla no modifica $x.key$; por lo tanto, en dos rondas, todo vértice x a una distancia $d + 1$ se concatena con G^s . \square

Corolario 3. Sea G^s el anillo global seguro de G ; la función $sane(x)$ es verdadera para cada $x \in V^s$.

Demostración. Dado que el vértice x ejecuta la regla UNIR para concatenarse con G^s , y en la siguiente ronda ejecuta la regla ESPERAR, el valor de la variable $x.padre = best_ldr(x)$ y el de la variable $x.fsn = 0$. Además, $y.ldr = x.ldr$ para cada vecino $y \in N(x)$; por lo tanto x es cuerdo; *i.e.* $sane(x)$ es verdadero. \square

Corolario 4. Sea G^s el anillo global seguro G y sea x un vértice no maestro de G^s . En una ronda adicional, x se vuelve un vértice maestro.

Demostración. Dado que G^s es el anillo global seguro, algún vértice $x \in V^s$ cuya variable $x.cls$ es incorrecta, ejecuta la regla CERRAR durante el algoritmo ELECCION-LIDER. Esta regla actualiza la variable $x.cls$ con el valor que regresa la función $opt(x)$ en una ronda. Por lo tanto, todos los vértices en V^s son vértices maestros y G^s es ahora el anillo global seguro maestro G^m . \square

Ahora se demuestra la propiedad de cerradura y el análisis de complejidad del algoritmo propuesto en este capítulo.

Corolario 5. Sea $G^m = (V^m, E^m)$ el anillo global seguro maestro de G . Ninguna

regla del algoritmo ELECCION-LIDER privilegia algún vértice de V^m .

Demostración. Se sigue la demostración de las definiciones de las reglas en el algoritmo ELECCION-LIDER. \square

Teorema 1. Sea G un anillo arbitrario. G se convierte en un anillo global seguro maestro G^m en a lo más $O(\text{diam})$ rondas.

Demostración. La demostración es una consecuencia directa de los Lemas 1 y 2 y del Corolario 4. \square

En el Capítulo III se presentan dos algoritmos auto-estabilizantes para el problema del conjunto independiente fuerte en un anillo y en un cactus. Ambos algoritmos utilizan el algoritmo ELECCION-LIDER.

Capítulo 3. Algoritmos auto-estabilizantes para el CIF

En este capítulo se presentan dos algoritmos auto-estabilizantes para encontrar el conjunto independiente fuerte (CIF). Primero se presenta un algoritmo auto-estabilizante para encontrar el conjunto independiente fuerte (CIF) máximo en un anillo de n vértices. Posteriormente, se generaliza este resultado para encontrar un CIF maximal en un grafo cactus. Los grafos cactus tienen importantes aplicaciones en redes de telecomunicaciones, problemas de localización y biotecnología, entre otras. Hasta donde el autor del presente documento tiene conocimiento, los algoritmos presentados en este capítulo mejoran el trabajo previo para el CIF en estas topologías.

3.1 Introducción

Este capítulo describe dos algoritmos auto-estabilizantes para el CIF. El primero de ellos recibe como entrada un anillo de n vértices y regresa un CIF máximo. Posteriormente, se generaliza este resultado y se propone un algoritmo auto-estabilizante para encontrar un CIF maximal en un grafo cactus. Ambos algoritmos utilizan una fase de elección de líder en el grafo. Para dicha fase se considera el algoritmo ELECCION-LIDER del Capítulo II. Adicionalmente, para la construcción del CIF se utiliza una fase denominada *fase de optimización*.

La idea de la fase de optimización consiste en construir el CIF a partir del líder, considerando el recorrido definido por el árbol de expansión generado en la fase de elección de líder. Una particularidad de la fase de elección de líder es que identifica aquellos vértices que se encuentran en un cierre de cadena. Esta información es indispensable para la ejecución de la fase de optimización. De esta forma, este capítulo presenta dos

algoritmos auto-estabilizantes óptimos para encontrar el CIF máximo en un anillo y el CIF maximal en un cactus, respectivamente. Estos algoritmos convergen en $O(D_K)$ rondas, donde D_K es el diámetro del grafo G . Hasta donde el autor de este documento tiene conocimiento, estos algoritmos mejoran el rendimiento de los algoritmos previos presentados en la literatura, cuando éstos se ejecutan en un anillo o en un cactus. Los algoritmos descritos en este capítulo utilizan un calendarizador síncrono.

El resto de este capítulo se organiza de la siguiente manera. La Sección 3.2 describe el algoritmo CIF-ANILLO que encuentra el CIF máximo en un anillo. La Sección 3.3 describe el algoritmo CIF-CACTUS, que consiste en una generalización del CIF-ANILLO. Finalmente, la Sección 3.4 describe la demostración de que el algoritmo es correcto y se hace un análisis de la complejidad temporal.

3.2 Algoritmo CIF-ANILLO

Sea $G = (V, E)$ un anillo no dirigido, donde el tamaño del anillo es $|V| = n$. Se supone que cada vértice de V tiene un identificador único. Cada vértice $x \in V$ puede leer las variables en $N[x]$, pero sólo puede escribir en sus propias variables.

El algoritmo CIF-ANILLO consiste de dos fases. Primero, durante la fase de elección de líder, el algoritmo elimina los líderes ficticios del anillo. Esta fase regresa como salida el anillo global seguro maestro (ver Capítulo II). Posteriormente, la fase de optimización calcula un CIF máximo S en el anillo G . El Pseudocódigo 1 describe este algoritmo.

Pseudocódigo 1. Algoritmo 2-CIF-ANILLO(G)

Entrada: Un anillo no dirigido $G = (V, E)$.

Salida: Un CIF maximo $S \subseteq V$.

begin

$G^m \leftarrow$ fase ELECCION-LIDER(G)

$S \leftarrow$ fase OPTIMIZACION(G^m)

end

Para la fase de elección de líder, se utiliza el algoritmo ELECCION-LIDER del Capítulo

II. Para la fase de OPTIMIZACIÓN se utilizan las siguientes variables:

- $x.clr$ toma un valor Booleano. Se dice que esta variable toma el color rojo cuando x pertenece al CIF y el azul en caso contrario.
- $x.ptr$ toma un valor entero no negativo. Esta variable apunta a un elemento del vecindario cerrado $N[x]$.

Ahora se describen las reglas de la fase de optimización.

3.2.1 Fase de optimización

Algunas de las definiciones que se utilizan en esta sección, se describen en la Sección 3.4.

Sea G^m el anillo global seguro maestro. Después de ejecutar la fase de optimización en G^m , solo aquellos vértices $x \in V^m$ cuyas variables $x.clr = rojo$ pertenecen al CIF máximo.

La idea de la fase de optimización es la siguiente. Primeramente el líder se colorea como rojo indicando que pertenece al CIF. Luego, a partir del líder, los vértices que siguen al líder se colorean consecutivamente como *rojo, azul, azul, rojo, ...*, exceptuando aquellos que se encuentran en un cierre de cadena. Los vértices en un cierre de cadena, deben considerar el coloreado de su oponente para garantizar que se obtenga un CIF válido. De la línea 10 a la línea 19 del Pseudocódigo de la función AGREGA-ANILLO se consideran todos los casos posibles para el coloreado de dichos vértices.

La fase de optimización consiste de dos reglas: RAIZ y SEGUIR. Adicionalmente se utiliza la función AGREGA-ANILLO(x). Las reglas en esta fase solo privilegian vértices maestros y se definen a continuación.

Regla RAIZ. Esta regla corrige el color y el apuntador del líder r de G^m de cualquier vértice distinto del líder.

```

RAIZ:    if ( $safe\_ldr(x) \wedge mst(x) \wedge x.clr = azul \wedge x.ptr \neq x$ ) then
        {
             $x.clr \leftarrow rojo$ 
             $x.ptr \leftarrow x$ 
        }

```

Regla SEGUIR. Esta regla corrige los valores de $x.clr$ y de $x.ptr$.

```

SEGUIR:  if ( $\neg safe\_ldr(x) \wedge mst(x) \wedge AGREGA-ANILLO(x) \neq (x.clr, x.ptr)$ ) then
        {
             $(x.clr, x.ptr) \leftarrow AGREGA-ANILLO(x)$ 
        }

```

El Pseudocódigo 2 describe la función AGREGA-ANILLO(x) calcula los valores correctos de las variables $x.clr$ y $x.ptr$. Se incluyen comentarios en color azul.

Pseudocódigo 2. Función AGREGA-ANILLO(x)

Entrada: Un anillo no dirigido $G = (V, E)$.

Salida: Un CIF maximo $S \subseteq V$.

begin

1. sea $y \leftarrow x.padre$
// En caso de que el color del padre sea rojo
2. **If** ($y.clr = rojo$) **then**
3. $\{x.clr \leftarrow azul, x.ptr \leftarrow y\}$
// En caso de que el color del padre sea azul y no apunte a x
4. **if** ($y.clr = azul \wedge y.ptr \neq x$) **then**
5. Sea $z = \{N(x) \setminus \{y\}\}$
// x apunta a su hijo
6. $\{x.clr \leftarrow azul, x.ptr \leftarrow z\}$
// En caso de que el color del padre sea azul y apunte a x
7. **if** ($y.clr = azul \wedge y.ptr = x$) **then**

```

8.   if ( $x.cls = x$ ) then
9.     { $x.clr \leftarrow rojo, x.ptr \leftarrow x$ }
10.  else
11.    //  $z$  es el oponente de  $x$ 
12.    Sea  $z \leftarrow x.cls$ 
13.    //En caso de que  $z$  sea rojo y más cercano al líder
14.    //o bien tenga la misma distancia al líder pero que  $z < x$ 
15.    if ( $z.clr = rojo \wedge (z.dst < x.dst \vee (z.dst = x.dst \wedge z < x))$ ) then
16.      { $x.clr \leftarrow azul, x.ptr \leftarrow x$ }
17.    else
18.      //  $z$  es azul y apunta a su padre
19.      if ( $z.clr = azul \wedge z.ptr = z.padre$ ) then
20.        { $x.clr \leftarrow azul, x.ptr \leftarrow x$ }
21.      else
22.        { $x.clr \leftarrow rojo, x.ptr \leftarrow x$ }
23.    Return ( $x.clr, x.ptr$ ) end

```

La Sección 3.4 contiene la demostración de convergencia y análisis de complejidad del algoritmo. Ahora se presenta una extensión del algoritmo para el caso en el que el grafo de entrada es un cactus.

3.3 Algoritmo CIF-CACTUS

Un grafo $K = (V_K, E_K)$ es un cactus si cada arista de E_K pertenece a lo más a un ciclo (un par de ciclos arbitrarios del cactus tienen a lo más un vértice en común). La Figura 10 ilustra un grafo cactus. Los grafos cactus tienen importantes aplicaciones en redes de telecomunicaciones como se describe en Lan y Wang (2000) y Koontz (1980), problemas de localización como se menciona en Kariv y Hakimi (1979), problemas prácticos de redes de computadoras descritos en Zmazek y Žerovnik (2004), y en biotecnología como mencionan Paten *et al.* (2011). Recientemente, los cactus han llamado la atención de la comunidad científica al encontrarse que algunos problemas

que pertenecen a la clase de problemas NP-difícil para topologías arbitrarias, admiten soluciones en tiempo polinomial para los cactus, así lo mencionan Zmazek y Žerovnik (2004) y Ben-Moshe *et al.* (2005).

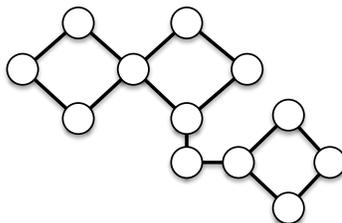


Figura 10. Grafo cactus

Sea $K = (V_K, E_K)$ un cactus no dirigido, donde el tamaño del cactus es $|V_K| = n$. Las suposiciones respecto a los identificadores y las capacidades de lectura y escritura de los vértices son las mismas que para el algoritmo CIF-ANILLO.

De forma similar al algoritmo CIF-ANILLO, el algoritmo CIF-CACTUS consiste de dos fases. Primero, durante la fase de elección de líder, el algoritmo elimina los líderes ficticios del cactus. Esta fase regresa como salida el cactus global seguro maestro (ver Capítulo II). Posteriormente, la fase de optimización calcula un CIF máximo S en el cactus K .

Para la fase de elección de líder, se utiliza el algoritmo ELECCION-LIDER descrito en el Capítulo II. Para la fase de optimización, se utilizan las mismas reglas que se utilizan en la fase de optimización del algoritmo CIF-ANILLO; sin embargo, debe redefinirse la función AGREGA-ANILLO(x). Esta función utiliza como rutina una nueva función llamada APUNTA(x). La diferencia radica en que aquel vértice azul x cuyo padre es azul debe seleccionar de entre más de un hijo a cuál de ellos lo propone como rojo. Cuando la función AGREGA(x) llama a la función APUNTA(x), esta última determina a cuál de los hijos de x (si es que tuviera hijos) le indica la posibilidad de ser rojo. Ahora

se describen estas nuevas funciones.

El Pseudocódigo 3 describe la función $\text{AGREGA-CACTUS}(x)$ calcula los valores correctos de las variables $x.clr$ y $x.ptr$ para todos los vértices del grafo. Los comentarios se ponen en color azul.

Pseudocódigo 3. Función $\text{AGREGA-CACTUS}(x)$

Entrada: Un cactus no dirigido $G = (V, E)$.

Salida: Un CIF máximo $S \subseteq V$.

begin

```

1.   sea  $y \leftarrow x$ 
      // En caso de que el color del padre sea rojo
2.   If ( $y.clr = rojo$ ) then
3.     { $x.clr \leftarrow azul, x.ptr \leftarrow y$ }
      // En caso de que el color del padre sea azul y no apunte a  $x$ 
4.   if ( $y.clr = azul \wedge y.ptr \neq x$ ) then
      //  $x$  apunta a su hijo
5.     { $x.clr \leftarrow azul, x.ptr \leftarrow \text{APUNTA}(x)$ }
6.   if ( $y.clr = azul \wedge y.ptr = x$ ) then
7.     if ( $x.cls = x$ ) then
8.       { $x.clr \leftarrow rojo, x.ptr \leftarrow x$ }
9.     else
      //  $z$  es el oponente de  $x$ 
10.    Sea  $z \leftarrow x.cls$ 
      //En caso de que  $z$  sea rojo y más cercano al líder
      //o bien tenga la misma distancia al líder pero que  $z < x$ 
11.    if ( $z.clr = rojo \wedge (z.dst < x.dst \vee (z.dst = x.dst \wedge z < x))$ ) then
12.      { $x.clr \leftarrow azul, x.ptr \leftarrow x$ }
13.    else
      // $z$  es azul y apunta a su padre
14.      if ( $z.clr = azul \wedge z.ptr = z.padre$ ) then
15.        { $x.clr \leftarrow azul, x.ptr \leftarrow \text{APUNTA}(x)$ }
16.      else
17.        { $x.clr \leftarrow rojo, x.ptr \leftarrow x$ }
18.    Return ( $x.clr, x.ptr$ ) end

```

El Pseudocódigo 4 describe la función $\text{APUNTA}(x)$ define un criterio claro para

escoger la configuración de $x.ptr$. (La notación $z!$ indica que existe únicamente un vértice z). Los comentarios se ponen de color azul.

Pseudocódigo 4. Función $APUNTA(x)$

Entrada: Un cactus no dirigido $G = (V, E)$ con el vecindario cerrado $N[x]$ de un vértice x .

Salida: Un vértice $y \in N[x]$.

```

begin
    //Si x no tiene hijos o bien el oponente de x es rojo
1.   if ( $\nexists z \in N(x), z.padre = x$ )  $\cup$  ( $\exists z \in N(x), opt(x) = z \wedge z.clr = rojo$ ) then
2.       {aux  $\leftarrow x$ }
3.   else
4.       if ( $\exists z! \in N(x), z.padre = x$ ) then
5.           {aux  $\leftarrow z$ }
6.       else
            //x apunta de entre sus hijos al de menor identificador
            //que no pertenece a un cierre de cadena. En caso de que todos
            //sus hijos pertenezcan a un cierre de cadena, selecciona al menor
7.           {aux  $\leftarrow \min(m \in N(x) | m.padre = x \wedge (m.cls = m \cup m.cls \in N(x)))$ }
8.   Return aux end

```

A continuación se presenta la demostración de convergencia y el análisis de complejidad del algoritmo CIF-CACTUS, la cual es válida para el CIF-ANILLO.

3.4 Convergencia de la fase de optimización

Dado el cactus global seguro maestro K^m , un vértice $x \in V_K^m$ es azul₁ si su variable $x.ptr = x.padre$; el vértice x es azul₂ en cualquier otro caso.

Un subconjunto $S \subseteq V_K^m$ es un CIF maximal si se cumplen las siguientes dos condiciones:

1. Si $x, y \in S$, $dist(x, y) \geq 3$

2. Si $x \notin S$, existe al menos un vértice $y \in S$ tal que distancia entre x y y es menor o igual a 2.

Sea r el identificador más pequeño de todos los vértices de V_K^m . Sea K_r^d el subgrafo inducido de K^m que contiene todos los vértices $x \in V_K^m$ cuya $dist(x, r) \leq d$. Sea S_r^d el conjunto de vértices rojos de K_r^d . Sea B^d el conjunto de vértices $x \in V_K^m$ cuya $dist(x, r) = d$.

Dado el subgrafo K_r^{d+1} de K^m , un *conjunto de cerradura* C^{d+1} es un subgrafo conectado maximal que consiste de aquellos vértices x a una distancia $d + 1$ de r tales que $x.cls \neq x$ y solamente se cumple una de las siguientes condiciones.

- El vértice x tiene un vecino y tal que $y \in C^{d+1}$.
- El vértice x tiene un vecino y tal que $y \in B^d$ y $y.cls \neq y$.

El vecindario en la frontera de cerradura NB^d de un conjunto de cerradura C^{d+1} consiste de aquellos vecinos de C^{d+1} en la frontera B^d ; note que $|NB^d| = 2$. Se denotan los dos vértices de NB^d como a y b . La Figura 11 muestra los conjuntos de cerradura de tamaño uno y dos.

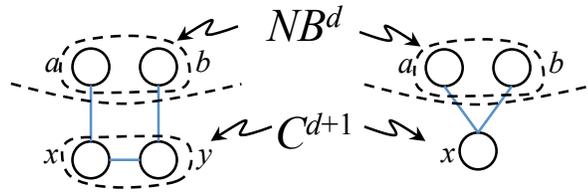


Figura 11. Conjuntos de cerradura de tamaño uno y dos.

Se divide la prueba de convergencia en tres partes. En la primera parte (Lema 3), se muestra que el algoritmo CIF-CACTUS incrementalmente genera un CIF basado en la unión de un conjunto de vértices y un conjunto de cerradura. En la segunda parte

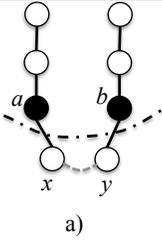
(Lema 4), se muestra por inducción que el algoritmo CIF-CACTUS calcula un CIF en $O(D_K)$ rondas en un cactus general K , donde D_K es el diámetro del cactus. En la tercera parte (Lema 5), se muestra que el CIF generado es maximal.

Lema 3. Sea S_r^d un CIF de K_r^d para algún d y sea C^{d+1} un conjunto de cerradura de K^m . El algoritmo CIF-CACTUS genera un CIF en el grafo inducido generado por la unión de los vértices de K_r^d y C^{d+1} en a lo más dos rondas.

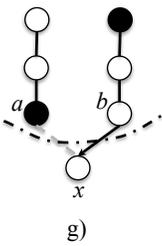
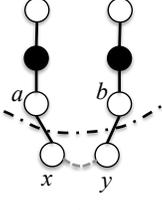
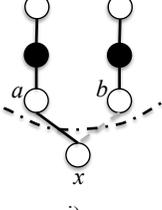
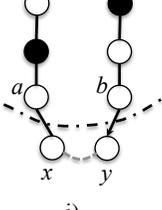
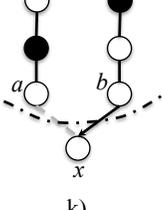
Demostración. Se demuestra por contradicción. Supongase que el algoritmo CIF-CACTUS no genera un CIF en el subgrafo inducido generado por la unión de los vértices de K_r^d y C^{d+1} después de dos rondas. Suponga además, sin falta de generalidad, que los colores de los vértices de C^{d+1} son azul₂.

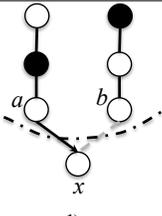
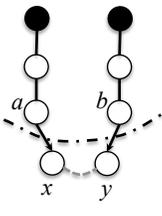
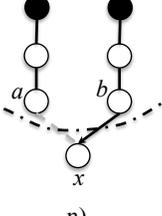
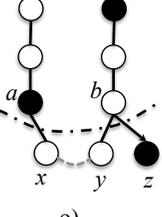
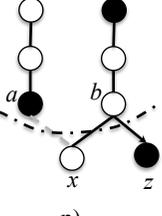
Existen 21 configuraciones diferentes para el grafo inducido por $\{K_r^d \cup C^{d+1}\}$. En todos estos casos, en a lo más dos rondas, al ejecutar la regla SEGUIR (en uno o dos vértices de $\{C^{d+1}\}$), el algoritmo CIF-CACTUS siempre corrige, si es necesario, los colores de los vértices en este conjunto. El conjunto de vértices rojos en el subgrafo inducido generado por la unión de los vértices de K_r^d y C^{d+1} es también un CIF. Por lo tanto existe una contradicción para cada caso. A continuación la Tabla 3 muestra la descripción de cada uno de estos casos. \square

Tabla 3. Configuraciones del grafo inducido por $\{K_r^d \cup C^{d+1}\}$

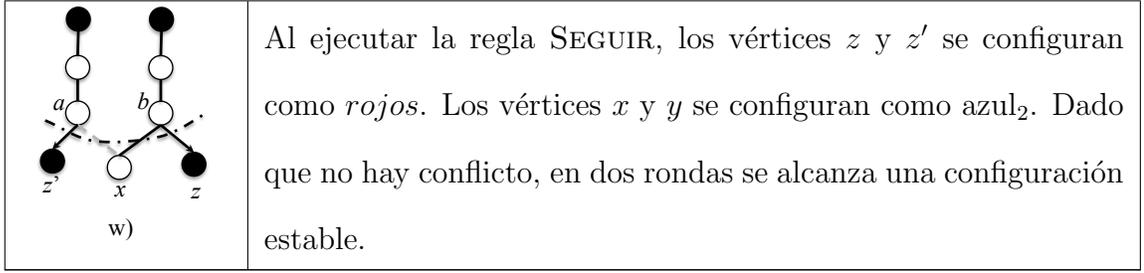
Configuración	Descripción
 <p>a)</p>	<p>Cuando x y y ejecutan la regla SEGUIR ambos vértices se configuran como azul₁ y apuntan a sus respectivos padres. Se alcanza una configuración estable después de una ronda.</p>

<p>b)</p>	<p>Al ejecutar la regla SEGUIR, el vértice x se configura como azul₁ y apunta a su padre. En una ronda converge a una configuración estable.</p>
<p>c)</p>	<p>Después de ejecutar la regla SEGUIR, x se configura como azul₁ y apunta a su padre. El vértice y se configura como azul₂. Dado que no hay conflicto, en una ronda se alcanza una configuración estable.</p>
<p>d)</p>	<p>Al ejecutar la regla SEGUIR, x se configura como azul₁ y apunta a su padre. Dado que no hay conflicto, en una ronda se alcanza una configuración estable.</p>
<p>e)</p>	<p>En la primera ronda el vértice x se configura como azul₁ y apunta a su padre; el vértice y se configura como rojo. En una ronda adicional, el vértice y se configura como azul₂ dado que el padre de su oponente es rojo. Después de dos rondas no hay conflicto, y se alcanza una configuración estable.</p>
<p>f)</p>	<p>Después de una ronda el vértice x se configura como azul₁. Dado que no hay conflicto, en una ronda se alcanza una configuración estable.</p>

 <p>g)</p>	<p>Después de una ronda el vértice x se configura como rojo al ejecutar la regla SEGUIR. En la siguiente ronda, se ejecuta nuevamente la regla SEGUIR en el vértice x y se configura como azul₂. Después de esta ronda adicional no hay conflicto, por lo que en dos rondas se alcanza una configuración estable.</p>
 <p>h)</p>	<p>Al ejecutar la regla SEGUIR, los vértices x y y se configuran como azul₂. Dado que no hay conflicto, en una ronda se alcanza una configuración estable.</p>
 <p>i)</p>	<p>Al ejecutar la regla SEGUIR, el vértice x se configura como azul₂. Dado que no hay conflicto, en una ronda se alcanza una configuración estable.</p>
 <p>j)</p>	<p>Al ejecutar la regla SEGUIR, el vértice x se configura como azul₂. El vértice y se configura como rojo. Dado que no hay conflicto, en una ronda se alcanza una configuración estable.</p>
 <p>k)</p>	<p>Al ejecutar la regla SEGUIR, el vértice x se configura como rojo. Dado que no hay conflicto, en una ronda se alcanza una configuración estable.</p>

 <p>l)</p>	<p>Al ejecutar la regla SEGUIR, el vértice x se configura como azul₂. Dado que no hay conflicto, en una ronda se alcanza una configuración estable.</p>
 <p>m)</p>	<p>Al ejecutar la regla SEGUIR, el vértice x se configura como rojo. El vértice y se configura como rojo. En una ronda adicional uno de los dos(el de menor prioridad) se configura como azul₂. Dado que no hay conflicto, en dos rondas se alcanza una configuración estable.</p>
 <p>n)</p>	<p>Al ejecutar la regla SEGUIR, el vértice x se configura como rojo. Dado que no hay conflicto, en una ronda se alcanza una configuración estable.</p>
 <p>o)</p>	<p>Al ejecutar la regla SEGUIR, el vértice x se configura como azul₁, el vértice y se configura como azul₂ y el vértice z se configura como rojo. Dado que no hay conflicto, en una ronda se alcanza una configuración estable.</p>
 <p>p)</p>	<p>Al ejecutar la regla SEGUIR, el vértice x se configura como azul₂ y el vértice z se configura como rojo. Dado que no hay conflicto, en una ronda se alcanza una configuración estable.</p>

<p>q)</p>	<p>Al ejecutar la regla SEGUIR, el vértice x se configura como azul₂, el vértice y se configura como azul₂ y el vértice z se configura como rojo. Dado que no hay conflicto, en una ronda se alcanza una configuración estable.</p>
<p>r)</p>	<p>Al ejecutar la regla SEGUIR, el vértice x se configura como azul₂ y el vértice z se configura como rojo. Dado que no hay conflicto, en una ronda se alcanza una configuración estable.</p>
<p>s)</p>	<p>Al ejecutar la regla SEGUIR, el vértice x se configura como rojo, el vértice y se configura como azul₂ y el vértice z se configura como rojo. Dado que no hay conflicto, en una ronda se alcanza una configuración estable.</p>
<p>t)</p>	<p>Al ejecutar la regla SEGUIR, el vértice x se configura como rojo y el vértice z se configura como rojo. La siguiente ronda, el vértice x se configura como azul₂. Dado que no hay conflicto, en dos rondas se alcanza una configuración estable.</p>
<p>u)</p>	<p>Al ejecutar la regla SEGUIR, el vértice x se configura como azul₂ y el vértice z se configura como rojo. Dado que no hay conflicto, en dos rondas se alcanza una configuración estable.</p>
<p>v)</p>	<p>Al ejecutar la regla SEGUIR, los vértices z y z' se configuran como rojos. Los vértices x y y se configuran como azul₂. Dado que no hay conflicto, en dos rondas se alcanza una configuración estable.</p>



Corolario 6. Sea S_r^d un CIF de K_r^d para algún d y sea C_i^{d+1} un conjunto de cerradura i , para algún $1 \leq i \leq t$, en K^m . El algoritmo CIF-CACTUS calcula un CIF en el subgrafo inducido generado por la unión de vértices de K_r^d y C_i^{d+1} , para $1 \leq i \leq t$, en a lo más dos rondas.

Demostración. Note que para dos vértices $x \in C_i^{d+1}$ y $y \in C_j^{d+1}$, e $dist(x, y) \geq 4$, para algún $i \neq j$. Así, el coloreado del conjunto C_j^{d+1} es independiente del coloreado del conjunto C_i^{d+1} . Por lo tanto, los resultados del Lema 3 se generalizan para más de un conjunto de cerradura. \square

Lema 4. Dado el cactus global seguro maestro K^m , el algoritmo CIF-CACTUS genera un CIF en $O(D_K)$ rondas.

Demostración. Se demuestra por inducción en d , donde $d = dist(x, r)$ para algún $x \in V_K^m$. Suponga sin pérdida de generalidad que todos los vértices son azul₂.

Caso base. En la primera ronda, los vértices a una distancia $d = 0$ ejecutan la regla RAIZ. El vértice r es el único que satisface esta condición. Dicho vértice configura sus variables $r.clr \leftarrow rojo$ y la variable $r.ptr \leftarrow r$. Note que S_r^0 es un CIF trivial de cardinalidad uno.

En la ronda dos, cada vértice x a una distancia $d = 1$ ejecuta la regla SEGUIR y configura sus variables como azul₁. Note que S_r^1 es un CIF de cardinalidad uno.

En la tercera ronda, cada vértice x a una distancia $d = 2$, que no pertenece a un

conjunto de cerradura, ejecuta la regla SEGUIR y configura sus variables como azul₂. El algoritmo corrige cualquier vértice que pertenece a un conjunto de cerradura C_i^2 (por el Corolario 6). Note, que a lo más al final de la cuarta ronda, S_r^2 es un CIF de cardinalidad uno y cualquier vértice x a una distancia $d = 2$ es azul₂.

A lo más en la ronda cinco, cada vértice x a una distancia $d = 3$, que no pertenece a un conjunto de cerradura, y que es hijo único, ejecuta la regla SEGUIR y configura sus variables como $x.clr \leftarrow rojo$ y $x.ptr \leftarrow x$.

Note que r es el único vértice rojo en K_r^2 y que la distancia entre r y algún vértice B^3 es tres. Si el padre y de x tiene más de un hijo, sólo uno de dichos hijos de y fija su color como rojo; cualquier otro hijo de y fija su color como azul₂ (la distancia entre cualquier par de hermanos es dos). Note que la distancia entre cualquier par de vértices rojos en B^3 es al menos cuatro. El algoritmo colorea correctamente cualquier vértice que pertenece a un conjunto de cerradura C_i^3 (por el Corolario 6) en a lo más dos rondas.

Por lo tanto, en a lo más el final de la ronda seis, S_r^3 es un CIF que consiste del vértice r y los vértices rojos de B^3 . Cualquier vértice x , a una distancia $d = 3$ puede ser rojo o azul₂.

A lo más en la séptima ronda, cada vértice x a una distancia $d = 4$, que no pertenece a un conjunto de cerradura y tiene la característica de que su padre y es azul₂, tiene el mismo comportamiento que los vértices a una distancia $d = 3$. Si su padre y es rojo, su comportamiento es similar a aquellos vértices a una distancia $d = 1$. El algoritmo colorea correctamente cualquier vértice que pertenece al conjunto de cerradura C_i^4 (por el Corolario 6) en a lo más dos rondas. Note que a lo más al final de la octava ronda S_r^4 es un CIF que consiste del vértice r y de los vértices rojos en B^3 y B^4 ; la distancia entre cualquier par de vértices rojos en B^4 es al menos cuatro. La distancia entre

cualquier vértice rojo en B^3 y cualquier vértice rojo en B^4 es al menos tres. En este punto, cualquier vértice x a una distancia $d = 4$, puede ser rojo o azul₁ o azul₂.

Hipótesis de inducción. Supongase que al final de la ronda $2d$, donde $d \geq 5$, el conjunto S_r^d es un CIF.

Paso inductivo. Ahora se demuestra que al final de la ronda $2d + 2$, el conjunto S_r^{d+1} es un CIF.

Note que cada uno de los vértice a una distancia d de la raíz son rojo o azul₁ o azul₂. En la ronda $2d + 1$, cada vértice x a una distancia $d + 1$, que no pertenece a un conjunto de cerradura y cuyo padre es rojo (azul₁, azul₂) tiene un comportamiento similar a aquellos vértices a una distancia uno (dos, tres) de r (ver el caso base). El algoritmo colorea correctamente cualquier vértice que pertenece a un conjunto de cerradura C_i^{d+1} (por el Corolario 6) en a lo más dos rondas. Por lo tanto, a lo más al final de la ronda $2d + 2$, el conjunto S_r^{d+1} es un CIF. \square

Lema 5. Después de ejecutar la fase de optimización, existe un vértice rojo a una distancia a lo más dos de cualquier vértice azul₁ o azul₂.

Demostración. Si x es azul₁, por definición su padre es rojo. Si x es azul₂ existen tres casos.

Caso 1. El padre y de x es azul₁; entonces, existe un vértice rojo (el padre de y) a una distancia 2 de x .

Caso 2. El padre y de x es azul₂ y y tiene más de un hijo; en este caso algún hijo $z \neq x$ debe ser rojo. Entonces, existe un hermano de x que es rojo, localizado a una distancia dos de x .

Caso 3. El padre y de x es azul₂ y x es el único hijo; este caso ocurre únicamente cuando x está en un cierre de cadena y su oponente es rojo o algún vecino de su oponente es rojo. Entonces, existe un vértice rojo localizado a una distancia dos de x . \square

Corolario 7. Dado un cactus K , el algoritmo CIF-CACTUS calcula un CIF maximal en $O(D_K)$ rondas. Este conjunto consiste de los vértices *rojos* de K .

Demostración. Por el Lema 4, todos los vértices *rojos* satisfacen la condición 1 del CIF maximal, y por el Lema 5, todos los vértices de color azul satisfacen la condición 2 del CIF maximal. \square

Corolario 8. Sea G un anillo de tamaño n . El algoritmo CIF-ANILLO calcula un CIF máximo en el conjunto S en $O(D_K)$ rondas. El conjunto S consiste de de vértices *rojos* de G .

Demostración. Dado G_r^d , donde $d = \lfloor \frac{n}{2} \rfloor - 1$ el algoritmo CIF-ANILLO asigna el color rojo al vértice r y a cada vértice x cuya $dist(x, r)$ es un múltiplo de tres; el resto de los vértices G_r^d son de color azul; por lo tanto, S^d es un CIF máximo. Finalmente el algoritmo colorea el resto de los vértices de K (un conjunto de cerradura C^{d+1} que tienen a lo más dos vértices). Por lo tanto el algoritmo CIF-ANILLO calcula un CIF máximo en un anillo G . \square

Ahora se demuestra la propiedad de cerradura y el análisis de complejidad del algoritmo propuesto en este capítulo.

Lema 6. Una vez que el algoritmo CIF-CACTUS (CIF-ANILLO) calcula el CIF maximal en un cactus K (máximo en un anillo G), este CIF prevalece indefinidamente en ausencia de fallas.

Demostración. Dado que el algoritmo es silencioso (ninguna regla se privilegia después de que converge a una configuración estable), el CIF calculado permanece en una configuración legal siempre y cuando no existan fallas transitorias. \square

Teorema 2. Dado un cactus K (anillo G), el algoritmo CIF-CACTUS (CIF-ANILLO) calcula un CIF maximal (máximo) en $O(D_K)$ rondas.

Demostración. Dado que las fases de elección de líder y la fase de optimización

tardan $O(D_K)$ rondas cada una, la complejidad temporal total es $O(D_K)$ rondas, ya que son fases independientes. \square

En el siguiente capítulo se presenta un algoritmo distribuido para el problema del CIF en una topología outerplanar, la cuál es más general que el cactus.

Capítulo 4. Algoritmo Distribuido para el CIF Maximal en Grafos Outerplanares Geométricos

En este capítulo se presenta un algoritmo distribuido determinístico que calcula el conjunto independiente fuerte (CIF) maximal en un grafo outerplanar geométrico. En estos grafos, cada vértice conoce la ubicación de sus coordenadas geométricas y además se encuentra en la frontera con la cara externa del grafo. El algoritmo que se presenta en este capítulo consta de tres fases. La primera fase elige un vértice como líder en el grafo de entrada. La segunda fase explora el grafo de entrada para determinar la información relevante acerca de su estructura. La tercera fase, con la información obtenida en la segunda fase, calcula un CIF maximal. Cuando el grafo de entrada es un anillo, el algoritmo calcula un CIF máximo. El tiempo de ejecución de este algoritmo es $O(n)$ pasos y utiliza $O(n \log n)$ mensajes. Este algoritmo no requiere conocer el tamaño del grafo de entrada. Hasta donde el autor del presente documento tiene conocimiento, éste es el primer algoritmo distribuido determinístico que resuelve este problema para un grafo geométrico outerplanar en tiempo lineal.

4.1 Introducción

Un *sistema distribuido* es una colección de procesadores independientes que se comunican entre sí utilizando una red de comunicación. Tales procesadores cooperan entre sí para alcanzar un objetivo global común. Los grafos modelan naturalmente los sistemas distribuidos y los científicos han estudiado y resuelto muchos problemas en grafos relacionados a estos problemas. Este capítulo se enfoca en el diseño de un algoritmo distribuido que calcula un CIF maximal en grafos geométricos outerplanares.

Un concepto muy bien conocido en la teoría de grafos es el conjunto independiente

definido en la Introducción. La mayor parte de la investigación en esta área se centra en el conjunto independiente maximal (MIS por sus siglas en inglés). Algunos algoritmos distribuidos relevantes para encontrar un MIS son los propuestos en (Panconesi y Srinivasan, 1996), (Barenboim y Elkin, 2009), (Kuhn, 2009), (Barenboim y Elkin, 2010), (Schneider y Wattenhofer, 2010), (Korman *et al.*, 2011).

Los algoritmos distribuidos más rápidos para resolver el problema MIS son los presentados por Panconesi y Srinivasan (1996) (para grafos generales) que corre en $O(2^{O(\sqrt{\log n})})$ rondas; los de Barenboim y Elkin (2009) y Kuhn (2009) (para grafos de grado acotado) que corren en $O(\Delta + \log^* n)$ rondas, donde Δ es el grado máximo en G ; y el de Barenboim y Elkin (2010) para grafos de arboricidad acotada (la *arboricidad* de un grafo no dirigido, es el mínimo número de bosques en el que su conjunto de aristas se puede particionar. Puede verse también como el mínimo número de bosques de expansión necesarios para cubrir todas las aristas del grafo) que corre en $O(\log n / \log \log n)$ rondas. Todos estos algoritmos tienen tiempo de ejecución sublineal y la mayoría de éstos utilizan la técnica de descomposición de redes descrita en Awerbuch *et al.* (1989). Esta técnica genera una partición de los vértices del grafo de entrada y generan un conjunto de clusters de características específicas. Ciertos problemas de grafos pueden resolverse parcialmente, independientemente y en paralelo, en cada cluster. Entonces, al combinar la solución parcial de cada cluster, es posible generar una solución para el problema original. Esta técnica es la que usan muchos de los algoritmos que corren en tiempo sublineal para el MIS. Todos estos algoritmos requieren que los vértices conozcan el número n de vértices en el grafo de entrada o al menos una cota superior de este parámetro. Esta es una suposición fuerte para cierto tipo de redes (por ejemplo, redes dinámicas), dado que calcular el número de vértices en un grafo requiere $\Omega(n)$ unidades como puede verse en (Awerbuch, 1987). El algoritmo propuesto en este capítulo, supone

que los vértices no conocen el tamaño n de G .

Existen muchas similitudes entre los problemas del MIS y del CIF maximal. Aunque existen algunos algoritmos distribuidos que corren en tiempo sublineal para el MIS en grafos generales, hasta donde el autor de la presente tesis tiene conocimiento, no existe un algoritmo sublineal para el CIF maximal en grafos generales. La principal dificultad para la implementación del problema del CIF maximal es que los vértices necesitan conocer información de otros vértices a una distancia dos. Obtener esta información puede ser muy costoso para grafos que no tienen grado constante. Por esta razón, al menos desde el punto de vista teórico, calcular el CIF maximal parece ser más complicado que calcular el MIS.

Una forma fácil de calcular un CIF maximal para un grafo G , basándose en el algoritmo diseñado por Barenboim y Elkin (2010), es la siguiente. Primero, calcular el grafo G^2 ; posteriormente, ejecutar el algoritmo para encontrar el MIS de Barenboim y Elkin (2010) en G^2 . El MIS resultante en G^2 es un CIF maximal en G . Note que el tiempo de ejecución de este algoritmo es sublineal sólo si G tiene grado constante. Para grafos de grado alto (incluso para algunos outerplanares), G^2 puede ser un grafo denso por lo que calcular todas sus aristas requiere $\Omega(n^2)$ operaciones. Por esta razón, el tiempo de ejecución de este procedimiento ya no es sublineal al utilizar n procesadores.

Cuando el valor de n no se conoce, muchos de los algoritmos actuales para calcular el MIS no se pueden aplicar directamente para calcular el CIF maximal. Sin embargo, existen algunos métodos que permiten ejecutar estos algoritmos sin el requisito de conocer n . Korman *et al.* (2011) proporciona un método que elimina, en algoritmos distribuidos, el requerimiento de conocer n (sin ninguna sobrecarga en la complejidad temporal). La principal desventaja de este método es que supone que el tamaño de cualquier mensaje es ilimitado (que es una suposición no factible). Los algoritmos

de Panconesi y Srinivasan (1996), Barenboim y Elkin (2009), y Barenboim y Elkin (2010) pueden utilizar este método para eliminar el requisito de saber n o un estimador polinomial de n . Sin embargo, implica una sobrecarga de $\Omega(\Delta^2)$ para transformar el tamaño de los mensajes a $O(\log n)$ bits (debido a que el método en (Korman *et al.*, 2011) requiere que algunos vértices consulten información de otros vértices a una distancia dos). Recuerde que Δ puede ser tan grande como $O(n)$ para grafos outerplanares. Schneider y Wattenhofer (2010) diseñaron un algoritmo distribuido óptimo para el cálculo de un MIS en $O(\log^* n)$ rondas para grafos con independencia acotada (un grafo $G = (V, E)$ es de *independencia acotada*, si existe una función acotada $f(r)$, tal que para cada vértice $v \in V$, el tamaño del conjunto independiente máximo considerando todos los vértices a una distancia menor o igual a r de v es a lo más $f(r)$, $\forall r \geq 0$), este algoritmo no requiere que los vértices conozcan el valor de n . Para grafos generales, este algoritmo puede ser tan lento como $O(n)$ rondas. Note que los grafos outerplanares no son necesariamente grafos de independencia acotada.

Incluso si se conoce el valor de n , un procedimiento similar al propuesto por Barenboim y Elkin (2010) para calcular un CIF maximal fallaría durante el proceso de combinación de las soluciones parciales de los clusters (ya que algunos vértices con grado alto tendrían que leer la información a una distancia dos en otros clusters).

Un grafo G es un *grafo plano* si se puede dibujar de tal forma que dos aristas solo se intersectan en los vértices. Dado un grafo plano G , sus *caras* son las regiones delimitadas por las aristas de G . Sea G un grafo plano, G es *outerplanar* si todos los vértices tocan la cara externa del grafo. A la región externa del grafo se denomina como *cara externa*. Los grafos outerplanares han atraído la atención de la comunidad científica, dado que algunos problemas que pertenecen a la clase de problemas NP-difícil para grafos arbitrarios, pueden resolverse en tiempo polinomial para grafos outerplanares

(Horváth *et al.*, 2010) y (Winter, 1986). Note que los árboles, anillos y cactus son subclases de grafos outerplanares.

El algoritmo que se propone en este capítulo recibe como entrada un grafo geométrico outerplanar. Los *grafos geométricos* suponen que cada vértice conoce sus propias coordenadas en el plano. Las aristas son líneas rectas que unen cualquier par de vértices que pueden comunicarse entre sí. Kranakis *et al.* (1999) utilizan grafos geométricos para descubrir rutas en un grafo no dirigido. Chávez *et al.* (2006) presentan un algoritmo para el descubrimiento de rutas para grafos outerplanares geométricos dirigidos. Algunas de las ideas detrás de la exploración del grafo outerplanar geométrico en nuestro algoritmo también están presentes en estos trabajos. Se utiliza la información de ubicación de los grafos geométricos para simplificar la exploración del grafo de entrada.

Los resultados obtenidos mediante el uso de grafos geométricos han demostrado ser útiles para la solución de diversos problemas combinatorios y de computación geométrica (Pach, 1991). Conocer la ubicación de cada vértice en el plano es una suposición real, dado que esta información está fácilmente disponible por el uso de sistemas de posicionamiento geográfico (GPS, por sus siglas en inglés).

Aunque el algoritmo propuesto en este capítulo resuelve un problema que se define para un grafo no dirigido, se modela el sistema distribuido donde se ejecuta como un grafo dirigido $D = (V, A)$, donde V y A son el conjunto de vértices y aristas dirigidas, respectivamente, y $|V| = n$. Cada vértice del grafo representa un procesador y cada arista dirigida un enlace de comunicación entre los procesadores. Se supone que los procesadores se comunican entre si utilizando el modelo de *paso de mensajes* (Attiya y Welch, 2004) en el que cada procesador se comunica con sus vecinos mediante el envío de mensajes a través de canales de comunicación. Se supone que cada vértice u tiene un identificador único de $O(\log n)$ bits y conoce su propia ubicación y la ubicación de

todos sus vecinos. La longitud del mensaje es de $O(\log n)$ bits (este modelo se conoce como el modelo *CONGEST* (Peleg, 2000)). Se supone un *modelo asíncrono* en el cual no existe un límite en el tiempo que tardan los pasos consecutivos en n procesadores (Attiya y Welch, 2004). Se miden las complejidades de mensajes y de tiempo como el máximo número de mensajes y el máximo número de pasos que el algoritmo requiere, respectivamente.

En este capítulo se propone el algoritmo determinístico CIF-OUTERPLANAR, para encontrar un CIF maximal en un grafo outerplanar geométrico G . El tiempo de ejecución del algoritmo CIF-OUTERPLANAR, es $O(n)$ pasos, donde n es el número de vértices en el grafo de entrada. La complejidad de mensajes es $O(n \log n)$ (debido al algoritmo de elección de líder que utiliza el algoritmo CIF-OUTERPLANAR). Hasta donde el autor del presente trabajo tiene conocimiento, el algoritmo CIF-OUTERPLANAR es el primer algoritmo distribuido determinístico que calcula un CIF maximal para un grafo outerplanar geométrico en un número de pasos lineal. Adicionalmente, este algoritmo calcula un CIF máximo cuando G es un anillo.

El algoritmo CIF-OUTERPLANAR se divide en tres fases independientes. La primera fase designa un único vértice como el líder. La segunda fase realiza un procedimiento que explora el grafo de entrada para obtener información relevante para la siguiente fase (el número n de vértices en el grafo de entrada, la frontera de la cara externa de G , determina si el grafo es o no es outerplanar). La tercera fase designa cuáles vértices pertenecen al CIF maximal. El procedimiento de la tercera fase se asemeja a una descomposición de orejas (el concepto de descomposición de orejas se describe en la Sección 4.2).

Se organiza el resto de este capítulo como sigue. La Sección 4.2 describe algunos conceptos básicos que son útiles para la descripción del algoritmo CIF-OUTERPLANAR. La

Sección 4.3 describe el algoritmo CIF-OUTERPLANAR. En la Sección 4.4 se demuestra que el algoritmo propuesto es correcto y se presenta su análisis de complejidad.

4.2 Preliminares

Esta sección presenta terminología básica y conceptos en grafos que se utilizarán en la descripción del algoritmo CIF-OUTERPLANAR. La *distancia* entre dos vértices de un grafo G es la longitud del camino más corto que conecta dichos vértices. Un *vértice de corte* es aquel que al removerlo desconecta G (Diestel, 2005). Un *punte* es una arista que al removerla desconecta G (Diestel, 2005). Un grafo G es *biconectado* si no tienen vértices de corte (Diestel, 2005).

El *grafo dual* $G^* = (V^*, E^*)$ de un grafo outerplanar no dirigido G es el grafo obtenido de G de la siguiente manera. Reemplazar cada cara de G por un vértice de V^* e insertar una arista entre dos vértices $u, v \in V^*$ si y solo si las fronteras de sus caras que corresponden a u y v comparten al menos una arista. El *grafo dual débil* $T^*(G) = (V_T^*, E_T^*)$ de G es el subgrafo inducido de G^* obtenido al remover el vértice que corresponde a la cara externa (Diestel, 2005). Particularmente, si el grafo outerplanar es biconectado, el grafo dual débil es un árbol (Agnarsson y Halldórsson, 2004). La Figura 12 muestra un grafo outerplanar biconectado G con ocho vértices y cuatro caras internas. El árbol en línea punteada y vértices en gris es el grafo dual débil $T^*(G)$ de G .

Sea $G = (V, E)$ un grafo no dirigido, y sea P_0 un ciclo simple arbitrario en G . Una *descomposición de orejas* de G , iniciando en P_0 , es una partición ordenada del conjunto de aristas $E = P_0 \cup P_1 \cup \dots \cup P_k$, tal que $\forall i (1 \leq i \leq k)$, P_i es un camino simple cuyos vértices finales pertenecen a $P_0 \cup P_1 \cup \dots \cup P_{i-1}$, pero ninguno de sus vértices internos

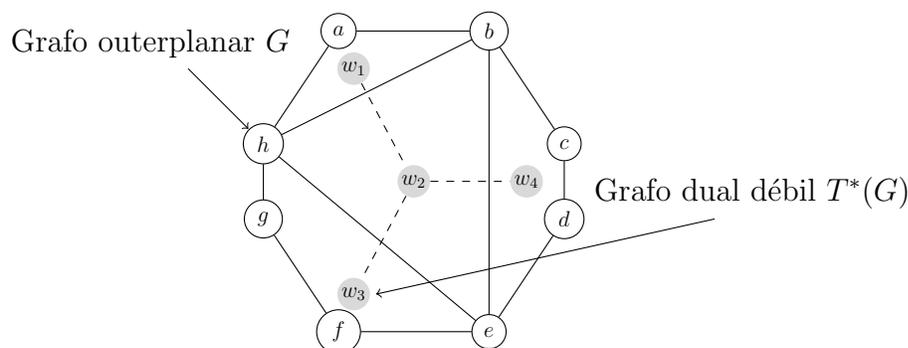


Figura 12. Grafo dual débil $T^*(G)$ del grafo outerplanar G .

pertenece a $P_0 \cup P_1 \cup \dots \cup P_{i-1}$ (JáJá,1992). Note que puede representarse una descomposición de orejas P_i como un conjunto ordenado de vértices $P_i = v_1, v_2, \dots, v_k$. En la Figura 12, una descomposición de orejas sería: si $P_0 = a, b, h, a$; entonces $P_1 = h, e, b$; $P_2 = h, g, f, e$; y $P_3 = e, d, c, b$.

La descomposición de orejas es fundamental para muchas aplicaciones en problemas de grafos, tales como pruebas de planaridad y de conectividad de un grafo. Algunos algoritmos distribuidos para descomposición de orejas se encuentran en (Kazmierczak y Radhakrishnan, 2000) y en (Tsin, 2004). Durante la tercera fase del algoritmo propuesto, se realiza un procedimiento que se asemeja a una descomposición de orejas para cada componente biconectado del grafo outerplanar geométrico G .

En este capítulo, G se refiere al grafo outerplanar geométrico no dirigido que es la entrada para el algoritmo CIF-OUTERPLANAR. Se modela el sistema distribuido donde el algoritmo CIF-OUTERPLANAR se ejecuta como un grafo D . Ambos grafos G y D tienen básicamente la misma topología. Se puede ver a D como una versión dirigida de G . Se obtiene D al reemplazar cada arista (u, v) de G por dos aristas dirigidas $\langle u, v \rangle$ y $\langle v, u \rangle$. En algunas partes del algoritmo, se hace referencia a G , pero a veces, se utiliza explícitamente D , dado que el algoritmo necesita distinguir entre las aristas $\langle u, v \rangle$ y $\langle v, u \rangle$.

Sea el vecindario cerrado de u , $N[u] = N(u) \cup u$. Sean ahora los conjuntos $A(u)^{\rightarrow} = \{e(u)_0^{\rightarrow}, \dots, e(u)_{\{|N(u)|-1\}}^{\rightarrow}\}$ y $A(u)^{\leftarrow} = \{e(u)_0^{\leftarrow}, \dots, e(u)_{\{|N(u)|-1\}}^{\leftarrow}\}$ sea el conjunto de aristas de salida y entrada incidentes a u , respectivamente. La asignación de etiquetas para las aristas es como sigue: desde un punto de vista global, la etiqueta para cada arista es un par ordenado de vértices (i.e. la arista $\langle u, v \rangle$ es una arista dirigida que va desde u a v); se denomina esta etiqueta como la *etiqueta global* de la arista. Al mismo tiempo, cada vértice asigna una *etiqueta local* a cada una de sus aristas incidentes. Para cada u , la etiqueta local $e(u)_0^{\rightarrow}$ corresponde a la arista $\langle u, v \rangle$, donde $v = \min \{w | w \in N(u)\}$; la arista $e(u)_1^{\rightarrow}$ es la arista de salida más cercana a $e(u)_0^{\rightarrow}$ en el sentido contrario a las manecillas del reloj. Es posible conocer esta información ya que el grafo de entrada es geométrico. El sistema establece las etiquetas locales de las aristas restantes utilizando el mismo procedimiento. En general, la arista de salida que sigue a $e(u)_i^{\rightarrow}$, en el sentido contrario a las manecillas del reloj, es la arista $e(u)_{(i+1) \bmod (|N(u)|)}^{\rightarrow}$. Se utiliza un procedimiento similar para etiquetar las aristas de entrada. La función $vtx_u(i)$ regresa el vecino $v \in N(u)$ que conecta las aristas $e(u)_i^{\rightarrow}$ y $e(u)_i^{\leftarrow}$. La función $vtx_inversa_u(v)$ regresa el índice i de las aristas $e(u)_i^{\rightarrow}$ y $e(u)_i^{\leftarrow}$ que conectan u a v y v a u , respectivamente.

Cualquier arista que toque la frontera del grafo outerplanar es una *arista externa*; de otra forma es una *arista interna*. Una *caminata* W en un grafo es una secuencia no vacía de vértices $W = v_0, v_1, \dots, v_n$. Si en W , $v_0 = v_n$, entonces W es una *caminata cerrada*. Se dice que una caminata cerrada W es una *caminata cerrada hacia la izquierda*, si para una arista arbitraria $\langle u, v \rangle \in W$, la siguiente arista en W , la arista $\langle v, w \rangle$, es una arista incidente a v que es la más cercana a $\langle u, v \rangle$ en el sentido de las manecillas del reloj. Una *caminata externa cerrada hacia la izquierda* en D es una caminata cerrada hacia la izquierda que consiste exclusivamente de aristas de salida. Si D es conectado,

existe una caminata de salida cerrada hacia la izquierda, denominada W^O , que contiene todas las aristas de salida de D . Note que W^O es la *caminata de salida cerrada hacia la izquierda maximal* en D . Dado una caminata W_1 que termina en el vértice u y una caminata W_2 que inicia en u , la *concatenación* de W_1 y W_2 (denotada como $W_1 \circ W_2$) es la caminata que se obtiene cuando la caminata W_2 se une a la caminata W_1 mediante la fusión de estas caminatas en el vértice u . Una forma de simular una caminata hacia la izquierda en un grafo es a través del uso de un token de recorrido. Un *token* es un mensaje que genera un líder global o un líder local. El mensaje está dividido en cierto número de campos. Cada vértice que recibe el token puede modificar la información de alguno de los campos del token. El vértice que originó el token es el único con privilegios para eliminarlo del sistema. Un token que recorre una caminata hacia la izquierda es un *token hacia la izquierda*. Todos los tokens del presente capítulo son *tokens hacia la izquierda*, así que se refieren simplemente como tokens.

El líder de un grafo es un vértice único que realiza actividades especiales. La primera fase del algoritmo CIF-OUTERPLANAR elige un líder, r , en el grafo de entrada G . Suponga que el líder r es un vértice de corte en G (en el caso en que G no sea biconectado). Sea $\mathcal{C}' = \{\mathcal{C}'_1, \mathcal{C}'_2, \dots, \mathcal{C}'_f\}$ el conjunto de componentes conectados de G después de remover r . Sea $Z_i = V_{\mathcal{C}'_i} \cup r$ y sea $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_f\}$ el conjunto de todos los r -componentes de G . Los r -componentes $\mathcal{C}_i = G[Z_i]$, donde $G[Z_i]$ es el subgrafo de G inducido por el conjunto Z_i . Se utiliza el concepto de r -componentes en las Secciones 4.3 y 4.4 para extender el algoritmo CIF-OUTERPLANAR a grafos outerplanares geométricos arbitrarios.

4.3 Algoritmo CIF-OUTERPLANAR

Esta sección presenta una descripción general del algoritmo CIF-OUTERPLANAR, que consiste de tres fases independientes. La *fase de elección de líder* configura un único vértice como el líder. En la *fase de exploración*, el líder obtiene información acerca de la estructura del grafo de entrada utilizando un conjunto de tokens. La *fase de coloreado* utiliza un conjunto de tokens y la información proporcionada por la fase de exploración para identificar los vértices que pertenecen al CIF maximal. El Pseudocódigo 5 presenta el algoritmo CIF-OUTERPLANAR.

Pseudocódigo 5. Algoritmo CIF-OUTERPLANAR(G)

Entrada: Un grafo outerplanar geométrico no dirigido conectado $G = (V, E)$.

Salida: Un CIF maximal $S \subseteq V$.

```

begin
     $D^L \leftarrow \text{FASE-ELECCION-LIDER}(G)$ 
     $D^E \leftarrow \text{FASE-EXPLORACION}(D^L)$ 
     $G^C \leftarrow \text{FASE-COLOREADO}(D^E)$ 
end

```

Existen muchos algoritmos en la literatura para implementar la fase de elección de líder (Tel, 2000) y (Attiya y Welch, 2004). Para la fase de elección de líder del algoritmo propuesto, se utiliza el algoritmo diseñado por Awerbuch (1987), dado que es óptimo en tiempo y en mensajes. Este algoritmo supone un modelo de paso de mensajes en una red asíncrona. La entrada de la fase de elección de líder es un grafo outerplanar geométrico no dirigido conectado G (la fase de elección de líder también funciona en grafos que no son geométricos). La salida de esta fase es un grafo outerplanar geométrico $D^L = (V^L, A^L)$, el cual es el grafo D con un vértice r (al que se llama líder). El tiempo de ejecución de esta fase es $O(n)$ pasos, donde n es el número de vértices del grafo de entrada. La complejidad de mensajes es $O(n \log n)$ mensajes.

Se describe la fase de exploración en la Sección 4.3.1 y la fase de coloreado en la Sección 4.3.2 La Sección 4.3.3 muestra cómo se extiende la fase de coloreado que funcione en cualquier grafo outerplanar. Ahora, se describe la fase de exploración para el algoritmo CIF-OUTERPLANAR.

4.3.1 Fase de exploración

La fase de exploración recibe como entrada un grafo outerplanar geométrico D^L como se describió previamente. La fase de exploración utiliza dos tokens, creados por r , para recorrer D^L . La salida de esta fase es un grafo $D^E = (V^E, A^E)$. D^E es el grafo D en el cual se identifican, el líder r , todos los vértices de corte, y todas las aristas externas y las aristas internas de D .

Se divide la fase de exploración en las siguientes subfases:

1. Durante la *primera subfase de exploración*, r envía el primer token de exploración T_1 a través de cada una de sus aristas de salida $e(r)_i^{\rightarrow}$, para $i = 0, \dots, |N(r)| - 1$. Utilizando información obtenida por T_1 , r determina las aristas externas de salida incidentes a r y también determina si r es o no un vértice de corte. Esta subfase utiliza el siguiente procedimiento para calcular dicha información:

- Suponga que r envía T_1 a través de la arista $e(r)_i^{\rightarrow}$ y que regresa a r a través de la arista $e(r)_j^{\leftarrow}$. Si $j \neq (i + 1) \bmod |N(r)|$, entonces la arista $e(r)_i^{\rightarrow}$ es una arista de salida externa. Particularmente, si $j = i$, entonces la arista $(r, vtx_r(i)) \in E$ es también un puente en G . Incluso, si $|N(r)| > 1$, entonces r es un vértice de corte.
- Supóngase que r envía T_1 a través de la arista $e(r)_i^{\rightarrow}$ y regresa a r a través de la arista $e(r)_{(i+1) \bmod |N(r)|}^{\leftarrow}$, para todo $i = 0, \dots, |N(r)| - 1$. En este caso,

existe solo una arista de salida externa. Esta arista de salida externa está incluida en la caminata de salida cerrada a la izquierda maximal W^O .

2. Durante la *segunda subfase de exploración*, r envía un segundo token de exploración T_2 sólo a través de cada una de sus aristas de salida externas para identificar todas las aristas externas e internas en D^L . Con la información proporcionada por el token T_2 , el algoritmo identifica W^O y todos los vértices de corte en D^L . Esta subfase utiliza el siguiente procedimiento:

- El algoritmo marca u como vértice de corte cuando un vértice $u \neq r$ recibe T_2 dos o más veces.
- El algoritmo marca como arista externa solo aquellas aristas que recorre el token T_2 en D^L ; de otra forma, marca estas aristas como internas.

Los token de exploración T_1 y T_2 contienen el identificador del líder r . Adicionalmente, T_1 contiene un contador que cada vértice incrementa cada vez que T_1 lo visita.

Observación 1. Note que el token T_2 se puede utilizar para determinar si el grafo de entrada G es o no outerplanar. Dado que T_2 viaja exclusivamente a través de la cara externa de G , todos los vértices en G deben recibir T_2 si G es outerplanar; de otra forma, G no es outerplanar.

Ahora se describe la fase de exploración del algoritmo. El Pseudocódigo 6 presenta la FASE DE EXPLORACION.

Pseudocódigo 6. FASE DE EXPLORACION**Entrada:** Un grafo geométrico outerplanar $D^L = (V^L, A^L)$.**Salida:** Un grafo geométrico outerplanar $D^E = (V^E, A^E)$.▷ Inicialmente, $maxLongitud$ is 0.

▷ Inicialmente, todos los vértices no son de corte.

▷ Inicialmente, cada arista es interna.

{para el líder r .}**begin**

1. **If** (\exists una arista $e(r)_i^{\rightarrow}$ tal que el token T_1 no ha viajado por de dicha arista y T_1 no está en tránsito) **then**
2. El vértice r envía T_1 a través de la arista $e(r)_i^{\rightarrow}$
3. **end_if**
4. **If** r recibe T_1 desde $e(r)_j^{\leftarrow}$ **then**
5. **If** ($j = (i + 1) \bmod |N(r)|$) **then**
6. **If** la longitud de la caminata cerrada recorrida por T_1 es mayor que $maxLongitud$ **then**
7. $max_indice \leftarrow i$
8. $maxLongitud \leftarrow$ la longitud de la caminata cerrada recorrida por T_1
9. **end_if**
10. **else**
11. El vértice r marca la arista $e(r)_i^{\rightarrow}$ como una arista de salida externa
12. El vértice r se configura como vértice de corte si $|N(r)| > 1$
13. **end_if**
14. **end_if**
15. **If** (r no es un vértice de corte AND r envía T_1 a través de todas sus aristas de salida AND T_1 no está en tránsito) **then**
16. $z \leftarrow max_indice$
17. El vértice r marca la arista $e(r)_z^{\rightarrow}$ como una arista de salida externa
18. **end_if**
19. **Para** todas las aristas de salida externas $e(r)_i^{\rightarrow}$ **do**
20. El vértice r envía el token de exploración T_2
21. **end_for**
22. **If** todos los tokens T_2 regresaron a r **then**
23. El vértice r inicia la fase de coloreado
24. **end_if**

{Para cualquier otro vértice $u \neq r$ }

25. **If** u recibe un token T_1 de $e(u)_i^{\leftarrow}$ **then**
26. El vértice u incrementa el contador de T_1 y reenvía T_1 a la arista $e(u)_{(i-1) \bmod |N(r)|}^{\rightarrow}$
27. **end_if**

```

28. If  $u$  recibe un token  $T_2$  desde  $e(u)_i^{\leftarrow}$  then
29.   If  $u$  recibió previamente el token  $T_2$  then
30.     El vértice  $u$  se configura como vértice de corte
31.   end_if
32.   El vértice  $u$  incrementa el contador de  $T_2$  y reenvía  $T_2$  a la arista  $e(u)_{(i-1) \bmod |N(r)|}^{\rightarrow}$ 
33.   El vértice  $u$  marca la arista  $e(u)_i^{\leftarrow}$  como una arista de entrada externa
34.   El vértice  $u$  marca la arista  $e(u)_{(i-1) \bmod |N(r)|}^{\rightarrow}$  como una arista de salida externa
35. end_if
end

```

La Sección 4.4.1 presenta el análisis de la fase de exploración.

4.3.2 Fase de coloreado

El objetivo de la fase de coloreado es calcular un CIF maximal, S , en G . Esta fase colorea como *rojo* todos los vértices que pertenecen a S ; de otra forma, los colorea como azul. Dependiendo de que tan lejos están los vértices azules de un vértice rojo, se distinguen dos clases de vértices azules. Un vértice es *azul₁* si alguno de sus vecinos es rojo; de otra forma, es *azul₂*.

La entrada de esta fase es un grafo geométrico outerplanar D^E proporcionado por la fase de exploración. La salida de esta fase es el grafo $G^C = (V^C, E^C)$, el cual es el grafo G con la siguiente información adicional para un vértice $u \in V$.

- Si $u \in S$, entonces sus variables $u.clr = rojo$, $u.ptr = u$ (u es un vértice rojo).
- Si $u \notin S$ y existe un vértice $v \in N(u)$, tal que $v \in S$, entonces sus variables son $u.clr = azul$ y $u.ptr = v$ (u es un vértice azul₁).
- Si $u \notin S$ y para cada $v \in N(u)$, $v \notin S$, entonces sus variables son $u.clr = azul$ y $u.ptr = u$ (u es un vértice azul₂).

Es importante notar que en un grafo outerplanar, un par de ciclos adyacentes comparten a lo más una arista. Cuando este grafo es biconectado, cualquier par de ciclos adyacentes comparte exactamente una arista. Para facilitar la explicación de la fase de coloreado, se supone que G es biconectado y que $|V| \geq 3$; después, en la Sección 4.3.3, se extiende el algoritmo para considerar grafos geométricos outerplanares arbitrarios.

La idea general del algoritmo para esta fase es realizar un procedimiento que asemeje una descomposición de orejas de un grafo no dirigido G . Dado que G es biconectado, la primera oreja es un ciclo y cualquier oreja siguiente es un camino abierto. Después de descubrir el primer ciclo, el algoritmo cuenta el número de vértices en el ciclo y entonces les asigna colores del conjunto {rojo, azul₁, azul₂}. El primer ciclo (y su cara respectiva) corresponden a la raíz del grafo dual débil $T^*(G)$ de G . Entonces, este algoritmo determina si existe un ciclo adyacente al ciclo procesado previamente. Este algoritmo encuentra el siguiente ciclo al realizar un procedimiento que es equivalente a realizar una búsqueda en profundidad en pre-orden de $T^*(G)$. (Un recorrido en pre-orden visita primero la raíz del árbol y entonces, recursivamente, sigue un recorrido en pre-orden de cada uno de sus subárboles de izquierda a derecha). Si dicho ciclo existe, este algoritmo realiza el mismo procedimiento como en el ciclo previo y continúa de forma recursiva. Esta fase termina cuando todos los vértices de todas las orejas de G están coloreados. Ahora, se introducen nuevas variables y definiciones básicas para la implementación de esta fase.

El primer ciclo, C_0 (que también es la oreja P_0), es incidente a r y contiene una única arista de salida externa de r en D^E . Cualquier ciclo siguiente C_i es la concatenación de una oreja abierta P_i con la arista definida por los dos vértices finales de P_i . La *raíz de ciclo*, r_i , del ciclo C_i es el vértice que descubre la existencia de C_i . El vértice r_i coordina el coloreado de cualquier vértice no coloreado en C_i . Nótese que r es también la raíz de

ciclo de C_0 . Suponga que los vértices en C_i son la secuencia $c_{i,1}, c_{i,2}, \dots, c_{i,j}, c_{i,1}$, donde $c_{i,1} = r_i$.

Se presentan ahora detalles más específicos sobre la implementación de esta fase. Inicialmente, el algoritmo colorea el vértice r como rojo. Se supone que todos los otros vértices no están coloreados. Cuando la raíz de ciclo, r_i , descubre un nuevo ciclo C_i , el algoritmo ejecuta las siguientes tres subfases para todo $i \geq 1$.

- *Subfase de conteo.* La raíz de ciclo, r_i , primero crea un *token de conteo* T_E que recorre el ciclo C_i . T_E incrementa su contador interno cada vez que visita un vértice en C_i (este contador se inició durante la creación de T_E). Cuando el vértice r_i recibe T_E , éste obtiene el número de vértices en C_i . El token T_E también obtiene el color del vértice $c_{i,j}$ durante su recorrido a través de C_i .
- *Subfase rojo/azul.* La raíz de ciclo, r_i , crea un *token rojo/azul* T_C que recorre el ciclo C_i . T_C contiene el número de vértices de C_i y el color de cualquier vértice coloreado de C_i . Si r_i es azul₂, se recolorea como azul₁ y apunta a $c_{i,2}$ y C_i antes de enviar T_C . Cuando un vértice $c_{i,k}$ de C_i , para un $2 \leq k \leq j - 4$, recibe T_C , $c_{i,k}$ determina su color y su apuntador basado en el color del vértice $c_{i,k-1}$ (seleccionando un color apropiado del conjunto {rojo, azul₁}) y su apuntador. Este coloreado forma una secuencia como la siguiente: \dots , rojo, azul₁, azul₁, rojo, azul₁, azul₁, \dots , y así sucesivamente. Para $k \geq j - 3$, el algoritmo ejecuta las siguientes reglas secuencialmente cuando $c_{i,k}$ recibe T_C . El objetivo de estas reglas es evitar que la distancia entre dos vértices rojos sea menor que tres.

– Para el vértice $c_{i,j-3}$.

- * Si $c_{i,j-4}$ es rojo, entonces $c_{i,j-3}$ se colorea como azul₁ y configura su apuntador para señalar a $c_{i,j-4}$.

- * Si $c_{i,j-4}$ es azul₁ y apunta a $c_{i,j-3}$, entonces $c_{i,j-3}$ se colorea como rojo.
 - * Si $c_{i,j-4}$ es azul₁ y no apunta a $c_{i,j-3}$ y $c_{i,j}$ es rojo, entonces $c_{i,j-3}$ se colorea como azul₂.
 - * De otra forma, $c_{i,j-3}$ se colorea como azul₁ y configura su apuntador para señalar a $c_{i,j-2}$.
- Para el vértice $c_{i,j-2}$.
- * Si $c_{i,j-3}$ es rojo, entonces $c_{i,j-2}$ se colorea como azul₁ y configura su apuntador para señalar a $c_{i,j-3}$.
 - * Si $c_{i,j-3}$ es azul₁ y apunta a $c_{i,j-2}$ y $c_{i,j}$ no es rojo, entonces $c_{i,j-2}$ se colorea como rojo.
 - * Si $c_{i,j-3}$ es azul₁ y no apunta a $c_{i,j-2}$ y ($c_{i,j}$ o $c_{i,1}$ es rojo), entonces $c_{i,j-2}$ se colorea como azul₂.
 - * De otra forma, $c_{i,j-2}$ se colorea como azul₁ y configura su apuntador para señalar a $c_{i,j-1}$.
- Para el vértice $c_{i,j-1}$.
- * Si $c_{i,j-2}$ es rojo, entonces $c_{i,j-1}$ se colorea como azul₁ y configura su apuntador para señalar a $c_{i,j-2}$.
 - * Si $c_{i,j-2}$ es azul₁ y apunta a $c_{i,j-1}$ y (ni $c_{i,j}$ ni $c_{i,1}$ es rojo), entonces $c_{i,j-1}$ se colorea como rojo.
 - * Si $c_{i,j-2}$ es azul₂ y $c_{i,1}$ es rojo, entonces $c_{i,j-1}$ se colorea como azul₂.
 - * Si $c_{i,j-2}$ es azul₂, $c_{i,1}$ no es rojo y ($c_{i,j}$ es rojo o no coloreado), entonces $c_{i,j-1}$ se colorea como azul₁ y configura su apuntador para señalar a $c_{i,j}$.
 - * Si $c_{i,j-2}$ es azul₁ y no apunta a $c_{i,j-1}$ y ($c_{i,j}$ es rojo o no coloreado), entonces $c_{i,j-1}$ se colorea como azul₁ y configura su apuntador para señalar

- a $c_{i,j}$.
- * De otra forma, $c_{i,j-1}$ se colorea como azul₂.
- Para un vértice no coloreado $c_{i,j}$.
 - * Si $c_{i,j-1}$ es rojo, entonces $c_{i,j}$ se colorea como azul₁ y configura su apuntador para señalar a $c_{i,j-1}$.
 - * Si $c_{i,j-1}$ es azul₁ y apunta a $c_{i,j}$, entonces $c_{i,j}$ se colorea como rojo.
 - * Si $c_{i,j-1}$ es azul₂ y $c_{i,1}$ es rojo, entonces $c_{i,j}$ se colorea como azul₁ y configura su apuntador para señalar a $c_{i,1}$.
 - * Si $c_{i,j-1}$ es azul₁ y no apunta a $c_{i,j}$ y $c_{i,1}$ es rojo, entonces $c_{i,j}$ se colorea como azul₁ y configura su apuntador para señalar a $c_{i,1}$.
 - * De otra forma, $c_{i,j-1}$ se colorea como azul₂.
- Para un vértice coloreado $c_{i,j}$.
 - * Si $c_{i,j}$ es azul₂ y $c_{i,j-1}$ es rojo, entonces $c_{i,j}$ se recolorea como azul₁ y configura su apuntador para señalar a $c_{i,j-1}$.
- *Subfase de búsqueda.* La raíz r_i del ciclo C_i crea un *token de búsqueda* T_S que recorre el ciclo C_i . Cuando un vértice u recibe T_S , determina si la siguiente arista, (u, v) , en C_i la comparten entre C_i y un nuevo ciclo C_j (el ciclo C_j es la concatenación de la oreja P_j y la arista (u, v)). Si C_j existe, el algoritmo ejecuta estas tres subfases en C_j recursivamente, iniciando en u . Esta fase termina cuando el token T_S regresa a r_i .

El Pseudocódigo 7, denominado FASE-COLOREADO, presenta el algoritmo para esta fase.

Pseudocódigo 7. FASE-COLOREADO**Entrada:** Un grafo geométrico D^E .**Salida:** Un grafo geométrico coloreado G^C .▷ Para referirse a un *atributo* de un token T como $T \langle \text{campo} \rangle$.▷ El vértice v es el vecino de r tal que la arista $\langle r, v \rangle$ es la única arista de salida externa de r en D^E .▷ La función Booleana *nuevo_ciclo* recibe como entrada un índice k y determina, dada la arista $e(u)_k^{\rightarrow}$, si la arista $e(u)_k^{\leftarrow}$ pertenece a un nuevo ciclo C_j .{FASE-COLOREADO para un vértice r }**begin**

1. **If** r no ha recibido aún el token T_S **then**
2. Configura el coloreado de r como rojo
3. $k \leftarrow vtx_inversa_r(v)$
4. *procesa_ciclo*(k)
5. **end_if**

{FASE-COLOREADO para $u \neq r$ }

6. Esperar hasta que T_E llegue a u
7. $k \leftarrow$ es el índice de la arista de entrada por donde T_E llega a u
8. $T_E \langle \text{contador} \rangle = T_E \langle \text{contador} \rangle + 1$
9. Reenviar T_E a través de $e(u)_{(k-1) \bmod |N(u)|}^{\rightarrow}$
10. esperar hasta que T_C llegue a u
11. $k \leftarrow$ es el índice de la arista de entrada por donde T_C llega a u
12. Asigna correctamente $u.clr$, y $u.ptr$
13. $T_C \langle \text{contador} \rangle = T_C \langle \text{contador} \rangle + 1$
14. Reenviar T_C a través de $e(u)_{(k-1) \bmod |N(u)|}^{\rightarrow}$
15. Esperar hasta que T_S llegue a u
16. $k \leftarrow$ es el índice de la arista de entrada por donde T_S llega a u
17. **If** (*nuevo_ciclo*(($k - 1$) \bmod $|N(u)|$)) **then**
18. *procesa_ciclo*(($k - 1$) \bmod $|N(u)|$)
19. **end_if**
20. Reenvía T_S a $e(u)^{\rightarrow (k-1) \bmod |N(u)|}$

{**La función** *process_cyle*(k) **para algún vértice** u }

21. El vértice u crea y envía un token de conteo T_E a través de la arista $e(u)_{(k-1) \bmod |N(u)|}^{\rightarrow}$ para contar el número de vértices de su ciclo actual y obtiene información acerca de los vértices que fueron coloreados previamente.

22. Esperar hasta que T_E regresa a u .
 23. El vértice u crea y envía un token rojo/azul T_C a través de la arista $e(u)_{(k-1) \bmod |N(u)|}^{\rightarrow}$ para colorear los vértices no coloreados de su ciclo actual.
 24. Esperar hasta que T_C regrese a u .
 25. El vértice u crea un token de búsqueda T_S .
 26. **If** ($nuevo_ciclo((k-1) \bmod |N(u)|)$) **then**
 27. $procesa_ciclo((k-1) \bmod |N(u)|)$
 28. **end_if**
 29. El vértice u envía T_S a través de la arista $e(u)_{(k-1) \bmod |N(u)|}^{\rightarrow}$.
 30. Esperar hasta que T_S regrese a u .
 31. Exit.
- end**

4.3.3 Fase de coloreado extendida

Ahora, se extiende el algoritmo CIF-OUTERPLANAR para un grafo geométrico outerplanar G que no necesariamente es biconectado (*i.e.* cuando G tiene vértices de corte). Las fases de elección de líder y de exploración funcionan para este caso. Ahora se presentan los cambios a realizar.

Se dice que un vértice u es la *raíz bi-comp* de un componente biconectado K si u es el vértice que descubre K . Note que r es la raíz bi-comp de todos los componentes biconectados a la que pertenece. Suponga que u es un vértice de corte en G y pertenece a un componente biconectado K . Suponga que el algoritmo previamente colorea u durante el procesamiento del componente biconectado K . Cuando u recibe el token de búsqueda T_S , el algoritmo detiene temporalmente el procesamiento de K e inicia el procesamiento del nuevo componente biconectado K' descubierto por u . Note que K' puede ser un puente o puede no serlo y u puede ser la raíz bi-comp de más de un componente biconectado. Dependiendo del tipo y número de componentes biconectados, se tienen tres casos. El primer caso es para un componente biconectado que no es un

puede. El segundo caso es para un puente. El tercer caso es cuando existe más de un componente biconectado. Ahora, se explican estos tres casos.

- Suponga que el vértice u es la raíz bi-comp de un nuevo componente biconectado K' que no es puente. El algoritmo procesa K' al ejecutar las subfases de conteo, rojo/azul y de búsqueda. Si durante la subfase de búsqueda de K' , el algoritmo encuentra otro vértice de corte v , el algoritmo detiene el procesamiento de K' y realiza recursivamente el procesamiento de un nuevo componente biconectado. Después de que u recibe el token T_S de la subfase de búsqueda del componente biconectado K' , el algoritmo continúa con el procesamiento del componente K .
- Suponga que u es la raíz bi-comp de un nuevo componente biconectado K' que es el puente (u, v) de G . El algoritmo omite la subfase de conteo para este caso, dado que es trivial. Entonces, u crea el token rojo/azul T_C y lo envía a través de la arista $\langle u, v \rangle$ de D^E . Cuando v recibe T_C , se colorea de acuerdo a una regla simple:
 - Si u es rojo, entonces v se colorea como azul₁ y configura su apuntador a u .
 - Si u es azul₁, entonces v se colorea como azul₂.
 - Si u es azul₂, entonces v se colorea como rojo y u se recolora como azul₁ cuando T_C regresa a u .

Después de visitar v , T_C regresa a u . Después, u envía un token de búsqueda T_S a través de la arista $\langle u, v \rangle$. Después de que v recibe T_S , si es necesario, realiza recursivamente el mismo procedimiento de u en un nuevo componente. Después de que u recibe T_S a través de la arista $\langle v, u \rangle$, el algoritmo continúa con el procesamiento del componente biconectado K .

- Supóngase que u es la raíz bi-comp de s componentes biconectados nuevos. De la misma forma que los casos previos, el algoritmo detiene temporalmente el procesamiento de K . Entonces, el algoritmo realiza el procesamiento de cada uno de esos s componentes biconectados, uno a la vez (cada componente biconectado puede ser un puente o puede no serlo). Después de finalizar el procesamiento del último componente biconectado (cuando u recibe el último T_S), el algoritmo continúa con el procesamiento de K .

Ahora, se proporciona el análisis de algoritmo CIF-OUTERPLANAR.

4.4 Corrección del algoritmo

Dado que todas las fases del algoritmo CIF-OUTERPLANAR son independientes, se demuestra separadamente la corrección y la complejidad de la fase de exploración en la Sección 4.4.1 y de la fase de coloreado (que incluye la versión extendida) en la Sección 4.4.2.

4.4.1 Corrección y análisis de la fase de exploración

Ahora, se demuestra la corrección de la fase de exploración. Los lemas 7, 8, 9, 10 y el Corolario 9 son resultados auxiliares para el Lema 11, el cual demuestra que la primera subfase de exploración identifica las aristas de salida externas incidentes al líder r y si r es o no un vértice de corte. El Lema 12 y el Corolario 10 indican que la segunda subfase de exploración encuentra todos los vértices de corte y todas las aristas externas e internas de D^L . Los corolarios 11 y 12 establecen las complejidades temporal y de mensajes de esta fase.

Lema 7. Sea r el líder de D^L y sea $\langle r, v_i \rangle$ una de sus aristas de salida. Si r envía

externa cerrada hacia la izquierda, entonces W y W^O deben separarse en algún vértice v_j . Sea $\langle v_j, q \rangle$ la primera arista de W que difiere de aquella en W^O (la cual tiene, en su lugar, la arista $\langle v_j, v_{j+1} \rangle$). Ambas caminatas comparten la arista previa $\langle v_{j-1}, v_j \rangle$ y dado que ambas caminatas siguen la convención hacia la izquierda, entonces la arista que sigue a $\langle v_{j-1}, v_j \rangle$ debe ser la misma en ambas caminatas. Por lo tanto existe una contradicción y W debe ser una caminata externa cerrada hacia la izquierda. \square

Lema 9. Sea r el líder de D^L y sea W una caminata cerrada hacia la izquierda que el primer token de exploración T_1 sigue después de que r lo envía a través de la arista $\langle r, i \rangle$. La arista (r, i) es un puente en G si y solo si T_1 regresa a r a través de la arista $\langle i, r \rangle$.

Demostración.

\Rightarrow Se demuestra por contradicción. Suponga que la arista (r, i) es un puente en G , que r inicia una caminata W hacia la izquierda al enviar un token T_1 a través de la arista $\langle r, i \rangle$, y que T_1 regresa a r por una arista diferente a $\langle i, r \rangle$. Si (r, i) es un puente en G , la única forma de que T_1 regrese a r (después de irse a través de la arista $\langle r, i \rangle$) es a través de la arista $\langle i, r \rangle$. Esta es una contradicción de la suposición inicial.

\Leftarrow Se demuestra por contradicción. Suponga que r inicia la caminata cerrada hacia la izquierda $W = r, i, w_1, w_2, \dots, w_s, i, r$ al enviar T_1 a través de la arista $\langle r, i \rangle$, que T_1 regresa a r a través de la arista $\langle i, r \rangle$, y que la arista (r, i) no es un puente en G .

Dado que la arista (r, i) no es un puente en G , entonces existe una caminata $Q = i, q_1, q_2, \dots, q_t, r$ que va desde i hasta r y que no viaja a través de la arista $\langle i, r \rangle$. Suponga que $w_m = q_z$ es el último vértice que pertenece tanto a W como a Q , respectivamente, antes de que se separen como se muestra en la Figura 14 (este vértice debe existir dado que ambas caminatas contienen al vértice i y ambas llegan a r desde diferentes vértices). Dado que T_1 es un token hacia la izquierda, debe viajar a través de los

vértices r, i, \dots, w_m , y desde w_m a q_{z+1} en lugar de w_{m+1} , lo cual es una contradicción de la suposición inicial. Por lo tanto, (r, i) debe ser un puente en G . \square

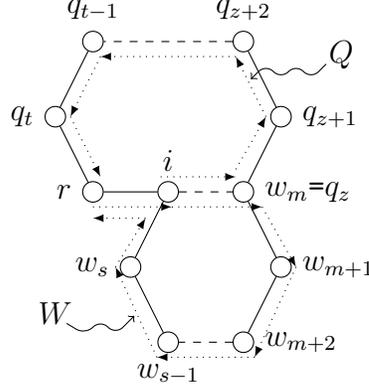


Figura 14. Diagrama auxiliar para la demostración del Lema 9.

Corolario 9. Sea r el líder de D^L . Sea W la caminata cerrada hacia la izquierda que r inicia al enviar el primer token de exploración T_1 a través de la arista $\langle r, i \rangle$. Si la arista (r, i) es un puente en G , entonces W es una caminata externa cerrada hacia la izquierda.

Demostración. Dado que la arista (r, i) es un puente en G , la arista $\langle r, i \rangle$ es una arista de salida externa. Por el Lema 8, W es una caminata externa cerrada hacia la izquierda. \square

Por simplicidad, se utiliza la siguiente notación auxiliar para los lemas 10 y 11. Sea $v^i = vtx_r(i)$ el vértice que está conectado a la i -ésima arista de r .

Lema 10. Sea r el líder de D^L y también un vértice que no es de corte. Sea W_i la caminata cerrada hacia la izquierda que r inicia al enviar el primer token de exploración T_1 a través de $\langle r, v^i \rangle$, para $0 \leq i \leq \{N(r) - 1\}$. Entonces, si $W_x = \max(|W_i|)$, para toda i , entonces $W_x = W^O$, y $\langle r, v^x \rangle$ es la única arista de salida externa incidente a r .

Demostración. Se demuestra por contradicción. Suponga que $W_x = \max(|W_i|)$, para toda i , y $W_x \neq W^O$. Note que W^O visita todas las aristas externas de D^L y

exactamente una arista de salida incidente a r pertenece a W^O . Por lo tanto, una de las caminatas cerradas hacia la izquierda debe ser W^O .

Dado que $W_x \neq W^O$, entonces $|W_x| > |W^O|$. Dado que W^O incluye todas las aristas externas de D^L , entonces W_x debe incluir al menos una arista interna. W_x no puede ser una caminata cerrada hacia la izquierda que consista exclusivamente de aristas internas, porque este tipo de caminatas cerradas hacia la izquierda es a lo más tan grande como W^O (cuando G es un anillo). Consecuentemente, W_x debe incluir al menos una arista externa. Por el Lema 8, cualquier caminata cerrada hacia la izquierda que inicia en una arista externa es una caminata externa cerrada hacia la izquierda y consiste exclusivamente de aristas externas. Por lo tanto, existe una contradicción dado que W_x no puede tener aristas internas. Por lo tanto, $W_x = W^O$. \square

Lema 11. Sea r el líder de D^L y también un vértice que no es de corte. Sea W_i la caminata cerrada hacia la izquierda que r inicia al enviar el primer token de exploración T_1 a través de la arista $\langle r, v^i \rangle$, para $0 \leq i \leq |N(r)| - 1$. Solo una de las siguientes tres expresiones es verdadera:

1. $|N(r)| = 1$, la arista (r, v^0) es un puente en G , $\langle r, v^0 \rangle$ es una arista externa en D^L , y $W_0 = W^O$.
2. $|N(r)| = 2$.
 - Si $|W_0| > |W_1|$, entonces $W_0 = W^O$ y $\langle r, v^0 \rangle$ es una arista de salida externa en D^L .
 - Si $|W_0| < |W_1|$, entonces $W_1 = W^O$ y $\langle r, v^1 \rangle$ es una arista de salida externa en D^L .
 - Si $|W_0| = |W_1|$, entonces $W_0 = W^O$, $\langle r, v^0 \rangle$ es una arista de salida externa en D^L , y G es un anillo.

3. $|N(r)| \geq 3$. Si $W_x = \max(|W_i|)$, para toda i , entonces $W_x = W^O$, y $\langle r, v^x \rangle$ es una arista de salida externa en D^L . Adicionalmente, para cada caminata cerrada hacia la izquierda que r inicia al enviar el primer token de exploración T_1 a través de la arista $\langle r, v^i \rangle$, el token T_1 regresa a r a través de la arista $\langle v^{(i+1) \bmod |N(r)|}, r \rangle$.

Demostración. Se demuestra cada expresión por separado

1. Dado que $|N(r)| = 1$, entonces v^0 es el único vecino de r . Si se remueve la arista (r, v^0) de G , r se desconecta de v^0 , y así la arista (r, v^0) es un puente en G . Dado que la arista (r, v^0) es un puente en G , la arista $\langle r, v^0 \rangle$ es una arista de salida externa en D^L . Por el Lema 8, $W_0 = W^O$.
2. Dado que $|N(r)| = 2$ y que r no es un vértice de corte, entonces las aristas (r, v^0) y (r, v^1) no son puentes en G . Dado que estas aristas no son puentes, por los lemas 7 y 9, cualquier caminata cerrada hacia la izquierda, que inicia en r cuando éste envía un token a través de la arista $\langle r, v^0 \rangle$ (o $\langle r, v^1 \rangle$), debe llegar a la arista $\langle v^1, r \rangle$ (o $\langle v^0, r \rangle$). Por el Lema 10, si $W_x = \max(|W_0|, |W_1|)$, entonces $W_x = W^O$ y $\langle r, v^x \rangle$ es una arista de salida externa en D^L . Cuando $|W_0| = |W_1|$, el grafo G es un anillo y $|W^O| = |W^I|$, donde W^I es la caminata hacia la izquierda más larga que consiste exclusivamente de aristas internas.
3. Dado que $W_x = \max(|W_i|)$, para toda i , entonces por el Lema 10, $W_x = W^O$ y $\langle r, v^x \rangle$ es una arista de salida externa D^L .

Se demuestra por contradicción el resto de la Expresión 3.

Suponga, que existe una caminata hacia la izquierda W_i que r inicia al enviar el primer token de exploración T_1 a través de la arista $\langle r, v^i \rangle$, y que tal caminata regresa a r por una arista diferente de $\langle v^{(i+1) \bmod |N(r)|}, r \rangle$.

Dado que r no es un vértice de corte y que $|N(r)| \geq 3$, entonces (r, v^i) no puede ser un puente en G ; por el Lema 9, el token T_1 no puede regresar a través de la arista $\langle v^i, r \rangle$.

Por lo tanto, suponga que T_1 regresa a través de la arista $\langle v^{(i+j) \bmod |N(r)|}, r \rangle$, donde $j \notin \{0, 1\}$. Por la suposición inicial, existe una caminata cerrada hacia la izquierda que inicia en $\langle r, v^i \rangle$ y termina en $\langle v^{(i+j) \bmod |N(r)|}, r \rangle$, para algún $j \notin \{0, 1\}$.

Para mostrar la existencia de una contradicción, es necesario analizar las posiciones posibles del vértice $v^{(i+1) \bmod |N(r)|}$ en el plano. Existen tres casos:

Caso a. Cuando el vértice $v^{(i+1) \bmod |N(r)|}$ está dentro de una cara limitada por las aristas recorridas por W_i . Existe una contradicción, dado que se supone que G es outerplanar y, en tales grafos, cada vértice de G toca la frontera del grafo.

Caso b. Cuando el vértice $v^{(i+1) \bmod |N(r)|}$ está en la frontera definida por W_i , *i.e.*

$$W_i = r, v^i, q_1, \dots, q_s, v^{(i+1) \bmod |N(r)|}, q_{s+1}, \dots, q_{s+t}, v^{(i+j) \bmod |N(r)|}, r.$$

Existe una contradicción, dado que $Q_i = r, v^i, q_1, \dots, q_s, v^{(i+1) \bmod |N(r)|}, r$ define una cara interna de G y un token debería ir a través de Q_i en lugar de W_i . La Figura 15 ilustra este caso.

Caso c. Cuando el vértice $v^{(i+1) \bmod |N(r)|}$ está afuera de la cara acotada por las aristas recorridas por W_i . Dado que W_i es una caminata cerrada hacia la izquierda, no existe un camino que vaya desde un vértice diferente de r en W_i a $v^{(i+1) \bmod |N(r)|}$. Esto implica que r es un vértice de corte, lo cual es una contradicción.

Existen contradicciones en todos los casos. Por lo tanto, el token T_1 debe regresar a través de la arista $\langle v^{(i+1) \bmod |N(r)|}, r \rangle$. \square

Observación 2. Note que en cada r -componente C_i de G (ver la Sección 4.2), el

\Leftarrow Se demuestra por contradicción. Suponga que un vértice $v_{i,k} \neq r$, $v_{i,k}$ aparece al menos dos veces en $W_{C_i}^O$, y que $v_{i,k}$ no es un vértice de corte. Sea $W_k = v_{i,k}, w_{i,1}, w_{i,2}, \dots, w_{i,t}, v_{i,k}$ una caminata externa cerrada hacia la izquierda propia de $W_{C_i}^O$ que inicia y finaliza en $v_{i,k}$. Sea $v_{i,k+1}$ el siguiente vértice de $v_{i,k}$ después del final de W_k . Dado que $v_{i,k}$ no es vértice de corte, existe una caminata $Q = w_{i,t}, q_1, \dots, q_s, v_{i,k+1}$ que va desde $w_{i,t}$ hasta $v_{i,k+1}$ sin pasar a través de $v_{i,k}$. Dado que T_2 es un token hacia la izquierda, debe viajar a través del siguiente camino $\dots, v_{i,k-1}, v_{i,k}, w_{i,1}, w_{i,2}, \dots, w_{i,t}, q_1, \dots, q_s, v_{i,k+1}, \dots$, lo cual es una contradicción dado que se supuso que T_2 visita $v_{i,k}$ dos veces. Por lo tanto, $v_{i,k}$ debe ser un vértice de corte (La Figura 16 ilustra este caso). \square

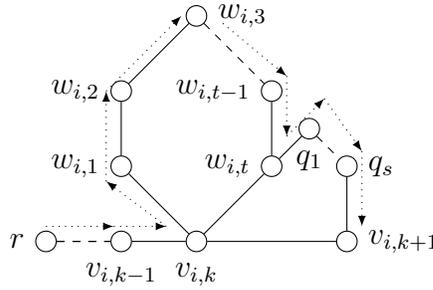


Figura 16. Diagrama auxiliar para la demostración del Lema 12.

Corolario 10. La segunda subfase de exploración identifica todas las aristas externas e internas de D^L .

Demostración. Por el Lema 8, cuando r envía el segundo token de exploración T_2 para generar la caminata $W_{C_i}^O$ en un r -componente C_i , T_2 viaja exclusivamente y a través de aristas externas del r -componente C_i . Por lo tanto, si T_2 viaja a través de la arista $\langle u, v \rangle$, esta arista es una arista externa en C_i ; de otra forma, es una arista interna. Dado que r realiza este procedimiento para todos los r -componentes del grafo de entrada, la fase de exploración identifica todas las aristas externas e internas de D^L . \square

Corolario 11. La fase de exploración calcula el grafo D^E en $O(n)$ pasos.

Demostración. Durante la primera subfase, el token T_1 recorre cada arista de D^L a lo más una vez antes de que desaparezca. Un procedimiento similar ocurre durante la segunda subfase de exploración cuando r envía el token T_2 . Por lo tanto, el tiempo de ejecución de esta fase es $O(n)$ pasos, dado que un grafo outerplanar con n vértices tiene $O(n)$ aristas. \square

Corolario 12. La fase de exploración requiere $O(n)$ mensajes.

Demostración. Cada vez que un token visita una arista es debido a que el algoritmo generó un mensaje. Por lo tanto, la fase de exploración genera $O(n)$ mensajes. \square

4.4.2 Corrección y análisis de la fase de coloreado

Ahora, se demuestra la corrección de la fase de coloreado para dos casos. Primero, cuando G es un componente biconectado que no contiene puentes y tiene m ciclos (ver la Sección 4.3.2). Segundo, cuando G tiene vértices de corte (ver la Sección 4.3.3). En ambos casos, se demuestra que el algoritmo genera un CIF maximal en G .

Lema 13: Sea r el líder de D^E . La fase de coloreado genera un CIF maximal en G , cuando G es un componente biconectado sin puentes.

Demostración. Se prueba por inducción en el número de ciclos en G .

Caso base: Considerar el primer ciclo $C_0 = v_1, v_2, \dots, v_j, v_1$ de G , donde $r = v_1$. Nótese que el algoritmo inicialmente colorea el vértice r como rojo. Primero, r crea un token de conteo T_E que recorre C_0 ; después de que r recibe T_E , obtiene el número j de vértices en C_0 . Posteriormente, el algoritmo asigna una secuencia de colores a los vértices de C_0 como sigue. rojo, azul₁, azul₁, rojo, ..., para los primeros $j - 4$ vértices de C_0 .

Nótese que la asignación de los primeros $j - 4$ vértices de C_0 genera un CIF maximal en esos vértices. Para asegurar un CIF maximal en C_0 , la asignación de colores en $v_{j-3}, v_{j-2}, v_{j-1}$ y v_j debe ser cuidadosa. El coloreado de estos cuatro vértices depende del valor de j y el algoritmo usa las reglas de la Sección 4.3.2 para asignar los colores a esos vértices. Es sencillo comprobar que estas reglas siempre generan un CIF maximal en un ciclo. Note que un CIF maximal en C_0 también es máximo.

Hipótesis de inducción: Supóngase que el algoritmo calcula un CIF maximal para los primeros k ciclos de G , donde $k < m$.

Paso inductivo. Sea $C_k = w_1, w_2, \dots, w_h, w_1$ el $k + 1$ -ésimo ciclo de G , donde w_1 es la raíz de ciclo. Nótese que C_k comparte exactamente una arista, (w_1, w_h) , con el ciclo C_s , para algún s , donde $0 \leq s \leq k - 1$. Por la hipótesis inductiva los vértices w_1 y w_h ya están coloreados. Primero, w_1 genera un token de conteo T_E que recorre C_k ; después w_1 recibe T_E , el vértice w_1 obtiene el número de vértices h en C_k y el color de w_h . Después, el algoritmo asigna una secuencias de colores, iniciando en w_2 y terminando en w_{h-4} , similarmente a la secuencia del caso base, pero considerando el coloreado de w_1 . Similarmente al caso base para asignar color a los vértices $w_{h-3}, w_{h-2}, w_{h-1}$ y recolorear w_h (en caso de ser necesario), el algoritmo considera el valor de h y los colores de w_1 y w_h (utilizando las reglas de la Sección 4.3.2). Similarmente al caso base, es fácil comprobar que estas reglas siempre generan un CIF maximal en estos últimos vértices y el algoritmo genera un CIF maximal en el ciclo C_k . \square

Cuando G tiene vértices de corte, se puede extender la demostración del Lema 13 como sigue. Hay dos casos a considerar.

Caso 1. Cuando el vértice de corte u descubre un nuevo componente biconectado sin puentes. El vértice u se comporta como la raíz de un nuevo componente biconectado y el algoritmo colorea sus vértices como se muestra en el Lema 13. La única diferencia

es que ahora u no necesariamente es rojo.

Caso 2. Cuando el vértice de corte u descubre el puente (u, v) . Dado que u está coloreado, la asignación del color en v es trivial. \square

Corolario 13. La fase de coloreado calcula el grafo G^C en $O(n)$ pasos.

Demostración. Durante la subfase de conteo, el token T_E viaja a través de cada arista de D^E a lo más una vez antes de desaparecer. Un procedimiento similar ocurre durante la subfase rojo/azul y durante la subfase de búsqueda cuando r envía tokens T_C y T_S , respectivamente. Por lo tanto, el tiempo de ejecución de esta fase es $O(n)$ pasos, dado que un grafo outerplanar con n vértices tiene $O(n)$ aristas. \square

Corolario 14. La fase de coloreado requiere $O(n)$ mensajes.

Demostración. Cada vez que un token visita una arista es debido a que el algoritmo generó un mensaje. Por tanto el algoritmo generó $O(n)$ mensajes. \square

El Teorema 3 resume los resultados de este capítulo.

Teorema 3. El algoritmo CIF-OUTERPLANAR con la extensión descrita en la Sección 4.3.3, para la Fase de coloreado, calcula en $O(n)$ pasos un CIF maximal cuando el grafo de entrada es un grafo outerplanar geométrico conectado con n vértices.

Conclusiones

En este capítulo se presentan los resultados y conclusiones del presente trabajo de tesis y se proponen nuevas actividades y problemáticas a resolver.

Conclusiones

El presente trabajo de tesis se presentan en tres etapas. La primera etapa consistió en diseñar un algoritmo auto-estabilizante para la elección de líder (ver Capítulo II). El algoritmo ELECCION-LIDER utiliza un esquema de calendarización distribuido síncrono; es decir, todos los vértices privilegiados en la ronda i se ejecutan en dicha ronda. Aunque el algoritmo ELECCION-LIDER es óptimo, dado que la complejidad temporal es de $O(diam)$ rondas, este algoritmo no converge bajo la suposición del calendarizador adversario. El calendarizador adversario solo habilitaría la regla RESET (la regla que descontamina aquellos vértices que reconocen a un líder ficticio) en caso de ser la única regla privilegiada. De esta forma el procedimiento del algoritmo ELECCION-LIDER que, en el esquema de calendarizador síncrono, habilita la regla RESET dos veces más rápido que cualquier otra regla, no es válido en el esquema de calendarizador adversario.

La segunda etapa consistió en diseñar dos algoritmos auto-estabilizante para calcular el conjunto independiente fuerte (CIF) en un grafo anillo y en un grafo cactus respectivamente (ver Capítulo III). La complejidad temporal de dicho algoritmo es $O(diam)$ rondas, que en el caso del anillo y del cactus es $O(n)$. No es posible que estos algoritmos se ejecuten en tiempo sublineal, dado que utilizan la elección de un líder y Awerbuch (1987) demuestra que se requieren al menos $\Omega(n)$ rondas para elegir un líder. El algoritmo CIF-CACTUS considera todas las configuraciones posibles en los cierres de cadena. No es tan claro que este mecanismo se pueda extender a los grafos

outerplanares, dado que el número de configuraciones posibles en los cierres de cadena se incrementa considerablemente.

La tercera etapa consistió en diseñar un algoritmo distribuido para calcular un CIF maximal en un grafo outerplanar geométrico (ver Capítulo IV) en un número lineal de pasos. El algoritmo CIF-OUTERPLANAR identifica los vértices de corte y los puentes de G y valida si G es o no outerplanar. Con una modificación simple, el algoritmo también genera una descomposición de orejas de G . Al igual que los algoritmos auto-estabilizantes para el CIF maximal, una limitante de este algoritmo es que requiere elegir un líder, por lo que el algoritmo no puede hacerse en tiempo sublineal. Incluso si se tuviera un líder, el algoritmo CIF-OUTERPLANAR no toma ventaja del paralelismo implícito en los sistemas distribuidos ya que requiere el envío de tokens que habilitan la ejecución del algoritmo sólo cuando un vértice recibe el token. Otra limitante del algoritmo es que supone que el grafo es geométrico. Sin esa restricción los vértices del grafo no sabrían a cuál de sus vecinos conviene enviar el token en cualquiera de las fases del algoritmo.

Trabajo futuro

Existen diversos retos a seguir como trabajo futuro. A continuación se explican brevemente los principales problemas para trabajo futuro.

- En el Capítulo III se diseñaron dos algoritmos auto-estabilizantes para calcular un CIF máximo en un anillo y un CIF maximal en un cactus. Dichos algoritmos funcionan correctamente bajo la suposición de que el calendarizador es síncrono. Se pretende como trabajo futuro extender estos algoritmos hacia un esquema de calendarización adversario para analizar si se mantiene la complejidad temporal

al utilizar este esquema. La idea es utilizar tres fases mutuamente excluyentes que funcionen bajo el esquema del calendarizador adversario. La primera fase (fase de elección de líder) consiste en adaptar el algoritmo de elección de líder de Datta *et al.* (2011). Se utiliza dicho algoritmo ya que es óptimo en tiempo y espacio y considera el calendarizador adversario. Se propone una fase de cerradura que permita identificar aquellos vértices que se encuentran en un cierre de cadena. Finalmente, se propone una fase de optimización que tome como entrada la salida de la fase de cerradura y calcule un CIF máximo para un grafo anillo y un CIF maximal en un grafo cactus. Se conjetura que es posible diseñar un algoritmo que calcule un CIF maximal en un cactus en $O(n)$ rondas para el calendarizador adversario. Se conjetura que dicho algoritmo calcula el CIF máximo si el cactus de entrada es un anillo.

- En el Capítulo IV se presenta un algoritmo distribuido que en $O(n)$ pasos calcula un CIF maximal en un grafo outerplanar geométrico. Se propone analizar la posibilidad de diseñar un algoritmo distribuido que calcule un CIF maximal en un outerplanar en tiempo sublineal. La idea se basa en utilizar la técnica de descomposición de redes propuesta en Barenboim y Elkin (2010). La dificultad de esta propuesta es diseñar un mecanismo para poder visualizar información a una distancia dos utilizando mensajes de tamaño reducido.
- Diseñar un algoritmo auto-estabilizante para encontrar un CIF maximal en un grafo outerplanar geométrico en $O(n)$ rondas. Se propone utilizar tres fases independientes. Para la fase de elección de líder puede utilizarse el algoritmo del Capítulo II. Es necesario analizar el uso de tokens para las fases de exploración y para la fase de coloreado. Este algoritmo es interesante porque también aportaría

información importante acerca del grafo outerplanar, tal como la conectividad del mismo.

Además del trabajo futuro descrito en los renglones anteriores, existen diversos problemas abiertos a explorar a futuro. Dichos problemas se enfocan a diseñar algoritmos para encontrar un CIF máximo en topologías más generales que el anillo.

Referencias bibliográficas

- Afek, Y. y Bremler, A. (1997). Self-stabilizing unidirectional network algorithms by power-supply. En *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, páginas 111–120. Society for Industrial and Applied Mathematics.
- Agnarsson, G. y Halldórsson, M. M. (2004). On colorings of squares of outerplanar graphs. En *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, páginas 244–253. Society for Industrial and Applied Mathematics, SIAM.
- Antonoiu, G. y Srimani, P. K. (1997). Self-stabilization: A new paradigm for fault tolerance in distributed algorithm design. *Colorado State University, Computer Science Technical Report*.
- Arora, A. y Gouda, M. (1994). Distributed reset. *Computers, IEEE Transactions on*, **43**(9): 1026–1038.
- Attiya, H. y Welch, J. (2004). *Distributed Computing: Fundamentals, Simulations and Advanced Topics*.
- 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*, páginas 230–240. ACM.
- Awerbuch, B., Luby, M., Goldberg, A., y Plotkin, S. A. (1989). Network decomposition and locality in distributed computation. En *Foundations of Computer Science, 30th Annual Symposium on*, páginas 364–369. IEEE.
- Barenboim, L. y Elkin, M. (2009). Distributed $(\delta+1)$ -coloring in linear (in δ) time. En *Proceedings of the 41st annual ACM symposium on Theory of computing*, páginas 111–120. ACM.
- Barenboim, L. y Elkin, M. (2010). Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition. *Distributed Computing*, **22**(5-6): 363–379.
- 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, editores, *Algorithms and Computation*, páginas 693–703. Springer.
- Berard, B., Beauquier, J., Beauquier, J., Fribourg, L., Fribourg, L., Magniette, F., y Magniette, F. (2000). Proving convergence of self-stabilizing systems using first-order rewriting and regular languages. *Distributed Computing*, **14**(14): 83–95.
- Brukman, O., Dolev, S., Haviv, Y., y Yagel, R. (2007). Self-stabilization as a foundation for autonomic computing. En *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*, páginas 991–998. IEEE.

- Butenko, S. (2003). *Maximum independent set and related problems, with applications*. Tesis de doctorado. University of Florida.
- Cameron, K. y Edmonds, J. (2005). Finding a strong stable set or a meyniel obstruction in any graph. En S. Felsner, editor, *European Conference on Combinatorics, Graph Theory and Applications*, páginas 203–206.
- Chávez, E., Dobrev, S., Kranakis, E., Opatrny, J., Stacho, L., y Urrutia, J. (2006). Route discovery with constant memory in oriented planar geometric networks. *Networks*, **48**(1): 7–15.
- Datta, A. K., Gradinariu, M., y Tixeuil, S. (2000). Self-stabilizing mutual exclusion using unfair distributed scheduler. En *IPDPS*, páginas 465–. IEEE Computer Society.
- Datta, A. K., Larmore, L. L., y Vemula, P. (2011). Self-stabilizing leader election in optimal space under an arbitrary scheduler. *Theoretical Computer Science*, **412**(40): 5541–5561.
- Diestel, R. (2005). *Graph Theory*. Springer.
- Dijkstra, E. W. (1974). Self-stabilizing systems in spite of distributed control. *Commun. ACM*, **17**(11): 643–644.
- Dolev, S. (2000). *Self-stabilization*. MIT press.
- Dolev, S. y Herman, T. (1995). Superstabilizing protocols for dynamic distributed systems. En *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, página 255. ACM.
- Dolev, S. y Yagel, R. (2005). Memory management for self-stabilizing operating systems. En *Self-Stabilizing Systems*, Lecture Notes in Computer Science, páginas 113–127. Springer.
- Fribourg, L., Messika, S., y Picaronny, C. (2006). Coupling and self-stabilization. *Distrib. Comput.*, **18**(3): 221–232.
- Funk, P. y Zinnikus, I. (2002). Self-stabilization as multiagent systems property. En *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3*, páginas 1413–1414. ACM.
- Gairing, M., Hedetniemi, S. T., Kristiansen, P., y McRae, A. A. (2003). Self-stabilizing algorithms for k-domination. En *Self-Stabilizing Systems*, Vol. 2704 de *Lecture Notes in Computer Science*, páginas 49–60. Springer.
- 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.
- Ghosh, S. (2006). *Distributed systems: an algorithmic approach*. CRC press.

- Hale, W. (1980). Frequency assignment: theory and applications. *Proceedings of the IEEE*, **68**(12): 1497–1514.
- Haynes, T., Hedetniemi, S., y Slater, P. (1998). Domination in graphs: Advanced topics. 1998. *Marcel Decker*.
- Hedetniemi, S., Hedetniemi, S. T., Jacobs, D., y Srimani, P. (2003). Self-stabilizing algorithms for minimal dominating sets and maximal independent sets. *Comput. Math. Appl*, **46**: 805–811.
- Hedetniemi, S. M., Hedetniemi, S. T., Jiang, H., Kennedy, K., y McRae, A. A. (2012). A self-stabilizing algorithm for optimally efficient sets in graphs. *Inf. Process. Lett.*, **112**(16): 621–623.
- Hochbaum, D. S. y Shmoys, D. B. (1985). A best possible heuristic for the k-center problem. *Mathematics of operations research*, **10**(2): 180–184.
- Horváth, T., Ramon, J., y Wrobel, S. (2010). Frequent subgraph mining in outerplanar graphs. *Data Min. Knowl. Discov.*, **21**(3): 472–508.
- Jain, A. y Gupta, A. (2005). A distributed self-stabilizing algorithm for finding a connected dominating set in a graph. En *PDCAT '05: Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies*.
- JáJá, J. (1992). *An introduction to parallel algorithms*. Addison Wesley.
- Kariv, O. y Hakimi, S. L. (1979). An algorithmic approach to network location problems. ii: The p-medians. *SIAM Journal on Applied Mathematics*, **37**(3): 539–560.
- Kazmierczak, A. y Radhakrishnan, S. (2000). An optimal distributed ear decomposition algorithm with applications to biconnectivity and outerplanarity testing. *IEEE Trans. Parallel Distrib. Syst.*, **11**(2): 110–118.
- Kessels, J. L. W. (1988). An exercise in proving self-stabilization with a variant function. *Inf. Process. Lett.*, **29**(1): 39–42.
- Koontz, W. (1980). Economic evaluation of loop feeder relief alternatives. *Bell System Technical Journal*, **59**(3): 277–293.
- Korman, A., Sereni, J.-S., y Viennot, L. (2011). Toward more localized local algorithms: removing assumptions concerning global knowledge. En *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, páginas 49–58. ACM.
- Kranakis, E., Singh, H., y Urrutia, J. (1999). Compass routing on geometric networks. En *Proceedings of the 11th Canadian Conference on Computational Geometry, UBC, Vancouver, British Columbia, Canada*, páginas 1–4. CCCG.

- Kuhn, F. (2009). Weak graph colorings: distributed algorithms and applications. En *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, páginas 138–144. ACM.
- Lan, Y.-F. y Wang, Y.-L. (2000). An optimal algorithm for solving the 1-median problem on weighted 4-cactus graphs. *European Journal of Operational Research*, **122**(3): 602–610.
- Manne, F. y Mjelde, M. (2006). A memory efficient self-stabilizing algorithm for maximal k-packing. En *Stabilization, Safety, and Security of Distributed Systems*, Lecture Notes in Computer Science, páginas 428–439. Springer.
- Mjelde, M. (2004). K-packings and k-dominations on tree graphs. Tesis de maestría. University of Bergen, Norway.
- Pach, J. (1991). Notes on geometric graph theory. *Discrete and Computational Geometry: Papers from DIMACS special year*, **6**: 273–285.
- Panconesi, A. y Srinivasan, A. (1996). On the complexity of distributed network decomposition. *Journal of Algorithms*, **20**(2): 356–374.
- Paten, B., Diekhans, M., Earl, D., John, J. S., Ma, J., Suh, B., y Haussler, D. (2011). Cactus graphs for genome comparisons. *Journal of Computational Biology*, **18**(3): 469–481.
- Peleg, D. (2000). *Distributed computing: a locality-sensitive approach*, Vol. 5. SIAM.
- Schneider, J. y Wattenhofer, R. (2010). An optimal maximal independent set algorithm for bounded-independence graphs. *Distributed Computing*, **22**(5-6): 349–361.
- Schneider, M. (1993). Self-stabilization. *ACM Comput. Surv.*, **25**(1): 45–67.
- Shi, Z. (2012). A self-stabilizing algorithm to maximal 2-packing with improved complexity. *Inf. Process. Lett.*, **112**(13): 525–531.
- Shi, Z., Goddard, W., y Hedetniemi, S. T. (2004). An anonymous self-stabilizing algorithm for 1-maximal independent set in trees. *Inf. Process. Lett.*, **91**(2): 77–83.
- Shukla, S., Rosenkrantz, D., y Ravi, S. (1994). Developing self-stabilizing coloring algorithms via systematic randomization. En *Proceedings of the International Workshop on Parallel Processing*, páginas 668–673.
- Tel, G. (2000). *Introduction to distributed algorithms*. Cambridge university press.
- Theel, O. (2001). A new verification technique for self-stabilizing distributed algorithms based on variable structure systems and l-japunov theory. *Hawaii International Conference on System Sciences*, **9**: 9069.

- 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*, páginas 863–871.
- Trejo-Sánchez, J. A. y Fernández-Zepeda, J. A. (2014). Distributed algorithm for the maximal 2-packing in geometric outerplanar graphs. *J. Parallel Distrib. Comput.*, **74**(3): 2193–2202.
- Tsin, Y. H. (2004). On finding an ear decomposition of an undirected graph distributively. *Inf. Process. Lett.*, **91**(3): 147–153.
- Turau, V. (2007). Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler. *Inf. Process. Lett.*, **103**(3): 88–93.
- Turau, V. (2012). Efficient transformation of distance-2 self-stabilizing algorithms. *J. Parallel Distrib. Comput.*, **72**(4): 603–612.
- Winter, P. (1986). Generalized steiner problem in series-parallel networks. *J. Algorithms*, **7**(4): 549–566.
- Zmazek, B. y Žerovnik, J. (2004). The obnoxious center problem on weighted cactus graphs. *Discrete applied mathematics*, **136**(2): 377–386.