VNIVERSIDAD NACIONAL
AVTNOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

Algoritmos locales para detectar conjuntos de corte

Que para optar por el grado de:
Maestro en ciencias (computación)

Presenta:
Josué Manuel Rivera Velázquez

Tutor:
Dr. Jorge Urrutia Galicia
Instituto de Matemáticas

Co-Tutor:
Dr. José David Flores Peñaloza
Facultad de Ciencias

MÉXICO, D. F. MAYO 2014

Capítulo 1

Introducción

En años recientes se ha prestado mucha atención al estudio de problemas en gráficas geométricas, debido a que se ha comprobado que muchas situaciones de la vida real pueden ser representadas y analizadas bajo este modelo. Por otro lado, se ha observado que las gráficas geométricas son ideales para modelar redes de computadoras, en las que la posición es conocida, tal como redes *ad hoc*, redes celulares y redes de sensores [18].

Estas redes se caracterizan por tener un alto grado de movilidad, por lo que normalmente se catalogan como redes naturalmente dinámicas. Como consecuencia, la aplicación de algoritmos con conocimiento global de la red empiezan a ser ineficientes para este tipo de escenarios [15].

Muchos algoritmos trabajan bajo la suposición de que se cuenta con información global y centralizada sobre la topología de una red [20]. Para los fines de este trabajo no consideraremos algoritmos de este tipo, ya que el uso de memoria y el mantenimiento de información cuando la red cambia de forma dinámica es muy costoso. Por otro lado se pierde la esencia de sistemas distribuidos al centralizar cómputo en uno o pocos nodos de la red. Más aún, la construcción y mantenimiento distribuido (local por nodo) es preferible debido a los recursos limitados y la movilidad de los nodos [20].

Un algoritmo local se modela como un agente con memoria constante, que tiene la capacidad de migrar de un nodo a otro, y que toma decisiones basado sólo en la información que contiene en su memoria, y la ubicación y conectividad del nodo en el que se encuentra.

Los algoritmos locales son algoritmos distribuidos, lo cuales son altamente escalables, además de ser tolerantes a fallas y cambios sobre la topología de la red [16]. Debido a esto, los algoritmos locales son empleados para mantener una solución factible en redes altamente dinámicas. Por otro lado, pese a ser tolerantes a fallas, una de las restricciones que tienen los algoritmos locales es la falla del nodo en el que se encuentran, lo cual des-

conectaría por completo al agente de la gráfica.

Si los nodos de una gráfica geométrica plana conocen su posición en el plano, muchos problemas pueden ser resueltos mediante algoritmos locales [8], tales como ruteo, donde se tiene un agente que desea viajar de un nodo u a un nodo v sólo recordando una cantidad constante de información, sin necesidad de conocer la topología de la red en su totalidad [4]. También, las soluciones de muchos otros problemas clásicos en gráficas tales como encontrar un conjunto dominante, la elección de un líder, o el coloreo de una gráfica, han logrado ser trasladadas a algoritmos locales.

En este trabajo se presenta una propuesta de solución al problema de detección de conjuntos de corte en una gráfica, mediante la utilización de algoritmos locales. Con esto, se intenta dar una solución distribuida y altamente escalable a este problema tan bien conocido.

En una gráfica, una arista de corte es una arista que al eliminarse incrementa el número de componentes conexas de esta. Un conjunto de aristas de corte o conjunto de corte, es un conjunto de aristas que, al ser eliminado desconecta a la gráfica, es decir, incrementa su número de componentes conexas.

Sea G una gráfica geométrica plana, la cual modela un sistema computacional donde sus elementos tienen un alto grado de movilidad, por ejemplo, una red de sensores, una red de celulares o una red *ad hoc*.

Problema 1. Encontrar el o los conjuntos de corte de cardinalidad mínima en G de manera local.

Este problema es de gran importancia dentro del estudio de teoría de gráficas, ya que los conjuntos de corte nos ayudan a detectar el grado de conectividad de una gráfica, es decir, cuantas aristas necesitan ser removidas para generar nuevas componentes conexas. Además, es importante señalar que la conexidad de una red se asocia a la vulnerabilidad de la misma. Por lo tanto, si logramos encontrar en una gráfica que el conjunto de corte de cardinalidad mínima contiene tres elementos por ejemplo, estamos asegurando que al romper esas tres conexiones, la comunicación en la red se verá afectada de forma dramática. Más aún, con este resultado, estamos garantizando que, al desconectar cualesquiera dos aristas, la red seguirá siendo conexa, manteniendo integra la comunicación entre todos sus nodos.

En este trabajo se presenta una solución al problema 1 mediante el desarrollo de un grupo de algoritmos locales, cuya combinación y desempeño en conjunto sigue siendo local.

Como se sabe, en toda gráfica plana existe por lo menos un vértice de grado menor o igual a 5 [3], por lo tanto, para el peor de los casos, el conjunto de corte de cardinalidad mínima será de tamaño 5. Por esta razón, sólo se presentarán algoritmos para la detección de conjuntos de corte de cardinalidad menor o igual a 5.

En la sección 2 se presenta una definición más detallada de algoritmos locales, así como algunos algoritmos que sirvieron como marco referencial para este trabajo. En la sección 3 se presenta el marco teórico, las definiciones y teoremas que se utilizan como bases, y finalmente en la sección 4 se presentan los algoritmos de identificación de conjuntos de corte, los cuales son nuestra propuesta de solución al problema 1.

Capítulo 2

Algoritmos locales

Un *algoritmo local* se modela como un agente con memoria constante, que tiene la capacidad de migrar de un nodo a otro, y que toma decisiones basado sólo en la información que contiene en su memoria, y la ubicación y conectividad del nodo en el que se encuentra. Por lo tanto, todo nodo o vértice de una gráfica sólo tiene información de vértices que se encuentran a un salto de distancia de él [18]. Los algoritmos locales son altamente escalables y robustos a cambios en la red, además pueden ser usados como subrutinas en el diseño de algoritmos centralizados y distribuidos [16].

Sea G una gráfica plana. Asuma que cada vértice v en G conoce su posición en la gráfica (sus coordenadas) y la posición de sus vecinos. Existe un algoritmo determinístico que permitirá a un agente A colocado en un vértice u viajar a un vértice v bajo las siguientes condiciones:

- A tiene memoria constante. En todo momento A conoce sólo la posición de u y v, y la posición de un número constante de nodos.
- Cuando el agente visita un vértice w de G, este puede usar la lista de vértices adyacentes a w (y sus posiciones).
- A no deja marcas en su recorrido.

Para ser más claros en la primera restricción, bajo esta condición A tiene un espacio de memoria constante, en la cual puede almacenar un número constante de posiciones de los elementos de G. Es fácil ver que esto restringe a que el agente nunca posea el conocimiento global de G [18].

Podría parecer que algunas de las restricciones anteriores son innecesarias, sin embargo, si consideramos el comportamiento de redes tan grandes como Internet, entonces éstas se convierten claramente en condiciones relevantes.

Por ejemplo, si un mensaje deja una marca cada vez que pasa a través de un nodo en una red, toda la memoria disponible en el nodo sería rápidamente consumida, de aquí la necesidad de la tercera condición.

2.1. Redes de sensores y redes *ad hoc*

La razón por la cual este tipo de redes son mencionadas en este trabajo es muy simple, los algoritmos que se emplean para el manejo y control de estas redes, por las características propias de la red, necesitan ser locales.

Redes donde los nodos son pequeños sensores, conocida como *redes de sensores*, permiten monitorear y analizar complejos fenómenos sobre una región muy extensa y por un periodo de tiempo relativamente largo. Recientes avances tecnológicos han permitido la construcción de sensores más pequeños y mucho más baratos, los cuales pueden obtener una gran cantidad de información acerca de fenómenos físicos, tales como la temperatura, la humedad, presión, niveles de ruido, niveles de dióxido de carbono, niveles de oxígeno, campos magnéticos, velocidades de los objetos, dirección, tamaño, la presencia de objetos, etc. Esta gran variedad de opciones permiten que los sensores sean usados en un gran número de aplicaciones, y bajo una enorme cantidad de posibles escenarios. El monitoreo ambiental, monitoreo de la salud, vigilancia militar, vigilancia de maquinaria industrial y automatización de inmuebles son sólo algunas de las aplicaciones más recientes, en las cuales las redes de sensores han sido la herramienta principal [19].

El criterio de diseño y requerimientos de una red de sensores varía según el uso o la aplicación para la cual se desea emplear. Sin embargo, algunos de los requerimientos típicos en las redes de sensores son:

- Tolerancia a fallas. Las redes de sensores son propensas a la falla de los nodos. Por esto, es crucial que los algoritmos empleados para la administración de estas redes puedan lidiar con este problema de manera eficiente.
- Escalabilidad. Gracias a que los sensores son cada vez más baratos, es altamente probable que las redes de sensores consistan en un enorme número de nodos. Por lo tanto, los algoritmos en redes de sensores deben ser también escalables, teniendo en cuenta que las redes en las que se van a emplear pueden tener miles e incluso decenas de miles de nodos.
- Simplicidad. Debido al diseño de los sensores, los recursos disponibles para la aplicación de los algoritmos están severamente restringidos. Como consecuencia, los algoritmos para redes de sensores deben considerar la capacidad de cómputo, así como el espacio de memoria restringida.

- Independencia de sus operaciones. Las redes de sensores usualmente son utilizadas en regiones complicadas, donde la atención individual de los nodos es casi imposible. Por esto, se necesita que los algoritmos en redes de sensores trabajen de forma oportuna e independiente, sin necesidad de recibir constantemente indicaciones.
- Dinamismo. Los nodos deben adaptarse a cambios repentinos en su entorno, por ejemplo, cambios en su conectividad.

Por otro lado, una *red ad hoc* es un tipo de red descentralizada. La red es *ad hoc* porque no depende de una infraestructura pre-existente, como *routers* (en redes cableadas) o de puntos de accesos en redes inalámbricas administradas. En lugar de ello, cada nodo participa en el encaminamiento mediante el reenvío de datos hacia otros nodos [17].

Una red *ad hoc* se refiere típicamente a cualquier conjunto de redes donde todos los nodos tienen el mismo estado dentro de la red, además, son libres de asociarse con cualquier otro dispositivo de red *ad hoc* en el rango de enlace [12].

Este tipo de red permite la adhesión de nuevos dispositivos, con el sólo hecho de estar en el rango de alcance de un nodo ya perteneciente a la red establecida. El principal inconveniente de este tipo de redes radica en el número de saltos que debe recorrer la información antes de llegar a su destino. Cada nodo que retransmite la información implica un salto, cuanto más saltos mayor es el tiempo que tarda en llegar la información a su destino. Otra de las características determinantes de este tipo de redes es que no se tiene conocimiento de la topología de la red [12].

Es fácil ver que, todas las características que requieren los algoritmos para emplearse en las redes de sensores o en redes *ad hoc*, las cumplen en su totalidad los algoritmos locales. Debido a esto, los algoritmos locales son objeto de gran interés dentro del estudio de la computación. A lo largo de este trabajo, nos referiremos a las redes de sensores o las redes *ad hoc* para ilustrar algunas características de los ambientes que requieren soluciones locales.

2.2. Algoritmos locales para el control de la topología en una red

En este tipo de algoritmos, cada nodo explora de forma independiente el área en el que se encuentra y los nodos que lo rodean, todo esto lo hace mediante su rango de transmisión. El algoritmo local debe ser suficientemente simple, tal que cada nodo pueda ejecutarlo sin problemas; por ejemplo, en una red *ad hoc*, cada dispositivo móvil deberá ser capaz de ejecutar el algoritmo. En este tipo de algoritmos locales, al igual que en todos los demás, no se consideran las soluciones centralizadas, ya que su ejecución y mantenimiento son casi

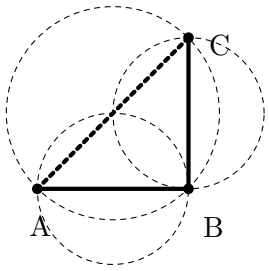


Figura 2.1: La arista {A,B} y {B,C} se encuentran en la gráfica de Gabriel, ya que sus círculos no contienen ningún otro vértice, mientras que la arista {A,C} no forma parte de ésta, ya que el círculo con diametro {A,C} contiene al vértice B.

imposible de realizarse en redes de sensores por ejemplo, debido a los limitados recursos con los que se cuenta (energía, memoria, procesamiento) y la movilidad de los nodos [19].

En general, los algoritmos locales que se utilizan en el control de la topología de una red no requieren que los nodos conozcan sus coordenadas, por otro lado, muchos de estos algoritmos necesitan que los nodos puedan determinar las distancias o direcciones hacia sus nodos vecinos, e inclusive algunos otros algoritmos necesitan de un identificador único en cada nodo. Sin embargo, pese a parecer una gran lista de requerimientos, las características de redes como las de sensores o las *ad hoc*, permiten que se pueda proveer facilmente esta información.

Uno de los métodos de construcción de gráficas más conocidos, y que se puede llevar a cabo de manera local es el de la *gráfica de Gabriel* [19], el cual construye una gráfica de la siguiente manera: dado un conjunto V de vértices, dos elementos de V están conectados por una arista si el círculo con esos dos puntos como diámetro no contiene más elementos de V (figura 2.1).

Es fácil ver que cada nodo sólo necesita revisar que vértices se encuentran dentro de su rango de transmisión, y mediante sencillos cálculos, determinar para cada uno de estos si es posible crear una conexión con él. Este método de construcción es completamente local. Por otro lado, bajo este modelo, se tiene (entre muchas otras) una pequeña restricción, la cual nos pide que cada nodo tenga el mismo rango de transmisión, requerimiento que fácilmente puede ser cumplido, ya que en una red de sensores por ejemplo, todos los nodos tienen las mismas características, y entre ellas, su rango de comunicación.

Otros de los métodos para la construcción de gráficas de manera local son los *basados en las triangulaciones de Delaunay* [19]. Li y sus co-autores propusieron un gran número de

algoritmos de control de topología basados en una triangulación de Delaunay k-localizada, donde k normalmente es 1 o 2 [13]. La triangulación de Delaunay 1-localizada no necesariamente entrega una gráfica plana, pero una subgráfica con ésta característica puede ser extraída de ella. Para $k \geq 2$, La triangulación de Delaunay k-localizada siempre es plana. A continuación, el algoritmo 1 presenta de manera muy superficial, el método que se lleva a cabo para el control de la topología de una red mediante una triangulación de Delaunay k-localizada.

Algoritmo 1 Control de topología mediante *Delaunay k-localizado*

1: Recabar información de todos los vecinos que se encuentren a k-saltos de distancia de cada nodo
2: Encontrar triángulos incidentes a un nodo al realizar una triangulación local de Delaunay
3: Enviar y recibir información de los vecinos que se encuentran en el triángulo
4: Checar si los triángulos incidentes han sido confirmados por los tres vértices
5: Agregar las aristas de la gráfica de Gabriel
6: Agregar las aristas de los triángulos confirmados

2.3. Algoritmos locales de ruteo

Probablemente, el algoritmo más sencillo de ruteo geográfico es el *greedy forwarding*, también conocido como *greedy routing* [19], donde cada nodo envía el mensaje (si él no es el destinatario) a su vecino con la mejor posición (el nodo más cercano) con respecto al nodo destino. El *greedy forwarding* es presentado en el algoritmo 2.

Algoritmo 2 Algoritmo *Greedy Routing*

1: Empieza en s
2: Envía el mensaje al vecino más cercano a t
3: Repite el paso 2 hasta alcanzar el nodo t o encontrar una mínima local con respecto a la distancia de t, es decir, llegar a un nodo v sin algún vecino posicionado más cerca del nodo t

El algoritmo de *greedy routing* claramente refleja un alto grado de simplicidad, tanto en concepto como en implementación, sin embargo, como se indica en el paso número tres del algoritmo 2, este presenta un gran problema: es posible que el mensaje quede "atorado" en un nodo con mínima local, es decir, un nodo sin un vecino mejor posicionado que él.

El primer algoritmo de ruteo geográfico (o ruteo por coordenadas) con garantía de entrega de mensajes fue el *face routing*, presentado en [8]. Poco tiempo después, una mejora del *face routing* fue presentada en *FACE-2* [4]. En ambos casos, el algoritmo se basa en lo siguiente: una gráfica plana G divide al plano en caras que son regiones clausuradas por poligonos hechos por aristas de G. Dado un vértice v en una cara f, se puede atravesar la frontera de f en sentido positivo (sentido negativo si f es la cara externa) usando la regla de la mano derecha [3]. Utilizando esta técnica para atravesar las caras como subrutina, en [4] se presenta el siguiente algoritmo para el envío de un paquete del nodo s al nodo t, el cual toma $O(n)$.

Algoritmo 3 Algoritmo de *Face Routing*

1: $p \rightarrow s$
2: **repetir**
3: sea f la cara de G con p en su frontera y que intersecta el segmento de linea (p, t)
4: atraviesa f hasta alcanzar una arista (u, v) que intersecte (p, t) en algún punto $p' \neq p$
5: $p \rightarrow p'$
6: **hasta que** $p = t$

Después de la aparición del *face routing*, muchos otros algoritmos basados en la misma idea y que también garantizaban la entrega de mensajes fueron presentados, como el *AFR (Adaptative Face Routing)* presentado en [10], el cual consiste en una variante del *face routing*, cuya modificación consiste en restringir las aristas para su búsqueda de un camino de s a t, mediante una elipse dada por las distancias entre estos, o el algoritmo $GOAFR^+$ presentado en [11], el cual realiza una combinación de *greedy routing* y *face routing* de la siguiente manera: el algoritmo empieza y utiliza el algoritmo de *greedy routing* hasta llegar o al nodo destino o a un nodo con mínima local, en cuyo caso utiliza el algoritmo de *face routing* para resolver ese problema, e inmediatamente después continuar con el recorrido empleando de nuevo *greedy routing*. Al final, todos los algoritmos antes mencionados siguen teniendo complejidad de $O(n)$, mismo orden de complejidad del *face routing*.

2.4. ¿Qué puede ser calculado de manera local?

No todos los problemas en gráficas pueden ser resueltos de manera local. Intuitivamente, todo aquel problema que necesite información global de la red, o todo aquel cómputo que necesite almacenamiento proporcional al tamaño de gráfica, se dice que es inherentemente no local [9]. Un buen ejemplo de este tipo de problemas es la búsqueda del camino más corto.

Aunque algunos problemas no puedan ser calculados localmente, se ha trabajado sobre

algoritmos locales que entregan aproximaciones al resultado óptimo, por ejemplo, en [9] se muestra una cota mínima de aproximación para el problema del conjunto mínimo de cobertura de vértices. Dicho resultado se extiende también para el problema del mínimo conjunto dominante.

Existe un gran número de algoritmos locales de aproximación que dan solución a problemas recurrentes en teoría de gráficas, problemas como búsqueda de árboles generadores de peso mínimo euclidiano, diagramas de Voronoi, y construcción de gráficas generadoras.

Como puede observar, el campo de los algoritmos locales ha tomado gran auge debido a sus características, como la escalabilidad, tolerancia a fallas, y sobre todo su fácil adaptación en cuanto a algoritmos distribuidos se refiere.

Capítulo 3

Resultados preliminares

A continuación, se presentan definiciones y resultados importantes en teoría de gráficas. El primer trabajo sobre la teoría de gráficas fue escrito por el matemático suizo Leonard Euler en 1736 [5]. Desde un punto de vista matemático, la teoría de gráficas fue insignificante en un principio, ya que se le asociaba con algunos juegos de entretenimiento. Pero algunos descubrimientos recientes en matemáticas, y particularmente en sus aplicaciones, le han dado un gran impulso. Actualmente hay algunos temas en matemáticas donde la teoría de gráficas es una herramienta indispensable. Este trabajo está totalmente basado en resultados conocidos dentro de la teoría de gráficas, por lo tanto, es vital hacer mención de algunos de estos resultados.

3.1. Gráficas

3.1.1. Definiciones básicas

Una *gráfica* es un par $G = (V, E)$ de conjuntos $V = V(G)$ y $E = E(G)$. El primero, $V(G)$, es una colección no vacía de puntos a los que se llama *vértices* de G y el segundo, $E(G)$, es una colección de líneas que unen elementos de $V(G)$, llamadas *aristas* de G [3]. Una gráfica es *finita* si tanto su conjunto de vértices como su conjunto de aristas son finitos.

Se dice que dos vértices $u, v \in V$ son *adyacentes* si están unidos por una arista, generalmente representada por $\{uv\}$, y se dice que estos vértices son los extremos de dicha arista, ó dicho de otra manera, esta arista es incidente en u y v.

Una manera muy habitual de representar una gráfica consiste en dibujar un punto por cada vértice de la gráfica, y unir dos de estos puntos con una linea si existe una arista que una los dos vértices correspondientes. En la Figura 3.1 se puede ver la representación de la gráfica $G = (V, E)$ con $V = \{u, v, w, x, y\}$ y $E = \{uv, uw, xu, vx\}$, se pude observar que v y u son adyacentes mientras que w y x no lo son, uw es incidente en u y en w y ninguna

arista incide en y. Aunque las aristas $\{v, x\}$ y $\{u, w\}$ se crucen en el dibujo su intersección no es un vértice de G.

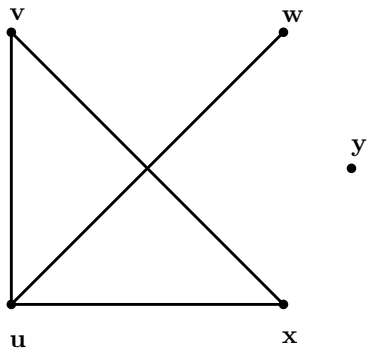


Figura 3.1: Representación gráfica de G.

Cuando dos aristas se cruzan en la representación de una gráfica se dice que se cortan y se llama *corte* al punto de intersección entre ambas. Por ejemplo, en la Figura 3.1 las aristas $\{v, x\}$ y $\{u, w\}$ se cortan.

Si dos vértices están unidos por más de una arista, se dice que estas son *aristas paralelas*. Una arista que tiene como extremos a un mismo vértice se denomina *lazo*. Una gráfica se dice que es *simple* cuando no tiene ni aristas paralelas ni lazos. La gráfica G_1 de la Figura 3.2 es una gráfica simple, mientras que G_2 no lo es por dos razones: la arista e es un lazo y las aristas f y g son paralelas.

Es conveniente observar que, en una gráfica simple con n vértices, el máximo número posible de aristas es

$$\frac{n(n-1)}{2} = \binom{n}{2}$$

pues cada vértice puede tener, a lo más, $n-1$ vértices adyacentes, esto es, $n-1$ aristas incidentes, y si se cuentan todas estas aristas por cada uno de los n vértices, en realidad se está contando cada arista dos veces.

Una *gráfica geométrica* es una gráfica en la cual los vértices o nodos son representados por puntos en el plano, y las aristas son segmentos de líneas rectas que conectan a dos de

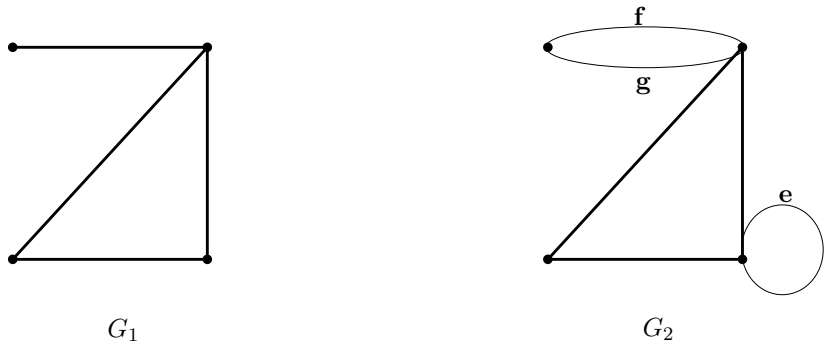


Figura 3.2: Ejemplos de gráficas simples, aristas paralelas y lazos.

estos. Las gráficas geométricas también son conocidas como *redes geométricas*.

Una *gráfica dirigida* $G' = (V', E')$ consta de un conjunto V' de vértices y un conjunto E' de arcos o aristas, que son pares ordenados de elementos de V'. Se dice que $a = (u, v)$ denota el arco con origen u y final en v, también se dice que u domina a v. Los arcos o aristas dirigidas también son denominadas *flechas*.

Para un vértice $u \in V'$, una *flecha de entrada* será toda aquella flecha donde u es el final, mientras que una *flecha de salida* será toda aquella flecha donde u es el origen.

3.1.2. Caminos y conexión

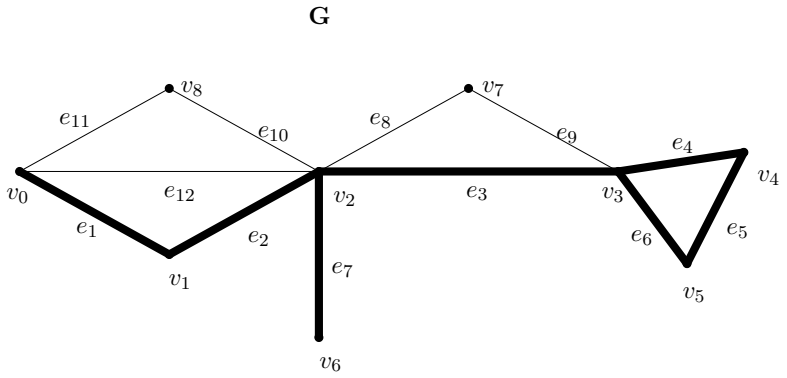
Un *camino* de longitud k en una gráfica G es una sucesión alternada de $k+1$ vértices y k aristas $W = v_0e_1v_1e_2v_2 \dots v_{k-1}e_kv_k$, donde v_i son los vértices y e_j son las aristas, tal que los extremos de e_i son v_{i-1} y v_i para $1 \leq i \leq k$ [3]. Se dice que W es un camino de v_0 a v_k. Los vértices v_0 y v_k se llaman el origen y el final del camino W respectivamente, y el resto de vértices $v_1, ..., v_{k-1}$ son los vértices interiores de W. A veces se denotará también a W como $W = v_0v_1...v_{k-1}v_k$.

Observación 3.1.1. La definición anterior permite que se repitan vértices y/o aristas en un mismo camino.

Se dice que un camino es un *paseo* cuando no hay repetición de aristas, y que es una *trayectoria* cuando no hay repetición de vértices (y en consecuencia, no hay repetición de

aristas). *Un camino cerrado* en G es un camino cuyos vértices origen y final son el mismo; cuando no exista repetición de aristas dicho camino cerrado se llamará *circuito*. Si no existe repetición de vértices (y por consiguiente de aristas) dicho camino cerrado se llamará *ciclo*.

En las siguientes figuras se muestran ejemplos de todos estos tipos de caminos, marcados con líneas más gruesas. Por ejemplo, el camino $W_1 = v_0e_1v_1e_2v_2e_3v_3e_4v_4e_5v_5e_6v_3e_3v_2e_7v_6$ que se muestra en la figura 3.3 tiene longitud 8, con origen v_0, final v_6 y vértices interiores $\{v_1, v_2, v_3, v_4, v_5\}$. W_1 no es una trayectoria, ni un paseo, ni un camino cerrado.



(a)

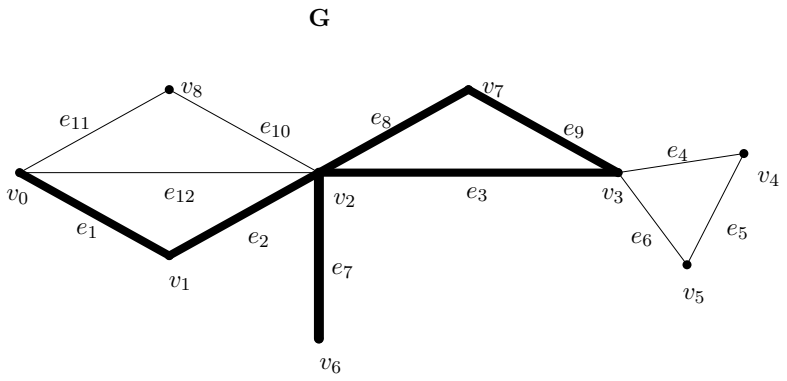
Figura 3.3: $W_1 = v_0e_1v_1e_2v_2e_3v_3e_4v_4e_5v_5e_6v_3e_3v_2e_7v_6$. W_1 es un ejemplo de un camino que no es trayectoria, ni paseo, ni camino cerrado.

$W_2 = v_0e_1v_1e_2v_2e_8v_7e_9v_3e_3v_2e_7v_6$, que se muestra en la figura 3.4, es un paseo. $W_3 = v_0e_1v_1e_2v_2e_7v_6$ de la figura 3.5 es una trayectoria. $W_4 = v_0e_1v_1e_2v_2e_3v_3e_9v_7e_8v_2e_{10}v_8e_{11}$ que se muestra en la figura 3.6 es un camino cerrado de longitud 7 pero no es un ciclo. $W_5 = v_0e_1v_1e_2v_2e_{10}v_8e_{11}v_0$ que se muestra en la figura 3.7 es un ciclo de longitud 4.

Una gráfica G es *conexa* si todo par de vértices $u, v \in G$ están unidos por una trayectoria de u a v. Una consecuencia automática de esta definición es que toda gráfica es la unión disjunta de gráficas conexas, a los que llamaremos *componentes conexas*. En la Figura 3.8 G_1 es una gráfica conexa y G_2 no lo es, pues tiene 2 componentes conexas.

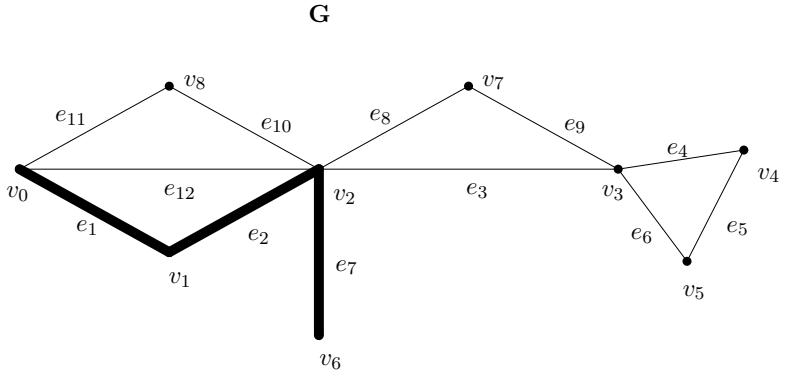
3.1.3. Subgráficas, eliminación y adición

Aparte del estudio de las características o propiedades de una gráfica en su totalidad, también se puede estudiar solamente una región o parte de la misma. Por ejemplo, dada



(b)

Figura 3.4: $W_2 = v_0e_1v_1e_2v_2e_8v_7e_9v_3e_3v_2e_7v_6$. W_2 es un paseo.



(c)

Figura 3.5: $W_3 = v_0e_1v_1e_2v_2e_7v_6$. W_3 es una trayectoria.

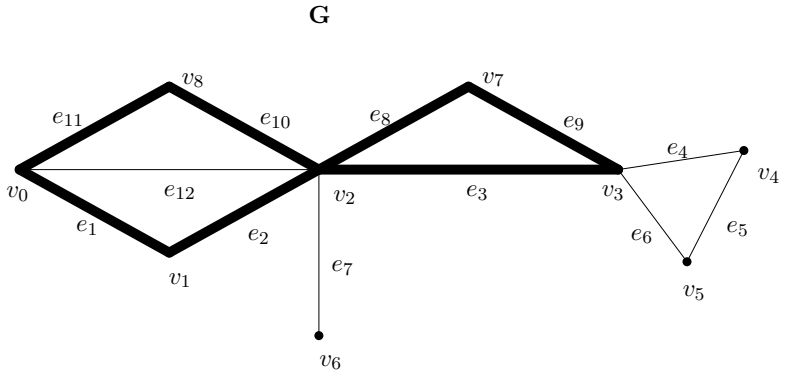


Figura 3.6: $W_4 = v_0e_1v_1e_2v_2e_3v_3e_9v_7e_8v_2e_{10}v_8e_{11}v_0$. W_4 es un camino cerrado de longitud 7, pero no es un ciclo.

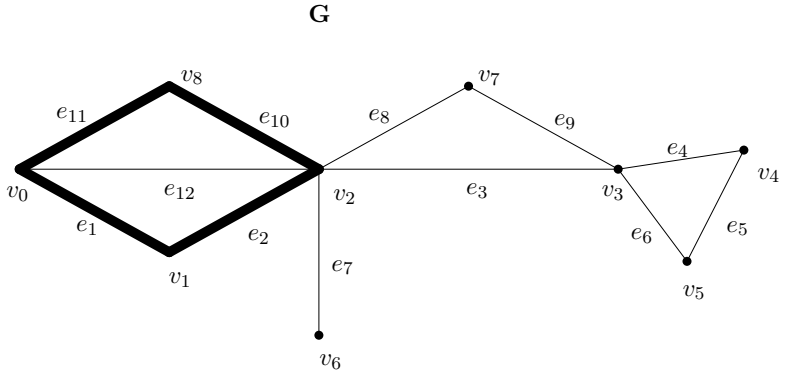


Figura 3.7: $W_5 = v_0e_1v_1e_2v_2e_{10}v_8e_{11}v_0$. W_5 es un ejemplo de un ciclo de longitud 4.

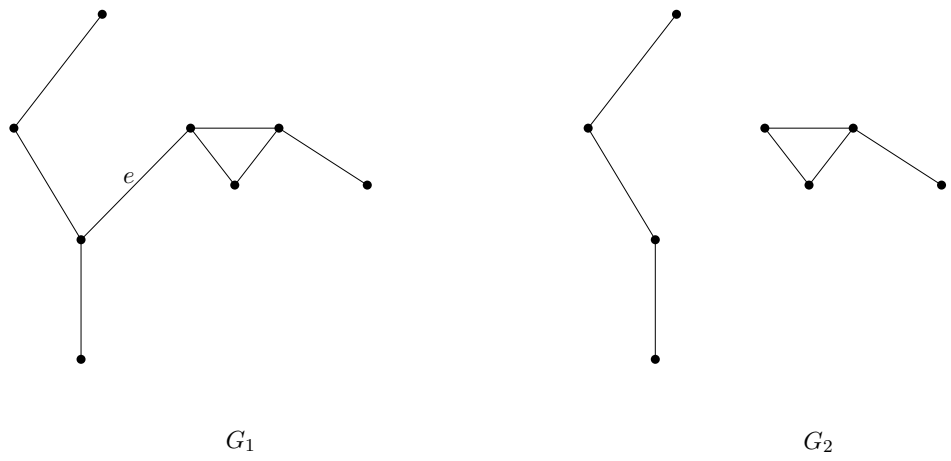


Figura 3.8: Componentes conexas en G_1 y G_2. G_1 es una gráfica conexa, G_2 no.

una gráfica no conexa a veces es conveniente estudiar cada una de sus componentes conexas por separado. Más aún, podemos estudiar conjuntos arbitrarios de vértices y aristas de una gráfica cualquiera. Una *subgráfica* G' de una gráfica G es una gráfica que tiene todos sus vértices y aristas en G, de manera que toda arista de G' incida en vértices de G.

La mayor parte de las subgráficas que vale la pena estudiar son aquellas que difieren de manera mínima de la gráfica de partida, pues conservan casi todas sus propiedades y son las pequeñas diferencias las que muestran detalles importantes. Es por ello que existen ciertas maneras de modificar mínimamente una gráfica, como se muestra en los siguientes párrafos.

Eliminar una arista e de una gráfica G es quitarla del conjunto de aristas de G, obteniendo una subgráfica de G denotada por $G\backslash e$. Las gráficas también se pueden modificar añadiendo elementos: por ejemplo, la *adición de una arista* e en una gráfica G es el resultado de añadir una arista al conjunto $E(G)$ tal que una dos vértices cualesquiera ya existentes en la gráfica. Se escribe $G \cup e$.

Dada una gráfica $G = (V, E)$, una arista $e \in E(G)$ es una *arista de corte* de G si al eliminarla, la subgráfica obtenida tiene más componentes conexas que G. Por ejemplo, retomando la figura 3.8, la arista e de G_1 es una arista de corte y al eliminarla se obtiene la gráfica G_2 que, efectivamente, tiene una componente conexa más que G_1.

Prescindir de un vértice en una gráfica no es tan simple como eliminar una arista, pues al quitar un vértice todas las aristas incidentes en este pierden un extremo. En consecuencia,

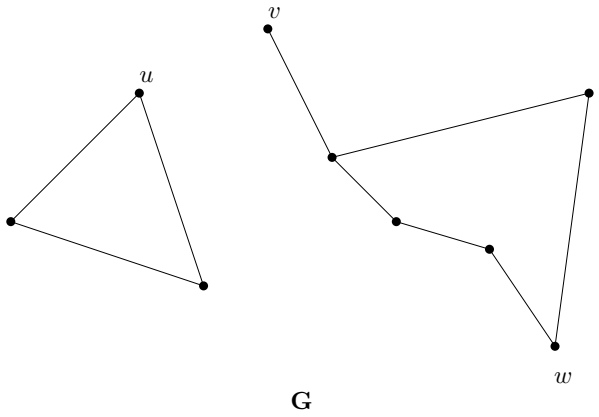


Figura 3.9: Ejemplo de distancias entre vértices. la distancia entre w y v es 3, y la distancia entre u y v es ∞.

es necesaria una buena definición de esta acción: la *eliminación de un vértice* v de una gráfica G consiste en quitar v del conjunto de vértices $V(G)$ y todas las aristas incidentes a v del conjunto de aristas $E(G)$, obteniendo una subgráfica de G denotada por $G\backslash v$.

La eliminación de un conjunto de aristas y/o vértices de una gráfica viene definida por la eliminación uno a uno de estos elementos, donde sin dificultad se puede probar que cualquier orden de eliminación lleva a una misma gráfica final.

3.1.4. Distancias

La *distancia* $d(u, v)$ entre dos vértices u y v de una gráfica G es la longitud menor de todas las trayectorias que los unen. Si u y v no se encuentran en la misma componente conexa, se dice que $d(u, v) = \infty$. En la figura 3.9 la distancia entre los vétices w y v de la gráfica G es 3, y la distancia entre u y v es ∞ porque no se encuentran conectados por ninguna trayectoria.

3.1.5. Árboles

Se dice que una gráfica es un *árbol* si es conexa y no tiene ciclos. Esta definición, que a simple vista no parece ser demasiado restrictiva, en realidad lo es, ya que el número de aristas de un árbol determina su número de vértices y viceversa. Además, en los árboles se tienen características muy particulares, como por ejemplo, en un árbol toda arista es una arista de corte, además el número de vértices es el número de aristas más uno, es decir $|V(G)| = |E(G)| + 1$.

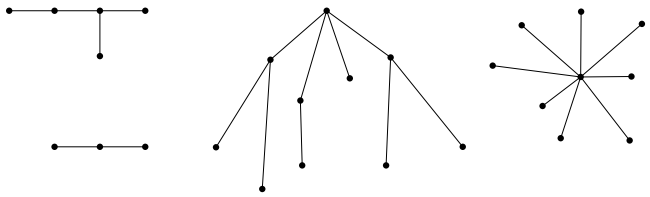


Figura 3.10: Ejemplos de árboles.

En la figura 3.10 se pueden ver diferentes tipos de árboles.

3.2. Planaridad

La *planaridad* es, a grandes rasgos y de manera intuitiva, la propiedad que tiene un objeto de poder vivir dentro del plano, esto es, de poder incluirse dentro del mismo sin perder ninguna de sus propiedades.

3.2.1. Gráficas planas y planares

Se dice que una gráfica es *planar* o que tiene una representación en el plano si esta puede ser dibujada sobre el plano asignando puntos a los vértices y líneas a las aristas (tal como se definió la representación gráfica en la sección 3.1.1), de tal manera que las líneas se intersecten entre ellas solamente en sus extremos. Tal representación gráfica de una gráfica G se llama dibujo sin cortes o *representación planar* de G, y se puede entender como el resultado de una transformación de la gráfica G en el plano tal que a cada vértice le asigna un punto y a cada arista una línea que une los puntos asociados a sus extremos.

Una *gráfica plana* es, abusando del lenguaje, una representación planar de una gráfica planar. La figura 3.11 muestra una representación planar de la gráfica planar G. Es importante observar que cualquier subgráfica de una gráfica plana es también plana, pues si la gráfica de origen no presenta cortes entre aristas tampoco los presentará ninguna de sus subgráficas.

La topología juega un papel muy importante en el estudio de gráficas. Los resultados topológicos que son especialmente relevantes en el estudio de las gráficas planas, son aquellos que tratan sobre las curvas de Jordan.

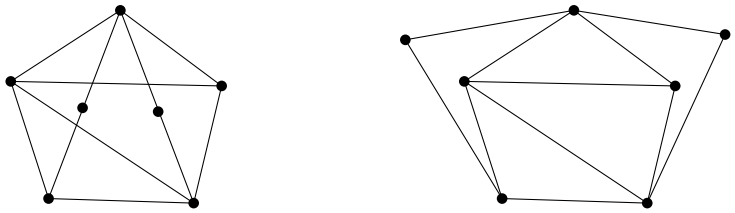


Figura 3.11: Una gráfica planar G, y una representación planar de ella.

Una *curva de Jordan* es una curva cerrada y continua que no se autocorta, es decir, no se intersecta a sí misma. La unión de las aristas en un ciclo de una gráfica plana constituyen una curva de Jordan en el plano; esta es la razón de que se use en esta sección un conocido teorema relacionado con estas curvas para demostrar que las caras en una gráfica plana son regiones totalmente acotadas por un conjunto de aristas que forman un ciclo (a excepción de una única región no acotada denominada cara externa). Sin embargo, antes se deben definir ciertos conceptos para poder entender el teorema.

Sea J una curva de Jordan en el plano. Entonces, el resto del plano está dividido en dos conjuntos abiertos mutuamente ajenos, uno acotado y el otro no, a los cuales llamaremos el *interior* de J y el *exterior* de J respectivamente. Se escribe $\text{int}(J)$ para denotar el interior de J y $\text{ext}(J)$ para denotar el exterior, donde claramente, $\text{int}(J) \cap \text{ext}(J) = J$.

Ahora podemos enunciar el teorema de la curva de Jordan:

Teorema 3.2.1 (Teorema de la curva de Jordan)**.** Dada una curva de Jordan J, cualquier línea que une un punto de $\text{int}(J)$ con un punto de $\text{ext}(J)$ debe cortar a J en al menos un punto (figura 3.12).

Este teorema trata con la topología del plano y, pese a ser intuitivo, la demostración formal del mismo tiene un alto grado de dificultad. La demostración de este teorema no se presentará en este trabajo. Se puede encontrar una demostración formal del teorema de la curva de Jordan en [6].

3.2.2. Gráficas duales

Hasta ahora se ha podido observar que toda gráfica plana G divide el plano en cierto número de regiones conectadas; de hecho, estas regiones son siempre el interior o exterior de algún ciclo de la gráfica. Las clausuras de estas regiones se llaman las *caras* de G [3].

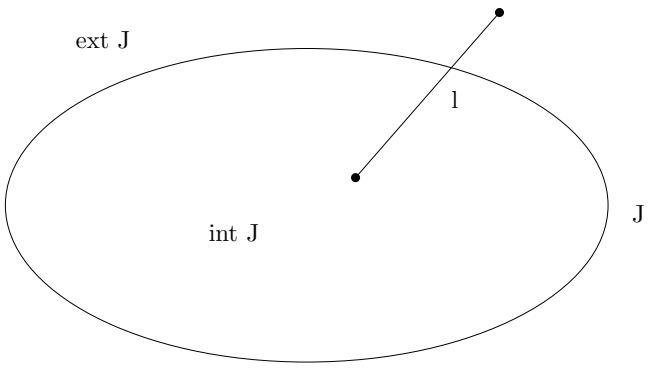


Figura 3.12: La línea l corta a J en un punto.

La figura 3.13 muestra una gráfica plana G con 6 caras, f_1, f_2, f_3, f_4, f_5 y f_6.

Se denota por $F(G)$ y $\phi(G)$ al *conjunto de caras* y al *número de caras*, respectivamente, de una gráfica plana G. Toda gráfica plana tiene exactamente una cara no acotada, a la cual denominamos *cara externa* o *exterior*. Por ejemplo, en la figura 3.13 la cara exterior de G es f_1. Las demás caras de G (f_2, f_3, f_4, f_5 y f_6 en la figura 3.13) se denominan *caras internas* o *interiores*.

En una gráfica plana G se denota *frontera de una cara* f por $b(f)$. Si G es conexa, entonces $b(f)$ se puede ver como un camino cerrado en el que se pasa dos veces por cada arista de corte de G; cuando $b(f)$ no contiene aristas de corte es un ciclo de G. Por ejemplo, en la gráfica plana de la figura 3.13 se tiene que $b(f_2) = v_1e_3v_2e_4v_3e_5v_4e_1v_1$ y $b(f_5) = v_7e_{10}v_5e_{11}v_8e_{12}v_8e_{11}v_5e_8v_6e_9v_7$. Se dice que una cara f es incidente en los vértices y aristas de su frontera.

Definición 3.2.1 (Arista frontera)**.** Una *arista frontera* es toda aquella arista que pertenece a dos caras diferentes.

Definición 3.2.2 (Arista puente)**.** Una arista e es una *arista puente* si al realizar $G\backslash e$ se incrementa en uno el número de componentes conexas en la gráfica resultante.

No es difícil demostrar que una arista puente pertenece a una sola cara de G.

El subconjunto de aristas frontera en G cumplen con el siguiente lema.

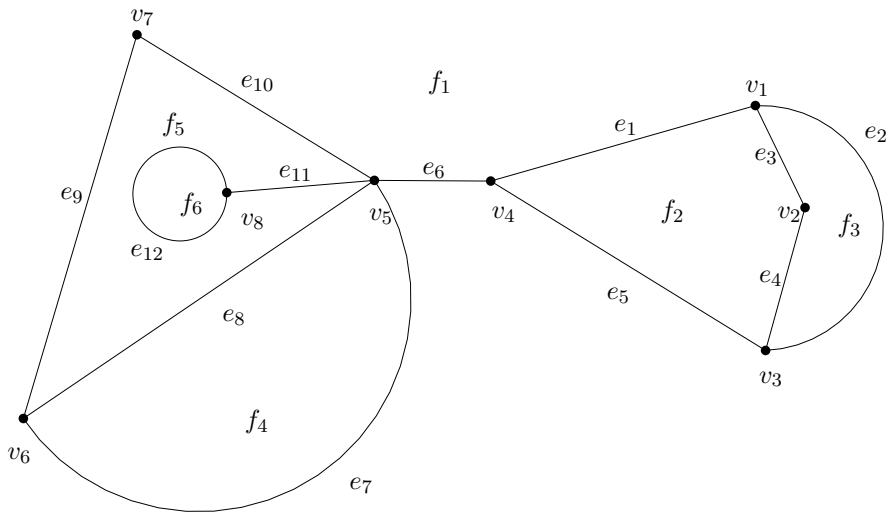


Figura 3.13: Ejemplo de caras de una gráfica.

Lema 3.2.1. Para cualquier par de vértices en G unidos por una arista frontera, existen dos caminos disjuntos que los unen.

Prueba. Sean u y v dos vértices en G unidos por una arista frontera e. Por ser una arista frontera, e pertenece a dos caras diferentes f y f', donde por lo menos una de las caras es una cara interna. Sin pérdida de generalidad diremos que f es una cara interna. u y v pertenecen al camino cerrado descrito por $b(f)$, y es fácil ver que, ignorando las aristas puente en $b(f)$, u y v pertenecen a un mismo ciclo. □

Definiremos como *vértice frontera* a aquél vértice en f que sólo es adyacente a aristas frontera. Así mismo, un *vértice puente* será aquel vértice adyacente sólo a aristas puente. Por otro lado, un *vértice híbrido* será aquel que sea adyacente tanto a aristas frontera como aristas puente.

El grado de una cara f en una gráfica plana G, $dG(f)$, es el número de aristas en las que ella incide, esto es, el número de aristas en $b(f)$, donde los puentes se cuentan dos veces. Por ejemplo, en la figura 3.13, $dG(f_5) = 6$.

Dada una gráfica plana G, se puede construir otra gráfica G^* de la siguiente manera: G^* tiene un vértice f^* por cada cara f de G, y una arista e^* por cada arista e de G; dos vértices f^* y g^* están unidos por la arista e^* en G^* si y sólo si, sus correspondientes caras f y g están separadas por la arista e en G. Se dice que G^* es la *gráfica dual* de G.

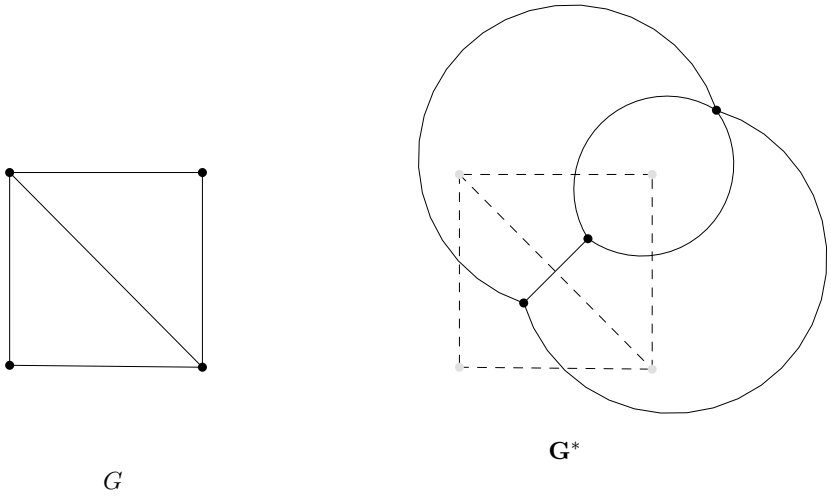


Figura 3.14: Ejemplo de una gráfica y su dual.

Existe una manera natural de crear una representación de G^* en el plano, que consiste en colocar cada vértice f^* en el interior de la cara f de la representación planar de G y dibujar cada arista e^* de manera que se corte con la correspondiente arista e de G exactamente una vez y sin cortar ninguna otra arista de G. En la figura 3.14 se muestra una gráfica y su dual, dibujada por el procedimiento natural aquí descrito.

Las siguientes relaciones son consecuencia directa de la definición de la gráfica dual:

Observación 3.2.1. Si G es una gráfica plana con dual G^*, se cumple:

- $v(G^*) = \phi(G)$
- $e(G^*) = e(G)$

3.2.3. Fórmula de Euler

Existe una fórmula sencilla que relaciona el número de vértices, el número de aristas y el número de caras de una gráfica plana conexa. Esta fórmula se conoce como *Fórmula de Euler*, en honor al matemático Leonhard Euler. Euler formuló esta relación para gráficas planas definidas por los vértices y aristas de los poliedros, además mostró que esta se podía extender para gráficas planas con una sola componente conexa.

Teorema 3.2.2 (Fórmula de Euler)**.** Si G es una gráfica plana conexa, entonces

$$v - e + f = 2$$

Prueba. Por inducción en el número de aristas en la gráfica. Caso base: Si $e = 0$, la gráfica consiste en un solo vértice y una sola cara. Por lo tanto tenemos $1 - 0 + 1 = 2$ con lo que se cumple la igualdad. Paso inductivo: Suponemos que la fórmula se cumple para todas las gráficas con a lo más n aristas. Sea G una gráfica con $n + 1$ aristas.

Caso 1: G no contiene ciclos. Entonces G es un árbol, por lo tanto se tiene que $v = e + 1$ y $f = 1$, por lo tanto se tiene que

$$(e + 1) - e + (1) = 2$$

con lo que la fórmula se cumple.

Caso 2: G contiene por lo menos un ciclo. Se elige una arista p que pertenezca a un ciclo. Removemos p para crear una nueva gráfica G'. Ya que el ciclo separa el plano en dos caras, las caras a ambos lados de la arista p deben ser distintas. Cuando se remueve la arista p, se mezclan esas dos caras. Entonces G' tiene una cara menos que G. Ya que G' tiene n aristas, la fórmula se cumple para G' por hipótesis de inducción. Esto es $v' - e' + f' = 2$. Pero $v' = v$, $e' = e - 1$, y $f' = f - 1$. Sustituyendo se tiene que

$$v - (e - 1) + (f - 1) = 2$$

y simplificando se obtiene

$$v - e + f = 2$$

□

Corolario 3.2.2.1. Si G es una gráfica simple plana conexa, con v vértices, e aristas y f caras, donde $v \geq 3$. Entonces $e \leq 3v - 6$.

La suma de $dG()$ (los grados de las caras) en G es igual a dos veces el número de aristas. Pero cada cara debe tener grado ≥ 3 (ya que G es simple). Por lo tanto se tiene $3f \leq 2e$.

La fórmula de Euler dice que $v - e + f = 2$, por tanto $f = e - v + 2$, y así $3f = 3e - 3v + 6$. Combinando esto con $3f \leq 2e$, se tiene que $2e \geq 3e - 3v + 6$. Entonces $e \leq 3v - 6$. □

Ahora es necesario mencionar el siguiente teorema, esto con el fin de utilizarlo en el siguiente corolario.

Teorema 3.2.3 (the handshaking theorem)**.** Si G es una gráfica simple no dirigida plana conexa con e aristas, entonces

$$\sum_{v \in V} deg(v) = 2e$$

es decir, la suma de los grados de todos los vértices en G es dos veces el número de aristas en G.

Prueba. Claramente esto es cierto para todas las gráficas sin vértices, así como para las gráficas con uno y dos vértices, ya que, para el caso de $|V| = 2$, debido a que G es simple y no dirigida, sólo existe una arista que une a los dos únicos vértices en G, por tanto cada vértice tendrá grado 1, lo cual da como resultado

$$\sum_{v \in V} 2 = 2 \cdot 1$$

Ahora se asume que esta propiedad se cumple para cualquier gráfica con menos de k aristas. Sea G una gráfica con k aristas. Se removerá una arista de G, esto dará como resultado una subgráfica inducida G' con $k-1$ aristas, por lo tanto en G' la suma de los grados de todos los vértices sera igual a $2(k-1)$ (por hipótesis de inducción). Al colocar de nuevo la arista, la suma de los grados será igual a $2(k-1)+2 = 2k$, por lo tanto el teorema se cumple para k aristas. □

Corolario 3.2.3.1. Si G es una gráfica simple plana conexa, entonces G tiene un nodo de grado menor a seis.

Prueba. Esto es claramente cierto si G tiene uno o dos vértices. Si G tiene al menos tres vértices, entonces, para una prueba por contradicción, se supondrá que cada vértice en G tiene al menos grado 6. Por el teorema 3.2.3, $2e$ es igual a la suma de los grados de todos los vértices, por tanto se tiene que $2e \geq 6v$. Por el corolario 3.2.2.1 tenemos que $e < 3v \Leftrightarrow 2e < 6v$, con lo cual hemos obtenido una contradicción. Por lo tanto, debe existir por lo menos un vértice de grado menor a 6. □

3.3. Conjuntos de corte en gráficas planas

Definición 3.3.1 (Arista de corte)**.** Una arista $e \in E$ se denomina *arista de corte* si es una arista puente.

Definición 3.3.2 (Conjunto de corte)**.** Un *conjunto de corte* c es un conjunto de aristas de G tal que $G \backslash c$ incrementa el número de componentes conexas en la subgráfica restante, mientras que la eliminación de cualquier subconjunto propio no lo hace.

Además, $G \backslash c$ genera exactamente una componente conexa más en la gráfica resultante.

Definición 3.3.3. Una gráfica G se denomina *k-conexa* si el o los conjuntos de corte de cardinalidad mínima en G son de tamaño k.

Definición 3.3.4. Sea G una gráfica 2-conexa, y $c = \{e_1, e_2\}$ un conjunto de corte de cardinalidad 2. Nos referiremos a c como *par de corte* (figura 3.15).

De la definición 3.3.1 podemos concluir lo siguiente.

Lema 3.3.1. Para un par de corte $e_1 \in f_i, f_j$ y $e_2 \in f'_i, f'_j$ se debe cumplir que $f_i \neq f_j$, y $f'_i \neq f'_j$, donde f_x denota una cara en G.

Teorema 3.3.1. Para toda gráfica plana 2-conexa, las aristas de un par de corte pertenecen a las mismas dos caras.

Prueba. Sea G una gráfica plana conexa, y e_1 e_2 un par de corte. e_1 no es de corte, por lo tanto, e_1 pertenece a dos caras en G, que sin pérdida de generalidad llamaremos f_1 y f_2.

Eliminamos e_1 de G, sea G' la gráfica resultante. Notemos que el único cambio en G' fue la fusión de f_1 y f_2, lo cual da como resultado una nueva cara, que sin pérdida de generalidad nombraremos f_3. Observemos que en G', e_2 es de corte, mientras que en G no es así. Por lo tanto, en G', e_2 pertenece a una única cara, y sus caras de G tuvieron un cambio al eliminar e_1, pues si sus caras en G' fueran las mismas de G, entonces e_2 sería de corte en G. Por lo tanto en G', e_2 pertenece sólo a f_3.

Ahora en G', regresamos a e_1 para reconstruir G. En G, e_2 no es de corte, por lo tanto sus caras debieron cambiar, y las únicas caras que cambiaron fueron f_1 y f_2, por lo tanto e_2 debe pertenecer a f_1 y f_2. □

La prueba anterior se basa en la siguiente observación. Para una gráfica plana G, al eliminar una arista frontera, sólo las dos caras a las que pertenece son modificadas, las demás caras en G permanecen idénticas.

Lema 3.3.2. Sea G una gráfica plana conexa, c un conjunto de corte de cardinalidad $k \geq 2$ y sea e_j un elemento de c. Al realizar $G \backslash e_j$, en la gráfica resultante, $c \backslash e_j$ sigue siendo un conjunto de corte de cardinalidad $k - 1$.

Prueba. Por inducción sobre el tamaño de k. Para el caso base sea $k = 2$, al eliminar una de las aristas del par de corte, la arista aún perteneciente a la gráfica resultante será una arista de corte, con lo que se cumple la proposición. Suponemos cierto para $k = n$. Para $k = n + 1$, por ser un conjunto de corte, para que aumenten el número de componentes conexas en G se deben eliminar las $n + 1$ aristas del conjunto c, sin importar el orden, por lo tanto, al eliminar una arista, sin importar cual sea, aún se deben eliminar

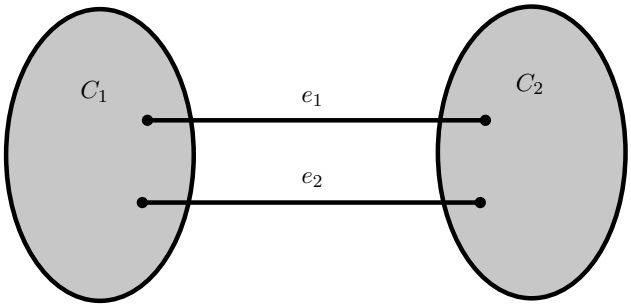


Figura 3.15: Par de corte

las n aristas restantes en c para que la gráfica aumente el número de sus componentes conexas, por lo tanto el conjunto $c' = c \backslash e$ sigue siendo un conjunto de corte, ahora de cardinalidad $k = n$. □

Lema 3.3.3. Sea G una gráfica plana conexa con F caras, c un conjunto de corte de cardinalidad k, y sea $G' = G \backslash c$. El número de caras en G' será $F' = F - (k - 1)$.

Prueba. Por ser una gráfica plana conexa, la gráfica G cumple con la fórmula de Euler(teorema 3.2.2). Por lo tanto tenemos:

$$F = 2 + e - v$$

Donde v es el número de vértices y e el número de aristas en G.

Si eliminamos de G $k - 1$ aristas pertenecientes a c la gráfica resultante seguirá siendo conexa, y tendremos

$$F' = 2 + e - (k - 1) - v$$

De donde, sustituyendo el valor de F tenemos

$$F' = F - (k - 1)$$

Ahora, al eliminar de la gráfica resultante la arista restante del conjunto c, la gráfica dejará de ser conexa y el número de caras no cambiará, pues la arista eliminada era de corte. □

Teorema 3.3.2. Sea c un conjunto de corte y c_f el conjunto de caras a las que pertenecen las aristas de c. El conjunto c y el conjunto c_f tienen la misma cardinalidad.

Prueba. Sea k la cardinalidad del conjunto c. La prueba se hará por inducción sobre el tamaño de k. Para el caso base $k = 1$, por definición, la arista de corte pertenece a sólo una cara de la gráfica, por lo tanto la propiedad se cumple. Suponemos cierta la propiedad para $k = n$.

Sea $k = n + 1$. Eliminamos una arista e de G tal que $e \in c$, sean $c' = c \backslash e$ y $G' = G \backslash e$. En G', c' es un conjunto de corte de cardinalidad $k = n$, por lo que cumple la hipótesis de inducción. Al regresar a G' la arista e, esta formará dos nuevas caras a partir de una cara existente en G'. Las dos caras se agregarán a c_f. Por lo menos una de las dos nuevas caras es una cara interna, y en la cara interna debe de existir por lo menos otra arista diferente de e que pertenezca a c, ya que de no ser así el conjunto c no sería de corte. Por lo tanto, la cara que se modificó al reinsertar e en G' debió ser una cara perteneciente a c_f. Por lo tanto c_f incrementará en uno su cardinalidad, por lo que $|c_f| = k + 1$.□

Teorema 3.3.3. Sea c un conjunto de corte de cardinalidad $k \geq 2$ y c_f el conjunto de las caras a las que pertenece c. En la gráfica dual, los elementos de c_f forman un ciclo.

Prueba. Por inducción sobre el tamaño de c. Sea $k = 2$ con $c_f = \{f_1, f_2\}$. En la gráfica dual, f_1 y f_2 son adyacentes a dos aristas paralelas que las conectan, por lo cual forman un ciclo. Suponemos cierto para $k = n$.

Para $k = n+1$. Eliminamos una arista e de G tal que $e \in c$, sean $c' = c \backslash e$ y $G' = G \backslash e$. En G', c' es un conjunto de corte de cardinalidad $k = n$, por lo que cumple la hipótesis de inducción. Al regresar a G ($G' \cup e$), e formará dos nuevas caras (f_1 y f_2) a partir de una cara de $G'(f_i)$. f_1 y f_2 se agregarán a c_f. Como se dedujo en el lema anterior, la cara f_i pertenece a c_f en G', por lo que f_1 y f_2 tomarán el lugar de f_i en la gráfica dual, y serán adyacentes por la representación de e, además formarán parte del ciclo formado por los elementos de c_f. □

3.4. Algoritmos locales en gráficas planas

3.4.1. Recorrido de una cara en una gráfica plana siguiendo la regla de la mano derecha y la regla de la mano izquierda

La bien conocida *regla de la mano derecha*, utilizada en muchos algoritmos locales para el ruteo en gráficas, es de hecho un algoritmo que fue inspirado en el recorrido de laberintos. El algoritmo es simple, indica que si un jugador en un laberinto camina por este colocando su mano derecha sobre el muro siempre cuidando de no despegarla, entonces el jugador eventualmente visitará cada muro del laberinto [14]. Más específicamente, si el

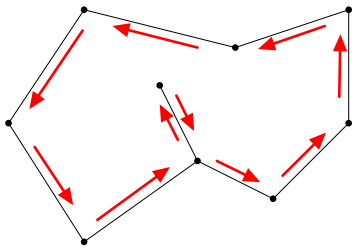


Figura 3.16: Recorrido de una cara interna usando la "regla de la mano derecha".

laberinto es la cara de una gráfica plana, el jugador visitará cada arista y cada vértice de la cara (figura 3.16).

Así mismo, podemos utilizar la *regla de la mano izquierda* para realizar un recorrido sobre la cara f en sentido contrario al mostrado en la figura 3.16, el cual mantendrá las propiedades y características del recorrido hecho siguiendo la regla de la mano derecha, salvo el orden de visita [14] (figura 3.17).

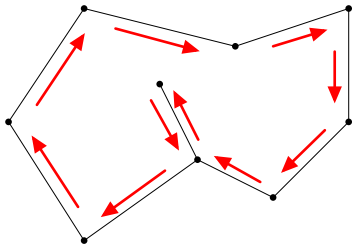


Figura 3.17: Recorrido de una cara interna usando la "regla de la mano izquierda".

Debe notarse que, al realizar un recorrido de f siguiendo la regla de la mano derecha, si f es una cara interna el recorrido se hará en *sentido positivo* o contrario al giro de las manecillas del reloj, mientras que el recorrido se hará en *sentido negativo* o en el mismo sentido al giro de las manecillas del reloj si f es la cara externa. Lo mismo ocurrirá para un recorrido de f siguiendo la regla de la mano izquierda, donde si f es una cara interna el recorrido se hará en sentido negativo, y el recorrido se hará en sentido positivo si f es la cara externa.

A lo largo de este trabajo se supondrá que todos los recorridos de una cara, sin importar que regla se utilice, empezarán en un vértice y no en una arista. Se puede ver que cuando el punto de inicio de un recorrido se encuentra en el interior de una arista, basta

con recorrer ese punto hacia uno de los extremos de la arista, en general el punto de inicio se recorrerá al vértice que se encuentra en sentido contrario del recorrido.

Definición 3.4.1. Sea G una gráfica geométrica plana conexa y f una cara de G. Denominaremos *vértice de inicio* al vértice donde comience el recorrido de f utilizando alguna de las dos reglas.

Definición 3.4.2. Denominaremos *arista de inicio* a la primer arista que se encuentre al realizar el recorrido de f utilizando alguna de las dos reglas.

Definición 3.4.3. Si el recorrido se realiza siguiendo la regla de la mano derecha, denominaremos *arista de término* a la arista que sea adyacente al vértice de inicio y que forme el ángulo menor con la arista de inicio. Si el recorrido sigue la regla de la mano izquierda, la arista de término será la arista que sea adyacente al vértice de inicio y que forme el ángulo menor en valor absoluto con la arista de inicio, al medir los ángulos en sentido negativo.

Teorema 3.4.1. El recorrido de una cara en G siguiendo alguna de las dos reglas es de $O(1)$ en cuanto al uso de memoria.

Prueba. Para que un agente a realice el recorrido completo (recorrer todos los vértices y aristas) de una cara en G de manera local, sólo es necesario seguir una de las dos reglas, las cuales utilizan un número constante de localidades de memoria, ya que a sólo necesita recordar el vértice de inicio y la arista de término. La condición de paro en el recorrido será si a llega al vértice de inicio mediante la arista de término. □

Teorema 3.4.2. El recorrido de una cara en G siguiendo alguna de las dos reglas es de $O(n)$ en cuanto al tiempo de ejecución.

Prueba. Al realizar un recorrido siguiendo alguna de las dos reglas, una arista puede ser visitada a lo más dos veces durante el recorrido (una en cada sentido), por lo tanto, si una cara tiene n aristas, el recorrido completo se realizará en $O(n)$ visitas. Si se asume que cada visita se realizará en una unidad de tiempo, al final se tendrá que el recorrido de la cara será realizado en tiempo de $O(n)$. □

3.4.2. Clasificación de una cara en una gráfica plana

Una gráfica geométrica plana conexa divide el plano en cierto número de regiones conectadas, las cerraduras de estas regiones se denominan caras. Podemos definir dos tipos de caras, las caras internas y la cara externa. Una cara es interna si su región queda totalmente acotada por un conjunto de aristas que forman un ciclo. En caso de no ser así se denominará cara externa. Una representación de una gráfica plana admite solamente una cara externa.

Una manera fácil de saber si una cara es interna o externa es mediante la suma de sus ángulos internos.

Teorema 3.4.3. La suma de los ángulos internos de un polígono simple de n lados es igual a $\pi \cdot (n-2)$.

Corolario 3.4.3.1. La suma de los ángulos externos de un polígono simple de n lados es igual a $\pi \cdot (n+2)$.

Como sabemos, no todas las caras internas de una gráfica se pueden ver como polígonos simples, ya que una cara puede tener una o más aristas puente. A continuación se presenta una generalización del teorema 3.4.3 a todas las caras internas de una gráfica geométrica plana.

Sea G una gráfica geométrica plana y f una cara interna de G con n vértices y sin aristas puente (figura 3.18). Ya que f es un polígono simple, cumple con el teorema 3.4.3. Sea α el ángulo exterior entre las aristas e y e'. Si $\alpha = 0$ entonces v y v' tendrán la misma posición. Si se superponen los vértices v y v', también lo harán las aristas e y e' (figura 3.19). Si se considera que f sigue siendo un polígono de n vértices y n aristas, f seguirá cumpliendo con el teorema 3.4.3.

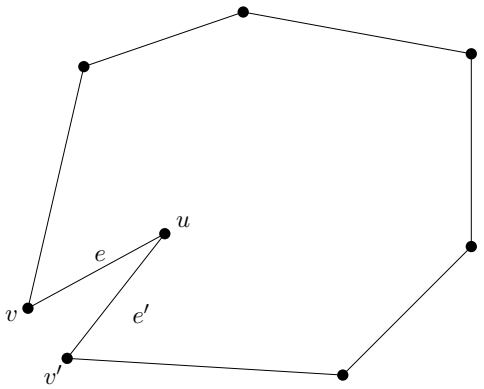


Figura 3.18: Cara interna f con n vértices y sin aristas puente.

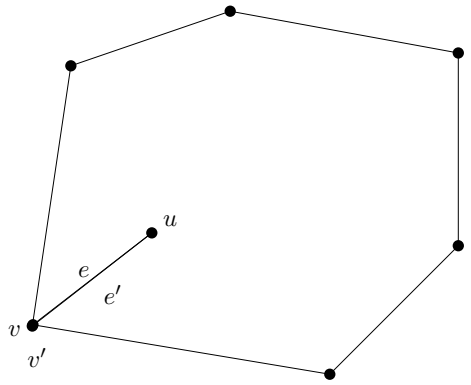


Figura 3.19: Cara interna f con $\alpha = 0$.

Ahora consideremos el caso donde se superponen más de dos vértices. Sea f_1 el polígono mostrado en la figura 3.20, donde se tienen m aristas y m vértices, y sea f_1'' el polígono resultante de superponer los vértices v, v' y v'' (figuras 3.21 y 3.22). Si se considera que f_1'' sigue siendo un polígono de m aristas y m vértices, el teorema 3.4.3 se siguirá cumpliendo. Como resultado tenemos que, cada vértice frontera se contará una sola vez, mientras que cada vértice híbrido debe ser contado según el número de aristas puente que sean adyacentes a él, más uno por formar parte de la frontera.

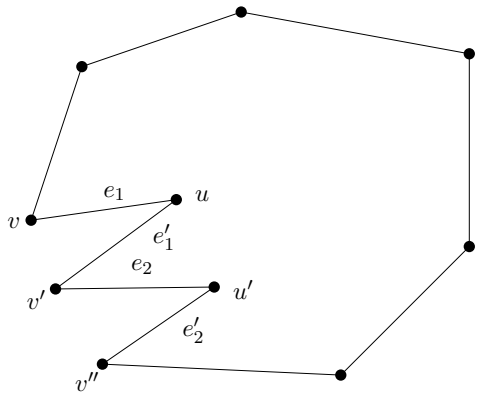


Figura 3.20: Cara interna f con m aristas y m vértices.

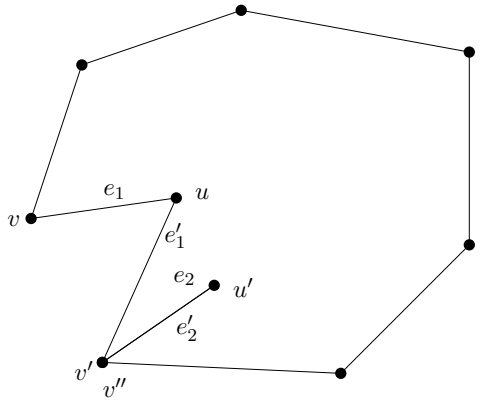


Figura 3.21: Cara interna f con un vértice superpuesto.

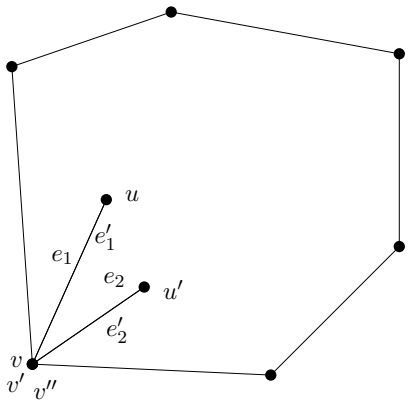


Figura 3.22: Cara interna f con dos vértices superpuestos.

Por último los vértices puente, los cuales pueden ser vistos como resultado de la superposición de un número d de vértices, donde d es el grado del vértice puente en G.

Al final podemos decir que, una cara con aristas puente en G puede ser vista como un polígono con vértices y aristas superpuestos. Por lo tanto, acomodando los resultados anteriores tenemos el siguiente teorema.

Teorema 3.4.4. La suma de los ángulos internos de una cara interna en G es igual a $\pi \cdot (n' - 2)$, donde n' = número de vértices frontera + número de vértices híbridos + número de aristas puente adyacentes a vértices híbridos + la suma de los grados de los vértices puente.

Corolario 3.4.4.1. La suma de los ángulos de la cara externa en G es igual a $\pi \cdot (n' + 2)$.

Clasificación local de una cara en una gráfica plana

Existe una manera fácil de identificar en una gráfica geométrica plana si una cara es interna o externa, además este método puede realizarse de forma local.

El método consiste en realizar el recorrido de la cara siguiendo alguna de las dos reglas mencionadas en la sección 3.4.1, contando los saltos durante el recorrido y midiendo en cada vértice el ángulo que se forma entre la arista por la cual se arribó y la arista por la cual se dejará el vértice (figura 3.23). El algoritmo 4 muestra el pseudocódigo que describe a este método.

Algoritmo 4 Dada una arista $e \in f$, determinar si f es una cara interna o externa

Entrada: Gráfica geométrica plana $G = (V, E)$, y una arista $e = \{u, v\}$ tal que $e \in f$.

Salida: Sea f la cara que se quiere revisar. La salida será **cierto** si f es una cara interna, y **falso** si es la cara externa.

1: Sea u el vértice de orden mayor en e (el vértice con coordena mayor sobre el eje horizontal, y en caso de empate, coordenada mayor sobre el eje vertical al comparar los vértices empatados). u será el vértice de inicio del recorrido.
2: Si f se encuentra a la derecha al realizar el recorrido de e en sentido $u \to v$ entonces se utilizará para el recorrido la regla de la mano izquierda. En caso contrario se utilizará la regla de la mano derecha. En ambos casos e será la arista de inicio.
3: El número de vértices empezará en uno
4: ángulo total $\leftarrow 0$
5: Al ángulo total se le sumará el valor absoluto del ángulo que forman la arista de inicio y la arista término, precisando que se trata del ángulo del lado del recorrido
6: **mientras** no se arribe a u a través de la arista de término **hacer**
7: Se avanzará al siguiente vértice en el recorrido siguiendo la regla correspondiente
8: El número de vértices se incrementará en uno
9: Al ángulo total se le sumará el valor absoluto del ángulo que forman la arista siguiente en el recorrido y la última arista recorrida, precisando que se trata del ángulo del lado del recorrido
10: **fin mientras**
11: **si** ángulo total $= \pi \cdot$ (número de vértices-2) **entonces**
12: **devolver cierto**
13: **si no**
14: **devolver falso**
15: **fin si**

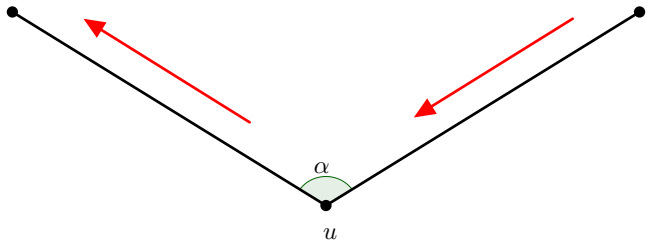


Figura 3.23: Recorrido de una cara siguiendo la regla de la mano izquierda. Cuando se arriba al vértice u, se cuenta el vértice y se mide el ángulo α.

Teorema 3.4.5. El algoritmo 4 reporta una vez cada vértice frontera de f, cada vértice híbrido lo reporta una vez por cada arista puente adyacente a él más una vez por ser parte de la frontera, y cada vértice puente es reportado una vez por cada arista incidente a él.

Prueba. Se puede observar que el algoritmo 4 contará cada vértice que se encuentre en el recorrido, sin importar si ya fueron contados o no. Los vértices frontera solo serán contados una vez, pues el recorrido pasará a través de ellos una sola vez. Los vértices híbridos se contarán una vez cuando el recorrido se encuentra en la frontera, y una vez por cada arista puente que sea adyacente a él, ya que por cada una de estas aristas, el recorrido llegará de nuevo al vértice híbrido, agregandolo cada una de estas veces a la cuenta total. Por último, cada arista puente será recorrida dos veces, contando cada vértice puente una vez por cada arista adyacente a él. □

Teorema 3.4.6. El algoritmo 4 tendrá complejidad de $O(n)$ en cuanto al tiempo de ejecución.

Prueba. El recorrido de la cara se realiza en tiempo $O(n)$ (teorema 3.4.2), y el algoritmo 4 realiza dos recorridos de la cara, uno para saber si f se encuentra a la derecha de la arista e en sentido $u \to v$, y otro más para contar los vértices y sumar los ángulos. □

3.4.3. Elección de líder

En algunos algoritmos es necesario designar un vértice o nodo especial, el cual se denomina como nodo "líder". El proceso de escoger un líder es conocido como *elección de líder*. La elección de líder no sólo es una forma de romper la simetría en una gráfica o red, también permite traducir algunos problemas a modelos computacionales notables [1].

En este caso nos concentraremos en el problema de la elección de líder en la topología de anillos.

Problema 2 (Elección de líder)**.** Cada nodo eventualmente decide si es o no el nodo líder, remarcando el hecho de que al final existe sólo un líder.

Observaciones:

- De manera más formal, los nodos están en uno de los tres estados siguientes: no decidido, líder o no líder. Inicialmente cada nodo se encuentra en el estado de no decidido. Una vez que deja el estado de no decidido, un nodo llega hasta un estado final (líder o no líder).

Definición 3.4.4 (Uniforme)**.** Un algoritmo es llamado *uniforme* si el número de nodos n no es conocido por el algoritmo (o por los nodos). Si n es conocido, el algoritmo es denominado *no uniforme* [1].

3.4.4. Algoritmo de elección de líder en un anillo asíncrono uniforme

En este modelo, cada nodo tiene un identificador único. Teniendo esta información, el problema de elección de líder se reduce a la elección de un nodo con, por ejemplo, el identificador con el valor más alto.

Observación 3.4.1 (Definición de ronda)**.** Una *ronda* en el algoritmo 5 será equivalente al envío de un mensaje de un nodo a su vecino inmediato en el anillo. Si dos o mas nodos realizan un envío de mensajes de manera síncrona, esos envíos se realizarán en una sola ronda. Por el contrario, si el envío se realiza de manera asíncrona, entonces cada envío de mensaje se realizará en una ronda diferente.

Teorema 3.4.7 (Análisis del algoritmo 5)**.** El algoritmo 5 es correcto. La complejidad en cuanto a tiempo es de $O(n)$. La complejidad en cuanto a número de mensajes es $O(n^2)$.

Prueba. Sea z el nodo con el identificador con el valor más alto dentro de un anillo. El nodo z envía su identificador en el mismo sentido al giro de las manecillas del reloj, y ya que ningún otro nodo tiene un valor mayor, eventualmente su mensaje llegará a el mismo indicando que él es el nodo líder. Cada uno de los nodos restantes podrá ser declarado como no líder almenos cuando termina el recorrido del mensaje de z. Ya que

Algoritmo 5 Elección de líder por paso de mensajes en un solo sentido

1: Cada nodo v ejecuta lo siguiente:
2: v envía un mensaje con su identificador (envía v) a su vecino que se encuentra en sentido negativo. Si el nodo v ya ha recibido un mensaje w con $w > v$, entonces puede omitir este paso.
3: **si** v recibe un mensaje w con $w > v$ **entonces**
4: v envía w a su vecino que se encuentra en sentido negativo
5: v decide no ser el líder
6: **si no**
7: **si** v recibe su propio identificador v **entonces**
8: v decide ser el lider
9: v avisa a todos los nodos del anillo que él es líder
10: **fin si**
11: **fin si**

hay n identificadores en el sistema, cada nodo enviará a lo más n mensajes, dando como resultado una complejidad en el número de mensajes de $O(n^2)$. Para la complejidad en tiempo, se supondrá que cada mensaje tomará tiempo de a lo más una unidad de tiempo en arribar a su destino, además se comenzará midiendo el tiempo desde el instante en que el primer nodo que despierte envíe su identificador a través del anillo. Después de a lo más $n-1$ unidades de tiempo el mensaje llegará al nodo z, despertándolo. El nodo z enviará el mensaje a través del anillo, lo cual tomará a lo más n unidades de tiempo. Por lo tanto el nodo z decidirá en un tiempo no mayor a $2n-1$ que él es el líder. Los demás nodos decidirán en un tiempo menor. □

En la figura 3.24 se muestra el recorrido de los mensajes de todos los nodos o vértices del anillo al querer realizar una elección de líder, cabe señalar que el hecho de que aparezca el envío de todos los mensajes al mismo tiempo (de manera síncrona) no implica que esto deba ser así. El nodo 10 se elige como líder al recibir su propio mensaje en la ronda 6.

Observación:

- En el algoritmo 5 un vértice necesita distinguir entre vecinos que se encuentran en sentido al giro de las manecillas del reloj y los que se encuentran en contrasentido. De hecho esto no es necesario, basta con que un nodo simplemente mande su mensaje con su identificador a cualquiera de sus vecinos, y reenvie un mensaje m al vecino del cual no recibió este mensaje. Por lo tanto los nodos necesitan sólo ser capaces de distinguir entre sus dos vecinos.

¿Se podrá mejorar el rendimiento de este algoritmo? A continuación veremos que el algoritmo 6 tiene un mejor desempeño.

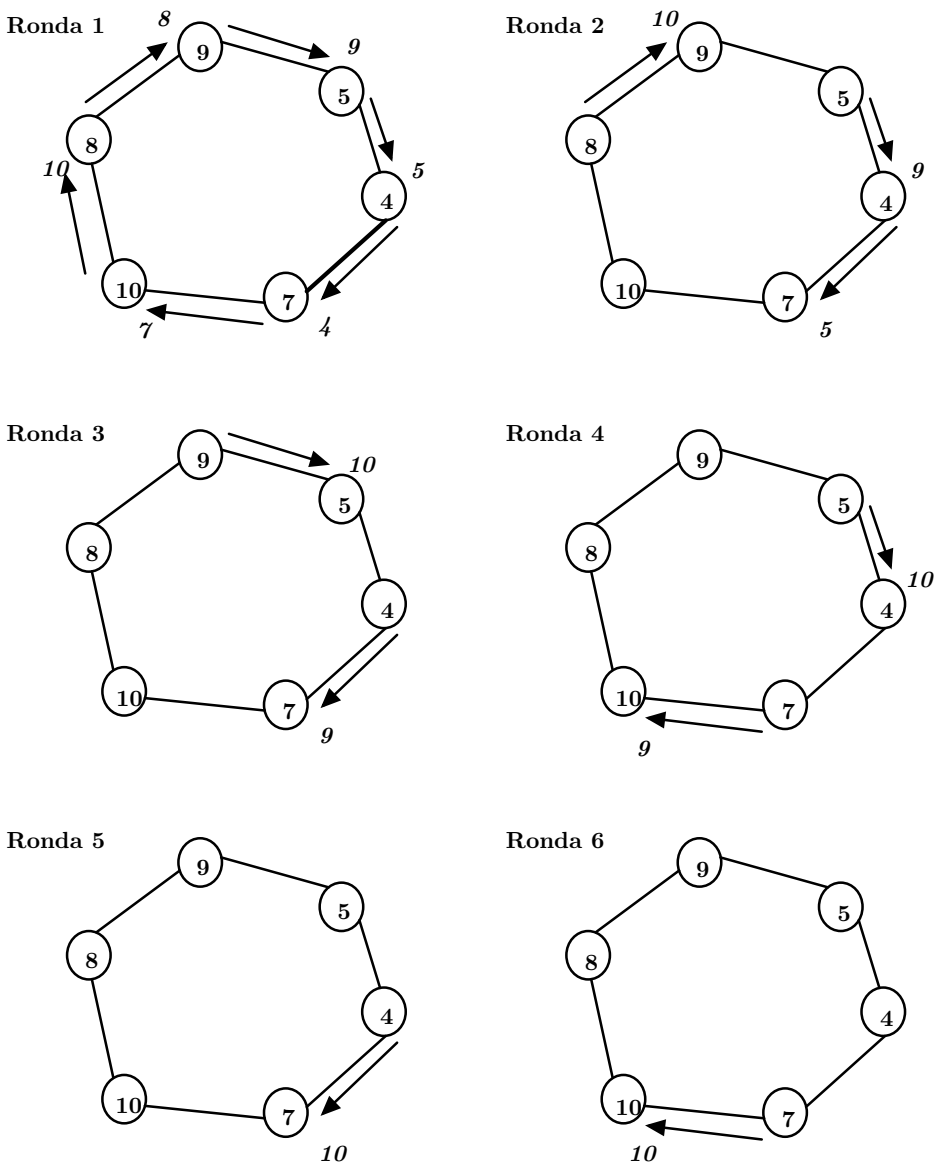


Figura 3.24: Ejemplo del algoritmo de elección de líder con mensajes en un solo sentido.

Algoritmo 6 Elección de líder por paso de mensajes en ambos sentidos

1: Cada nodo v ejecuta lo siguiente:
2: Inicialmente todos los nodos son candidatos a ser líder.
3: Cuando un nodo con identificador v recibe un mensaje con un identificador $w > v$, entonces v decide no ser líder y deja de ser candidato.
4: Los nodos candidato buscan en un vecindario de crecimiento exponencial (un vecindario que crece en potencias de 2 de manera simétrica en ambos sentidos del anillo) ser los nodos con el identificador más alto, mandando un mensaje con el valor de su identificador. Cada mensaje incluye el valor del identificador del nodo fuente, un bit de líder, un contador y un número de alcance. El número de alcance será 1 y el contador tendrá un valor de 0 al iniciar la ejecución del algoritmo (la primera ronda que ejecute el nodo).
5: Un nodo al recibir un mensaje comparará el valor del identificador en el mensaje con el valor de su propio identificador, si el valor del identificador del nodo es mayor al valor del identificador en el mensaje entonces el nodo encenderá el bit de líder en el mensaje (sólo si aún no está encendido). Después, el nodo incrementará el contador en el mensaje, si el valor del contador es igual al valor del número de alcance, el mensaje será regresado como un mensaje *replay* hasta llegar a su nodo fuente usando el paso de mensajes, de lo contrario el nodo enviará el mensaje a su siguiente vecino
6: Una vez que un nodo recibe el mensaje de *reply*, este verificará si no existe un nodo candidato con un identificador mayor al suyo en el área de búsqueda (verificando si se encuentra encendido alguno de los bits de líder en los mensajes), si no existe dicho nodo entonces se duplicará el valor de *número de alcance*; el contador se pondrá en cero, y dos nuevos mensajes serán enviados (uno en cada dirección). Si durante la búsqueda se encontró un candidato con un identificador mayor (el bit de líder de alguno de los mensajes se encontró encendido), entonces sólo se marcará el nodo como no líder y el nodo dejará de generar mensajes propios (con el valor de su identificador).
7: Si un nodo v recibe su propio mensaje (el mensaje original, no el mensaje *reply*) v decidirá convertirse en líder.

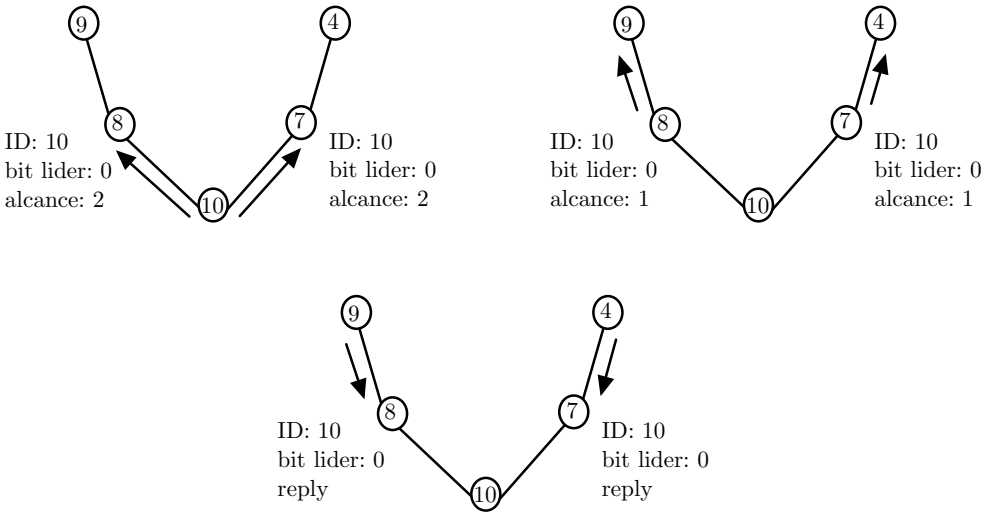


Figura 3.25: Ejemplo del paso de mensajes en el algoritmo 6.

En la Figura 3.26 se muestran todos los mensajes generados por los nodos del anillo al querer realizar una elección de líder, cabe señalar que el hecho de que aparezca el envío de todos los mensajes al mismo tiempo (de manera síncrona) no implica que esto deba ser así [7]. El nodo 10 se elige como líder al recibir su mensaje (no el replay) en la ronda 4 (se debe notar que en este algoritmo la ejecución de la i-ésima ronda abarca todo el proceso desde la salida del mensaje, el arribo de este a hasta los nodos a distancia 2^i, y el regreso del mensaje a su nodo fuente).

Teorema 3.4.8 (Análisis del algoritmo 6)**.** El algoritmo 6 es correcto. La complejidad en cuanto a tiempo es de $O(n)$. La complejidad en cuanto a número de mensajes es $O(n \log n)$.

Prueba. La correctez se verifica de la misma manera que en el teorema 3.4.7. La complejidad en cuanto a tiempo es de $O(n)$, ya que el nodo z con el identificador con el valor más grande envía mensajes de ida y vuelta tomando $4, 8, 16, 32, \ldots, 2 \cdot 2^k$ unidades de tiempo, con $k \leq 1 + \log(n)$. Probar la complejidad en cuanto al número de mensajes es más complicado: si un nodo v es candidato a líder al inicio de la ronda i, ningún otro nodo a distancia $\leq 2^{i-1}$ (por ambos lados) será candidato en la siguiente ronda. Dentro de cualquier grupo consecutivo de $2^{i-1}+1$ nodos, por lo menos uno de ellos es candidato a líder en la *i-ésima* ronda. Aunque posiblemente todos los n nodos serán candidatos a líder (y por lo tanto participarán en el envío de mensajes) en la primer ronda (de mensajes a distancia 1), a lo más $\frac{n}{2}$ nodos podrán participar en la segunda ronda (mensajes a distancia 2), a lo más $\frac{n}{3}$ en la tercera ronda (mensajes a distancia 4), a lo más $\frac{n}{5}$ en la cuarta ronda

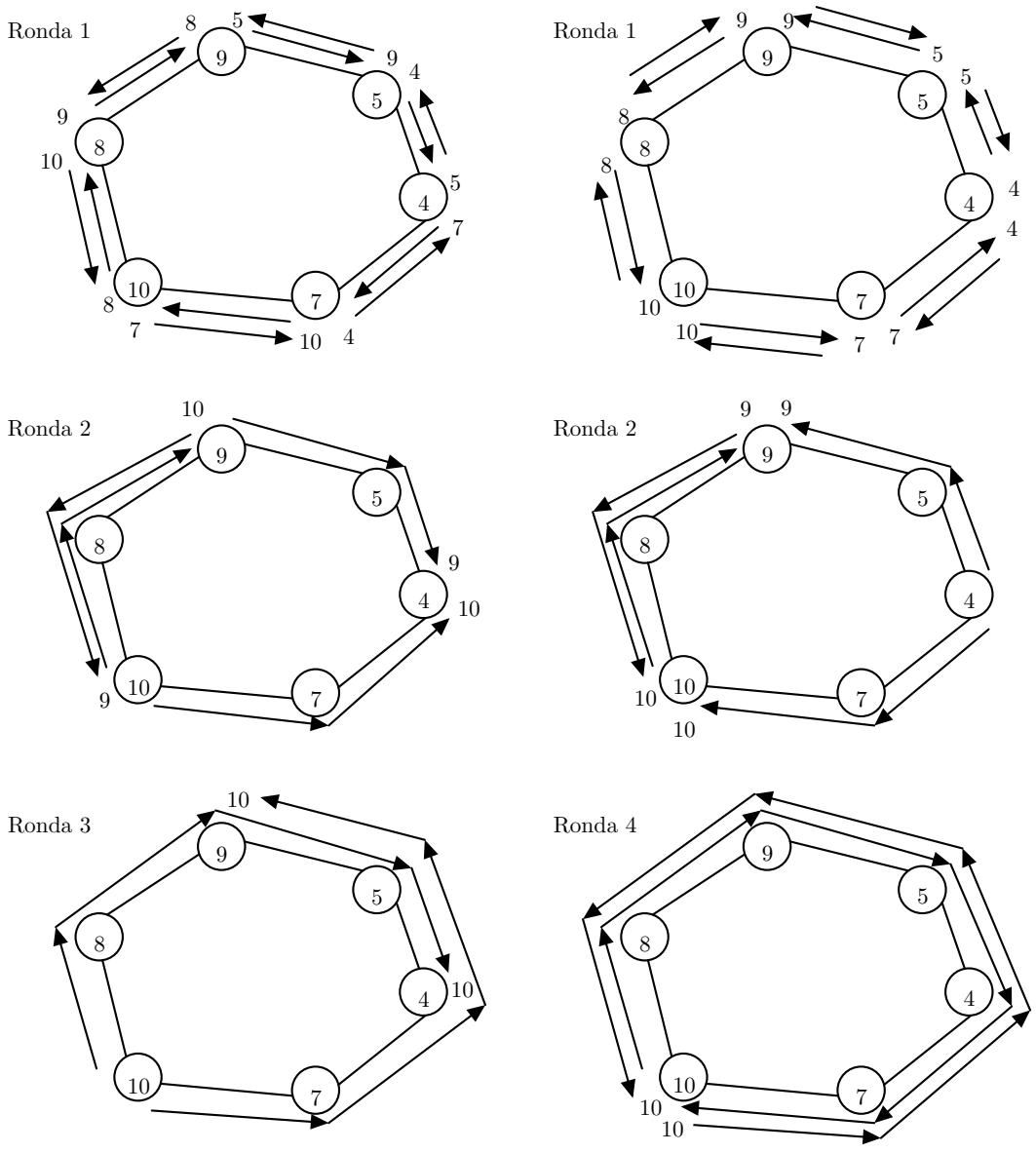


Figura 3.26: Ejemplo del algoritmo de elección de líder con mensajes en ambos sentidos.

(mensajes a distancia 8), etc.

Un nodo candidato a líder en la i-ésima ronda envía dos mensajes (un mensaje en cada sentido) a distancia 2^i, y la ronda termina al regreso de estos mensajes. Por lo tanto, al término de la i-ésima ronda cada nodo candidato generó el envío de a lo más $4 \cdot 2^i$ mensajes. La suma total de todos los mensajes generados (enviados) durante la ejecución del algoritmo es

$$4 \cdot (1 \cdot n + 2 \cdot \frac{n}{2} + 4 \cdot \frac{n}{3} + 8 \cdot \frac{n}{5} + \cdots + 2^i \cdot \frac{n}{2^{i-1}+1} + \dots)$$

Cada uno de los términos dentro del paréntesis tiene un valor $\leq 2n$, y se tienen a lo más $1 + (\log n)$ términos. (Ningún nodo generará el envío de mensajes a distancia mayor a $2n$, ya que, una vez que un nodo inicia el envío de mensajes a distancia n o mayor y el mensaje da toda la vuelta al anillo, el nodo recibe su propio mensaje (no el replay) y se marca como líder, terminando el envío de mensajes.) Así, el número total de mensajes generados es menor a

$$8n + 8(n \log n) = O(n \log n). \square$$

Capítulo 4

Identificación de conjuntos de corte

En el presente capítulo se encuentra nuestra propuesta de solución al problema de la identificación de conjuntos de corte de cardinalidad mínima en gráficas geométricas planas, mediante el uso de algoritmos locales. Todos los algoritmos presentados en esta sección están basados en resultados anteriores dentro del área de teoría de gráficas (capítulo 3).

4.1. Algoritmo de enumeración efectiva

En esta sección presentaremos un algoritmo local para reportar una vez cada vértice y cada arista de una gráfica geométrica plana. En [2] se presenta un algoritmo que realiza esto en $O(n \log n)$. El algoritmo que aquí presentamos tiene la misma complejidad temporal, sin embargo, consideramos que presenta mayor simplicidad, además de estar diseñado para ser un algoritmo completamente local.

4.1.1. Arista dominante de un vértice

Sea $G = (V, E)$ una gráfica geométrica plana conexa, donde V denota al conjunto de todos los vértices que pertenecen a G, y E es el conjunto de todas las aristas.

Para todo $u \in V$, u estará caracterizado por sus coordenadas en el plano, además se trabajará bajo el supuesto de que un punto es único en su posición, es decir, dada cualquier posición en el plano existirá a lo más un vértice en esta.

Para definir la arista dominante de un vértice en la gráfica G se realizará lo siguiente. Sea $u \in V$, se traza una semirecta vertical a partir de u en la dirección $(0, 1)$, este rayo se denominará *línea de barrido* (figura 4.1).

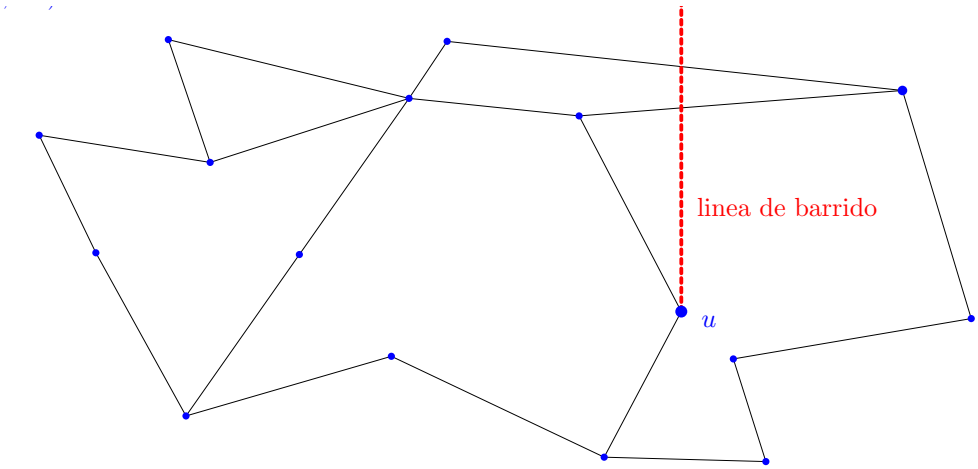


Figura 4.1: Definición de la linea de barrido del vértice u.

Definición 4.1.1. Sea $u \in V$, y sea c el conjunto de todas las aristas en G que son adyacentes al vértice u. Se denominará *arista dominante* de u a aquella arista que pertenezca a c y que forme el ángulo menor con la linea de barrido (figura 4.2).

Lema 4.1.1. Para todo vértice $u \in V$, está definida exactamente una arista dominante.

Prueba. Sea u cualquier vértice en G. Dado que G es conexa, el conjunto c de aristas adyacentes a u contiene por lo menos un elemento. Por definición, sin importar la cardinalidad de c, la arista dominante de u será la arista que pertenezca a c y que forme el ángulo menor con la linea de barrido. No pueden existir dos o más aristas que generen el mismo ángulo con la linea de barrido, ya que G es geométrica y plana. □

4.1.2. Arista de entrada

Una gráfica geométrica plana conexa divide el plano en cierto número de regiones conexas, las cerraduras de estas regiones se denominan caras. Si se elimina una de las aristas frontera de una de las regiones, esta se mezclará y formará una nueva región (cara) con la región colindante a la arista que se eliminó. Si este procedimiento se realiza con todas las caras internas, al final quedará sólo una cara en la gráfica, que será la externa. Además, si durante el procedimiento se cuida de no romper la conexidad de la gráfica (no formar más componentes conexas) entonces al final del procedimiento la gráfica resultante será un árbol. Por tanto, se busca definir un conjunto de aristas que, al eliminarlas de una

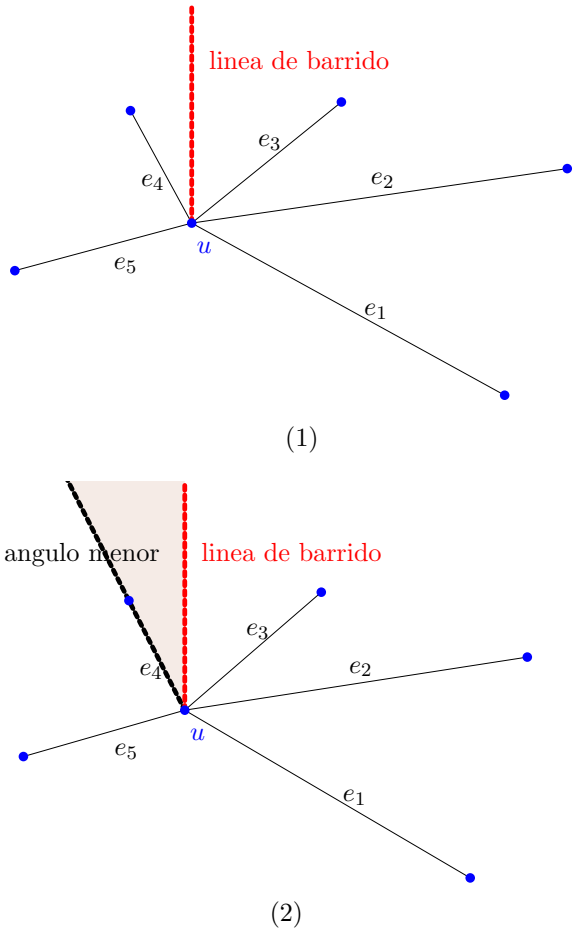


Figura 4.2: Definición de arista dominante del vértice u. En este caso la arista dominante de u es la arista e_4.

gráfica geométrica plana conexa, resulte un árbol generador de la gráfica. Este conjunto se denominará conjunto de aristas de entrada.

Definición 4.1.2. Sea f una cara interna en G y u el vértice con abscisa mayor en f (en caso de empate, u será el vértice con la ordenada menor de los vértices empatados). Entonces se dirá que u es el *vértice de extrema derecha* en f.

Definición 4.1.3. Sea $e = \{u, v\}$ con $u = (x_u, y_u)$ y $v = (x_v, y_v)$. Se dirá que v es el *vértice de orden mayor* en e si $x_v > x_u$ o $x_v = x_u$ y $y_v < y_u$.

Ahora, se utilizará una variante de la definición de linea de barrido.

Sea $u \in V$, se traza una semirecta vertical a partir de u en la dirección $(0, -1)$, este rayo se denominará *línea de barrido invertida*.

Definición 4.1.4. Sea f una cara interna en G y u el vértice de extrema derecha en f. La *arista de entrada* en f será aquella arista de f que sea adyacente a u y que, al trazar la linea de barrido invertida en u y medir los ángulos en sentido negativo, forma el ángulo menor en valor absoluto con la linea de barrido invertida.

Teorema 4.1.1. Toda arista de entrada en G es una arista frontera.

Prueba. Sea f una cara interna y u el vértice de extrema derecha en f. Al trazar la línea de barrido invertida en u la linea debe encontrarse completamente fuera de f (a excepción de su intersección con u), de no ser así implicaría que u no es el vértice de extrema derecha en f. Por lo tanto, al trazar la linea de barrido invertida en u y medir los ángulos en sentido negativo, la arista de f que forme el ángulo menor en valor absoluto con la linea de barrido invertida debe ser una arista frontera. □

Corolario 4.1.1.1. Al eliminar una arista de entrada el número de caras en G disminuye en 1.

Lema 4.1.2. Sean f_i y f_j dos caras internas cualesquiera en G, y e_i y e_j sus respectivas aristas de entrada. Si $f_i \neq f_j$ entonces $e_i \neq e_j$.

Prueba. Por contradicción. Suponemos que $f_i \neq f_j$ y $e_i = e_j$. Sea $e_i = \{u, v\}$, donde sin pérdida de generalidad diremos que u es el vértice de orden mayor en e_i. Ya que la arista de entrada es la misma para las dos caras, u es el vértice de extrema derecha en ambos casos, además e_i debe ser colindante por un lado a f_i y por el otro a f_j. Al trazar la línea de barrido invertida sobre u, en la región que se encuentra en el ángulo que forman esta línea y la arista e_i se debe encontrar una de las caras a las que es adyacente e_i, que sin pérdida de generalidad diremos que es f_j. Por esto, o la linea de barrido se encuentra contenida en f_j implicando que u no es el vértice de extrema derecha de f_j, o existe una arista frontera de f_j que forma un ángulo menor con la linea de barrido, y claramente ninguno de los dos casos son válidos. Por lo tanto, para que $e_i = e_j$, se debe cumplir que $f_i = f_j$, lo cual es una contradicción. □

Lema 4.1.3. Sea F el conjunto de las caras internas en G con $|F| = k$, y sea A_e el conjunto de aristas de entrada en G. Al realizar $E \backslash A_e$ la gráfica resultante será un árbol generador de G.

Prueba. Por inducción sobre el número de caras en G. Sea $G' = (V', E')$ la gráfica resultante, donde claramente $V' = V$ y $E' = E \backslash A_e$. Caso base: $|F| = 0$. G es conexa y ya que no existen caras internas $G' = G$, además G' no contiene ciclos, por lo tanto G' es un árbol generador.

Se supondrá cierta la propiedad para $|F| = k$, y se comprobará para $|F| = k + 1$. Sea c el conjunto de vértices de extrema derecha de todas las caras internas en G, y u el vértice con abscisa mayor en c (y en caso de empate, con la ordenada menor de los vértices empatados). Trazamos la línea de barrido invertida en u. Sea e la arista de entrada adyacente a u que, al medir los ángulos en sentido negativo forma el ángulo menor en valor absoluto con la línea, y sea f_e la cara para la cual e es la arista de entrada. La arista e es adyacente a la cara externa (por la posición de u), por lo tanto al eliminar e, f_e se mezclará con la cara externa, reduciendo en uno el número de caras en G (corolario 4.1.1.1) y dejando el resto de las caras internas iguales, por lo que los elementos de $c \backslash e$ siguen siendo aristas de entradas válidas en $G \backslash e$. Ahora, por hipótesis de inducción, al eliminar el resto de las aristas de entrada de $G \backslash e$ lo que queda es un árbol generador. □

Búsqueda de la arista de entrada en una cara interna

A continuación se presenta el algoritmo 7, el cual muestra el procedimiento que se debe llevar a cabo para, dada una arista, una dirección sobre la arista y una regla de recorrido (mano izquierda o derecha), encontrar la arista de entrada de una cara a la que pertenece. Se puede observar que el algoritmo es local, ya que está basado en el algoritmo 5 presentado en la sección 3.4.3.

Es fácil ver que el funcionamiento del algoritmo 7 es idéntico al descrito en el algoritmo de elección de líder por paso de mensajes en un solo sentido en la sección 3.4.3 (a excepción de la clasificación de una cara en interna o externa, cuyo costo es de $O(n)$ (algoritmo 4), lo cual no afecta el orden de complejidad del algoritmo), por lo que ni la complejidad ni la corrección de este algoritmo serán analizadas en esta sección.

También se presenta el algoritmo 8, el cual muestra el procedimiento para, dada una arista, saber si es la arista de entrada de una cara a la que pertenece. Se puede observar que el algoritmo es local, ya que está basado en el algoritmo 6 presentado en la sección 3.4.3.

Algoritmo 7 Dada una arista e, encontrar la arista de entrada de una cara a la que pertenece

Entrada: Gráfica geométrica plana conexa $G = (V, E)$, $e = \{u, v\}$ tal que $e \in E$, una dirección $a = (u, v)$, y una regla de recorrido r (regla de la mano derecha o regla de la mano izquierda).

Salida: Sea f la cara a la que se encuentra a la derecha de la arista e al recorrerla en sentido a siguiendo la regla de la mano izquierda, o sea f la cara a la que se encuentra a la izquierda de la arista e al recorrerla en sentido a siguiendo la regla de la mano derecha (según el valor de r). La salida será o la arista de entrada de f si es una cara interna, o el vértice de extrema derecha de f si es la cara externa.

1: u será el vértice de inicio del recorrido.
2: u generará un mensaje que contendrá su identificador (sus coordenadas), este mensaje lo enviará a v siguiendo la regla r.
3: **mientras** no llegue el mensaje a u a través de la arista de término (definición 3.4.3) **hacer**
4: **si** el vértice w recibe un mensaje con identificador z donde $z > w$ **entonces**
5: w enviará un mensaje que contendrá a z a su vecino siguiendo la regla r con respecto a la arista por la cual recibió el mensaje
6: **si no**
7: w enviará un mensaje que contendrá a w a su vecino siguiendo la regla r con respecto a la arista por la cual recibió el mensaje
8: **fin si**
9: **fin mientras**
10: sea x el identificador en el mensaje
11: **si** f es una cara interna **entonces**
12: devolver e', donde e' es la arista adyacente a x que pertenecen a f y que, al medir los ángulos en sentido negativo forma el ángulo menor en valor absoluto con respecto a la línea de barrido invertida trazada en x
13: **si no**
14: devolver x
15: **fin si**

Algoritmo 8 Dada una arista, revisar si es la arista de entrada de una cara a la que pertenece.

Entrada: Gráfica geométrica plana conexa $G = (V, E)$, $e = \{u, v\}$ tal que $e \in E$, una dirección $a = (u, v)$, y una regla de recorrido r (regla de la mano derecha o regla de la mano izquierda).

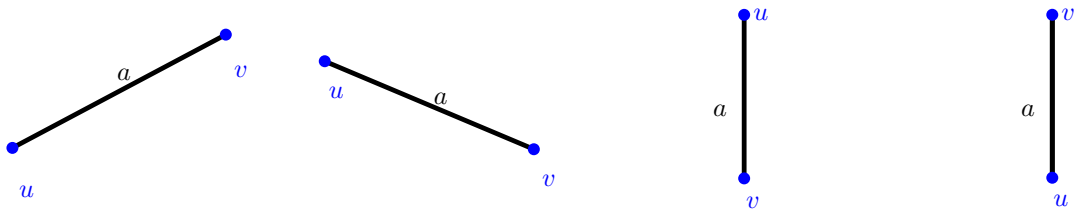
Salida: Sea f la cara a la que se encuentra a la derecha de la arista e al recorrerla en sentido a siguiendo la regla de la mano izquierda, o sea f la cara a la que se encuentra a la izquierda de la arista e al recorrerla en sentido a siguiendo la regla de la mano derecha (según el valor de r). La salida será **cierto** si e es la arista de entrada en f, y **falso** en caso contrario.

1: u será el vértice de inicio del recorrido.
2: u enviará dos mensajes, un mensaje siguiendo la regla r al vértice v, y el otro a su vecino que se encuentra al seguir el recorrido, utilizando la regla opuesta a r y el sentido contrario a a, todo con respecto a e
3: Identificar para cada mensaje su arista de inicio y su arista de término
4: Cada mensaje incluirá el identificador de u(sus coordenadas), un número de saltos y una variable llamada alcance. Cuando u genera el mensaje el número de saltos es cero. En la primera ejecución (ronda) la variable alcance tendrá el valor de uno.
5: **mientras** el mensaje m original no llegue a u a través de su arista de término o el mensaje m reply no llegue a u a través de su arista de inicio **hacer**
6: **si** el mensaje no es reply **entonces**
7: Incrementar en uno el número de saltos
8: Sea z el vértice en el que se encuentra m y sea w el identificador en m
9: **si** $z > w$ **entonces**
10: remplazar el identificador en m por el identificador z
11: **fin si**
12: **si** número de saltos = alcance **entonces**
13: invertir el sentido del recorrido de m (cambiar de dirección y de regla), marcar m como reply y enviar m al vecino siguiente en el recorrido ya invertido
14: **si no**
15: enviar m siguiendo la regla por la cual arribó el mensaje
16: **fin si**
17: **si no**
18: enviar m siguiendo la regla por la cual arribó el mensaje
19: **fin si**
20: **fin mientras**
21: **si** u recibe su mensaje original (no el mensaje reply) **entonces**
22: **si** f es una cara interna **entonces**
23: sea e' la arista de salida en u que pertenece a f y que, al medir los ángulos en sentido negativo, forma el ángulo menor en valor absoluto con respecto a la línea de barrido invertida trazada en u
24: **si** $e = e'$ **entonces**
25: **devolver cierto**
26: **si no**
27: **devolver falso**
28: **fin si**
29: **si no**
30: **devolver falso**
31: **fin si**
32: **si no**
33: **si** el identificador en el mensaje de reply es el identificador de u **entonces**
34: repetir el envío de mensajes (a partir del paso 1) aumentando al doble el valor de la variable alcance
35: **si no**
36: **devolver falso**
37: **fin si**
38: **fin si**

El funcionamiento del algoritmo 8 es idéntico al descrito en el algoritmo de elección de líder por paso de mensajes en ambos sentidos en la sección 3.4.3, por lo que ni la complejidad ni la corrección de este algoritmo serán analizadas en esta sección.

4.1.3. Otros preliminares

Sea $e = \{u, v\}$ con $e \in E$ y $u, v \in V$, cuyas coordenadas de los vértices en el plano son $u = (x_1, y_1)$ y $v = (x_2, y_2)$. Se dirá que la arista e esta siendo *revisada por la izquierda* si durante el algoritmo, al estar posicionado el agente sobre el vértice u, $x_2 > x_1$. Para el caso donde $x_1 = x_2$, se dirá que la arista e esta siendo revisada por la izquierda si durante el algoritmo, al estar posicionado el agente sobre el vértice u, $y_2 > y_1$. En la figura 4.3, suponiendo que el agente está posicionado sobre u, sólo para los casos 4.3a, 4.3b y 4.3d la arista e está siendo revisada por la izquierda, mientras que en 4.3c no es así.



(a) (b) (c) (d)

Figura 4.3: Las aristas en 4.3a, 4.3b y 4.3d están siendo revisadas por la izquierda, mientras que en 4.3c no es así.

Sea $u \in V$. La función $eDominante(u)$ recibe como entrada el vértice u y regresa la arista $e = \{u, v\}$, donde e será la arista dominante del vértice u.

Sea $u \in V$ y $e \in E$. La función $barridoRot(u, e)$ toma como entrada el vértice u y la arista e, y regresa el valor e', donde e' es la arista adyacente a u que forma el ángulo menor con la arista e al medir los ángulos en sentido positivo (figura 4.4) .

Sea $e = \{u, v\}$ una arista con $u, v \in V$ y $e \in E$. La función $esDeEntrada(e)$ la definiremos como aquella función que contestará con un valor booleano a la pregunta ¿es e la arista de entrada de alguna cara a la que pertenece?, si es así la función regresará *cierto*, de lo contrario regresará *falso*. La función $esDeEntrada()$ estará descrita por el algoritmo 8, por lo tanto tendrá una complejidad temporal de $O(n \log n)$ amortizado, es decir, la complejidad está dada por la ejecución de la función en todas las aristas de G, mientras que el tiempo de ejecución de la función para una sola arista es de $O(n)$.

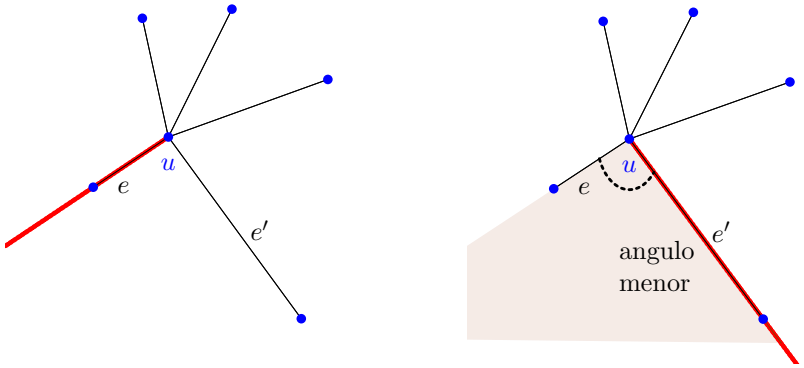


Figura 4.4: Definición de la función $barridoRot(u, e)$.

Es importante señalar que, para saber si una arista es de entrada para la o las caras a las que pertenece, es necesario ejecutar dos veces el algoritmo 8, en ambos casos la arista y la dirección serán las mismas, sin embargo la regla deberá cambiar, ya que esto realizará el cambio de cara que deseamos. Con estas condiciones se asegura que al realizar el algoritmo serán revisadas todas las caras a las que pertenece la arista.

La función $sucesor(e)$, toma como valor de entrada una arista $e = \{u, v\}$ y devuelve e', donde e' será la siguiente arista en el recorrido de una cara a la que pertenece e siguiendo la regla correspondiente. Para especificar la cara en la cual se está realizando el recorrido es necesario especificar una dirección sobre la arista e.

4.1.4. Algoritmo de enumeración efectiva

Sea $G = (V, E)$ una gráfica geométrica plana conexa. El algoritmo 9 describe el procedimiento para reportar cada arista y cada vértice de G exactamente una vez, a través de un recorrido completo de la gráfica. Se debe notar que con algunos cambios el algoritmo puede ser modificado para también reportar todas las caras en la gráfica (ya que cada cara se encuentra asociada a una sola arista de entrada).

Lema 4.1.4. El algoritmo 9 visita cada arista de la gráfica exactamente dos veces.

Prueba. Al realizar el recorrido de la gráfica se evita recorrer todas aquellas aristas que sean de entrada, por lo que el recorrido realizado sobre G será equivalente al recorrido (siguiendo alguna de las dos reglas) sobre un árbol generador en la gráfica. Por esto, cada arista será revisada dos veces, una en cada sentido, incluso las aristas de entrada, ya que

Algoritmo 9 Algoritmo de enumeración efectiva

Entrada: Gráfica geométrica plana conexa $G = (V, E)$, u tal que $u \in V$
Salida: Reporta cada vértice y cada arista de la gráfica G una sola vez

1: $e_u = \{u, v\} \leftarrow eDominante(u)$
2: $a = (u, v)$
3: $e_{inicio} \leftarrow e_u$
4: $a_{inicio} \leftarrow a$
5: **repetir**
6: Sea $a = (u, v)$
7: **si** $e_u = eDominante(u)$ **entonces**
8: Reportar el vértice u
9: **fin si**
10: **si** e_u está siendo revisada por la izquierda **entonces**
11: Reportar e_u
12: **fin si**
13: **si** $esDeEntrada(e_u)$ **entonces**
14: $e_u = \{u, v\} \leftarrow barridoRot(u, e_u)$
15: $a \leftarrow (a \cap \{u, v\}, \{u, v\} \backslash (a \cap \{u, v\}))$
16: **si no**
17: $e_u = \{u, v\} \leftarrow sucesor(e_u)$
18: $a \leftarrow (a \cap \{u, v\}, \{u, v\} \backslash (a \cap \{u, v\}))$
19: **fin si**
20: **hasta que** $e_u = e_{inicio}$ y $a = a_{inicio}$

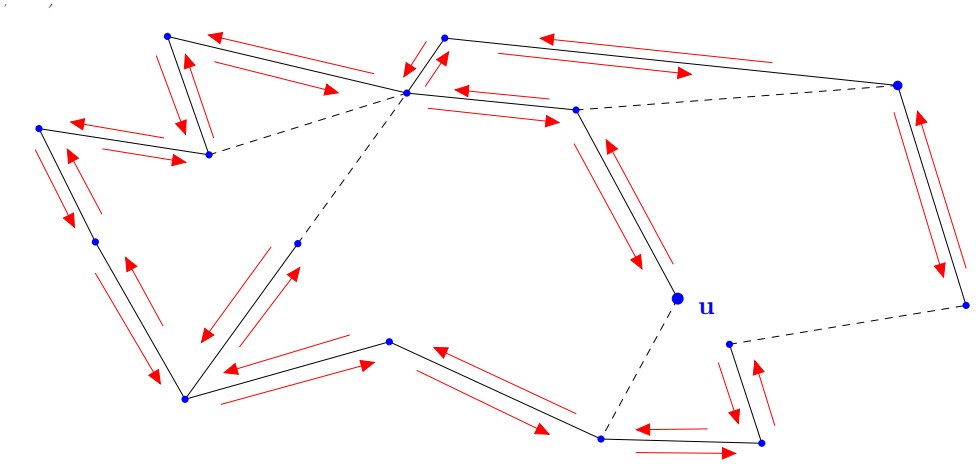


Figura 4.5: Recorrido en el algoritmo de enumeración efectiva. El vértice u es el vértice de inicio, y las aristas en lineas punteadas representan las aristas de entrada de la gráfica.

antes de ser descartadas en el recorrido tendrán que ser evaluadas para ver si son aristas dominantes de algún vértice o verificar si deben ser reportadas o no. □

Teorema 4.1.2. El algoritmo 9 reporta sólo una vez cada arista de la gráfica G.

Prueba. Al realizar el recorrido de G, se visitará cada arista $e = \{u, v\}$ dos veces (lema 4.1.4), una en sentido (u, v) y otra en sentido (v, u). Por construcción del algoritmo, la arista sólo será reportada si durante el recorrido está siendo revisada por la izquierda. □

Teorema 4.1.3. El algoritmo 9 reporta cada vértice una vez.

Prueba. Un vértice es reportado sólo si se desea salir de él a través de su arista dominante. Por el lema 4.1.4, al realizar el recorrido de la gráfica cada arista es revisada una sola vez en cada sentido, en particular todas las aristas que son dominantes para algún vértice. Por lo tanto cada vértice será reportado una sola vez, lo cual ocurrirá cuando la arista dominante del nodo en cuestión sea recorrida y él sea el vértice origen. □

Teorema 4.1.4. El algoritmo 9 tiene un tiempo de ejecución de $O(n \log n)$.

Prueba. Las funciones *aDominante*(), *barridoRot*() y *sucesor*() son de $O(1)$. El algoritmo 9 ejecuta estas funciones dos veces por cada arista, lo cual tiene una relación

lineal con el número de vértices en la gráfica, por lo tanto el tiempo total de ejecución de estas funciones será de $O(n)$. La función *esDeEntrada*() es una función cuyo tiempo de ejecución es de $O(n \log n)$ amortizado (costo total de ejecución (teorema 3.4.8)). Por lo tanto el algoritmo tendrá una complejidad de $O(n \log n)$. □

4.2. Detección de aristas de corte

Antes de presentar el algoritmo debemos definir dos conceptos más. *eEntrada(a)* es una función que recibe una arista $e = \{u, v\}$ y devuelve la arista de entrada de una cara a la que pertenece e. Para especificar la cara de la que se desea saber su arista de entrada es necesario especificar una dirección sobre la arista e y una regla de recorrido. Esta función está descrita por el algoritmo 7.

Por último, dada una dirección sobre una arista, $a = (u, v)$, diremos que $\overline{a} = (v, u)$, es decir, $\overline{a}$ será la dirección opuesta de a. Así también para r, donde, si r es la dirección de recorrido siguiendo la regla de la mano derecha, $\overline{r}$ será la dirección de recorrido siguiendo la regla de la mano izquierda.

A continuación se presenta el algoritmo 10, el cual realiza la detección de aristas de corte en una gráfica geométrica plana conexa. El algoritmo tomará como entrada una gráfica y una arista de la misma, y contestará si la arista es o no una arista de corte.

Algoritmo 10 Determinar si una arista es de corte

Entrada: Gráfica geométrica plana conexa $G = (V, E)$, $e = \{u, v\}$ tal que $e \in E$.
Salida: **cierto** si e es arista de corte en G y **falso** en caso contrario.

1: $a \leftarrow (u, v)$ (dirección inicial del recorrido)
2: r será la regla del recorrido (mano derecha o mano izquierda)
3: $e_{suc} = \{v, w\} \leftarrow sucesor(e)$ (valores a y r)
4: $a' \leftarrow (v, w)$
5: **mientras** $e_{suc} \neq e$ y $a' \neq a$ **hacer**
6: **si** $e_{suc} = e$ y $a' = \overline{a}$ **entonces**
7: **devolver cierto**
8: Terminar el algoritmo
9: **si no**
10: $e_{suc} = \{w, z\} \leftarrow sucesor(e_{suc})$ (valores a' y r)
11: $a' \leftarrow (a' \cap \{w, z\}, \{w, z\} \backslash (a' \cap \{w, z\}))$
12: **fin si**
13: **fin mientras**
14: **devolver falso**

4.2.1. Corrección

Sea $G = (V, E)$ una gráfica geométrica plana conexa, y $e = \{u, v\}$ tal que $e \in E$. El algoritmo 10 determina si e es una arista de corte en G.

Prueba. Por la definición 3.3.1, para determinar si e es una arista de corte basta con verificar si e pertenece a una sola cara.

Por construcción, en el algoritmo se realiza un recorrido de una cara a la que pertenece la arista e, la cual es determinada por la dirección y la regla de recorrido que se escoja.

Si al realizar el recorrido se visita la arista e más de dos veces (punto 6 del algoritmo), entonces e pertenece a una sola cara. El algoritmo responderá cierto si este es el caso. En caso de que e sea visitada sólo una vez durante el recorrido (termina el ciclo *while*), el algoritmo contestará falso. □

4.2.2. Complejidad

El algoritmo 10 toma tiempo $O(n)$ para decidir si e es una arista de corte en G.

Prueba. Por construcción del algoritmo 10, se realiza el recorrido de una cara a la que pertenece la arista e, para el peor de los casos el recorrido abarcará toda la gráfica.

Sin importar si la cara del recorrido es interna o externa, el recorrido nos llevará a recorrer una sola vez todas las aristas frontera de la cara, y dos veces todas las aristas puente, por lo tanto el recorrido será de orden lineal en cuanto al número de aristas de la gráfica, lo cual por ser gráfica plana nos dará un orden lineal en cuanto al número de vértices en el gráfica. □

Es fácil ver que, si el algoritmo 10 es ingresado como una subrutina en el algoritmo de enumeración efectiva (algoritmo 9), lo que se tendrá será un algoritmo que, para cada arista de la gráfica, verifique si es o no una arista de corte. Dicho algoritmo tendrá una complejidad de $O(n^2)$.

4.3. Detección de conjuntos de corte de cardinalidad 2

El algoritmo 11 toma como entrada una gráfica geométrica plana conexa y una arista $e = \{u, v\}$ perteneciente a esta, y encontrará (si es que existen) todos los pares de corte que forma e en la gráfica.

Algoritmo 11 Reportar los pares de corte a los que pertenece una arista

Entrada: Gráfica geométrica plana conexa $G = (V, E)$, $e = \{u, v\}$ tal que $e \in E$.

Salida: Reporta todos los pares de aristas de corte que forma e con las demás aristas de la gráfica

1: $a \leftarrow (u, v)$ (dirección inicial del recorrido)
2: r será la regla del recorrido (mano derecha o mano izquierda)
3: $Cara_{(e)} \leftarrow eEntrada(e)$ (valores a y r)
4: $Cara_{inv(e)} \leftarrow eEntrada(e)$ (valores a y $\overline{r}$)
5: $candidato = \{v, w\} \leftarrow sucesor(e)$ (valores a y r)
6: $a' \leftarrow (v, w)$
7: **mientras** $candidato \neq e$ y $a' \neq a$ **hacer**
8: $\quad Cara_{(c)} \leftarrow Cara_{(e)}$
9: $\quad Cara_{inv(c)} \leftarrow eEntrada(candidato)$ (valores a' y $\overline{r}$)
10: $\quad$ **si** $Cara_{(c)} \neq Cara_{inv(c)}$ **entonces**
11: $\quad\quad$ **si** $Cara_{inv(e)} = Cara_{inv(c)}$ **entonces**
12: $\quad\quad\quad$ **devolver** e y $candidato$ como un par de corte
13: $\quad\quad$ **fin si**
14: $\quad$ **fin si**
15: $\quad candidato = \{w, z\} \leftarrow sucesor(candidato)$ (valores a' y r)
16: $\quad a' \leftarrow (a' \cap \{w, z\}, \{w, z\} \backslash (a' \cap \{w, z\}))$
17: **fin mientras**

4.3.1. Corrección

Sea $G = (V, E)$ una gráfica geométrica plana conexa, y $e = \{u, v\}$ tal que $e \in E$. El algoritmo 11 encontrará (si es que existen) todos los pares de corte que forma la arista e en la gráfica G.

Prueba. Por el teorema 3.3.1, una arista $e = \{u, v\}$ sólo puede formar un par de corte con aristas que pertenezcan a sus mismas dos caras, por lo tanto, al recorrer una de las caras a las que pertenece e (la cara a la que se encuentra a la derecha de e al recorrerla en sentido a siguiendo la regla de la mano izquierda, o la cara a la que se encuentra a la izquierda de e al recorrerla en sentido a siguiendo la regla de la mano derecha, según el valor de r), todas las aristas que se recorren en esa cara son candidatas a formar un par de corte con e.

Para verificar si una arista candidata forma o no un par de corte con e, es necesario (para cada una de las aristas candidatas) revisar la otra cara a la que pertenece la arista candidata. Si se verifica que la arista candidata pertenece a las mismas dos caras a las que pertenece e ($Cara_{inv(e)} = Cara_{inv(c)}$), entonces se reporta esa arista candidata junto con la arista e como un par de corte y se sigue con el recorrido de la cara, en caso contrario sólo se descarta esa arista y se sigue la búsqueda en las aristas candidatas restantes.

Por último, es importante señalar que el algoritmo distingue un caso muy particular, cuando tanto la arista candidata como la arista e son aristas de corte (ver linea 10 del algoritmo 11).□

4.3.2. Complejidad

El algoritmo 11 tiene un tiempo de ejecución de $O(n^2)$.

Prueba. En el algoritmo 11 se realiza un recorrido de una de las caras a las que pertenece la arista e (la cara a la que se encuentra a la derecha de e al recorrerla en sentido a siguiendo la regla de la mano izquierda, o la cara a la que se encuentra a la izquierda de e al recorrerla en sentido a siguiendo la regla de la mano derecha, según el valor de r), este recorrido se realiza en tiempo de $O(n)$. Dentro del ciclo, se ejecuta la función $eEntrada()$, función que, dada una arista e', devuelve como valor la arista de entrada de una cara a la que pertenece e'. Para conocer dicha arista de entrada se tiene que realizar el recorrido completo de una cara, lo cual toma tiempo de $O(n)$. Esta función se tiene que realizar un número lineal de veces (una vez por cada arista que se visita en el recorrido de la cara), por lo tanto el tiempo final de ejecución del algoritmo es de $O(n^2)$. □

Una vez más, si el algoritmo 11 es ingresado como una subrutina en el algoritmo de enumeración efectiva (algoritmo 9), lo que se tendrá será un algoritmo que, para cada arista

de la gráfica, reporte los pares de corte a los que pertenece. Dicho algoritmo tendrá una complejidad de $O(n^3)$.

4.4. Detección de conjuntos de corte de cardinalidad m

El algoritmo 12 es la generalización del algoritmo presentado en la sección anterior. Es fácil ver que, según el tamaño del conjunto de corte es el número de caras que se deben recorrer para saber si forman un ciclo en la gráfica dual. Por lo tanto la complejidad del algoritmo estará dada por el tamaño del conjunto de corte que se desea obtener.

Algoritmo 12 Reportar los conjuntos de corte de cardinalidad m a los que pertenece una arista

Entrada: Gráfica geométrica plana conexa $G = (V, E)$, arista $e = \{u, v\}$ tal que $e \in E$.
Salida: Reporta todos los conjuntos de corte de cardinalidad m que forma e con las demás aristas de la gráfica.

1: Sean $Cara_e$ y $Cara_{inv(e)}$ las caras a las que pertenece e
2: Realizar el recorrido de una de las caras, en este caso el recorrido se hará sobre $Cara_e$
3: **para** cada arista de $Cara_e$ diferente de e **hacer**
4: La arista se nombrará $candidato1$
5: Sean $Cara_{c1}$ y $Cara_{inv(c1)}$ las caras a las que pertenece $candidato1$, con $Cara_{c1} = Cara_e$
6: Realizar un recorrido sobre $Cara_{inv(c1)}$
7: **para** cada arista de $Cara_{inv(c1)}$ diferente de $candidato1$ **hacer**
8: La arista se nombrará $candidato_{m-2}$
9: Sean $Cara_{c(m-2)}$ y $Cara_{invc(m-2)}$ las caras a las que pertenece $candidato_{m-2}$, con $Cara_{c(m-2)} = Cara_{inv(c1)}$
10: Realizar un recorrido sobre $cara_{invc(m-2)}$
11: **para** cada arista de $cara_{invc(m-2)}$ diferente de $candidato_{m-2}$ **hacer**
12: La arista se nombrará $candidato_{m-1}$
13: Sean $Cara_{c(m-1)}$ y $Cara_{invc(m-1)}$ las caras a las que pertenece $candidato_{m-1}$, con $Cara_{c(m-1)} = Cara_{invc(m-2)}$
14: **si** las caras $\{Cara_e, Cara_{inv(c1)}, \ldots, Cara_{invc(m-2)}, Cara_{invc(m-1)}\}$ son diferentes y $Cara_{invc(m-1)} = Cara_{inv(e)}$ **entonces**
15: **devolver** e, $candidato1$, ..., $candidato_{m-2}$ y $candidato_{m-1}$ como un conjunto de corte de cardinalidad m
16: **fin si**
17: **fin para**
18: **fin para**
19: **fin para**

4.4.1. Corrección

El algoritmo 12 basa su corrección en el teorema 3.3.3 y el teorema 3.3.2. El procedimiento que sigue es: dada una arista $e = \{u, v\}$, primero se verifica cuales son las caras a las que pertenece. Después se realiza el recorrido de una de las caras ($Cara_e$) y se toman las aristas de esa cara (diferentes de e) como el *candidato*1. A continuación se verifica cuales son las caras a las que pertenece *candidato*1 ($Cara_{c1}$ y $Cara_{inv(c1)}$). *candidato*1 y la arista e comparten la cara $Cara_{c1} = Cara_e$. Ahora, para la cara $Cara_{inv(c1)}$ se toma una de las aristas diferentes a *candidato*1 ($candidato_{m-2}$) y se verifican las caras a las que pertenece ($Cara_{c(m-2)}$ y $Cara_{invc(m-2)}$). Por último, se realizará el mismo procedimiento que se realizó para *candidato*1 sobre la arista $candidato_{m-2}$, obteniendo así a $candidato_{m-1}$.

Al final, para saber si las aristas candidatas y la arista e forman un conjunto de corte de cardinalidad m, sólo se debe verificar si todas las caras a las que pertenecen forman un ciclo en la gráfica dual de G. □

4.4.2. Complejidad

La complejidad en cuanto a tiempo de ejecución del algoritmo 12 es de $O(n^m)$.

Para un candidato $m-1$ toma tiempo $O(n)$ saber a que caras pertenece. Para un candidato $m-2$ sólo pueden existir un número lineal de candidatos $m-1$, por lo tanto, revisar los conjuntos de corte a partir de un candidato $m-2$ tiene una complejidad $O(n^2)$. Para cada candidato $m-3$ existen un número lineal de candidatos $m-2$, por lo tanto revisar los conjuntos de corte a partir de un candidato $m-3$ tiene una complejidad $O(n^3)$. Por lo tanto, revisar los conjuntos de corte a partir de un candidato 1 tiene una complejidad $O(n^{(m-1)})$, y dado que existe un número lineal de candidatos 1, al final la complejidad total del algoritmo será $O(n^m)$. □

4.5. Detección de conjuntos de corte de cardinalidad mínima

Por último, se presenta el algoritmo 13, el cual, mediante una búsqueda exhaustiva, reportará el o los conjuntos de corte de cardinalidad mínima en una gráfica plana conexa.

El algoritmo tomará como entrada una gráfica geométrica plana conexa y para cada una de las aristas de la gráfica revisará si son aristas de corte, si es así, las aristas de corte serán reportadas y el algoritmo terminará. De lo contrario, para cada una de las aristas de la gráfica se reportarán (si es que existen) los conjuntos de corte de cardinalidad 2 a los que pertenecen, si ningún conjunto de cardinalidad 2 es encontrado, entonces se realizará la búsqueda de conjuntos de corte de cardinalidad 3.

Este procedimiento se seguirá hasta encontrar por lo menos un conjunto de corte, que para el peor de los casos, será un conjunto de corte de cardinalidad 5 (ver sección 3.2.3).

Algoritmo 13 Algoritmo que reporta los conjuntos de corte de cardinalidad mínima

Entrada: Gráfica geométrica plana conexa $G = (V, E)$, u tal que $u \in V$.
Salida: Reporta el o los conjuntos de corte de cardinalidad mínima en G.

1: $m \leftarrow 0$
2: **repetir**
3: $e_u = \{u, v\} \leftarrow eDominante(u)$
4: $a = (u, v)$
5: $e_{inicio} \leftarrow e_u$
6: $a_{inicio} \leftarrow a$
7: $m \leftarrow m + 1$
8: **repetir**
9: **si** e_u está siendo revisada por la izquierda **entonces**
10: **si** e_u forma conjuntos de corte de cardinalidad m **entonces**
11: Reportar los conjuntos de corte que forma e_u
12: Terminar el algoritmo
13: **fin si**
14: **fin si**
15: **si** $esDeEntrada(e_u)$ **entonces**
16: $e_u = \{u, v\} \leftarrow barridoRot(u, e_u)$
17: $a \leftarrow (a \cap \{u, v\}, \{u, v\} \backslash (a \cap \{u, v\}))$
18: **si no**
19: $e_u = \{u, v\} \leftarrow sucesor(e_u)$
20: $a \leftarrow (a \cap \{u, v\}, \{u, v\} \backslash (a \cap \{u, v\}))$
21: **fin si**
22: **hasta que** $e_u = e_{inicio}$ y $a = a_{inicio}$
23: **hasta que** sea reportado algún conjunto de corte

4.5.1. Complejidad

El algoritmo 13 tiene un tiempo de ejecución de $O(n^6)$.

Prueba. Para verificar los conjuntos de corte de cardinalidad k en la gráfica, se debe realizar un recorrido completo de la misma. El recorrido de la gráfica tiene una complejidad de $O(n \log n)$. En el peor de los casos, se deberán buscar conjuntos de corte de cardinalidad 1, 2, 3, 4 y 5 para cada arista. Por lo tanto, la complejidad total del algoritmo queda de la siguiente forma

$$O(n \log n) + n(O(n)) + n(O(n^2)) + n(O(n^3)) + n(O(n^4)) + n(O(n^5))$$

lo cual resulta en la complejidad de $O(n^6)$. □

Capítulo 5

Conclusiones

A lo largo de la presente investigación, se estudió un problema muy conocido en gráficas como es la identificación de conjuntos de corte, y se ha presentado una propuesta de solución local a nuestro problema planteado en el capítulo 1, el cual consistía en encontrar el o los conjuntos de corte de cardinalidad mínima en una gráfica geométrica plana conexa, de manera local. Los algoritmos locales que se presentaron a lo largo de este trabajo son:

Algoritmo Local (salida)	Complejidad
Una vez cada vértice y cada arista de una gráfica plana	$O(n \log n)$
Si una arista es ó no de corte	$O(n)$
Los conjuntos de corte de cardinalidad 2 a los que pertenece una arista	$O(n^2)$
Los conjuntos de corte de cardinalidad m a los que pertenece una arista	$O(n^m)$
Los conjuntos de corte de cardinalidad mínima de una gráfica plana	$O(n^6)$

Como una propuesta de trabajo a futuro, se tiene la implementación de estos algoritmos locales en redes físicas que necesiten de algoritmos de este tipo, como las redes de sensores o las redes *ad hoc*. Otra propuesta es la mejora en la complejidad temporal de los algoritmos, o en su defecto, la demostración de que estas cotas son justas para este problema en particular. Cabe señalar que, las propuestas aquí presentadas se basan en la búsqueda exhaustiva, por lo tanto, cualquier método que mejore esto mejorará la complejidad de los algoritmos presentados en este trabajo.

Bibliografía

[1] H. Attiya and J. Welch. *Distributed Computing.* Wiley-Interscience, 2004.

[2] M. Berg, M.V. Kreveld, R.V. Oostrum, and M. Overmars. Simple traversal of a subdivision without extra storage. *International Journal of Geographical Information Science*, pages 359–373, 1997.

[3] J.A. Bondy and U.S.R. Murty. *Graduate Texts in Mathematics, Graph Theory.* Springer, 1988.

[4] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Kluwer Academic Publishers*, 2001.

[5] L.R. Foulds. *Graph theory applications.* Springer, 1992.

[6] F. García and M.L. Puertas. El teorema de la curva de jordan. *Divulgaciones Matemáticas 6*, pages 43–60, 1998.

[7] D.S. Hirschberg and J.B. Sinclair. Decentralized extrema-finding in circular configurations of processors. *Technical Note Operating Systems, Communications of the ACM*, 1980.

[8] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. *IN PROC. 11 TH Canadian Conference on Computational Geometry*, pages 51–54, 1999.

[9] F. Kuhn, R. Wattenhofer, and T. Moscibroda. What cannot be computed locally. *23th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 300–309, 2004.

[10] F. Kuhn, R. Wattenhofer, and A. Zollinger. Asymptotically optimal geometric mobile ad-hoc routing. *Proceedings of the 6th International Workshop onDiscrete Algorithms and Methods for Mobile Computing and Communications*, 2002.

[11] F. Kuhn, R. Wattenhofer, A. Zollinger, and Y. Zhang. Geometric ad-hoc routing: Of theory and practice. *Proceedings of the 22nd ACM Symposium onPrinciples of Distributed Computing*, 2003.

[12] J. Li, C. Blake, D.S.J. Couto, H.I. Lee, and R. Morris. Capacity of ad hoc wireless networks. *7ma Conferencia Internacional de la ACM*, 2001.

[13] X. Li, S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic. Topology control in wireless ad hoc networks. *Mobile Ad Hoc Networking, chapter 6*, 2003.

[14] F.P. Preparata and M.I. Shamos. *Computational Geometry, An introduction.* Springer-Verlag, 1985.

[15] S. Rajsbaum and J. Urrutia. Some problems in distributed computational geometry. *6th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 1999.

[16] J. Suomela. Survey of local algorithms. *ACM Computing Surveys (CSUR)*, 2013.

[17] C.K. Toh. *Ad Hoc Mobile Wireless Networks: Protocols and Systems.* Prentice Hall, 2001.

[18] J. Urrutia. Local solutions for global problems in wireless networks. *Instituto de Matemáticas, Universidad Nacional Autónoma de México*, 2006.

[19] D. Wagner and R. Wattenhofer. *Algorithms for Sensor and Ad HocNetworks.* Springer, 2007.

[20] D. Wagner and R. Wattenhofer. Local algorithms. *Springer*, 2007.

Índice alfabético